A.M. Turing Award Oral History Interview with
Robert (Bob) Endre Tarjan

by Roy Levin

San Mateo, California

July 12, 2017

Levin: My name is Roy Levin. Today is July 12th, 2017, and I'm in San Mateo, California at the home of Robert Tarjan, where I'll be interviewing him for the ACM Turing Award Winners project.

Good afternoon, Bob, and thanks for spending the time to talk to me today.

Tarjan: You're welcome.

Levin: I'd like to start by talking about your early technical interests and where they came from. When do you first recall being interested in what we might call technical things?

Tarjan: Well, the first thing I would say in that direction is my mom took me to the public library in Pomona, where I grew up, which opened up a huge world to me. I started reading science fiction books and stories. Originally, I wanted to be the first person on Mars, that was what I was thinking, and I got interested in astronomy, started reading a lot of science stuff.

I got to junior high school and I had an amazing math teacher. His name was Mr. Wall. I had him two years, in the eighth and ninth grade. He was teaching the New Math to us before there was such a thing as "New Math." He taught us Peano's axioms and things like that. It was a wonderful thing for a kid like me who was really excited about science and mathematics and so on.

The other thing that happened was I discovered *Scientific American* in the public library and started reading Martin Gardner's columns on mathematical games and was completely fascinated. I started creating my own board games and other kinds of mathematical puzzles. I kind of switched my interest from astronomy and outer space to mathematics.

My dad was working at a state mental hospital at the time. He was a superintendent. It was originally Pacific State Colony, but they changed… Pacific Colony. They changed the name to Pacific State Colony. We were living on the grounds. It was for what are now called developmentally disabled people, who used to be put away in institutions and taken care of. He started a research group, a research institute there. Instead of just housing these people, he wanted to understand causes and how to actually make people better. Among

other things, he started a collaboration with Linus Pauling who was at Caltech in those days who was interested in the chemistry of certain kinds of brain problems, phenylketonuria in particular, which is caused by an inability to metabolize a certain amino acid. He came over to our house from time to time and he left a Caltech catalogue when I was at some young age and played mathematical games with me, which of course I could beat him at because they were weird games that he didn't know anything about.

Levin: ***[laughs]*** **But he was game?**

Tarjan: Yes, yes, yes. When I got to high school, I started actually working in this research institute that my dad had started at the state hospital doing calculations with punched cards. They weren't really computers. They were IBM machines that you feed in the punched cards and you could program them by punchboards. I mean we were doing statistical calculations, so I got started a little bit in computation doing that.

Then my junior year in high school I guess, between my sophomore and my junior year, I went to a summer science program in Ojai, which involved again astronomy actually. The main project for the six weeks we were there was to take photos through a telescope of an asteroid, measure the photographic plates, figure out the orbit of the asteroid, which involved calculation again, in this case on Marchant calculators, which were these old typewriter-like devices. It's amazing how much computation has changed since those early days. That involved understanding Newton's laws and so on and so forth, and doing the calculations. We also had the opportunity to go to UCLA and see a real computer. It was an IBM machine of some sort and we got a chance to program in FORTRAN. That was really eye-opening. I got interested in trying to solve two-person zero-sum games. I was reading some game theory stuff at the time and I started doing some programming there.

I was also interested in the four-color problem, which was open at that point, and I did some experiments. Eventually the hospital graduated from IBM punched card equipment to a Bendix G-15, which was a paper tape machine, and I programmed that. Then they got what I think was some kind of Honeywell machine that was really something that you could program on, and I programmed in FORTRAN and I played around with trying to solve the four-color problem. Of course, my technical skills and the power of the machine were not good enough in those days, but it was the same techniques that Haken and Appel eventually used to solve the problem, so that was interesting.

Then when I graduated from… Well, my senior year in high school in Pomona, I ran out of the math classes in high school so I took some math classes at Harvey Mudd. Then I was off to college. It was going to be either Harvey Mudd or Caltech. I ended up going to Caltech, majoring in mathematics. Don Knuth was there, but I didn't get a chance to take a class from him. He left for Stanford fairly

early on when I was there. I was at Caltech from ’65 to ’69. But there were other people working in combinatorics, Herb Ryser in particular, with whom I worked quite a bit. I started getting interested in matching problems and network flow problems, in those days took all the mathematics courses that I could along with the… it wasn’t really computer science in those days but there were automata theory courses and so on.

I had another interesting professor who taught automata theory, Giorgio Ingargiola, who was this flamboyant Italian who prepared voluminous class notes that were riddled with errors. Some of us went through carefully and tried to find all the errors. There’s nothing like learning material by identifying other people’s errors.

I graduated from Caltech in mathematics and it was a question of whether to go to… Ever since I was a kid, I wanted to be a scientist. I wanted to do cool scientific stuff, both because I thought it was interesting and because I guess my goal was to become famous somehow. But I was interested in computers and I was interested in mathematics, so I applied both to mathematics departments and to computer science departments. I got in various places, eventually went to Stanford, decided to go to Stanford in computer science in 1969, which was the same year that Ron Rivest went there and several other amazing people. It was just marvelous because it was a whole new world to explore.

Levin: I want to talk about that in quite a bit of detail. There were indeed a number of people there I wanted to follow up on. But you’ve raised some quite interesting things that I’d like to follow up on before you got to Stanford. You said that your earliest experience was really being introduced to technology or shall we say technical things at the public library.

Tarjan: Yes.

Levin: Now roughly how old do you think you were?

Tarjan: I don’t… First grade probably. Age six, something like that.

Levin: So it was really going on…

Tarjan: I mean I started out in the kiddie section, but eventually I migrated into the adult science section.

Levin: And astronomy was an early interest. Was it sort of the descriptive aspect of it or do you think the mechanistic aspect of it that appealed to you? Or maybe both?

Tarjan: It was the adventure aspect, the sci-fi side of it. It was not the technical part.

Levin: So that was the "going to Mars" part?

Tarjan: Yes, yes, definitely.

Levin: Got it. And this notion of wanting to be a scientist from early on, that's intriguing to me. Where do you think that came from?

Tarjan: Well, I think my father always wanted to be a scientist, but he ended up as an MD. He certainly valued intellectual achievement. He was an Eastern European, Hungarian émigré. [0:10:00] I think that was a lot of it. And it was a small neighborhood. I spent a lot of time by myself. I spent a lot of time reading. So between my dad and my natural interests…

Levin: But it sounds like he encouraged you quite a bit – the contact with Linus Pauling for example that came through him.

Tarjan: Yes, yes, yes.

Levin: It sounds like he saw that this was something that appealed to you and interested you and supported it.

Tarjan: Yes. Also Mr. Wall. I mean teachers… I mean parents have a huge influence, but teachers too. This particular middle school math teacher was a really important influence on me.

Levin: Were there any other people that you can think of who were early influencers? Maybe family members or friends of the family?

Tarjan: No. Some of the people I met through my work at the state hospital, the director and various other people there: Harvey -- Mr. Dingman. I mean they had this kid, this teenager who was helping them with all this stuff. I mean it was really a wonderful thing for me. They were very tolerant and open and willing to let me play around.

Levin: Wow, that's great. This is the way these things begin to happen. I guess the other thing I wanted to ask you about was when you were doing this sort of quasi-programming job before you got real computers I think you said in high school…

Tarjan: Yes. And through college. I kept working at the state hospital all the way through college. That only ended after I graduated and went to Stanford.

Levin: Okay. But I'm thinking of just before you really got an opportunity to actually program something. I'm not counting the plugboards. Do you recall any sort of frustration at that point with the sort of primitiveness of the tool? Or was it, compared to the calculators that you had to use, it was such an improvement that that wasn't a concern?

Tarjan: It was certainly frustrating, but on the other hand, there's a huge jump going from programming with a plugboard to even programming in machine language or assembly language even if it's on punched tape. You've got so much freedom to express your ideas. So I'd say the opportunity trumped the frustrations. Then once we got the ability to program in FORTRAN, that's another jump of another level, because you get another level of abstraction.

Levin: Exactly. Did you teach yourself FORTRAN or was there some more formal way to learn it?

Tarjan: Well, I encountered FORTRAN in the summer science program. They gave us some books. I think McCracken had written a book on FORTRAN programming.

Levin: A classic book, yes.

Tarjan: Yes. I think we had a few lectures on it, but I read the book and then started trying to write some programs. So it was some combination of some formal training and me learning it myself.

Levin: Did you have any classmates in school who had similar interests with whom you would work on these things? Or was it mostly a solo pursuit?

Tarjan: It was mostly a solo pursuit all the way through high school. Yeah.

Levin: Interesting. Well, you were a driven fellow it sounds like.

Tarjan: *[laughs]*

Levin: *[laughs]* Let's come back to your undergraduate days at Caltech. It's interesting: were you aware of Knuth's, shall we say, residence at Caltech at that time even though you didn't take a course?

Tarjan: Only after the fact, interestingly.

Levin: I see. When you subsequently ended up working with him.

Tarjan: I didn't know who he was. I mean he was this professor in the math department, but… I mean he hadn't written his books yet, even though the

thought was in his mind I think. At some point, he made the decision that he wanted to help found a computer science department and it didn't seem to be happening at Caltech. This is probably a good question to ask Don at some point, but he moved to Stanford to this brand new computer science department. That's where I ended up going, but it was completely coincidental.

Levin: Huh, that's interesting. I was trying to remember when the first volume of his series came out. I think it was in the late '60s.

Tarjan: I believe it was mid to late '60s, I'm thinking '68 or something like that, yes.

Levin: Okay. That must have been…

Tarjan: I think this would have been about the time or just after he had moved to Stanford. I guess he must have been writing the volume…

Levin: He must have started working on it at least at Caltech, even if he hadn't published it by then.

Tarjan: Yes.

Levin: Okay. Well, but that's certainly something to pursue with him at some point.

The undergraduate courses that you were talking about sounded like they were not exactly computing but they were things for which computing obviously became very relevant, in things like… you talked about net flow problems and courses in automata theory.

Tarjan: Yeah. Well, the network flow stuff I was working… I wrote a paper with Herb Ryser on matching in various kinds of problems. I think at some point I looked at the wonderful book by Ford and Fulkerson on network flows and got some ideas out of that book. Mostly I want interested in combinatorics. I was also playing around with computability, automata theory, stuff like that. So really focusing on the algorithms, other than the stuff that I'd worked on related to the four-color problem. That came in grad school in fact.

I had… Well, so what happened was I got offered a Hertz Fellowship, which I accepted, to go to Stanford, and it came with the option to go to Lawrence Livermore Lab to work there for the summer, again doing computation, in this case simulating star interiors or hydrogen bomb interiors or something like that. But I had to get a security clearance, so the first month I sat in the library reading more stuff. Finally the security clearance came through and I got to actually start working and doing some programming and so on.

When I got to Stanford, well, I was interested in automata theory, but Stanford had John McCarthy and Ed Feigenbaum, these gods of artificial intelligence, so I kind of thought I would maybe do artificial intelligence. But what I was really thinking about was more complexity/computability. I remember a conversation with Michael Arbib my first year at Stanford. He was in the EE department, not in the computer science department. He said, "Don't do automata theory. Do complexity," so I started thinking about other stuff. I took some classes and I decided that at that point, artificial intelligence was too fuzzy, that it wasn't mathematical enough. I took a class with Bob Floyd, which was all about problem solving, algorithmic problem solving.

In my first year, I pretty much passed all the requirements needed for the PhD except actually doing the research and writing the thesis, so that kind of set me up for the next step. I talked to Don at that point. I mean Don at Stanford was another god by that time. I mean he became very important very fast in the field. Talked to him about taking me as a student once I decided I didn't want to do artificial intelligence. But he had a policy of taking two students every other year. I was in the off year. *[laughs]* So I ended up with him not as my advisor. Eventually Bob Floyd became my advisor.

