Scott Hanselman: This is ACM ByteCast a podcast series from the Association for Computing Machinery, the world's largest education in scientific computing society. We talk to researchers, practitioners, and innovators who are at the intersection of computing research and practice. They share their experiences, the lessons they've learned and their own visions for the future of computing. I'm your host today Scott Hanselman. Hi, I'm Scott Hanselman, this is another episode of Hanselminutes and we're doing this one in association with the ACM ByteCast so thank you to them for this partnership. And today we're talking with Dr. Jelani Nelson who's a professor of the Department of EECS UC Berkeley and a member of the Theory Group. How are you sir?

Jelani Nelson: I'm doing well, thanks for having me.

Scott Hanselman: Yeah. Thanks for hanging out. If people go and Google for you, you have been doing all kinds of really cool stuff online and in the computer science world for many, many years, it's really impressive. It's very different than I think a lot of people's backgrounds because I'm very much a practitioner who has very minimal computer science background and I'm really trying to get my head around, we're trying to build new coders, we're trying to build new developers, we're trying to build new computer science people, it's such a completely different spectrum between theory and algorithms and computer science and text boxes over data, which is what a lot of developers find themselves doing right now. Are these completely different concepts or how do we deal with that spectrum? How do we kind of reconcile that between the HTML of it all and the JavaScript of it all and the really good work that happens in the world of computer science itself?

Jelani Nelson: And when you ask are they really different, you mean software engineering versus doing theory?

Scott Hanselman: Yeah. Because I'm sure you code all the time certainly, but if you were going to make a website today what would you do versus when I go and make a website the algorithms that I might use would be very naive and very inefficient and never the twain shall meet.

Jelani Nelson: Yeah. Speaking of websites it's funny you asked because I learned HTML when I was maybe 12 or so, that was my knowledge of web design and then when I was a grad student at some point I wanted to have a website so I remembered that knowledge I acquired when I was 12 and made myself a website in Emax just typing the HTML by hand, it wasn't very good but it had the information that I wanted to put there so it was good enough.

Scott Hanselman: Yeah, exactly.

Jelani Nelson: And then more recently I wanted to make my website a little nicer so I learned a little Bootstrap and Hugo and put something together that's a little bit better, not a lot better but it's okay. That is very different from what I usually do, I am a

theorist, I've been a theorist since, well, even as an undergrad I would say I primarily took theory courses but there were some non theory courses I was required to take. And as a theorist and as an academic too my main output has been research papers, right? And not code. Although I did code for fun a lot starting undergrad, I was really into programming competitions carried that into grad school too.

Recently I actually did have an itch to do more stuff that's a little bit more connected to software engineering because I teach courses with titles like, algorithms for big data, for example, right? But as an academic it's like, well, I'm just in theory land, I'm not actually dealing with real data usually, right? So I wanted to do something a little bit more applied just to do something different. So starting this past summer, 2021, I actually spent a day a week at Google so I'm actually also a Google employee but 20% time. And it's been pretty interesting, I mean, I get a chance to use my theory knowledge to do real things and interface with other researchers at Google but also software engineers.

Scott Hanselman: What does that feel like? So you're spending 20 plus years doing theory and then suddenly you're in the weeds, a similar analogy, it's like you're a biologist and now you're in surgery and someone is cut open in front of you and you're like, mitosis.

Jelani Nelson: Yeah. It hasn't been exactly like that because the team that I'm working with there is within the research part of Google and my manager is someone I've known since I was a starting graduate student, we've written papers on similar topics or even the same topic. There's still some people there in that team who are still intermediaries between me and people actually working on products, although I do meet with the product people too. It's been pretty interesting in the sense that I knew that people were using some of the kinds of things I worked on in the real world outside of academia but I was a little removed from that and now I get to see why they actually care about these things. I mean, I kind of knew why they cared but I get to see it right in front of me and give my input and share my expertise in ways that hopefully help.

Scott Hanselman: That's cool. That is the goal, right? The Theory Group at Berkeley is trying to take those applications of the theoretical computer science and apply it, not just to software engineering but economics and physics and pure mathematics.

