Me: This is Edward Auttonberry interviewing <DocX> at Louisiana Tech. Okay let’s get started. So, what is algorithmic complexity and what are complexity classes?

<DocX>: Those are good questions. Um, algorithmic complexity is a [pause] – there are a lot of definitions and there are a lot of ways you can say it – but basically, what algorithmic complexity is, is it is a way of attempting to measure the uh… growth rate of the resources that an algorithm needs, and by that I mean that as um… different types of problem solutions, different algorithms, for the same problem like searching or sorting, will have different amounts of memory or different amounts of CPU time they will need to solve the same problems. So, computer scientists have come up with a mathematics that allows us to describe the uh… growth rate an alg-… a particular algorithm’s growth rate for its need for resources as problem size increases. Now what are complexity classes? Well, we can talk about um… general classes of algorithms. Every algorithm that were interested in generally is uh… inputted as a computer program, and it would be very, very difficult to compute the exact runtime of every algorithm unless you just went out and ran it on large sets of data. And even if you did that, you wouldn’t know its exact runtime on the next set of data you ran it on. So, what we try to do is we try to say “Okay, can we come up with a general trend to describe the algorithm?” And from that, we have come up with a number of complexity classes. Uh, for example, a computer scientist will talk about a constant time algorithm. That means there’s an algorithm that will always run in the approximately the same amount of time regardless of the size of the input. You might say, well, “That doesn’t even make any sense. How could you have such a thing?” Think about a list and removing the first item from that list. It doesn’t matter how long the list is, it will always be a constant time algorithm – take the first item off of it. And then we have other classes. We have classes, uh… that are linear, for example. That means that the – the time, uh… that it takes to do something is a linear function of the size of the problem. Uh, we have uh problems, that we-we group into, uh, polynomial classes like n-squared or n-cubed, and basically that’s describing the shape of the function that generally describes, uh… that algorithms need for resources given increasing problem sizes.

Me: Okay.

<DocX>: That was probably a way longer of an answer than you wanted but um…

[Knocking on <DocX>’s door]

<DocX>: You might want to pause that.

[Interview is paused while <DocX> responds to person at door.]

Me: So, what about more conceptual complexity classes, like NP?

<DocX>: Okay, good, because when you say complexity classes, I wasn’t one hundred percent sure what you meant. Um… but, yeah. Um… So, everything that I described in my first answer was talking about, um… practical problems, um… that are less than exponential. Well, what does that mean? Um, basically, you can say that if the growth rate of resources is exponential in nature for a particular algorithm, that’s not a very useful algorithm as you’ll only be able to solve it on very tiny problems. Um, many of the problems in artificial intelligence, uh… have fallen into that category. So what computer scientists have done is that they-they know that there is, uh… the polynomial problems, the ones we’ve been discussing, and they know that certain problems are exponential, uh… which means that… it’s all – any… any kind of algorithm to solve this problem is going to need exponential resources, but there is a gray area in between the exponential problems and the polynomial problems, and we call that NP, which stands for nondeterministic polynomial. And, what these problems are, in simple layman’s terms, are problems that are very, very hard to solve; the only existing algorithms we have are exponential; but they’re very, very easy to verify that you have a correct answer, which is kind of weird. So, if I ask you to solve the problem, it is going to take exponential time. If I ask you to uh… to, to… to verify that an answer is correct, you can do that very quickly, in polynomial time. So that’s the class NP.

Me: Okay. So, can you really quickly describe the difference between NP and NP-Complete?

<DocX>: Sure. Um, NP-Complete is a subset of NP, and all NP-Complete is, is the hardest problems in NP. Now what does that mean, “the hardest problems,” well, and why do we study these? Well, we… we study them because if you could solve any NP-Complete problem in polynomial time then you could solve all NP problems in polynomial time. And Conversely if any one of the NP-Complete problems turns out that you can prove that its exponential, then all of the NP-Complete problems will automatically move to the, uh… exponential class. And the reason we can do that is part of being NP-Complete means that you have come up with a mapping between problems that you can do in polynomial time.

Me: Okay. So, we’re going to move away from like the actu-… the questions with answers. So, in what kind of situation would an understanding or application of this concept be useful?