But after my first year in grad school, John Hopcroft came, on sabbatical. I think he came in the summer after my first year in grad school and I was all set to start doing research because I'd finished all the requirements. Then we started talking about algorithms and algorithmic problems and so on. I had to…

There's another story I should tell, which is I took a course from John McCarthy in LISP programming. There was a final project, which required implementing some non-trivial calculation or algorithm in LISP. One of the possibilities he suggested was planarity testing. He mentioned the famous criterion for "A graph is planar if and only if it does not contain five vertices all pairwise connected or two sets of three vertices all pairwise connected." Some people in the class tried to implement this test, but nobody succeeded in getting something to work. I was doing some reading on this problem. I found a wonderful paper by Lempel, Even, and Cederbaum [0:20:00] in which they described an algorithm for testing planarity in terms of formula manipulation. I implemented that algorithm in LISP and it was really fast. In fact, it ran in quadratic time. So I was the only one who got something to succeed on this particular problem, so I was kind of primed to look at planarity testing and graph search and stuff like that.

So we started talking about graph search and applications of graph search, and…

Levin: You and Bob Floyd at that point?

Tarjan: John Hopcroft.

Levin: Oh, Hopcroft. Excuse me.

Tarjan: Floyd came into the picture later because I needed an advisor. But John was there. We ended up sharing an office, as I recall, and we started working on graph search. He was an engineer, so he took a very engineering problem-solving perspective, but I was trained as a mathematician, so I wanted to sort of develop a theory – "What is graph search? What is depth-first search? What can you do with it? What does it do to a graph?" Anyway, between the two of us, we eventually… well, we started out by using depth-first search to find biconnected components.

Then I took this idea and went back to the planarity testing problem and worked on planarity testing for a year. We started out with a… I mean we were trying to get a fast algorithm. We knew that the Lempel–Even–Cederbaum algorithm took quadratic time in a number of vertices. I found another technique, which was… I'm not sure who developed it first. I think Shirey at Wisconsin. But the idea was to find a cycle and then remove the cycle from the graph and look at the connected pieces and figure out how to embed the connected pieces either on the inside of the cycle or on the outside of the cycle so it all fits together and is consistent. There's a natural recursive structure that you get out of this and we were trying to find a way to implement this algorithm using depth-first search.

So we played around with it. We got a rather complicated algorithm that ran in order n log n time, but there was no reason to believe it couldn't be done in linear time. So I spent most of a year dreaming a lot about ways to try to make this thing work in linear time and eventually managed to solve it. That turned into my PhD thesis.

So John was really my de facto advisor. But he was a Cornell professor. He was only on leave, so I needed an actual advisor and talked to Bob Floyd and he agreed to do it. Which was a wonderful thing, except that he was an amazing perfectionist. He read my thesis with a fine-tooth comb and corrected the grammar and all kinds of other stuff. Cost me an extra three months or something like that. But so be it. It was worthwhile.

So I kind of raced through grad school as it turns out, but it was a great time to be doing algorithms because there was all kinds of unsolved problems. This idea of… Knuth had developed the idea of analysis of algorithms, concrete analysis of algorithms, but he paid attention to constant factors. John and I decided that if we want machine independence, we should ignore constant factors. This turned out to be a very powerful idea.

Levin: Absolutely. My recollection of first being exposed to Knuth's work was in the context of analyzing not just algorithms but really programs. So the constant factors were the issues and then some abstraction…

Tarjan: And sometimes not even constant factors. Sometimes additive factors.

Levin: Yes, that's right. Taking that step back that you and John Hopcroft took seems like it made a huge difference to the kind of analysis that was both possible and ultimately more useful.

Tarjan: That's what it seemed to us at the time and it certainly led to a lot of interesting results.

Levin: That seems like a very important pivot point in terms of the work that you ended up doing. Do you remember at all how you and John came to this conclusion? Was it just sort something that you're sitting around one day talking? Or how did you get there?

Tarjan: I don't recall. That's a very good question. I think we decided that… We weren't doing things like sorting problems. We were doing things somewhat more abstract. I mean dealing with graph algorithms. And we were worried about the order of magnitude of growth, the asymptotics, and it just seemed like if you ignore the constant factors, just worry about the growth rate… If you think about it, you want to solve big problems, so as the problem grows, the asymptotics become more and more important -- the constant factors become less and less important -- and if you can abstract away the constant factors, you save yourself a lot of grief. So it's kind of a first cut, but it's a very important first cut and maybe it's enough in any situation.

Levin: Well, it seems evident today.

Tarjan: Yes, yes, yes.

Levin: But you're talking about the early '70s here and it wasn't so evident. People were much more concretely focused I guess.

Tarjan: I guess. I don't know that I want to claim credit for the idea, but it was certainly in the air and it seemed like a natural approach to us for what we were trying to do.

Levin: Indeed. I'd like to, before we move on to what happened after Stanford, you say you raced through it, but there still were a few other things that got done there.

Tarjan: That's true.

Levin: In particular, one of the things that I thought was intriguing was a paper that you ended up writing with Manuel Blum, Bob Floyd, Vaughan Pratt, and Ron Rivest. All those authors were listed as being at Stanford,

although I'm guessing that Blum might have been visiting because I think he was actually on the faculty at Berkeley.

Tarjan: Yes, he was.

Levin: But there are four eventual Turing Award winners for different things in that list of authors, so it's pretty interesting to go back and look at a paper from the early '70s on that topic. How did that come about, other than the fact that you were all there at the same time?

Tarjan: Well, Manny Blum came to visit. There was some constant back-and-forth of various people between Stanford and Berkeley. I mean Dick Karp was at Berkeley at the time. He came and we went up there. But Manny came to Stanford as I remember it and he had some ideas to… The question is "Can you find the median of a set of numbers faster than by just sorting the numbers and picking out the one in the middle?" "There's this well-known n log n lower bound for sorting. Does it apply to median finding as well?" that was the question. Manny had some ideas to use recursion in a clever way to show that you could do better than n log n time. He and Bob Floyd started talking and they had an idea that gave a log n log* n algorithm actually, log* being the number of times you have to logarithm to get down to 1. So very slowly growing but not constant.

And Rivest was working with Bob Floyd and he got in the conversation. And I was a good friend of Ron's and we started talking about this problem. The question then became "n log* n? Why not linear?" So the two of us play with it pretty intensely and eventually figured out how to tweak the recursion to get a truly linear bound. And Vaughan got involved also. At that point, it was a matter of tweaking the constant. This is an example where we actually paid attention to the…

Levin: I noticed that in the paper.

Tarjan: …constant factors. The way it's taught, it's a very simple but subtle idea to use recursion in two different ways and then you can get the linear time. But we went to some trouble to try to push down the constant factor. I think we got 5.43 or something like that as an upper bound. That was subsequently improved. The exact constant for that problem is still not known. But yeah, that was maybe over-optimization I would say.

Levin: Well, I went and read this paper recently in preparation for this interview, and I remember there's a comment in… It's not that long a paper, but the comment appeared twice in the paper where it says, "We're pretty confident that this number is not the lower bound."

Tarjan: *[laughs]*

Levin: So whatever tweaking had been done, obviously you thought there was room for more.

Tarjan: Definitely, definitely. Ron and Bob Floyd worked on an average time algorithm, and I think they got the exact constant. I think it's one and a half *n* [0:30:00] on average, and it's not clear for the worst case, somewhere between 3 and 2, less than 3 and greater than 2, and there's a gap.

Anyway, so my big passion is ignore the constant factors, but there are times when I've paid attention to constant factors and times when I've ignored log factors and stuff like that, which is I would say more of a trend than it used to be. If you're doing complicated calculations, people are willing to give up log factors and polylog factors and so on.

Levin: That's an interesting trend.

Tarjan: It's kind of a mixed bag. Well, here's what I think. I've always… I like simplicity and I like algorithms that can actually apply to real-world situations and fundamental problems. I think when you start ignoring log factors and polylog factors, you may end up with beautiful theoretical results, but if you take log of a billion, that's 30; $\log^2$, that's 30 times 30, that's 900. These factors start to get pretty large, so maybe not so practical. I mean the goal is to come up with good algorithms that you can apply in practice that actually run fast as well as being simple, beautiful, and analyzable.

Levin: I guess the moral is "You should ignore these things when they really don't matter, but not when they do." ***[laughs]***

Tarjan: It depends what your goal is.

Levin: It just depends what you're trying to do.

Tarjan: If you're trying to understand the theory, that's one thing, but if you're trying to actually get something to run… I've always been more interested in upper bounds than in lower bounds, although I dabbled in lower bounds too. For one thing, upper bounds are easier to get because all you have to do is design one algorithm and analyze it. If you want to prove a lower bound, you need to prove something about all possible algorithms. This is not so easy.

Levin: So St-…

Tarjan: Yeah…

Levin: I'm sorry, go ahead.

Tarjan: Yeah, that median finding paper, that kind of came at the end of my time at Stanford. That was a lot of fun though. I mean there was this ferment, the Berkeley–Stanford axis, everybody was talking about all these interesting problems and understanding that this idea that you could really analyze algorithms from a theoretical point of view, this was becoming very important. I mean the notion of NP-completeness emerged around that time. I think Cook's paper was maybe 1971.

Levin: So it's right in that same time.

Tarjan: So we started talking about that too – "What does this mean? What can be done with it?"

Levin: And the fact that Hopcroft was there visiting from… Was he on a…?

Tarjan: Cornell.

Levin: At Cornell at that point, yeah.

Tarjan: Yes.

Levin: For that same period of time, was that just sort of lucky coincidence? Or was it the fact that in the early days of the department, they were really trying to get people from more established departments to visit?

Tarjan: Well, I was a lowly grad student at that point, so I don't know the politics of how he came to Stanford. But he was a Stanford… I think he was a Stanford PhD actually.

Levin: Yes, I believe that's correct.

Tarjan: So I think he liked Stanford and that was a good reason for him to visit, and it was a great department. It was a very productive year for him.

Levin: Another happy coincidence about being people being there at the same time.

Tarjan: Yeah, definitely.

Levin: Did you and John continue to work together after the time at Stanford?

Tarjan: Yeah. Well, what happened was he went back to Cornell and I was ready to apply for a job, so I applied to many places. Eventually ended up taking

an assistant professorship at Cornell. For one, well, that was a great department and John was there, and I wanted to continue working with John.

Unfortunately the fact that Bob Floyd… Well, there were two things that happened. Stanford had a nine-term residency requirement, a term being a third of a year. I had to be in residence or else pay tuition for nine terms, which was two and a quarter years, which is kind of an odd thing. My PhD was more or less done after two years, then I thought I would be done at the end of the summer, but Bob Floyd took it apart and we put it back together again, so it cost me an extra three months. But that was okay because of this nine-term business.

Then I ended up delaying going to Cornell until the middle of winter. This was after two full years plus a little more at Stanford. I drove my fire-engine-red Mustang from Los Angeles, where my parents were, to Ithaca, New York in the middle of the winter. It's got rear-wheel drive, stuff is floating out of the sky. "What's this white stuff? Oh, and I can't…" My experience at Cornell was good professionally but not so good personally. Being a Southern California boy and going to a place like Ithaca, New York in the middle of winter is quite a shock.

Levin: Indeed. But you survived for a couple of years.

Tarjan: I survived for a couple of years. I more or less immediately decided I was not going to be able to tolerate the weather. I thought about coming back to California after the first semester, and I could have come back to Berkeley, but I decided that was not good form. So actually I waited till the next year. I spent a year and a half at Cornell and then I came back to Berkeley. I actually gave up an assistant professorship at Cornell and took a postdoc at Berkeley. They have something called a Miller Fellowship, which pays the equivalent of an assistant professorship but has no responsibilities.

Levin: That's pretty good. This still exists?

Tarjan: And it's a two-year… It still exists. It's a wonderful thing.

So I spent a year and a half in Ithaca, enjoying the department, suffering through the weather, doing more work with Hopcroft. I mean we took depth-first search and we tried to apply it to many problems. We did some more stuff with planarity testing, testing isomorphism of planar graphs. But then I couldn't deal with the weather at that point, so I retreated to California. The day I left, which was in May, big puffy white things were falling out of the sky again, so somehow I took this as a sign. Ithaca's a beautiful place. My wife majored in operations research in Ithaca actually. It's a great department. It's a little too isolated for me somehow.