Jelani Nelson: Yeah. I mean, I would say theory is very broad so yes. There are parts of theory that are more connected to software engineering and here's a faster algorithm, here's an algorithm that uses less memory, less bandwidth or whatever resource you care about. I think theory is more than that, right? Theoretical computer science is also philosophical, right? What is the power of randomness in computation? Do we need randomness to have efficient algorithms? Right? That's complexity theories, studying kind of the limits of computation and what resources are required to solve various computational problems, or quantum computing, right? We don't actually have quantum computers yet but, I mean,

so it's somewhere between the boundary of science and computing, right? So if we had them what would they be able to do? What would they not be able to do? There's actually one quantum complexity theorists, Scott Aaronson, who likes to just talk about quantum complexity theories about what we can't do with computers that we don't have, right? But, I mean, there's some philosophical aspect to it too.

There's also the theory CS footprint on areas that are not part of CS at all like economics, right? So now there are a lot of people who have been working in that intersection between economics and computer science and what does theoretical computer science have to say about economics, for example. There's someone who's an alumnus of UC Berkeley, he got his PhD here Constantinos Daskalakis who was working on, and a former faculty member here Christos Papadimitriou was his advisor, proving things about the hardness of finding Nash equilibrium, right? So Nash equilibrium that's a concept that was invented by economists, right? Which Nash I think won a Nobel Prize for in economics. But computer science came along or theorists from computer science came along and said, hey wait, you say that this equilibrium exists but is it computationally tractable to find one, right? And if it's not computationally tractable to find one then how important is the concept, right? Does it actually model what people really do if they can't efficiently compute this thing.

Scott Hanselman: And then you've got this interesting relation between the math world and the context of John Forbes Nash Jr. who's a mathematician then game theory which brings in economics and now you're bringing in computer science and it's this cross collaborative triangle of trying to figure that problem out.

Jelani Nelson: And there are a lot of these examples, right? Now, another big one is privacy, differential privacy, right? So the field of law cares about privacy, right? But then now computer science is coming in and saying, hey, there's this notion that was invented or introduced maybe more than 15 years ago called differential privacy, you can mathematically define what it means to maintain privacy and then design algorithms or mechanisms that provably maintain privacy, right? Under this mathematical definition. So, I mean, theory CS has really, I think, made impact, I mean, to software engineering, yes, but also beyond, even beyond computer science.

Scott Hanselman: So that means that it's not just people like me that are doing, like you said, text boxes over data, it's my parents and my non-technical friends that these things are all affecting, it's things that show up in the news all the time, the differential privacy one being a perfect example, when we have conversations about metadata versus data, trying to explain what appears to be subtle differences to my non-technical family members but is actually a huge difference. And even now dealing with data everyone seems to become a scientist now during this time of COVID it seems that people who are least likely to do any work on the group project in high school also seemed to be the ones that are telling me that I should do my own research.

Jelani Nelson: Right. Yeah, I've seen a lot of memes since around that.

Scott Hanselman: Yeah. So you're teaching a ton of courses at the undergraduate and graduate level but if you're in undergrad in computer science and you're taking one of your classes, for example, efficient algorithms and intractable problems, what kind of a student and what kind of a researcher are you creating? Where do you anticipate that the person would go? Would they go into the computer science theory direction or would they just become a more well rounded engineer?

Jelani Nelson: Right. So that course you just mentioned is the UC Berkeley introductory algorithms course so they learn techniques like various graph problems, Dykstras and Bellman-Ford, various dynamic programming algorithms and the technique of dynamic programming, max flow, there are many different things in there, some randomized algorithms. The class is quite big, this semester I have on the order of 730 students in that class.

Scott Hanselman: Wow.

Jelani Nelson: Yeah. The vast majority are not going to go into research, right? They're going to go into industry, a lot software engineers. A lot of students view the class as a way to train themselves for coding puzzles that the big software engineering companies like to give, right? Algorithmic questions. I think aside from puzzles there is value in knowing how to design efficient algorithms, minimizing resource consumption whether it be time or memory.

Scott Hanselman: How do you not lose 600 of them? I mean, that's a gauntlet, it's almost the definition of gatekeeping to have a 700 person class. How do you make an inclusive course with that many people and not lose people just in the chaff?

