Introduction

Below contain a number of written out walktthrough solutions of some selected tutorial questions that I best believe explains and trains the various lessons and mindsets depending on the topic. Some of these were assignment questions from when I studied these topics.

Also note that the answers I have written are to demonstrate the thought process, don't let their "word count" intimidate you!

I wholeheartedly recommend attempting these yourselves before you looking at the answer (or not, I'm not your mother).

Divide and Conquer

DC 1 - Skipped Numbers

Question

```
You are given an array containing a sequence of 2^n − 1 consecutive positive
integers starting with 1 except that one number was skipped; thus the sequence
is of the form 1, 2, 3, . . . , k − 1, k + 1, . . . , 2n.

You have to determine the missing term accessing at most O(n) many elements of
A.
```

Answer

With every divide and conquer question, the first step is to figure out the naive method. Since we need to find the skip number we would need to find the place index where the number is skipped.

Since we have array A, it would stand to reason that for all element prior to the skip value (let's call it **s**), the difference between the index and the value is of 1. However, every number after the skip value would have a different of 2.

Example array A: [1,2,3,4,5,6,7,9,10,11,12,13,14,15,16]
(Skipped value is 8)

Before the skip:

```
A[0] = 1, A[1] = 2, A[2] = 3, hence A[b] = b+1
```

After the skip:

```
A[7] = 9, A[8] = 10, A[9] = 11, hence A[a] = a+2
```

Hence the naive solution:

```
Scan every index from start to finish until you find 2 index position, where the
difference between the index and the element is changes from 1 to 2. Time
complexity is O(2^n)
```

However, the problem is that you are scanning 2^n elements (re-read the question).

Notice that you are searching a value in an ordered data set. The first thing you should think of is "binary search".

So lets try binary search, but what are the partition conditions?
You know that if the current index/element difference is 2, every subsequent values would also have to be 2. You also know that if the current index/element difference is 1, every prior value will have a difference to 1. So checking those values are redundant.

Hence you are trying to find indices where the index/difference is transits from 1 to 2.

Optimised solution:

```
Apply a binary search algorithm on the array, where we drop the upper partition
when the index/difference of the current index is 2 or we drop the lower
partition when the index/difference of the current index is 1.
```

Since the binary search algorithm is O(log(n)) and the array size is 2^n. The final time complexity is O(log(2^n)), which simplifies to O(n)

Note that this question is a tricky one for it primes you into thinking linear search due to the expected complexity of O(n). But remember that the complexity is not simply dependent on the algorithm but on the data set size being operated on as well.

DC 2 - Distinct Pairs

Question

```
You are given an array A of n distinct positive integers.

1) Design an algorithm which decides in time O(n^2 log n) (in the worst case) if
there exist four
distinct pairs of integers {m, s} and {k, p} in A such that m^2 + s = k + p^2

2) Solve the same problem but with an algorithm which runs in the expected time
of O(n^2).
```

Answer

Part 1

Naive solution:

```
We test ever combination of 4 numbers in array. Since we have n distinct
integers, that would mean n * (n-1) * (n-2) * (n-3) combination. Which would
mean your time complexity would be, asymptotically, O(n^4)
```

Well that was terrible but let's have a re-read of the question. There are thing that you may have noticed.

The equation (m^2 + s = k + p^2), the LHS and the RHS are actually identical, albeit written in a different order.
So let's firstly re-write it: (m^2 + s = p^2 + k). Lets generalise this formula structure as (a^2 + b)

But so what? Well, since they have the same structure, we don'tt have to search every 4 number combinations but instead look at every 2 number combination and check whether numbers produce the same 2 answer.

So if we test every combination of 2 numbers instead, that brings our search space to n^2. We then calculate the result of generalised formula (a^2 + b) with every combination of numbers, and lets place it in an array R (for result). If any 2 combination are equal and their numbers are unque, it would suggest that these 4 numbers are unique. This will take time complexity O(n^2).

To search, we can simply sort the result array (keeping track of which 2 combination of number were used, maybe using a tuple or something), and linearly scan through the array and checking for duplicates. The sort is O(n^2 log (n^2)), which by way of logartihmic laws, becomes O(n^2 log(n)). The linear search is O(n^2).

At each result value that are the same, you check the 4 integers between the two values for duplicates. If you do find duplicates integers, you continue the search. If none of them are duplicates, 4 distinct integer exist to satisfy the original formula. Since it's a comparison between 4 values regardless of space, that would mean time complexity is O(4), which becomes O(1).