Levin: She survived the weather I guess better.

Tarjan: Yes, and her family was in Ithaca too, so that helped. She lived at home.

Levin: Well, that changes the situation.

Tarjan: Yeah. And I think being an undergraduate there… Being a single, unmarried, fresh assistant professor, I was younger than most of the grad students at that point, because I went through Stanford really fast. It was very odd. I hung out more with the students than with the professors.

I would say the most interesting thing on the technical front besides the follow-on work that John and I did together was I got started on the disjoint set union problem at that point. There was an IBM workshop, Complexity of Computation or something like that. I forget the exact date. I think 1973. It was spring, maybe spring of 1973, 1973 or 1974. It was spring. I drove from Ithaca to… It was at T.J. Watson I guess, so this is suburbs of New York. It was spring down there. It was still winter in Ithaca. I noticed that.

But anyway, this problem, keeping track of disjoint sets using union find and path compression and so on, the question was how do you analyze this incredibly simple algorithm? Knuth had posed this as an open problem with the assumption that this thing actually runs in linear time, constant time per operation. In the amortized sense, which means that if you do a sequence of operations, [0:40:00] the total time for the operations is linear in their number, even though an individual operation may take more time than some of the other operations.

There was a false proof of a linear time bound, which Mike Fischer, who I think was at Yale at that point or he might have still been at MIT, he found the bug in the proof and he got an $n \log \log n$ upper bound, which he presented at this IBM workshop, which is also where Karp presented his famous paper on many different NP-complete problems. It's interesting that the Cook paper was published in STOC, Symposium on Theory of Computing conference, the Karp paper was published in this very obscure IBM workshop. Neither of these things made it to a journal somehow. This is a bad thing about computer science – a lot of great stuff does not in fact get published in journals.

Anyway, there were some amazing authors and papers in this workshop, including those two, and Karp's was the most important. But Fischer had this paper on this problem, so I took this problem back to Ithaca and started working on it, thinking about… And Hopcroft and Ullman were working on it at the same time. They managed to improve Mike Fischer's upper bound to a log* per operation upper bound. Again, log* is the number of times the log function has to be iterated to get down to 1.

So linear versus log*. Everybody was trying to beat down the upper bound. I asked the question "What if it's not linear? How would you prove that?" You

need to construct an example. For any fixed value *k*, you need to construct a problem in which you can do a linear number of operations, each of which takes *k* steps. If you can do that for all possible *k*, it can't be linear. I started playing with examples and I was able to get a double-recursive construction that showed that indeed for all possible *k*, there's a big enough *n* such that there's an example of size *n* where you can do linear number of operations all costing *k*. The growth rate of *n*, the recursion looks just like the Ackermann function, which is this crazy function that Ackermann invented way back when to show that primitive recursion is not powerful enough to do all possible computation. General recursion is more powerful than primitive recursion, and it's a diagonalization over the primitive recursive functions. So here's this crazy function. If you turn it around, that means that the running time of this algorithm is not linear, not constant per operation. It's at least inverse Ackermann function.

Then the natural question is "Is this right? Or is it log*? Or is it something crazy in between?" At that point, that's kind of when I moved to Berkeley. I spent a year or year and a half trying to prove that that bound was tight, because it's more interesting than linear or log*, because nobody had ever seen this function in the analysis of an algorithm at that point. Eventually I managed to prove there was an upper bound as well as a lower bound. That I thought was very, very cool. I was very proud of that accomplishment. Worked hard for a very long time on that problem.

Levin: That was one of the I guess infrequent examples of you working on lower bounds.

Tarjan: Yes, yes. It was a lower bound for a specific algorithm, although I later generalized it to a class of pointer algorithms. Then Mike Fredman and I think Mike Saks figured out how to prove it for an arbitrary random-access machine, algorithms in cell-probe model. So the lower bound holds in a very strong model.

Levin: As you said, this is the first time that the Ackermann function, or in this case its inverse, had turned up in the analysis of a real algorithm. Did that turn out to be the start of anything? Or is this some weird, isolated case?

Tarjan: Well, it turns out to be relevant to various computational geometry configurations. If you're looking at envelopes of intersecting functions, be they linear functions or quadratic functions in one dimension or several dimensions where you got a surface, the combinatorics of these things give rise to inverse Ackermann function and their variants. Micha Sharir and a number of other combinatorial… computational geometrists figured this out. That's another important place where this function comes up.

Levin: But this was the first time?

Tarjan: Yeah, yeah, yeah.

Levin: Interesting. Very interesting. I want to go back a little bit. You were talking about after having come back to the Bay Area you were at Berkeley on this Miller Fellowship, for a couple of years was it, I think?

Tarjan: Yes, two years.

Levin: But you also were spending time at Stanford during that, right?

Tarjan: Well, after the first year of the Miller Fellowship, people started suggesting I should become a professor again. Stanford was ready to offer me a professorship. And Berkeley too – they said they'd give me an acting associate professorship. Stanford offered me an assistant professorship. I talked to a lot of people and ended up accepting the Stanford offer but delaying for a year so I could finish the second year of the Miller Fellowship. So I did two years at Berkeley already knowing I was going to go to Stanford as an assistant professor. That's what happened.

Levin: You were then at Stanford again for several years.

Tarjan: About, yeah, five years. I was there as a student for two years and a quarter, and I was there as an assistant associate professor for about five years. Had great students. The best thing about being a professor at Stanford was the amazing students.

Levin: Who were some of your collaborators during that time?

Tarjan: Well, they were mostly my students. Danny Sleator comes to mind. Very creative guy. Tom Lengauer also, who's now in Germany, with whom I did the dominator algorithm, which is used in a lot of code optimization compilers, the code optimization part of compilers. But yeah, the student I worked most closely with, especially after he graduated, was Danny, but I think I had eight students, eight PhD students finish in the five years I was there.

Levin: Did you find that the students led you into areas that you might not otherwise have chosen to explore?

Tarjan: To some extent, I would say. We just kind of explored algorithmic-type problems together on a curiosity-driven basis, I would say. Lengauer is an interesting case because -- a very smart student. We worked on the graph algorithm stuff a little bit, but he ended up doing his PhD thesis on a complexity-related problem that has to do with pebble games actually. Then later on, he switched completely. He went into computational biology. Most of his career has been spent in computational biology. So we were dabbling in this and that, mostly centered around algorithms, but with the occasional diversion.

Yeah.

Levin: During the years that you were a professor at Stanford after your Miller Fellowship, you were just saying that you had a number of great students and that it was curiosity-driven research that occurred during that time. Can you highlight a few things that particularly stand out during that period that you worked on, either with your students or separately?

Tarjan: Well, I mentioned the pebbling results with Tom Lengauer, which were incredibly complicated combinatorial arguments. I got involved with Wolfgang Paul, who was a German professor, at that point because he was working on the same problem basically. We ended up collaborating a bit, which was… I love to collaborate with people. It's actually much more fun than working on one's own, shall we say. So I enjoyed that collaboration. I enjoyed the work with Lengauer on the dominator-finding algorithm. That was actually something I worked on back when I was a postdoc [0:50:00] at Berkeley. I had an algorithm, but we managed to improve it and simplify it to the point where it was easy to implement and amazingly fast. It used an extension of the disjoint set union data structure, so it in fact had the inverse Ackermann function time bound.

Then I was working with Danny on network flow algorithms, which was kind of a revisit of things I thought about back in my undergraduate days. But there was a beautiful paper that Edmonds and Karp wrote in 1972, I think, on finding maximum flows in networks. Then there were some improvements by various Russians and I started… I mean we were looking at all possible algorithmic problems to see what we could do in the way of getting improvements in the asymptotics.

So we started looking at the maximum flow problem and it was clear that underlying it was a data structure problem. I think the most recent result before what we ended up doing was by Zvi Galil. But the issue is you have a network. You want to send some material from a source *s* to a sink *t* through the network. Each edge has a capacity. You can't exceed the capacity of the edge. Any vertex in the middle, the amount of stuff coming in has to equal the amount going out. The classical Ford–Fulkerson algorithm finds a path along which to send flow, sends as much as possible so one or possibly more edges get saturated, finds another path, and keeps doing this until there are no more paths.

Now to make this correct and to make it work, you have to consider re-routing as well as pushing flow. Re-routing means if I send some flow through an edge from *v* to *w*, I can reduce the flow and that gives me a net flow from *w* to *v*. Every time I push flow in a forward direction, that creates the possibility of pushing flow in a backward direction and undoing what I did before. This gives the definition of augmenting paths, which were the basis of the Ford–Fulkerson algorithm, which is not a polynomial-time algorithm necessarily. All they proved was that

each augmentation gives you one increment of flow, but if you get big numbers, this could run for a long time.

Edmonds and Karp among other things proved that if you choose paths cleverly, namely always choose an augmenting path with the fewest edges, then you get a polynomial time bound. But the next question is how do you implement this algorithm? How do you keep track of which edges have flows on them and where you can increase the flows?

If you think about it for a little while, you end up with a set of trees, rooted trees where there's a set of paths that you can send flow through that come into a root, and then the goal is to connect these trees together so you get a complete path from the source to the sink. You end up with a collection of trees. You want to extend from the root of one tree to another tree and keeping doing that. That's a linking process of trees. When you finally get a connection from *s* to *t* and you send flow through it that saturates one or more edges, those edges get cut.

Now you've converted this algorithm into a data structure problem – maintain a collection of trees under link and cut operations. Then we started looking at how do we come up with an efficient data structure to keep track of these trees and the linking and cutting? So this is an extension of doing it for paths. If you don't have the branching problem, you just do it for paths, you can represent each path by a binary tree and get a log time solution per operation.

Danny and I played around with this and we discovered that you could represent these trees by trees of binary search trees, sort of a two-level data structure, and that gives a $\log^2$ solution if you use standard balanced binary trees. That was a reinterpretation of what Galil had already done. But then if you come up with a really clever implementation of… at the bottom level the search trees that exploits knowing where they are in the top-level structure, then you can do better. So we invented a data structure called a biased search tree where certain nodes are easier to access than other nodes, and we were able to get…

We got two results actually. We got this biased search tree data structure where the access time depends upon the access frequency or some weights or something like that. We were able to use that in this link-cut tree data structure to get a log time solution implementation of the… Edmonds, Karp, and also a Russian named Dinitz had more or less invented the same idea, so we implemented their algorithm and we got a fast network flow algorithm. That started us thinking about… There was one piece of the analysis in which we were not looking at individual operations. We were bounding things over the entire set of operations, this notion of amortization, which also came into play in the union-find algorithm.

Danny got his PhD. He went to Bell Labs. I had started collaborating with people at Bell Labs when I was still at Stanford, Mike Garey in particular and

Dave Johnson. They visited Stanford a lot. Ron Graham came to Stanford a lot. He brought Paul Erdős. It was amazing the number of people who were around. Anyway, I started visiting Bell Labs. Danny went to Bell Labs. I was up for a sabbatical. I took a sabbatical. I went to Bell Labs for a year. Then they made me a permanent offer and it was an offer I couldn't very well refuse, so I ended up staying at Bell Labs then. And that was the move from the west coast to the east coast.

Levin: East coast weather again. ***[laughs]***

Tarjan: *[laughs]* But New Jersey…

Levin: New Jersey is different.

Tarjan: …which is a lot milder than Ithaca, New York.

Levin: That's true. That is true.

Tarjan: Yes. I moved to… Well, I started at Bell Labs in 1980, September of 1980 I guess. It was the tail end of the heyday of Bell Labs. It was before the first breakup of the phone company. There were all kinds of great people there. Unix was happening.

Levin: ***[brief, inaudible]***

Yeah, yeah. It was fabulous.

To continue the story with Danny, we understood that this notion of amortization was important and we had this complicated data structure, this biased search tree kind of thing. Why not try to apply this idea to a search tree? That is, can you develop a search tree where the operations are not worst-case efficient but efficient in the amortized sense, so if you do an expensive lookup, the tree kind of readjusts itself so that the next operation on the same element is going to be cheap? That's the question. Start with the easier proposition. Forget search trees. What about linear lists?