Jelani Nelson: More teaching staff. So we have a teaching staff whose size is on the order of 50, lots of office hours, Piazza, I don't know if you know about Piazza, it's kind of a forum where students can ask questions possibly anonymously and get answers from teaching staff members and or other students.

Scott Hanselman: So hands on. So while it might sound like it's 700 people thrown into a pile and it's a big battle royale for algorithmic complexity superiority, it's really, there's a community within that organization itself, 50 TAs and beyond.

Jelani Nelson: Oh yeah. TAs and graders combined, yes, around 50.

Scott Hanselman: And are you finding that a certain flavor of person pops out the other side? Is it a certain personality that succeeds in a course like this or are we now in 2021 building more inclusive classes than I took in the late '80s in the early '90s?

Jelani Nelson: Well, I can't compare with the '80s and '90s because I wasn't in CS, I was a little kid back then.

Scott Hanselman: Of course, but you know what I'm saying. You've been teaching for some years, can you maybe think about how has your style changed in the last even 10 years to make sure that you make more successful theorists, more successful scientists and more successful engineers?

Jelani Nelson: My style has definitely changed. I think when I first started because I learned this stuff myself so long ago and it's second nature to me, I forgot how hard it is to absorb it the first time around. So I think the first time I taught a class like this I probably went too fast and tried to pack too many problems in each homework assignment. Actually, you mentioned this right before we started talking today, I run this program in Ethiopia where I train high school students there in algorithms, we've trained more than 500 students so far in the last 10 years, we started in 2011. And I remember the first time I ran that program, that was July, 2011, it was a four week program, we met Monday through Friday every weekday for about two and a half hours.

Almost none of the students knew how to write any code at the beginning of that program and by the last week I was showing them IOI problems, this is the high school programming Olympics, right? Kids who have been training in algorithms for years and were the top in their country, I was showing them problems from those Olympic contests and solutions to them. In retrospect I was going way too fast, right? For kids who had never coded a day in their lives, three weeks of training followed by a fourth week of Olympic level programming is probably a little too fast, but yeah, I've turned that down a little bit since then.

Scott Hanselman: The IOI being the International Olympiad in Informatics, as you said, the world championships the Olympics of algorithms. This is a four week course for high schoolers and it's not even 9:00 to 5:00, it's two or three hours a day.

Jelani Nelson: That's another change I made, now it is 9:00 to 5:00.

Scott Hanselman: It is 9:00 to 5:00. Okay. So this would be a bootcamp, it's an algorithmic bootcamp.

Jelani Nelson: Yes, that's right. Yes.

Scott Hanselman: Okay. And again, is it just one of those things like at bootcamp where if you survive you'll be a better person on the other side? How do you keep from losing people? And what's the secret sauce that makes your graduates so successful? Because if you take a look at the AddisCoder website that we'll link to in the show notes, you have not only 500 students but tons of them have gone on to MIT and become very successful and well thought of in their space.

Jelani Nelson: Yeah. Right. So let me back up a little bit. So you said kind of what helps to make the students successful?

Scott Hanselman: Yeah. What's the special sauce? Is it just you? How can someone become successful in four weeks? That seems like they may have had the right head space on the way in.

Jelani Nelson: Yeah. So first of all we recruit our students from all across the country, so the most populous city in Ethiopia is the capital Addis Ababa, less than 10% of our students come from the capital, we bring them from everywhere, from every region of the country. And we do that with help from our core organizing organization, it's called the Meles Zenawi Foundation, they're based in Ethiopia. They help us with the underground logistics, they help us work with the ministry of education, the government to get top students from all across the country. And Ethiopia has on the order of 110 million people, it's a big country, it's the second most populous country in Africa after Nigeria.

So you're pulling the top, nowadays, one summer program might train 175 students or so, so you're pulling top 175 students out of a population of size 110 million, right? So first of all, they're really smart, right? So that helps a lot. We do some admissions on our own outside from the ministry but we tell the ministry, the students you try to find for our program make sure though they're top in mathematics, top in physics, anything mathematical because algorithms is a mathematical thing, it's a mathematical field. And then when they come they're really not doing anything for those four weeks except our assignments, right? So they're really into it, we have a very large teaching staff, it's not just me, we usually have three to four instructors over the summer, I'm one of them.