<DocX>: That’s a very good question. Um, and maybe I’ll do it in kind of two parts. One is, in my answer to question one, when I was talking about “What is complexity?” okay, that is super critical that a computer scientist understand because, um… finding a better algorithm can give, uh… far more benefit than even buying a new computer or… or research into making computers faster. For example, the trivial example I always give in class is, um… there are two kinds of searching: one kind of search is called binary search and one kind of search is called sequential search. If we wanted – if… if the only algorithm we knew was sequential search, we wouldn’t be able to have uh, credit cards and debit cards because a transaction at a local supermarket or gas station where you slide in your card, uh that requires a lookup of your account balance, and if you had to do that with some sort of sequential search, considering the hundreds of millions of cards that are out there, it would literally take days to be able to process a transaction. But with something like binary search, there are more complex algorithms, but binary search is a simple example, it can be done in seconds or less. You do it every day every time you go to a Walmart or a service station. So that’s why al-… studying algorithms is so critical, because we need to be able to… to… to classify and find more efficient algorithms. Secondly, in addition to talking about the, uh… efficiency of an individual algorithm, we sometimes talk about the com-… uh, the complexity of a particular problem, like searching or sorting, and if we can prove mathematically that you’re never going to be able to do better than “n-log n,” for example, then, uh… we can stop wasting our time trying to find a better algorithm than “n-log n” if we’ve… if we’ve mathematically proven that there are… that there are no better algorithms. So that helps… that helps us focus our… our… our research. The second half if we talk about complexity classes like P and NP-Complete and exponential, why is that important in practice? Well, that… it’s a little bit more of a philosophical problem, but it could have real-world implications. Right now, most everyone believes that the NP-Complete problems are actually exponential – that there is… that there is no polynomial solution to any of those problems. But its just, people think. It’s not – no one has proven it. We might, at some point in time, find a polynomial algorithm for one of those NP-Complete problems, and if we did that, then suddenly all of those problems would be… would be solvable on a computer. And these aren’t just abstract problems. A lot of them have to do with, um… things like… like uh… finding efficient ways of shipping goods or uh efficient ways of configuring networks and, uh… things like that that are very critical problems to the way the world runs.

Me: Okay. In your opinion, do you think that this is an important topic for a computer scientist, software engineer, or cyber security indivi-… pro, uh… professional to understand?

<DocX>: All three uh... need to understand it and that, uh… is why in computer science the course that covers this material, 325, is required. Um, I don’t think it’s actually required in cyber engineering.

Me: No.

<DocX>: Um, but uh… um… I mean when you design a curriculum you have to make trade-offs. Um, I probably wouldn’t have made that particular trade-off. Um, I probably would have said – traded in – theory of cyber science, which I teach, um… and replaced that with 325, which is a far more critical course in my opinion.

Me: So, what… what would… how would you feel about introducing concepts like this more early on, not necessarily focused towards computers scientists but maybe like just as part of a science class in high school?

<DocX>: That’s a very good point. Um, if you look at the work that is being done in the living with cyber experience here at Louisiana Tech in computer science, um… that’s a freshman introduction to computing that’s very hands-on. Um, on a lot of those ideas there’s a lot of interest in translating those into a high school level and, in fact, there is a uh… a cyber boot camp type thing that’s run, uh… by many of the professors, Jean Gourd, for example, that’s involved with the freshman living with cyber experience to try to offer, um… a little bit of that as kind of a summer camp experience for students that are interested in computing or in cyber.

Me: Well, I mean more like making it part of a curriculum for high school students, maybe.

<DocX>: So, I think I see what you’re saying. Um, its challenging, um… I… uh, my approach to this is I think it certainly can be taught at the high school level, um… and probably should be taught at the high school level. There’s nothing really complicated about talking about talking about, uh… sequential search versus binary search. Every… Every child knows the higher-lower game where someone picks a number and you try to guess it by, you know, going higher – you telling them higher and lower. That’s basically binary search is how you can solve that problem. So that’s very intuitive, and uh it’s… it’s… it’s… clear even to a child that you’re going to get to the answer faster than guessing 1, 2, 3, 4, 5, which is a sequential search. So, yeah. It… it… it wouldn’t even have to be high school. You could… you could teach the basic ideas of here’s a problem, here are two algorithms for solving it, one is obviously a much better algorithm than another. You could… you could do that in day-school probably.

Me: Okay. Well, that’s it. That’s the end of my questions.