There had been some work actually by Ron Rivest and various other people analyzing the behavior of simple self-adjusting linear list schemes, in particular move-to-front. So a simpler way to implement a table. You can implement a table as a hash table or a binary search tree or even a linear list with sequential search. You just walk down the list until you find the item that you want. The move-to-front rule says, "When you find the item, move it then to the front of the list."

Now let's suppose you're in a situation where you got a whole bunch of items and they're accessed with different frequencies. The ideal situation would be you

figure out the access frequencies, you put the most frequently accessed item on the front and so on and so forth, build the optimum list. But what if you don't know the frequencies or they're changing over time, there's some kind of locality of reference, there's some context? You're looking at this group of items for a while and then this other group, so you want to move the other group up to the front and so on.

So move-to-front has really good behavior and Rivest for example analyzed it in a probabilistic model. But we brought this notion of amortization to the problem and looked at the amortized performance of move-to-front. We were able to prove that in fact, as compared to any algorithm among a certain class, no matter what the input sequence is, move-to-front comes within a constant factor of the best algorithm even knowing the entire future of the access sequence. This was a fairly powerful result…

Levin: Yeah, I can believe it.

Tarjan: …in a rather simplistic problem.

So it was a nice result. Then we went back to the binary search tree situation. We were asking the question, "What about binary search tree? Is there some analogue to move-to-front?" We played with that for a while and first attempts failed. But then we invented the splay algorithm, which in fact has properties much like move-to-front, gave us the splay tree data structure, for which we actually eventually won the Paris Kanellakis Award. Because it's a very simple data structure, there's no explicit balancing, this thing adjusts to match its usage, and we had this very audacious conjecture, which is essentially that splaying performs as well on binary search trees as move-to-front does on linear lists. Namely, for any binary search tree–based algorithm, you can't beat splay by more than a constant factor on any access sequence, even if you know the access sequence in advance. That is, knowing the future can't help. If this is true, it means that splaying in some sense is a general-purpose binary search tree algorithm, very simple. That problem is still open after 35 years or something like that. That's my favorite open problem, prove or disprove the dynamic optimality conjecture.

Levin: There's been plenty of work on it during that time I imagine. ***[phone rings in background]***

Tarjan: Yes, yes.

[As a result of the disruption from the ringing phone, which has been edited out, Tarjan revisits and expands on his discussion of splay trees.]

Tarjan: So we played around with algorithms for search trees, trying to reproduce on search trees the same behavior we had with move-to-front on

linear lists. So search trees, the local restructuring operation is a rotation which kind of changes things on a local area. We tried strictly bottom-up, strictly top-down, didn't work. But then if you do the rotations in pairs, we came up with a strategy that we called splaying, one definition of which is to spread out, which basically moves an item all the way to the root, moves everything on the search path about halfway toward the root, and spreads everything else out a little bit.

So we developed this self-adjusting search tree, which we were then able to prove had logarithmic worst-case… amortized time behavior. Standard balanced search trees have log time worst-case per operation. This data structure had log time amortized behavior. There could be expensive operations, but they're compensated for by cheap ones. But not only that, we were able to show that for various imbalance situations, biased situations, the data structure behaved very well. In particular for our network flow application, we could plug the splay tree in and we got the log time performance overall and we got rid of the complexity of the biased search tree data structure.

Then we made the audacious conjecture that this splay tree is essentially universal among binary search tree algorithms in the sense that if you're given a sequence of operations and you the adversary know the sequence ahead of time, no matter what algorithm you try, the splay algorithm, which does the operations one at a time in very simple fashion, will perform just as well to within a constant factor as the optimum algorithm.

This is the dynamic optimality conjecture. It's been open since we wrote the paper, which was early-mid 1980s, so 30 plus years. I think it's the most interesting open conjecture in the area of data structures. People have made some progress on it, but it's still open and it's very challenging. It's another example of a situation where you have a very simple algorithm, some local update process. When you repeat it, it displays very complicated behavior. That's true of path compression where we got the inverse Ackermann function bound. That seems to be true of splaying. I'm sure there are many other examples. It's kind of like the phenomenon of fractals, where you have a very simple function, but when you iterate it, you get incredibly intricate patterns. It's an example in algorithms of the same sort of thing, or at least that's the way I see it.

Levin: Have you personally continued to work on this conjecture?

Tarjan: Yes, off and on. In fact, I have a grad student now who's been working on it and he has some, I think, novel ideas, so we'll see where that goes.

Levin: Maybe this will be the time.

Tarjan: It's time, but you never know with these things. One thing about theory, you never know ahead of time what the answer is or whether you're going to be

able to find the answer. I always tell students, “If you have a problem and you don’t know what the answer is, work on both sides of it. Because even if you’re working on the wrong side, you’re going to get insights that will help you when you turn to try the other side of it. And if you happen to work on the wrong side the whole time, you’re never going to get the answer because you’re trying to prove the wrong thing.”

Levin: The voice of experience, it sounds like. You were at Bell Labs for several years. I think you talked about the fact that they gave you an offer and you ended up staying after your sabbatical.

Tarjan: Yes, that is true.

Levin: Obviously that was a pretty different environment than being in an academic department, even though there were close connections.

Tarjan: It was, and I had spent some time there on visits previously, so I had some understanding. The difference between an industrial research lab and a university is most of your collaborations are with your colleagues, whereas in a university most of your collaborations are with the students.

Bell Labs was a wonderful place and I worked a lot with Garey and Johnson on algorithms and all kinds of things. But actually I discovered after the first year I was there that I was losing my motivation to do research because I was not getting the exposure to students. So I took an adjunct professor position at NYU and taught a course there. I was able to teach a course, an advanced algorithms course one night a week, so I would take the train in, teach, talk to the students, stay overnight, go back out the next day. I found that very rewarding, rebuilt my motivation to work on problems.

I actually had one PhD student at NYU, Neil Sarnak, who worked on persistent data structures, the idea there being most dynamic data structures, you do an update and the old structure gets destroyed so you could never go back and if you make an error you’re out of luck, or if you want to go back and access previous versions you’re out of luck. We started looking at applications of search trees in situations where you want to be able to at least access if not modify old versions as well as new versions. This idea comes up in many contexts. First of all, if you want an undo operation in any kind of editing system, you need to be able to back up, and if you want multiple versions so you can go back and look at previous versions, you need this capability, and there are a whole bunch of applications in computational geometry.

We figured out a way to build persistent balanced search trees, giving up only a constant factor in efficiency, and with both good time and space bounds. Subsequently we generalized that. I was still working with Danny at the time and with one of his students, we generalized that to general techniques for making

data structures persistent, that is preserving past versions at least so you can access them and with more powerful techniques you can update them as well subject to certain constraints.

Levin: How has that area evolved? I'm interested in it because, from my own background in operating systems and so on, the technique that's typically used there is just checkpointing of data structures, [1:10:00] and then if you want to get back to an older state, you go back to the checkpoint. That's a big hammer. It works great for a certain class of things…

Tarjan: Yes. And it can be expensive.

Levin: Exactly.

Tarjan: I mean the first observation is keep track of the changes. I mean these ideas that…

Levin: Yes. Logging as well.

Tarjan: Logging, that's the first and most important idea. I think in practice that's almost as far as it's gotten. That is, there are some beautiful theoretical ideas that in the right context might actually be useful for something, but nobody's tried to apply them yet.

The dynamic tree data structure that Danny and I used for the network flow algorithm, it's a beautiful data structure. I did some experiments with one of my Princeton students, Renato Werneck, with this data structure in the network flow application and in some others. The conclusion was that using a naive data structure is at least as fast as using this sophisticated but asymptotically efficient data structure. Sometimes beautiful ideas, they have to wait for the right time.

Interestingly on that front, this I always thought was a beautiful data structure with not yet a practical application. I was talking to a colleague of mine at Princeton, Sebastian Seung, and it turns out that they're using this data structure. He's modeling mouse brains. He wants to figure out the interconnection pattern of mouse brains. He slices mouse brains and then each 2D slice, they have to determine… they have to do a pattern recognition to find the chunks of neurons and then they have to determine which chunk on this slide corresponds to which slice, which one corresponds to that slice, so they can piece together these neurons. And these neurons are long, stringy things. They're trees in fact. It turns out, they're using this data structure to keep track of these neurons, which I think is just amazing. The scale of the problem is such and the fact that these things are actually tree-like means it works well for them. So you never know when these ideas get used in practice.

Levin: Unexpected application indeed. Yes, exactly. Coming back to the persistent data structure idea and the extent to which it actually finds its way into practical use now, do you have any sense of that?

Tarjan: I don't. I should probably take a look and see.

Levin: It'd be interesting to know. Certainly the idea of logging, which has been around for a very long time and the combination with checkpointing as an efficiency mechanism is something that gets used in systems a lot. But the more general idea I don't know about.

Tarjan: Yes. The sort of first approach to binary search trees is just copy the access path every time you make a change. Then you get a log space overhead. We found a way around that. But this idea of just copying the access path, we found five or six independent inventions of the same idea in different fields, so it's a natural idea that emerged more or less at the same time.

Levin: Just so. Coming back to Bell Labs and the difference in the environment there from academia, in academia of course you have completely curiosity-driven research.

::

Levin: How did you find that at Bell Labs? Did you have any sense that there were things that you were supposed to work on and other things that were less well regarded?

Tarjan: Well, there were small parts of Bell Labs where it was in fact curiosity-driven research. I mean I was in the mathematics research department first under Ron Graham and then under Mike Garey, and we had the freedom to look at anything we wanted to. I mean we got exposure to problems that emerged from the business units, but there wasn't really any pressure to work on them necessarily. That part of it was curi-… I mean people were interested in trying to help solve real-world problems, but there was no mandate. I mean we were very privileged. It was a small group within a huge organization. If you have a huge organization, there's room for small outliers and amazing stuff came out.

Levin: How small a group was it?

Tarjan: Well, our department was a dozen people maximum, and there was a computer science theory group with Al Aho and various other people. Computer science, mathematics, altogether: 25 to 50 people were doing curiosity-driven research in an organization of several thousand. There were physicists too. Those were good times. The phone company was a monopoly. It had not yet split. The beginning of the end came in 1984 when AT&T split into Ma Bell… well, AT&T, the long distance part, and the seven operating companies and Bell

Labs split into AT&T, Bell Labs, and Bellcore, which was jointly owned by the local operating companies. There was a question of who would go where. I lived through that split. Things were quasi-stable for a while. It's hard to destroy a strong and large organization, but over time things didn't go so well unfortunately.

Levin: Well, the economics of the business had fundamentally changed…

Tarjan: That is certainly true.

Levin: …and sooner or later it was going to have its effect.

Tarjan: Yes. And Bellcore in particular. I mean some people thought that Bellcore would be successful longer term, but you can't have a research organization that is jointly owned by semi-competing organizations.

So I was at Bell Labs… Well, so I started in 1980 at Bell Labs. I took this adjunct position at NYU. I taught there for I think four years, just one course, one semester, one day a week. Then Texas started dangling… This was the oil boom, so Texas had a lot of oil money and they started dangling endowed chairs in front of people, and eventually they got around to me. I went to Austin and visited and I decided at that point, "Maybe it's time to look around and consider other alternatives." I looked at both industrial places and at academic places, and eventually chose the conservative path. I mean I was married at the time with two kids, one young one. I get an offer from Princeton, so I took a professorship at Princeton but kept an affiliation with Bell Labs part-time. Since then, I've been at Princeton full-time but with a part-time affiliation with some industrial research group or other subsequently.

Levin: Ah, yes. I want to explore those. But I have one more question about Bell Labs. During the time that you were actually there as a regular employee before you went to Princeton, Ken Thompson and Dennis Ritchie won the Turing Award for the work that they had done on Unix. I'm wondering if you recall whether that had any particular impact in the organization at the time.