Since operations are carried out in consecutive order and the O(n^2 log(n)) is the greatest time complexity. The time complexity of this algorithm is O(n^2 log(n))

Part 2

As you can see with this question, they indicate they want an **expected time** of O(n). The one data structure that is commonly referred to is hashmaps.

Since hashmap improves our lookup time, notice that initial answer too an additional log(n) from the merge sort. So instead, store every 2 number combination in a hashmap, where their key is their result when passed into the equation (a^2 + b).

Then, check by brute force, every element of the n^2 combination pairs until a pair is found to share the same result and have distinct integers.

Hence the time complexity is O(n^2) instead

Greedy

GR 1 - Alice's Party

```
Alice wants to host a massive party with her friends. However, in an attempt to
stop awkward silence from happening, she has 2 criteria for those she invites.

Everybody she invites must know at least 5 other people, and NOT know 5 other
people.

She has a list of pairs of names, each pair representing two people who know
each other. There are N number of unique people

Create an algorithm that finds the maximum number of people Alice can invite
```

Remember that greedy does not necessarily mean that the you are picking the largest or the smallest between any decision. Greedy refers to the method of decision making, where you make 1 decision irrespective to past decision at any given point.

Create a list of people's name, followed by the number of people they know and do not know base on the pairing. The greedy variable in this case is given to you in the constraint: A person who knows 5 people, and not know 5 people.

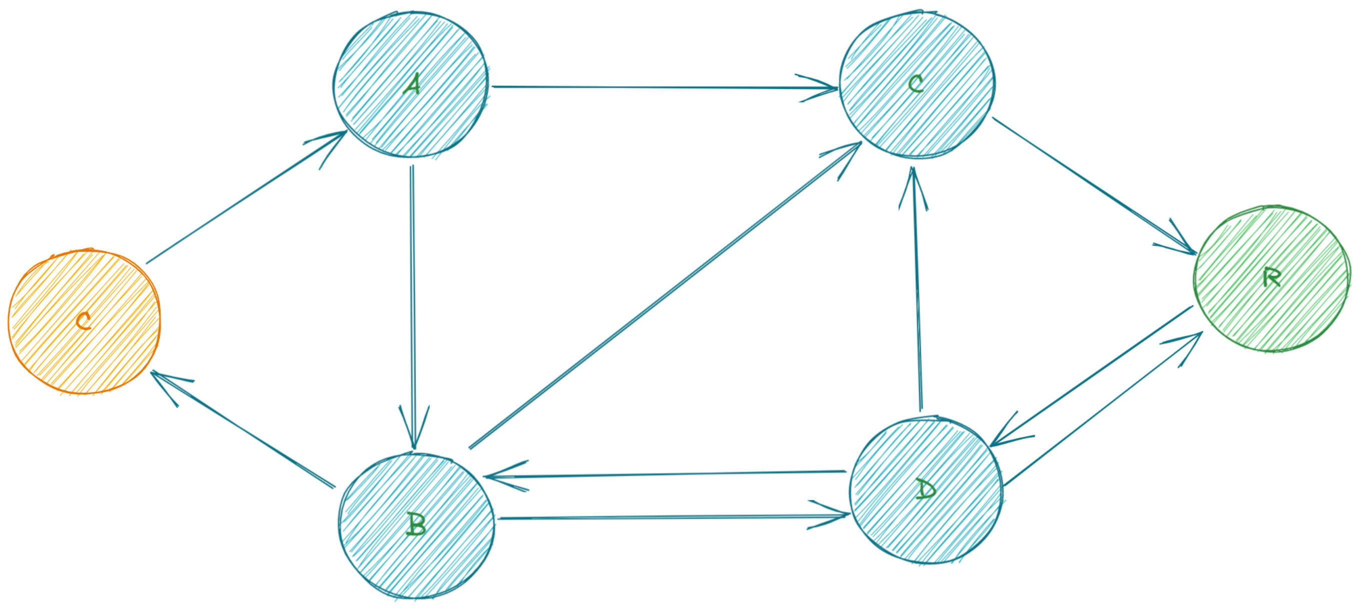
Eliminate anybody who do not fit within the constraint, each time recalculating the number of people which each person knows and do no know. Repeat this process until everybody who falls outside the constraint has been removed. The remaining people in the list would be the largest number of people that Alice can invite.

