**Homework 5: NP - The Job Interview**

**Question.** We have discussed your programming skills, a variety of algorithms, and some general topics in asymptotic runtime and runspace. One last topic we’d like you to talk about is intractability. Can you briefly describe your concept of intractability, NP-Hardness, and NP-Completeness? (Please answer in general terms.)

*When thinking about Computational Complexity one of the important factors to consider is runtime, which can be different for many algorithms and can help you determine which algorithms to use and when. For example, if you are given a decent input size and your algorithm is shown to have exponential runtime, you would obviously want to find a different way to solve that problem because that would take way too long. When designing or searching for algorithms to use, the main thing we are worried about is efficiency. We consider polynomial time to be “fast” and there for optimal for time complexity. Intractability simply means to me that you have a problem that can not be solved in an efficient manner or can not be solved in polynomial time.*

*NP hard is a little more difficult to explain because of all the unknowns. If something is described as NP hard you probably should not try to find a polynomial time solution for it, although if you did, you would be able solve any other NP problem in polynomial time (p=np) . Every NP hard problem is at least as hard as the hardest problem in the NP class.*

*NP complete is a problem that is both NP hard and a part of NP. These are the hardest problems in the NP class. There are many problems out there today that are shown to be NP complete. If we have a decision problem x, and x is in NP… for it to be NP complete, every problem in NP must be reducible to x.*

**Follow-Up #1**. What is an example of an NP Complete problem that you find surprising?

*The bin packing problem, taking items of different volumes and trying to pack them into a finite number of bins/containers each of volume V in a way that minimizes the number of bins used. It is similar to the knapsack problem but rather than having 1 bin we can have multiple (but finite) bins. This is an NP Hard problem, but the decision version (deciding yes or no if all items will fit in specific number of bins) is NP Complete.*

*When you just think about a problem like this in your head logically, it is pretty easy to convince yourself that this is something that is very simple and should easily be done in polynomial time… We just place each item in one of the bins and if all the bins are filled up and we still have items, the decision is no, and yes otherwise. This is clearly not the case, especially when thinking about very a large number of bins and items. Derivations of this problem like the greedy first fit algorithm have been proven to be fast but not optimal at all. Definitely an interesting problem to me in the NP complete class.*

**Follow-Up #2.** OK, that’s all well and good, but how does this whole mess of intractability affect the practical issues facing algorithm design and implementation?

*You obviously want to be practical and efficient when designing or implementing algorithms, If you come across a problem that is shown or known to be NP Hard or NP Complete what should you do? If you don’t want to settle for something less optimal, your only choice is to design a polynomial time algorithm for an NP Complete problem and prove that P=NP, but that is obviously unrealistic or unlikely rather. Some of the other things that you can try are coming up with a similar, but easier version of the problem and trying to solve that in polynomial time. You can do this by looking at special cases of the problem or maybe adding in some constraints. At this point you are going to have to accept other than optimal solutions and move on or hope someone else can solve the P=NP problem.*

**RESOURCES**

<https://algs4.cs.princeton.edu/66intractability/>

<https://en.wikipedia.org/wiki/Time_complexity#Exponential_time>

<http://jeffe.cs.illinois.edu/teaching/algorithms/book/12-nphard.pdf>

<https://www2.cs.duke.edu/courses/fall18/compsci570/nphardness.pdf>