Tarjan: I would say they were always well-regarded and this only increased their level of regard. I mean smart people who did great things were really appreciated in that organization. I mean strong leadership, that was important. I mean eventually… Arno Penzias was a head of Bell Labs. He was a physicist. He was not a manager. The people they promoted to management were strong researchers and then they tried to inculcate management skills in them. I mean I think this was critical to the success of the labs through its successful period.

Levin: Right. Put a Nobel Prize winner in charge of your labs, going to appreciate – maybe – "Nobel Prize," quote-unquote, in computing.

Tarjan: He rewarded serious academic-style research. And there were enough interesting industrial problems around that people could do both or do something and get a paper out of it or a system out of it and have it have a real impact in the business.

Levin: Do you remember how those problems came to the attention of people in the labs? Was there any kind of mechanism for it or is it just stuff that seeped in?

Tarjan: The responsibility to do it was the department heads'. They would visit people in the business units and so on and talk to them about what problems they were interested in, and then bring them back to the labs or else have us go and visit and try to get some discussion going. [1:20:00]

Levin: Did you end up having to… or not "having to" but … having the opportunity to visit in some part of the business in pursuit of one of these opportunities?

Tarjan: I visited the branch of Bell Labs in Illinois. Maybe it was in Naperville or some place like that. I don't remember the subject, but it had something to do with government stuff. I mean Bell Labs had its fingers is all kinds of pies.

Levin: I wanted to ask you about some other things that happened before you went to Princeton. During that time, I think it was that you wrote a book. Is that right?

Tarjan: Yes, yes.

Levin: *Data Structures and Network Algorithms*?

Tarjan: Yes. That's when I was at Bell Labs. I taught a summer school I think it was. Well, SIAM sponsored the book. I'm not sure that they sponsored the summer school. I gave a series of lectures and then I turned that into a small book.

Levin: That book was quite well regarded as I understand. Won a prize, in fact.

Tarjan: A lot of people loved it.

Levin: Is the subject matter such that it should still be in print?

Tarjan: I would love to redo that book. I covered many of the important problems. By now, there are better solutions to many of them, so it deserves a thorough rewriting. But still, it's got the right set of problems and it's got good

algorithms in there, so it's definitely worth reading if somebody is interested in getting started in the field, shall we say. That's what I think.

Levin: Is it available online in any form?

Tarjan: SIAM still sells it. There's probably an electronic version.

Levin: I mean it's the sort of thing that should be used at very least for background and perspective on these areas that have grown so much.

Tarjan: Yes.

Levin: Interesting.

Tarjan: I would say another interesting experience I had with respect to book writing was when I was at Stanford, I was 30 years old. George Pólya was still there. He was still alive and he was still occasionally teaching a course, so I had an opportunity to co-teach a combinatorics discrete mathematics course with him for one term. The notes for that course got turned into a book. I had a grad student, Don Woods, who collected the notes, wrote it up, and published it. It was a wonderful experience though. Sort of my ideal. If I'm still as functional at 90 as Professor Pólya was, I'll be thrilled. He was a wonderful, wonderful man and had a great way of explaining things.

Levin: A legendary figure at Stanford certainly and elsewhere.

Tarjan: Yes, yes. And his books, *How to Solve It* and *The Art of Plausible Reasoning*, those were part of my readings back in the early days.

Levin: Just so. This one I happened to stumble across. It may not be available in… because it was course notes, may not be available as a published book but is available electronically through iTunes of all things.

Tarjan: Ah. Excellent.

Levin: I'm guessing that means that it still gets significant use.

Tarjan: I think it still gets used, yeah. It's got classical material in it. That stuff is less obsolete than the algorithms, because mathematics moves at a slower pace and we were looking at very basic problems.

Levin: Was that your only opportunity to interact with Pólya?

Tarjan: Yes. I went over to his house a few times, but he was getting on in years, so…

Levin: It's impressive that he was still going.

Tarjan: Yes.

Levin: That's something. Coming back to the book you did, the SIAM book that you did at Bell Labs, obviously a book like that is a pretty substantial undertaking, even if it is derived from a set of class notes, that bookifying it is a big deal. Was that something that Bell Labs sort of actively supported or was it just assumed that you could do that sort of thing and no one was going to give you trouble about it?

Tarjan: Well, in those days there were no typesetting systems, so I had secretarial support and I had support from the art department to draw the figures. In that sense, I certainly had active support and they were happy to give me the time to do it also. I was not unique in that regard. I mean Brian Kernighan was Bell Labs, I'm sure published many books during the time he was there.

Levin: Oh yes. Certainly true. Well, the C book being the most famous one probably. Certainly the one that sold the most copies.

Let's turn now to a couple of things that you've mentioned, certainly Princeton and in particular the fact that beginning with your time at Bell Labs and continuing to the present day, you've always been involved with a corporate entity in parallel with your academic position. I think in my experience that's rather unusual, at least for someone who's worked in the more theoretical side of computing, the algorithms and so on, as opposed to as a more conventional consultant to a company. This is obviously something you've wanted to do, and I'd like to understand about that.

Tarjan: Yes, certainly. I've certainly enjoyed it over time. Yeah. How to explain it. Well, as I said, if one is in academia, one spends time collaborating with students; if one is in an industrial setting, one spends time collaborating with one's colleagues. I like to do both, so having the combination is a natural way to do both. Although I collaborate with plenty of academic colleagues also, but mostly in other universities, I would say. I mean the Internet is really good for long-distance collaboration, and it gives one an excuse to travel to fun places too.

Yes. At Bell Labs, I was never involved in management and I've never had an official management position. But I spent four years part-time, '85 to '89, at Bell Labs after I took the Princeton position, and then NEC started a research institute in Princeton, which was more convenient and intriguing because there was the possibility of building up a strong research group from scratch. My first marriage was kind of breaking up at the time, so in '89 I left Bell Labs, I took a part-time position with Research Institute which was right down the street. I could go back and forth, it was very convenient, and I got… They had fellow positions and we

were supposed to be the senior leadership, so we didn't have any official… Well, the official management role was to kind of advise and consent, help with the hiring, help with the evaluation and so on, as well as doing our own research. We managed to hire some very strong people. While it lasted, it was a wonderful place. And it was closer. Andrew Goldberg for example, who I had collaborated with when he was a grad student at MIT, again on network flow algorithms, took a position at NEC Research Institute.

I spent eight years there, something like that, until '97 maybe. Then several of us got involved in developing a technique for digital watermarking. I kind of got added in on the back end to make sure the thing was technically sound, but we were trying to actually commercialize this, several of us. We ran into some disputes with the management of NEC. Working for a Japanese company in the US I think is a bit of a challenge, for understandable reasons. Not good logistics.

We set out to look for a place where we could do that or some related kind of work, and we found this crazy company in California called Intertrust, which was a startup trying to do digital rights management. They were trying to start a research lab, which is completely crazy for a startup, right? [1:30:00] But several of us moved there, I part-time, but that's when I started doing my crazy commute between California and New Jersey, which I have been doing ever since. First I was at Intertrust, then I was at Hewlett-Packard, then I was at Microsoft for a couple of years, and now I'm actually back at Intertrust part-time.

It's a different environment. You get exposed to some real problems or what appear to be real problems. I like to work on problems that I think are fundamental and have some practical potential impact. Academia, I think it's a little bit easy to kind of get diverted into… I don't know what to call it. It's a little bit too theoretical or too "up in the clouds" kind of research. It's good to see what people are actually trying to do, what problems they're actually trying to solve. It's a little bit hard to do that if one is completely in the academic world. On the other hand, interacting with students both doing research and teaching is tremendously motivating. Somehow I've enjoyed both for a long time.

Levin: I can imagine that there could be some complications in trying to… Because companies tend to worry about intellectual property issues and stuff like that. Universities tend to try to stay away from that as much as possible. If you're working in both places and you've got students, I can imagine that there might have been some challenges. How did you manage that?

Tarjan: Try to keep them separate as much as possible. I would say that most significant issues had to do with Intertrust because they were IP-centric. But at Intertrust I was working mostly on stuff unrelated to my academic work, my algorithms work, specifically security-related stuff. We started with the digital watermarking, we started working on general techniques for software protection

actually, software watermarking and a number of other techniques, and we ended up writing several patents on that stuff. That stuff actually got into the Intertrust products eventually, but it was definitely separate from my academic work.

The other places were much more relaxed about this issue. And I'm kind of ambivalent about this idea of patent protection for software. I'm not convinced it's a good idea. My real objective is to get my ideas out there and get them used, and patents can get in the way sometimes.

Levin: Let's explore that, because I think this is a topic that's obviously far from settled in our society now. Of course the original idea of patents was to make technology available to the public by forcing publication. That hasn't worked as well perhaps as Thomas Jefferson and others thought it would when it comes to software, and you seem to have some doubts as well.

Tarjan: Well, is an algorithm a mathematical thing that you discover or is it an invention? This is one question. The other issue is the patent system doesn't work terribly well because it seems to be possible to patent ideas that are obvious, at least obvious to an expert, shall we say, although the criterion is supposed to be "non-obvious." So I think there are theoretical issues and practical issues with patents. I mean if I came up with a creative idea and the company wanted to do it, I was happy to patent it. I have a number of patents on things I did at HP. But getting something into practice is somehow different from patenting it, at least in the algorithmic space. I do think the patents are kind of… they get in the way more than they help, somehow.

Levin: Well, I think certainly when it comes to algorithms and even in some cases for things that are much more specific implementations, it seems to have gotten to the state where companies do it just for their own protection. It's a completely defensive activity.

Tarjan: Yes, yes, yes.

Levin: They end up trading patents and agreeing not to sue each other, and that's the end of the story, which is I'm sure not what Thomas Jefferson had in mind.

Tarjan: I suppose not, but that seems to be the way the system has evolved.

Levin: Do you see any hope that we might get to something better?

Tarjan: I have no idea. The other thing I think about patents is lawyers write these documents. I mean they're incredibly hard for a scientist to read, so the

ideas kind of get lost in there somewhere. It's a different business than the inventive process, or it's a different kind of creation, shall we say.

Levin: Yes, indeed. Well, it's unfortunate that it does get in the way of the flow between industry and academia, because I think that flow benefits both.

Tarjan: I completely agree.

Levin: Coming back to the sequence of companies that you worked for and did things, you mentioned that you got a lot of extra travel out of it. That was certainly one good thing. But the companies had undoubtedly different characters. NEC, as you said, was a Japanese company, Intertrust a startup, the other two were big American companies. To what extent did those differences influence the kind of work that was being done? Or was that really kind of orthogonal?

Tarjan: Well, NEC was set up on kind of a Bell Labs model, the pure side of Bell Labs. It was creation of an ideal and there was the hope that eventually some of their ideas would be commercialized, but somehow they could never really get it to work. It was a great idea and it lasted for 5 or 10 years, but then the company didn't get any benefit out of it. But we were free to do whatever we wanted to basically and it was really hard to… The Japanese had sort of a "not invented here" syndrome, so if one actually had a good idea, to get them to use it was hard and it was also hard to figure out what problems they really had that we might be able to help solve.

Intertrust was a startup. We had this lab. We actually managed to do some stuff that eventually became relevant to the product, but it was always a very IP-centric company so they made most of their money over time from licensing their original digital rights management patents. But the software self-protection stuff we did, as I said, did eventually get into products. Their problem was not so much getting research into practice but converting their ideas into practical running software. They always kind of wanted to be a products company but they could never quite figure out how to do it, but they succeeded just on the basis of the IP.

HP, in some ways, of the industrial labs I've been in, I had the most successful experiences and the most frustrating experiences trying to get ideas into practice. Because I was working on, well, a lot of optimization-related problems, scheduling of high-performance computers, and also we were working on some personnel-matching problems. HP bought a… EDS. Was it EDS? I think so -- this big services company. They had the challenge of "You have a project. It has certain requirements that need to be fulfilled by a set of people. How do you match the people to the slots in the project?" Classical assignment problem. We worked on some systems for that. We had some beautiful algorithms and ideas,

but getting it into practice was a real challenge. Because again you're in a big company, the people in the business units have their own processes. They're not willing to try something new unless what's going on is just failing miserably.