But we also have a teaching staff that I think typically has in the order of 40 TAs usually half of them are based in Ethiopia, various undergraduates from various institutions in Ethiopia or some of them are lecturers at universities or work at an industry in Ethiopia and then the other half we fly in from all over the world. And they're really strong TAs, we have a lot of PhD holders or current PhD students, or people who themselves did IOI when they were in high school or similar kinds of competitions. We've had early engineers at big companies, I think the second hire at Dropbox was a TA for AddisCoder a few years ago, 2018, as was another early Dropbox, and we've had people from Google, from Microsoft, from other companies so it helps that we have very high quality teaching staff.

Scott Hanselman: The importance of algorithms, do you think that amongst people like myself who are more on the practitioners side, do you think that we're sloppy? Do you think that maybe there's a style of coding right now that is not respecting algorithms and stuff like that? Because things like IOI that I hadn't thought of, maybe you were surprised that I hadn't thought about it, the rank and file developer doesn't think as much about algorithms as I think people in your world do. Is that something that you think that the computer scientists on Twitter or the software engineers on Twitter maybe need to be refreshing themselves on?

Jelani Nelson: Possibly. I mean, again, most of my life I've been in academia so it's only recently that I've started more closely interacting with industry. And computer science is a big space, right? So there's room for a lot of different kinds of people with different expertise. Not everyone I think needs to be an expert in making websites as you asked me earlier, not everyone needs to be an expert in algorithms. But even me as an algorithmist I have an idea and then I code it up and do some experiments and I want to have a pretty interface to display the results of my experiments, I still need to have some kind of UI there that is useful, so it still could be useful to have knowledge from something that's not your core.

Scott Hanselman: Right. Well, the reason that I ask that and the reason why I'm kind of pounding on that is trying to understand the lenses through which different people look at different problems and there's that idea that when all you have is a hammer everything looks like a nail. And having now been in software engineering, not with a computer science but with a software engineering degree and a background for now 30 years, I'm starting to realize looking back that I had and have blind spots around theory and algorithms. And as I'm trying to teach the next generation of practitioner I want to understand where are huge sections of knowledge that I need to be pulling from and maybe I need to go back and fill in some of those missing Tetris pieces in my own knowledge and help make a more well rounded either academic or a more rounded practitioner and that's why I'm picking on you a little bit with those questions.

Jelani Nelson: Yeah. Again though, I think that CS is so broad that maybe it's necessary, maybe it's not, I'm not sure. You were at Microsoft, you mentioned, I don't know, how long were you at Microsoft?

Scott Hanselman: 14 years.

Jelani Nelson: Okay. I interacted with some theorists who are at Microsoft, Microsoft has a pretty strong theory group, for example in Seattle and Redmond, there used to be more but I think there's still some in the Boston area right next to MIT. There was a really good lab that unfortunately Microsoft closed seven years ago that was in Silicon Valley, I visited that place before, I gave a talk there once, and I knew a lot of the people there. I want to say two of the researchers there, Parikshit Gopalan and Sergey Yekhanin, they had some expertise in an area called coding theory which is a part of theoretical computer science. And my understanding is that they designed some coding scheme that was implemented in a Azure storage and saved Microsoft a lot of money, I don't know the details myself, it's not exactly my field. But there's coding theory, right? Coding theory even traditionally is I think considered part of electrical engineering. There's room for I think all kinds of different people and different areas of expertise to contribute at a large company like that.

Scott Hanselman: ACM ByteCast is available on Apple Podcast, Google Podcast, Podbean, Spotify, Stitcher, and TuneIn. If you're enjoying this episode please do subscribe and

leave us a review on your favorite platform. What is the sense of achievement that you get? When do you feel like you're done or you've done a thing? Is it applying an algorithm or someone taking one of your research papers and taking it and saving someone money or saving someone's life or something like that? I know I feel achievement when I hit deploy and then the thing has a URL or the app is in the App Store, but if someone took an algorithm of yours and did something with it or a paper of yours and applied it, does that give an extra sense of accomplishment?

