Introduction

Here are simply a couple of tips and tricks that I found particularly useful when doing algorithms. Some of which applies more to some questions while others. I hope whoever sees this finds this helpful.🤗
There might be certain topics I did not cover just because either they changed the spec or was irrelevant when I did the course.

General Advice

- **Timing:** When dealing with any of the assignments, know that the hardest part is probably putting words to your mathematical approach. Give yourself sometime when dealing with this part, cramming in one night will almost guarantee missing a important variable
- **Tutorial reference:** If in doubt, refer to the tutorial questions and how they manage to answer said questions.
- **Tutorial Assignment Similarities:** Chances are, there are questions in the tutorial questionsthat are similar to your assignment, always check first before any assignment to have a starting point. **They might ask you for more details in the assignment** so always make sure.
- **More the merrier:** If you think you should include/mention somthing in the assignment, **do so** (within reason), most of the time when they mark you down is when you forgot to make something (that might seem trivial) known.
- **Fake but simple data:** Because of how general some the question may be, make up some fake data and test your algorithm with that. Try to make sure your data covers some extreme edge cases, one solution make work for only a subset, but there might be 1 edge case you are missing.
- **Practice:** Finding algorithm is a muscle, the more you do, the faster you are to spot the solution. Its one of the ffew courses where brute force practice work. Sooner or later, you would realise how similar a lot of question are to each other.

Divide and Conquer

- With divide and conquer typically you will have 3 main techniques:
    - Merge sort (O(nlogn))

    - Binary Search (O(logn))
    - Splitting
    - Eliminating redundant action

- **Advice 1 - Stupid Way First:** Start by doing question the "stupid" long way. Trying to look for the answer immediately in really hard. Then review your stupid solution and see where you can apply the 4 main techniques. You will be finetuning your dumb solution until it meets the time complexity

- **Advice 2 - Big Oh hint/red herring:** For most questions, you would be given the Big Oh time complexity for your solution meet. You can use this to guess what potential techniques they could be illuding to. (However, don't be too hard up with one technique, sometimes they can throw you off (E.G linear technique with binary search may result in O(nlogn), iits not always solely merge sort))

- **Advice 3 - Power and exponential:** Sometimes, they may disguise the use of a logarithmic techniques with an exponential variable. A question may ask for O(n) but have a variable within it that is 2^n where apply log will result will result in n. Questions will always give you a hint on what to do

---

Greedy

- Greedy is typically use for a naive way of find maximisation and minimisation solutions. Typically there is 1 variable to look out for to be greedy with.

- The hardest part is less the solving, more the proving optimality.

- Some Dynamic Programming Questions may seem like greedy at first since they both tackle minimisation and maximisation problems

- **Advice 1 - Finding the variable to be greedy with:** Sometimes, the question may present numerous variables and conditions to consider at once. However, most of the time, they should be able to be simplified to one variable. (E.G Given monster take some x energy and give some y energy when defeated, make an algorithm that determine whether you can defeat all monsters and if so in what order. Might seem like 2 variables, but you can simplify the two variables into 1 "net energy")

- **Advice 2 - Edge cases:** Sometimes, there are edge cases that you may need to consider in your solution. A greedy solution may sometimes exceed these edge cases and you must account for them or at least mention them. (E.G If the energy required to kill a monster

exceed initial energy, pick the first monster where its energy to kill is less than the initial energy)

- **Advice 3 - Optimality:** (Tbf, I'm so so on this part) When looking to prove optimality, you should be looking to prove that your solution is the optimal solution. This can be done by assuming there exist an optimal solution different to yours. Then using *facts and logic* show how the optimal solution could only be your solution because of condition in the question. (I.E suppose there is an optimal solution, it will have to do x given the condition, hence its exactly my solution)

Dynamic Programming

- Dynamic programming, to put it simply, is the act of using recursion to solve a maximisation/minimisation problem.
- There are 3 steps of any dynamic programming, you always do them:
    - Find the subproblem
    - Find the base case
    - Define and explain the recursion case
- (Tbf, this is one of the harder parts of the course so my advice might seem obvious because yeah)
- **Advice 1 - Draw a tree:** It miight seem obvious, but drawing a tree is always helpful when using it to visualise recursion style questions.
- **Advice 2 - Designing the recusive case:** Since you will most likely be writing in latex, you may take quite a bit of time to draw out the recursive case than most. I found myself dickin' around with latex to be very time consuming and error prone.

Max Flow Algorithm

- The challenge of max flow comes less of the actual solving, more of the setup.
- While you need not repeat all the Max Flow steps in all your write ups, make sure you have a good undderstanding of how the algorithm works and what results you get from them
- Max flow algorithms require one to model a situation as a graph. Decide on the what represent vertices and edges.

- **Advice 1 - Dual edges:** Sometimes, you may be given a question where you are given a bidirectional edge, ensure you pick up on this.
- **Advice 2 - Draw it out:** Its a given for questions that involves graph but still draw out the graph no matter how trivial you may think it is.
- **Advice 3 - Take your time to explain:** You would most likely need to describe the structure of your graph with words instead of actually drawing one. Take your time to make clear what element models to vertices and to edges and explain why you can apply the said Max Flow Algorithms

---

This guy is a legend

`https://www.youtube.com/channel/UCZCFT11CWBi3MHNlGf019nw`