There are an amazing number of brilliant people at HP, but somehow HP Labs… There were some successes. I think the labs did not get credit for the successes it had in getting things into the business units. Getting traction was very difficult. HP as an old-line company kind of didn't rise [1:40:00] to various challenges somehow, notably cloud computing. They had an opportunity, I think, to be second to Amazon in the cloud computing space and somehow they missed the boat. Then Microsoft came along -- game over at that point.

Levin: The thread that runs though this is that it's not always very… it's often difficult to get the ideas that are done even in some context that seems relevant to the company into the product.

Tarjan: There's a huge challenge. Yes. There is a huge challenge getting ideas into product. I mean big companies are notoriously resistant to change, for good reason I think. Where do startups come from? A lot of startups in the Valley come from people in big companies having some idea that they can't push through and going somewhere else and doing it themselves. It happens in academia too. I mean if you have a good idea and you want to commercialize it, you obviously can't do it in an academic setting. Can't do it in a big company either. You go do it yourself. I think the US is marvelous in this sense because startup culture is amazing, unique. Or it has been unique. Whether it stays that way going forward, that's an open question.

Levin: It does seem that corporate research, that is to say research of the more blue-sky, unfettered kind that we've been talking about done in corporate setting, is a dying breed.

Tarjan: Well, the classical examples were AT&T and IBM, which are both shadows of their former selves, if that. Microsoft created a research organization, which is the most obvious thing to compare, but they've had their ups and downs, I would say. Google and some of the other newer companies seem to have a different model of how research should work and how it should connect into their environment. I think there's room for many models, but certainly the old one, there are no or few examples of it these days.

Levin: Do you see that people who work in algorithms are having the same opportunities to influence companies that you've had over the course of your career?

Tarjan: That's a very good question and I don't really have an answer to it. I mean algorithms in the sense that I do it, it's kind of out of fashion at this point. The big thing now is big data, data analytics, neural nets, machine learning,

because it's been highly successful in certain domains. What I think though is that this is only part of the solution. Eventually people are going to have to come back and realize you can do the pattern-matching, you can get the coefficients, but if you've got a decision problem with some complicated combinatorics and you want something close to an optimum solution, I don't think machine learning is going to solve it. You need different techniques. I hope we get back to that.

Levin: Computing, like everything else, has its fads and trends.

Tarjan: It certainly has its fads, and they come and go. In my lifetime, AI has probably gone through at least three cycles. This one may be bigger than all the previous ones and they've certainly had remarkable success. But eventually I think the limitations of the current techniques… we're going to run into the limitations of the current techniques. But who knows? One needs to be open-minded. Predicting the future is completely impossible in technology, and things change faster and faster and faster.

Levin: So after being at several different universities and different other places, you settled down at Princeton for 32 years now.

Tarjan: Quite a while. Yes, indeed.

Levin: Quite a long time. Obviously it suits you, despite all the travel that it entails to get to some of these other places.

Tarjan: *[laughs]*

Levin: Over that really long period of time, can you identify a few highlights of work or students or anything that stand out in your mind from those years?

Tarjan: Yes. This story illustrates the advantage of moving to a new place or interacting with a new colleague or set of colleagues or something like that. When I got to Princeton, I started talking to Bill Thurston who was in the math department, who was an incredible creative genius, sort of the parent of hyperbolic geometry, but he was able to apply hyperbolic geometry in all kinds of strange and wonderful places. And Danny moved to Carnegie Mellon I think before I went to Princeton, but he was still visiting so he came from time to time. We were still talking about splay trees and problems related to that. Of course binary search trees, this is a rich… the structure of binary search trees is a rich combinatorial space and you can ask mathematical questions as well as computer science questions. It's more than a data structure.

There's a very basic question, which is "Given a search tree on *n* nodes, you can always convert it into another search tree on the same set of *n* nodes in the same order by doing rotations," which are these local transformations that

change one tree into another. The question is, "Is this graph… Let's imagine a graph whose vertices are the trees, with two trees connected if they're joined by a… if it can rotate at one place and get to the other one. So big graph, Catalan number of search trees. What are the properties of this graph? Is it connected? Yes. What's the diameter? Is there a Hamiltonian cycle?" I had one PhD student at Princeton, Joan Lucas, whose thesis among other things showed that there is a Hamiltonian cycle. You can go through all the trees, you can construct a Gray code of trees using rotations as the individual moves.

Danny and I started to get interested in this question of "How far apart can two trees be?" The answer is it's easy to get an upper bound which is about 2*n*, where *n* is the number of nodes and there's a subtractive constant. But proving that this is… And it's easy to prove a lower bound of *n*, but then there's this mysterious constant factor. Here again is an example where I did work on a constant factor, but I'll justify it by saying it's more a mathematical problem than a computer science problem.

We were trying to prove that 2*n* minus a constant is the correct answer and we started talking to Bill Thurston. He made the observation that they're equivalent structures. First of all, if you take a polygon, an *n*-sided polygon, there are various ways to slice it into triangles by connecting two of the vertices in the polygon. Let's think of a convex polygon. Triangulations correspond exactly to binary search trees. The *n* changes by 2 I think. There is a move that turns one triangulation of a polygon into another, which consists of taking a diagonal and the two triangles bounding it form a quadrilateral and you remove the one diagonal and replace it the other way. This corresponds exactly to rotations in binary search trees. So the diameter of the binary search tree graph is the same as the diameter of the polygon under diagonal flip transformation graph.

So these are exactly equivalent. Then you can make the observation that if you take two polygons with triangulations, glue the boundaries together and blow the thing up in the middle, you get a polyhedron with triangular faces. Now each of these diagonal flips corresponds to a tetrahedron, and now the sequence of diagonal flip moves corresponds to chopping this polyhedron up unto tetrahedra where all the vertices are on the original boundary. Now if you can prove that there are certain polyhedra such that you need lots of tetrahedra to fill in to do the packing, then you get a lower bound [1:50:00] for this packing problem, which translates to a lower bound for the diagonal flipping problem, which translates to a lower bound for the tree problem.

So you can do a volumetric argument – compute the volume of the polyhedron, compute the maximum volume of a tetrahedron, divide. This idea fails miserably if you do this in Euclidian geometry, but now comes the Thurston idea – hyperbolic space. If you embed this thing into hyperbolic geometry in the right way, the volume of a tetrahedron is a unit and the volume of the whole thing turns out to be 2*n* minus a constant, magic happens. You get the answer. So

hyperbolic geometry gave us -- via Bill knowing hyperbolic geometry -- the lower bound for this original semi-algorithmic question. This is an example of the mysterious, incredible power of mathematics. Also, it was a great experience because that was the first time I met Bill and got to work with him.

Levin: And having the right tool.

Tarjan: Yes, having the right tool, definitely. The more things you know, the better off you are, and/or the more smart colleagues you have who have other sets of tools, the better off you are. Collaboration is wonderful. Sharing problems is wonderful. I try to do as much sharing as possible with problems, because that's how they get solved and there are always more problems out there in the world.

Levin: No need to be proprietary.

Tarjan: No, no.

Levin: Any other highlights that stand out in your decades?

Tarjan: Well, while I was at Princeton, I did a lot of work with Andrew Goldberg on network flow algorithms and so on. As I said, he joined as a colleague at NEC. That was a wonderful time. I had other great students.

Monika Henzinger, who was at Google for quite a while in the early days running their small research group, was one of my students.

Jeff Westbrook was one of my students. The interesting thing about Jeff is that he was brilliant, he was a strong computer scientist, but he had been an undergraduate at Harvard and worked for the *Harvard Lampoon*. Eventually he decided… Well, he had colleagues from Harvard who had gone to Hollywood, were writing for *The Simpsons*. He decided… They enticed him into joining them, so he ended up in Hollywood as a writer for *Futurama* and *The Simpsons* and some of these other shows.

So you can never tell what paths your students will take. My student from NYU, Neil Sarnak, who worked on persistent search trees, eventually ended up at Goldman Sachs doing some of their IT stuff. He was doing banking in the IT space. He's now happily retired playing golf. So computer science PhD is a wonderful starting point for many interesting careers.

Levin: Well, it teaches you how to think creatively.

Tarjan: Definitely, yes.

Levin: Definitely useful in lots of areas.

Tarjan: Yes.

Levin: What I'd like to do now is: we've talked at various points about some of your specific results and work, and if we can kind of take a step back or a step up – I'm not quite sure which – and look at trends in your work. I know just looking at the amazing list of papers that you have and the breadth of topics that they cover, there have been quite a few trends in your work, things that you've come back to over time. I'd like to ask you to highlight some of the ones that you regard as the most significant or the ones that have been the most satisfying in some ways for you.

Tarjan: The first important trend or general approach I think we've talked about already is this notion of amortized efficiency, measuring the efficiency of an algorithm by generally running time, but it could be space or it could be counting some special set of operations, but ignoring constant factors. This allows a level of abstraction that allows you… It gives a nice level of abstraction to try to deal with the problem in the most important way and ignore messy details.

Then, well, I got started doing graph algorithms. The idea of graph search as a tool. What are the properties of graph search? What does it do to graphs? I worked with Hopcroft mostly on depth-first search, but I've also done work with breadth-first search and some extensions of breadth-first search like lexicographic search, maximum cardinality search. This is a powerful general technique for solving graph problems.

The idea of amortization. If you are in a situation where you are doing a long sequence of operations, generally you don't care about the individual efficiency. You care about the total efficiency. Then you can consider trading off expensive operations as long as they're balanced by cheap ones. This leads to much more freedom in the design space. In particular, it got us to splay trees and this analysis of move-to-front. It comes up both in data structures and in more complicated algorithms where you have some piece of the computation that you're doing repeatedly, and it doesn't have to be fast all of the time. All it has to be is fast enough of the time.

The idea of reducing a combinatorial problem, graph problem, or some other problem to a data structure problem. Extracting exactly the right operations that need to be done and then figuring out how to implement those operations in sort of a minimalistic way, that's a key idea.

We didn't mention randomization, but randomization is also a very powerful technique and I've done a few efficient randomized algorithms. Random sampling. The most interesting example is the randomized algorithm I developed with David Karger and Philip Klein to find minimum spanning tree in linear time. It uses random sampling and it uses recursion. Interestingly, it's got the same

property as the median finding algorithm that I did back in Stanford. It uses recursion in two places in fundamentally different ways, and you put the thing together in the right way, you get a linear-time algorithm. Randomization is critical in security -- crypto and so on -- and it helps in a lot of algorithmic situations. With the minimum spanning tree problem for example, that was actually something I did while I was at Princeton also, although I worked with Philip, who was at Brown but visiting Princeton, and with David, who at MIT at the time. I think David's PhD thesis was all about random sampling and its use in efficient algorithms.

Ah, the minimum spanning tree problem. We found a linear-time algorithm that uses randomization. It's still open whether this problem can be solved deterministically in linear time. It's a very peculiar situation, and again, inverse Ackermann function comes in here. Bernard Chazelle developed a deterministic algorithm that runs in linear-time inverse Ackermann function time. That algorithm was modified by Pettie and Ramachandran to produce an algorithm that runs in optimum time to within a constant factor. But they don't know what the bound is, so it might be…

So there is an optimum deterministic algorithm, but what its running time is, nobody knows. "How can this be?" you ask. The answer is there's another general technique. These minimum spanning tree algorithms use essentially parallelism in a very heavy way. They do a recursion that very rapidly reduces the problem to tiny little subproblems. If the subproblems are small enough, you can enumerate all possible algorithms, decide the best one, and apply it. The number of possible algorithms to solve the tiny subproblems, even though it's super exponential in the problem size, it can be arbitrarily sublinear in the size of the big problem. You can get the optimum algorithm for all little subproblems and just do table lookup. So you run this thing and the recursion turns out to be only constant depth. Then you get an algorithm, [2:00:00] but you don't know the bound for these optimum tiny algorithms that you computed by brute force. A very strange situation.