Jelani Nelson: I mean, it gives some sense of accomplishment, yes. I would say though, most of the senses of accomplishment I obtain are from just pursuit of knowledge, academic pursuit, right? So there's this problem that seems like a really foundational or an interesting problem, important problem and no one in the world knows how to do it, right? And you keep thinking about it and thinking about it maybe with some friends, some colleagues and finally you crack it, right? And then there's that feeling of cracking the problem and solving it I think is the sense of accomplishment, then you write it up and move on. Sometimes it might make practical impact, in my case most of the time I think it doesn't but that's okay.

Scott Hanselman: Yeah, it is okay. It does make me feel some kind of way though when you say, and then I write it up and move on, because it's out there, someone could need that information.

Jelani Nelson: Yes.

Scott Hanselman: Yeah. How are you thinking about that? I asked Leslie Lamport the same question and I'm interested, is this a thing where you are fingers steepled and you're just walking around the quad with your head down, or is this eureka moments in the shower or is it really just years of hard work and then it slowly cracks open?

Jelani Nelson: I've been in academia now since I started my PhD, it's like 15 years, so I've been involved in a number of projects, I would say both have occurred. I've had projects where we had the problem crisply defined and we're thinking about it, we got stuck, maybe we could solve some special cases. We didn't eventually solve it until a couple years later or in some cases we never solved it, right? And then I've had other projects where we read someone else's paper and realize there was some problem there and it's like, oh, we happen to really understand the tools really well that would be useful for this problem and we work it out and it just works, it's like a day or two, right? So, I mean, that's happened too so I've had the full range.

Scott Hanselman: Yeah. What are some of the problems that you've been most proud of over the last 15 years? I know you did a lot of work in streaming algorithms and things like that, is there anything that you're particularly proud of looking back?

Jelani Nelson: Yeah. I mean, some of the papers I like of my own I would say one was in my thesis, the distinct elements problem, so what is that problem? Right? You see your server, right? People visit your website, when they visit your website they have some IP address and you want to know what is the number of distinct IP addresses that have visited my website. So you don't want to just maintain a counter because if the same person visits your website multiple times you'll count them multiple times, I only want to count them once because I want to know distinct count, right? So how much memory do you need to solve this problem? So one of the things I did in my PhD thesis together with two other researchers, Daniel Kane and David Woodruff was to basically nail the complexity of this problem.

So the relevant quantities are the universe that the items in the stream are coming from so IP addresses say integers from zero to two of 32 minus one, right? So that's the universe. It's an approximation algorithm so it doesn't give you exactly the right answer but it gives you the right answer up to say 1% error or 5% error, a tunable parameter. So the memory goes up if you want better accuracy, right? And understanding how to trade off the memory with the accuracy. So basically we can understand exactly what the optimal space complexity was with this problem. Actually, one thing we didn't fully nail was the dependence on the failure probability because it's a randomized algorithm that has some chance of outputting a wrong answer. But one of my PhD students actually, when I became a professor, went back and revisited the problem and also nailed the dependence on the failure probability and he wrote a paper on that, it was in his thesis, he won a best student paper award the Student Best Paper Award. So yeah, I mean, I think that's something that has been valuable.

Related to counting, I had another counting paper actually which we're about to submit to a conference in a couple months and that is just maintain a counter. So imagine now I just want to know, how many visits have I had to my website? Distinct or not it doesn't matter. So every time I see a new user I just increment, increment, increment, and at the end query, tell me the value of the counter. So the naive thing is just maintain a counter, right? So if the number gets to be as big as N you would need login bits to write down a number that's as big as N, but it turns out that you can actually do better if you're willing to settle for approximation and randomness.

And there was an algorithm from the '70s due to this guy named Rob Morris, not Rob Morris the professor at MIT and the inventor of the Morris worm, but his father. The father Rob Morris, he worked for Bell Labs for a while, he was chief scientist at the NSA for some time. But back in I think the early days of Unix he wrote a spell checker and I mean the story is online, let me not go into it. But basically in the course of writing that spell checker he realized that he wanted to maintain a lot of different counts in his code and the computers he had back then didn't have enough memory to store all these counts explicitly so

he needed some way to compress them. So he needed to be able to store a 20 bit number in only 10 bits, right? What does that really mean?