Randomization? Yes, I think… Now trends that I have not worked so much on but that everybody asks me about and I think are very important is, well, two. One is multiple processors and the other is issues of memory hierarchy. Our model was always single processor, flat memory, random access memory. This gives us a level of abstraction in which it's very easy to work and we can get remarkably practically efficient algorithms as well as theoretically efficient ones. On the other hand, modern machines have a memory hierarchy and caching is a significant phenomenon, and if you have to go out to disk or flash or something like that, there's real slowdown involved. It is important to understand how you get efficient algorithms that deal well with the memory hierarchy. Also now there are multicore machines all over the place. How do we use parallelism effectively?

I think it's kind of wide open for both of these questions. Again, computer science cycles over topics. Back in the '80s, there was a lot of work on parallel random-access machines, flat shared memory but lots of processors all working together with a common clock to solve a problem. Plenty of beautiful work. People tried to build computers to actually do this, the Connection Machine for example. Conventional hardware turned out to be more efficient. Those ideas kind of got put aside. But now I think these ideas are coming back but in sort of a more general context. They're a small number of processors, relatively small, huge memory, which may be shared or it may be distributed, and not a common clock. If you're dealing with asynchronous parallel processors, even with shared memory it's a real challenge to get any kind of provable efficiency. I've been looking at data structures recently and trying to actually prove bounds in that setting. It's a step beyond to look at distributed memories, which is a form of memory hierarchy. I think there are huge challenges.

I guess my… I've worked on a lot of topics over time, but as I think I already said, I like problems that I view as fundamental, that are not sort of twiddles on twiddles on twiddles. The design space is incredibly rich. Computer science is a young field. We've found new problems, come up with pretty good solutions, but not necessarily completely explored the design space. A lot of what I've been doing recently has been revisiting old problems that I still don't fully understand and trying to really nail down what is the right solution to the problem, what can you really say is best possible?

I mean I've done enough work for at least my lifetime, so now I can do what I want, right? There are still these problems that intrigue me that I continue to work on, like dynamic optimality and like some of the basic data structure problems. I'm still interested in network flow. But also trying to bring the results into the modern age, the multiprocessor age, the big memory, distributed memory, hierarchical memory age. I think there are huge challenges there. There's been plenty of work, but there's plenty of work still to be done.

Levin: That's a… *[laughs]* You may say you've done plenty of work for a lifetime, but it sounds like there's plenty more you want to do.

Tarjan: Yeah. As long as I have problems to work on and I can keep up with my students, I see no interest to stop.

Levin: One area that you didn't mention but which I think is probably one that's gotten more attention recently is online algorithms and the competitive analysis that kind of goes with them, because certainly that stuff came up in the context of things like auctions and so on, all the game theory things that happen. Do you see that as something that's still trending up?

Tarjan: It was certainly a hot topic for a while. I mean I would say the real source of that was a lot of work on scheduling-type algorithms and bin-packing-type algorithms that people like Dave Johnson did and Mike Garey and Ron Graham, mostly people at Bell Labs. Then the notion, when we… the move-to-front result is a competitive result, and the splay tree analysis is competitive stuff. Danny and his students kind of picked up the bandwagon, coined the term "competitive," and did a lot of work in various settings. It was quite a lot of work for a long time, I guess late '80s, early to mid '90s, and then it got into game theory, computational economics, and so on. I think that's a very important concept. I don't know if it's tailed off, but it's not the hot area that it used to be. But it's fundamental, which means it will come back. It's not going to die out completely.

Levin: Let me try an idea on you and see what you think. You develop an algorithm and then you analyze it. Maybe the algorithm is complicated, maybe the analysis is simple, or maybe the other way around, or sometimes they're both complicated. Presumably the last is not a good state.

Tarjan: The best state is the algorithm is simple and the analysis is complicated. The analysis can be as complicated as it needs to be as long as it's correct. Make the algorithm simple. I'm not sure where you're going with this, but…

Levin: Well, you went there.

Tarjan: *[laughs]*

Levin: That was what I was hoping you were going to say. My point that I was trying to get to is: it seems that's actually been a trend in your work, is to pick up things that you did earlier, maybe were dissatisfied with the complexity, tried to figure out how to simplify them while still having the same performance characteristic or whatever, even if the analysis ends up being messier in order to do that.

Tarjan: Yes, I would say that's true, definitely. It's the practical thread, right? You want the algorithm to be simple so there's some hope that somebody will actually use it. So it's implementable, you can run it on a machine without huge torture. I mean that has to be doable and people have to want to do it. If it's not simple, they're not going to want to do it.

Levin: In some sense, the lack of simplicity in the algorithm gets you every time whereas the complexity of the analysis you only pay for once.

Tarjan: Right. It's another measure of complexity, the length of the code. It's sort of Kolmogorov complexity, or it is Kolmogorov complexity. How complicated is it to express the static steps of the algorithm? That's actually the most important kind of complexity, because if you have a problem to solve and it's

small enough that you can write a quick-and-dirty one-off algorithm to solve it, program to solve it, game over. It's only if you need to solve a problem repeatedly or on huge instances where the simple algorithm doesn't work. Then the stuff that I do becomes relevant.

Levin: *[briefly inaudible]* You mentioned randomization and its importance. I wanted just to touch back on that for a minute. If I look over the space of your work, it seems like randomization has played a relatively small but important part. Is that true?

Tarjan: I think that's true, yeah. I could point to three or four algorithms that I've helped develop that involve randomization.

Levin: Can you think of any particular reason why that is, other than that's the way you think about things? Or do you have a preference for approaching these things with deterministic algorithms?

Tarjan: Maybe I have a preference for determinism, but randomization is a beautiful idea. The tricky point is where do you actually get your random bits and how many do you need and how random do they have to be? It's a nice abstraction. The question of how does it work in practice is a little bit [2:10:00] complicated I think. But in some situations it simplifies things so much, you just got to go with it.

And sometimes, you know… Getting back to the inverse Ackermann function bound, the disjoint set union problem, I've done recent work on that. The classical algorithms, the ones that I analyzed, are deterministic. But it turns out there's a simple randomized strategy that you can analyze that gives the same bound. I worked with a professor at Stanford and another professor at Penn and one of my grad students on this, Dan Larkin, at Princeton.

Why would you care, if you got the deterministic solution? Well, the randomized solution is simpler for one thing. Furthermore, people have done experiments which suggest that you don't need a complicated linking strategy, which is one part of the algorithm. The question is "Is this an artifact of the experiments?" My hypothesis is that in effect, they're using the randomized strategy because they're generating random examples. They're getting the randomization for free, which is a nice situation, so it's nice to know that.

But then if you look at the concurrent version of the problem, which I started to do recently with an undergrad actually -- Siddhartha Jayanti at Princeton, who's now going to MIT as a grad student -- there the randomization gives you at least simplicity. It gives you, as far as we can tell, lots of simplicity. To get the bounds using determinism, the bounds that we're trying to get, things get way more complicated. So there are settings in which randomization is really powerful and

other settings where it's not so important. Use it when it's necessary. If you don't have to use it, you don't think about it.

Levin: Got it. *[pause 11 secs]* I'd like to turn now to the matter which in some sense was the original impetus for this interview, which is the Turing Award. You won the award jointly with John Hopcroft in '86 for work that you had done together in the '70s. I think we talked about that work quite a bit earlier on. Can you think… I mean you won it at a relatively early age in your career, even though it was already 10 years or more after the work.

Tarjan: Yeah, I was 38 at the time. 1986, right? I think I'm maybe the second-youngest winner. I think Don Knuth is probably the youngest. I am not sure about that, but…

Levin: That's been quite a few years now. Can you reflect back on the impact that winning the award might have had on your career?

Tarjan: It was a great thrill. I think it had less impact back then than it seems to now. I mean I think I get more benefit out of having won the Turing Award at all now than I did then, which is a great thing for the ACM and a great thing for the reward. It's just… I'm very happy I was working in the relatively young days of the field, because to win a Turing Award now is a much bigger challenge. Of course there's a much bigger reward too, but I get kind of the inherited glory of the thing now, which is wonderful, and I don't have to worry about it. It was a great honor. It was thrilling. I was a tenured, endowed professor at Princeton. In terms of academia, I was already at the highest level I could get, so I don't think it impacted my career in any serious way as far as I can tell.

I was never motivated primarily by awards either. You know, awards are a funny business. You have to have an intrinsic drive, intrinsic interest in doing research, because even in the early days when there were all these open problems, it was hard. I spent a year on planarity testing like banging my head against the wall figuratively. You've got to be willing to suffer, go away from a problem, come back to it later, and there's this mysterious process that's completely unconscious where things kind of realign, the gears shift, and all of a sudden they move in a new way. Doing research in theory is kind of terrifying because you never know if the problem you're trying to solve you can solve or not. It's good to have several things that you can go between when you get stuck on one thing, so you can go think about something else.

It's also good to have other things to do. Like I used to bicycle a lot, run long distance and so on. I love my family, my kids. We're now up to five, three of mine and two of my second wife's, three of whom are getting married this summer. Well, one already did and two are getting married. So it's nice to have distractions of various sorts as well as different problems to work on.

Levin: Frees the brain.

Tarjan: *[laughs]* Yes. It refreshes the brain, cleans out the brain. Put the stuff aside and come back to it later with a different point of view. This is also a good reason to have colleagues, because it's a lot harder to go down a rat hole collectively I think then. If you have different people with different points of view, you get out of the rat holes faster.

Levin: Yeah, very definitely. Let me ask you another question about awards. Obviously the Turing Award was this very significant thing, but it wasn't the only very significant thing that happened to you in a relatively small number of years.

Tarjan: Yeah, that's true.

Levin: You were selected for the National Academy of Engineering, National Academy of Sciences, AAAS, American Philosophical Society, all in the space of, what, five years or less? That must have been a pretty heady time.

Tarjan: *[laughs]* It was. Actually the first one was I think the Nevanlinna Prize, which was this new award created by the International Mathematical Union in information science. They must have… I don't know all the details, but they understood that computer science was important and was mathematical and they wanted to… they were talking about giving a Fields Medal to a theoretical computer scientist. So they got some money from Finland, created a new award, but more or less the same terms as the Fields Medals. I got the first one in 1982.

The IMU meeting, the International Congress of Mathematicians, was scheduled to be in Poland in Warsaw in '82, but there were uprisings and so on so it actually got delayed by a year. My first wife and I and our young daughter Alice, who would have been… maybe she was… I don't think she was quite pregnant with Sophie, but Alice was very young. Alice was like four or something. We go to Warsaw. It's still communist. The milk is sour, the poor kid is suffering. It was a great honor, but we were happy to leave communist Poland to go to Finland where they were having me, because I get to the hotel and there are bountiful plates of fish and stuff. It was kind of night and day. It was an interesting experience.

Levin: What I wanted to find out was whether particularly the National Academies, which it's certainly an honor to be a member of, but there's actually work, can be work. Has that led to any interesting projects that you've been involved in?

Tarjan: I was involved in one project to understand the state of mathematical teaching in this country, high school/undergraduate level. That was quite a while

ago. I would say I haven't done my fair share in that space. I guess my view has been: it takes up enough time teaching, interacting with my grad students, trying to do research. Life is short and finite, [2:20:00] and as long as things are going well on the personal front, I'll leave it to others to try to solve the bigger problems.

Levin: The thing that you mentioned about mathematics teaching, that's something that's intriguing to me personally. My sense is that this seems to be another one of these wheels that keeps turning at least at the public school level in American society. There's always another "New Math." There was the one that you and I had, and then there have been at least two or three since then. There doesn't seem to be any end in sight. Do you have a sense that this wheel is actually taking us anywhere?

Tarjan: Well, I have a spiral theory of the universe, which is that things kind of go around like this and occasionally you come back to the same place, but it's offset in an orthogonal direction, in a new dimension. It's probably like that with that mathematics education.

I mean part of the problem is mathematics is a very… the word I want to use is "esoteric," which is not quite right. But you need a special kind of… your mind has to work in a certain way to really get it, and there are the people who get it or can be trained to get it and then there's everybody else. I don't know. It somehow seems like you need two tracks and both people need their own… And there may be more than two. There probably are more than two. But different people need different ways of being taught. This suggests you can't treat… And I'm sure this goes for every other field too, but I think it's particularly true in mathematics. Individualized learning or small-group learning or something like that with programs somehow adapted to the abilities of the individuals. There's no one right solution for everybody.

That's hard to do. I mean it takes money and time and effort. On the other hand, we've got the resources of the web. You develop an amazing teaching tool and it's there forever. You can use it all over the planet. So there is at least the opportunity to create various teaching platforms and then somehow try to match the kids to the appropriate program.

Levin: One can hope.

Tarjan: I don't know if anybody's trying this sort of thing, but this seems more productive than… It's like the Buddhist cycle of life or something. You want to get off the cycle. How are we going to get off the cycle? You need to take multi-branching paths or something maybe. I don't know.

Levin: Well, the application of the web is one that probably hasn't been tried at the extent that it should have. I mean there are instances of it like Khan Academy and so on that seem to have their place.

Tarjan: Yeah, and technology has not done so well in education. I mean previously there was this big push for computer-aided instruction. It kind of failed miserably. Now there are online courses, Coursera and so on and so forth. I think the jury is open. It's a great new set of information, a new resource, but it has to be combined with some kind of interactive personalized approach. The best way to learn is to do, obviously, and the best way to learn material is to teach it yourself. You end up knowing it a whole lot better than the students do. That's another reason to be a teacher, actually.

Levin: Exactly. I have one more area I want to ask you about before we talk about the future a bit.

Tarjan: *[laughs]*

Levin: And that is your involvement in DIMACS, the NSF Center for Discrete Mathematics and Theoretical Computer Science, which has been around for quite a long time, since the late '80s I believe, at Rutgers. You've been involved with it for also quite a long time. Can you say something about how that came about and what your role has been?

Tarjan: Danny Gorenstein, who was a math professor at Rutgers, I think had some conversations with Ron Graham. The NSF started a program to create science and technology centers and there was some discussion between them and other people in the area about trying to start a center to do discrete math theoretical computer science. We put together a proposal and submitted it, and I ended up being the co-director and Danny was the director while he was alive. I was very active in the early days. We had money from the NSF for I guess 11 years altogether, which enabled us to pay something for grad students, most importantly run workshops on scientific programs over a year or sometimes two years. It was a great organization.

Mathematics has ongoing institutes that are funded by the government. This is not true in computer science. The NSF -- the computer science part -- resisted creating permanent institutes of this kind. At the end of the 11 years, they weren't going to fund us anymore, so the funding model changed. Fred Roberts took over as director after Danny died and spent most of his time scrounging for grants for specific projects. If we wanted to run a specific program, we would have to get a specific grant to help run the program. So the funding environment changed and things went in more applied directions naturally, because that was where the funding was.

I got less involved after the first 10 or 11 years when it was up and running. I think it's still a great institution, still doing good work, but it's much more applied than it used to be. Now there is the Simons Institute at Berkeley, which is somewhat related to the DIMACS model, and it's privately funded, so we'll see

what happens with them over time. They have a shorter-term program length too. It tends to be one semester or six months rather than a year or two.

Levin: But the charter is otherwise somewhat similar, that…?

Tarjan: That is my impression, yes. Pick topical areas. Bring visitors in. Collect students. Have joint research efforts, workshops, lectures. It's a great idea.

Levin: They've been running a few years now, right? Maybe four, five?

Tarjan: Yes, something like that. I think these things are very useful to the field.

Levin: One particular thing that I know DIMACS did and which I think you may have been involved was this thing that was called the DIMACS Challenge.

Tarjan: Yes.

Levin: Tell me a little bit about that.

Tarjan: David Johnson was the prime driver of that. I mean it was mostly Bell Labs, Rutgers, and Princeton, and we had some participation from people in other places. David was especially interested in experimental algorithms and came up with this idea to pose a computational problem and have people work on it, propose datasets, propose solutions, implement the solutions, run them on the datasets, have a workshop or two first to discuss preliminary ideas and then to talk about what they had done. I think that was a wonderful idea. The area of experimental algorithms is very important, very challenging, and kind of under-studied or it's kind of a neglected field. But I think that was one important institution that worked on this. I think they still do them from time to time, although David now unfortunately has passed away so he's no longer on the scene.

Levin: Well, I know as recently as maybe five or eight years ago, there was a DIMACS Challenge, so there's at least some…

Tarjan: Yes. There is the opportunity for anybody who wants to do one to run it through DIMACS, and they have the infrastructure set up so they can do it.

Levin: Sounds like a fine thing. Let's move on to some speculations about the future.

Tarjan: *[laughs]*

Levin: As they say, it's always difficult to make predictions, especially about the future.

Tarjan: A dangerous topic. An easy way to embarrass oneself.

Levin: *[laughs]* Let me ask a future question that's really more about extrapolating from the past, and that is the role that algorithms [2:30:00] research has played in the development of the field of computing more broadly, which I think in the past we can see that it had been absolutely fundamental to things that have happened today. Behind many of the absolutely essential systems that pervade our environment are central algorithms without which they would not exist and that came out of… Maybe the particular algorithm didn't spring from a research setting, but the things that led to it did. As you've already said, there's this drive that you personally feel and that I think other algorithms researchers do, to find problems, abstract them, create algorithms that address those problems, and so on. There's a potential virtuous cycle there between the development of the field and algorithms research. Do we think that cycle is going to continue, or do we see things that might interrupt it?

Tarjan: My own view is that it will continue, although it may be… Well, what is happening with computer science is it's expanding in all possible directions. Now of course the hot area is kind of big data, machine learning, deep neural nets, so on and so forth, which has led to interesting things: the rebirth or resurrection of statistics departments, because there's a lot of probability statistics going into the data analytics. It has made linear algebra, applied linear algebra really important. There are certain classical optimization techniques that are actually used in the neural net training, because it's just tuning parameters. Then there's the question…

And it's been successful, unexpectedly successful. Somehow the idea of neural nets got started in the '80s and it languished for a long time, and it seems to be because there wasn't enough data and the computers weren't big enough and fast enough. Now it turns out if you get enough data, you get big enough, fast enough computers, you can do amazing things in certain domains – pattern recognition, figuring out kitties on the web, natural language translation.

The question is "Where are the limits?" We don't know the limits yet. It's good for some things. I suspect it's not good for everything. I want to understand things. One downside of these systems is they are inherently non-understandable. You get parameters tuned. How do you extract meaning from what these things are actually doing, what they get trained? That's an interesting, non-trivial question.

They don't exploit the idea of hierarchy or abstraction. I mean the successful systems mostly tend to be pretty homogeneous. I think they're going to run into combinatorial explosion problems when they try to tackle problems where there's inherent combinatorial explosion. That is I actually think the area of optimization

algorithms is very important and has kind of been neglected, but if you're a decision-maker, you need to get your hands on the data and understand the patterns. The neural nets are great for this, but once you have the information and you need to make a decision, if it turns into a complicated combinatorial problem like Amazon with package delivery or project management where you have to manage, assign people to tasks and so on, these problems, I don't see neural needs as being the right tool to solve them. I could be wrong. Which is to say this is a wonderful technology. It's a really good solution to some of our computational problems. It's not the solution to everything. Things are going to cycle around at some point when we figure out the limitations of these things.

I also think another big trend is cloud computing. Yes, in some of these interfaces to allow you to do large-scale distributed computation like MapReduce -- Hadoop and so on -- very simple paradigms, very general paradigms, which makes them incredibly attractive at least to first cut. But – this gets back to this first solution versus better solution – if it works fine but some of these things result in incredible expense, computational expense on problems where you could do orders of magnitude better if you went back and designed the right algorithm or even pulled a simple algorithm off the shelf… It is not just about scalability. It's about absolute efficiency on the particular problems involved. I think there needs to be a lot more work and a lot more general point of view as to what are the possible tools one can use. If you're in a setting where you try the MapReduce algorithm and the thing is still running for hours, what if you can reduce the time to minutes or seconds with many smaller computer resources? Shouldn't you be trying to do that?

So I think the heyday of algorithms as I have done them is not yet over. We have to bring in multiprocessors. A really hard problem. We got to worry about memory hierarchies and distributed memories. But these issues are not going to go away. People thought that nuclear power would make electricity free. Didn't happen yet. Computation is not free. Eventually the problems will get to the scale where cleverness, clever algorithm design is going to really pay off. Actually the bigger the problems, the *more* clever algorithm design is going to pay off. I don't see this changing, although there will be these cycles.

Levin: Even during the era when computing power was growing exponentially, our appetites were growing faster. So one can readily believe that what we want to do with algorithms and systems will outgrow the ability to do them by brute force, which is...

Tarjan: That is my suspicion. Yes. And it's a much more interesting world if we don't do everything by brute force as well.

Levin: *[laughs]* That's for sure. Well, it gives us a job if nothing else.

Tarjan: *[laughs]* Yes.

Levin: Let me ask you a question that's not strictly related to your field of expertise but, just as someone who has been a participant and an observer in the computing industry for decades, I've seen it move from a specialty to a pervasive thing in our society. As that has happened, as computers have become… and the software that we run on them and the data that we feed them have expanded into all corners of, at least First World and eventually more the world, society, they increasingly run up against problems that are non-technical. They involve legal and regulatory issues that are complicated. Perhaps the most evident one we see these days is self-driving cars. I think we've seen enormous progress in the technology of autonomous vehicles, but not a lot of progress in the other issues that they raise in introducing them actually into society and making them practical. Do you have any thoughts on how that will play out over the coming decades and the society, how it will try to address these issues?

Tarjan: Well, it is certainly a serious issue. This was brought to mind in another context, because some people view "algorithm" as a dirty word now. I got invited to a conference at NYU on kind of the ethics of algorithms by Helen Nissenbaum, the point being that all these computer systems, Facebook, Google, and so on, they're using your information and other information to guide you in searching or showing you certain stuff. You have no idea what the algorithm is doing behind the scenes. These things were either programmed or tuned on parameters somehow. There are biases built into these things that we don't understand. How do we deal with that? "Fake news," right? Everybody's in their own [2:40:00] cocoon of news, so there's no common view of the world.

This is much scarier to me than the self-driving car problem, actually. We don't understand the implications of this technology from a societal, civil society point of view, I think. At least people are starting to be aware of the problem. You can't start to deal with it until you're aware of the problem. There needs to be some human oversight, some human intervention if necessary, some acknowledgement of these issues. I think it's a very unstable situation right now and the technology changes so fast, it's very hard for people to keep up with these things. I guess I'm still an optimist, but we'll see what happens.

Self-driving cars. I think that's a relatively simple issue, although I think these other issues are much subtle and much more important. I mean kids walk down the street glued… not just kids, adults, and me too, glued to their cell phones. It's like all the world's information, all the video, everything, people get addicted to this stuff. We don't understand what is that doing to human beings? Our kids are getting raised in a completely different world than we were, and their kids are going to get raised in a completely different world than them. This is going to change society. What are the impacts? Very important questions.

So I think that's a great area if you want to… I always tell my students who want to do algorithms work or theoretical work, "Go find an application area. Find a problem in the application area. Turn it into an algorithms problem." Maybe the thing to do, or one place to go study, if you want to get into computer science these days is study the ethics, study the history, study the political impact, the societal impact, and so on. I'd like to see the humanities play into our world the way our world is playing into the humanities for example, where studies in the humanities are using computers to do statistical analysis and so on. It's revolutionizing the humanities. But there should be a way for classical knowledge to come in and address some of these problems that our technology is creating. I don't have any of the answers, but we got to have some kind of back and forth.

Levin: Absolutely agree. Are there any other topics that you would like to raise that I haven't covered before we wrap this up?

Tarjan: No. Just thank you very much.

Levin: Well, thank you.

Tarjan: *[laughs]*

Levin: Thank you. Mine was the easy part.

Tarjan: The Turing Award is a great honor. Again, with these awards, there are so many more amazing candidates than can possibly get them. I worked very hard, I think I did great things, but I was certainly lucky in getting all the awards that I got. So I think it's just a great thing.

Levin: Well, thank you very much, Bob.

Tarjan: Thank you.

[2:43:38] ***[end of recording]***