But you can do it if you're willing to approximate so based on the 10 bits you won't know exactly what that 20 bit number was but you can know it up to say 1% error and how do you increment the counter when you're only maintaining this compressed representation, so he gave an algorithm for this back in the '70s. What me and this colleague of mine, he's now a faculty at Princeton, Huacheng Yu, well, we proved the lower bound showing any solution to this problem must use at least blah bits of memory and then we showed that that lower bound we proved is actually optimal, there was an algorithm, right? Maybe I should have said the other way around. There was an optimal algorithm for this problem that we can analyze and prove there is no better algorithm up to a small concept factor.

Scott Hanselman: That's very cool. And then you take these concepts though after you've written the paper then a couple years later, for example, you might go and teach something about that. You've taught the distinct elements problem in your Harvard CS 229 class, that's something that you worked on, a paper you worked on now becomes part of who you are and part of how you teach thereby moving things forward for everyone else.

Jelani Nelson: Yeah. In that class you mentioned I actually taught a version of it at Berkeley too in fall 2020, that's a graduate class. Whenever I teach graduate classes I always have a final project component which is almost half the grade and I tell the students, find an interesting problem or a research problem and try to solve it. If you fail to do so it's okay, research problems are hard, you can fall back on writing a survey of what you learned or doing an implementation project where you implement lots of different algorithms you learned and experimentally compare them. But I encourage them to try and actually solve an open problem in research and some of them actually do, there have been a few projects that turned into research papers that actually solved some open problem.

Scott Hanselman: That's amazing. That's kind of that Good Will Hunting idea where someone who you don't expect suddenly fills out the whole impossible problem outside of the hallway there.

Jelani Nelson: Right. Yeah. I remember seeing that movie. I think there is a little extreme where it was just written on a board in the hallway and he just solves it, right? Whereas it's like these students are working hard.

Scott Hanselman: Yeah. They're working very hard. Speaking of working very hard, back to this concept, as we get towards the end of our show here, we were talking about the IOI, the International Olympiad in Informatics and you had an interesting observation about who goes to these. And I was trying to get to that root idea of how do we make more people? How do we even have them come from

different places, be different backgrounds, age, race, socioeconomic, who goes to these events?

Jelani Nelson: Sure. Well, I can tell you a little bit about the United States, I know less about how the teams are selected in other countries. So just for everyone listening a little bit more about the IOI, it's a high school competition, it's a programming competition where most of the problems are algorithmic in nature meaning you need efficient code not just code that solves the problem. And it's like the Olympics, right? So every country selects a team and that team represents the country in these Olympics and the location of Olympics changes from year to year. Every team has four students on it, you ask the question, how do you get to be on the USA team? Right? How do you get to represent the United States in this Olympic? So there's an organization called the USACO, USA Computing Olympiad, OSACO, and they run a sequence of contest throughout the year.

Most of them or maybe all of them are online, anyone can go to their website I think Google for USACO training gateway, maybe train.usaco.org. You can make an account, you can do practice problems on their website, and then you can also participate in their contests. And you start off in some division, I forgot what, maybe it's silver or bronze, I don't really remember, and from there when you do well enough you get promoted a gold and when you do well enough there you're promoted to platinum. And if you're one of the top platinum high schoolers in America you get invited to a live training camp in South Carolina at Clemson I think because the head coach of the USA team is a faculty member there that's why they have it there. And on the order of 15 to 20, I forgot exactly the number of high schoolers, make it to that camp every summer and then they run some contests there and provide some training as well and have some invited speakers to give lectures.

And then based on your performance in that camp you either make the team or you don't, right? This team of four. Okay. So about, who participates in this? When I was in college, I was class of '05 at MIT, many of the former IOI USA team members would go to MIT. And I noticed this anecdotally, the ones I was meeting not all, but many of them came from this one particular high school in Virginia called Thomas Jefferson High School of Science and Technology, it's a really strong public high school, it's a magnet school in Northern Virginia. And I always found that odd because America is a big place and how did it happen that in a team of four two of them are from the same high school in Northern Virginia. I was invited to give a talk to the training camp team in summer 2020 and I mostly talked about my research and I tried to get them excited about modern algorithmic research but I also had a portion at the end of the talk on the following.

So in preparation for that talk I went to the USACO website and I looked at who made the training camp in the last decade. It had the list of all the student names and the high schools they had attended and I made a heat map for myself and I can actually send you a link to that heat map too if you want to put

it in a note for the podcast, and I just looked at, what states were the students coming from? And the first thing that really stood out was half the country had zero students, in the past decade half the states were such that no student from that state made it to the camp. Okay. So let's look at the half of the states which did send students, of those half of them all the students who came from that state came from one high school in the state so which was weird. I think it was 26 states had participated in a decade, 13 satisfied the property that only one high school ever had representation at this camp. Okay.

And then we look at, okay, who are the top performers in terms of sending kids to the camp? California was number one, it sent something like 60 people to the camp over a decade, Virginia was number two, sent maybe a dozen or a little more than a dozen and then it quickly started falling off, I think Massachusetts, Texas, and I don't remember who else. But New York sent three or four in a decade compared to California which sent 60 and I thought that was weird. And then I zoomed in a little more on the California kids and I looked at, where were they coming from? And of the 60 Californian kids I think 57 were from the Bay Area and from those 57 I believe all but one were from the South Bay so what that means is maybe a five to 10 mile radius around the Stanford University campus, right? Cupertino, Sunnyvale, et cetera, San Jose, and I thought that was really weird, right?

The same phenomenon that I noticed when I was in college now it's shifted, Virginia is still there, a lot of those kids came from TJ, but now we see a lot of kids from the Bay Area and I realized it was all about exposure and access. So turns out there's an afterschool training camp in the South Bay where kids can pay to train, right? So that camp, it's an after school program, they hire former Olympiad competitors and other really smart people to train these kids. And this is a kind of training that you're not going to get in public school or even in most private schools actually, right? You really can't access this kind of training unless you live in the Bay Area, right? So I think that's a big part of it, right?

First of all, even knowing that it exists, when I was in high school I didn't even know to train for this thing because I didn't know this thing existed, I didn't know what an algorithm was until I went to college let alone training for algorithms as a high school student. By the way, I didn't even know what MIT was before I applied to MIT, I found out that MIT existed the fall of my senior year in high school. My dad bought this US news called Ranking Magazine, he brought it home in September, I looked at it, I saw that MIT was ranked first at the time in CS and first in math and then I only applied to MIT. I also didn't know that it was very hard to get in, that reduced the stress in my life not knowing but it could have been a disaster.

Scott Hanselman: Yeah. I'm feeling a little bit inadequate myself having just now learned that this algorithmic Olympiad exists even now. So access organizational willpower, someone putting on that event, a parent or a teacher encouraging someone to go to that event, it becomes a snowball that goes downhill which is great for the

people within a five mile radius of one particular area of a certain demographic of a certain area in Southern California, but what does it mean for the rest of us? Getting that access to folks whether it'll be through programs like AddisCoder or other online resources seems essential for the future.

Jelani Nelson: Yeah. Actually now that you've mentioned AddisCoder again, access and awareness, we do surveys in AddisCoder, right? Where we ask them, prior to AddisCoder what did you want to major in, in college? Most popular answer, medicine. And then we ask them at the end of the program, now what do you want to major in college? And some of them still say medicine but a lot of them start saying computer science. I forgot exactly what the number was but it tripled, right? The number who said they want to do it before the program to after the program tripled or more than tripled.

And it's just because if you're growing up in rural Ethiopia what your parents tell you, what society tells you is, if you're really smart and you want to be successful and have a good salary, be a doctor, right? People just don't know that CS is out there and what CS really means, right? Maybe they've used email before or they have a Facebook account, I guess, Instagram or TikTok for the young folks now, but they don't really know what it means to be a computer scientist, right? And I think that's part of what AddisCoder does is it makes them aware and they realize, oh, this is actually interesting.

Scott Hanselman: Very cool. Access and awareness is really where it all starts.

Jelani Nelson: I think it plays a big role.

Scott Hanselman: Well, thank you so much Dr. Jelani Nelson for chatting with me today. In cooperation and in partnership with the ACM ByteCast this has been another episode of Hanselminutes and we'll see you again next week.

Jelani Nelson: Thank you.

Scott Hanselman: ACM ByteCast is a production of the Association for Computing Machinery Practitioner Board. To learn more about ACM and its activities visit acm.org. For more information about this and other episodes please do visit our website at learning.acm.org/bytecast, that's B-Y-T-E-C-A-S-T, learning.acm.org/bytecast.