GR 2 - Roadtrip

```
In Greedinia, cities are connected by one way roads and it takes exactly 1 day
to travel between any two connected cities. (Amazing civil engineering).

If you need to travel between two cities but their is no direct road, you would
need to stay 1 night in all intermediate cities.

You have a map showing all of Greedinia toll charges for each road and prices
for cheapest hotel.

Create an algorithm which travel from city C to resort R with the cheapest route
```

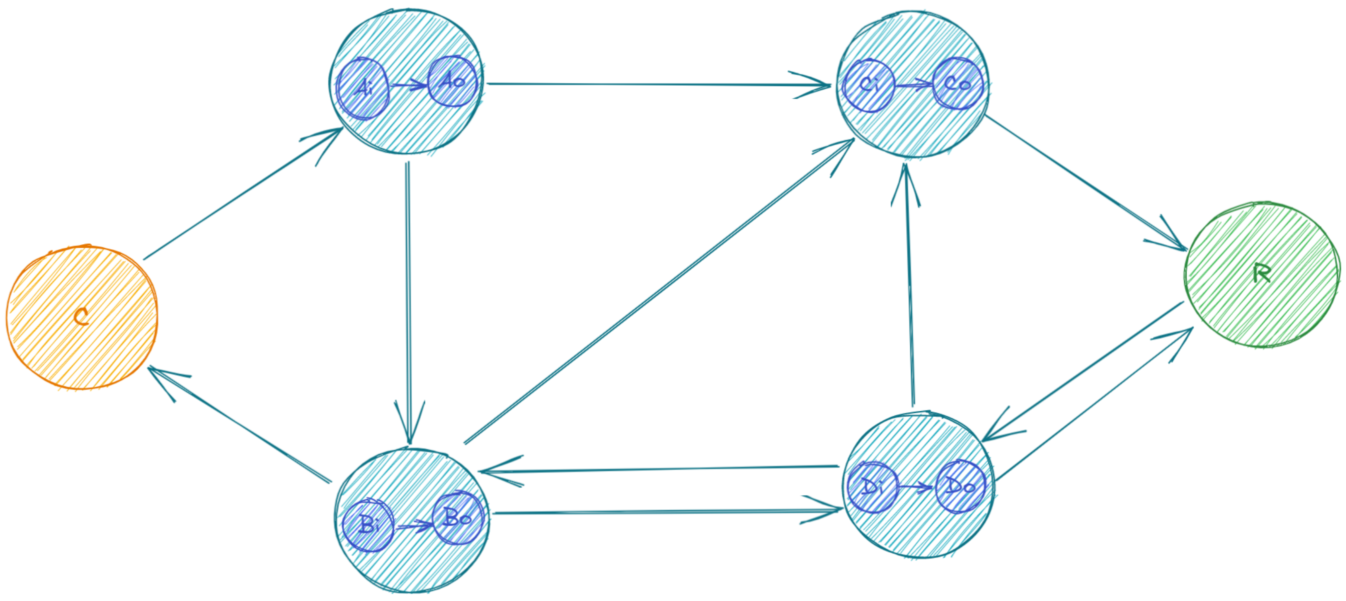
You can think of this question graphically, where each city is represented by a vertex and each roads is represented as an edge. The toll cost can be represented as weights on each road.

Dijkstra algorithm is a well known algorithm for finding the shorted path, under the assumption the weight of an edge is related to the weight. Hence if we were to use this algorithm on our graph, it should find the cheapest route.



However, you may notice that each city also incur a cost. The question is: "How can we attribute the cost of each town into the algorithm?"

Instead fo representing each city as a vertex, we can impose a weight by splitting each city into 2 vertices, an "in" vertex, and an "out" vertex, connected by a weight which is the cost of staying at that city. We hence impose a cost to each city.



Thus, by applying Dijkstra's Algorithm, we find the cheapest route from C to R with while considering both the cost of the toll and the cost of the city

Dynamic Programming

DP 1 - Holiday Planning

```
You are vacation for N days at a resort that has three possible activities
(Activity 1, 2, and 3)
For each day *i*, for each activity, you figured out how much enjoyment, *e*
(i,j), where *i* is the day and *j* is the activity you would derive from each
activity (Same activities on different day may offer different level of
enjoyment)
You are not allowed to do the same activity 2 days in a row. Determine maximum
enjoyment possible over the entire stay of N days and the sequence of activities
you should do each day.
```

Subproblem

What was the maximum enjoyment you could derive each day instead of the entire stay

Base case

Given that you start out with not acheiving any enjoyment since no activities has been done, your base case would be 0

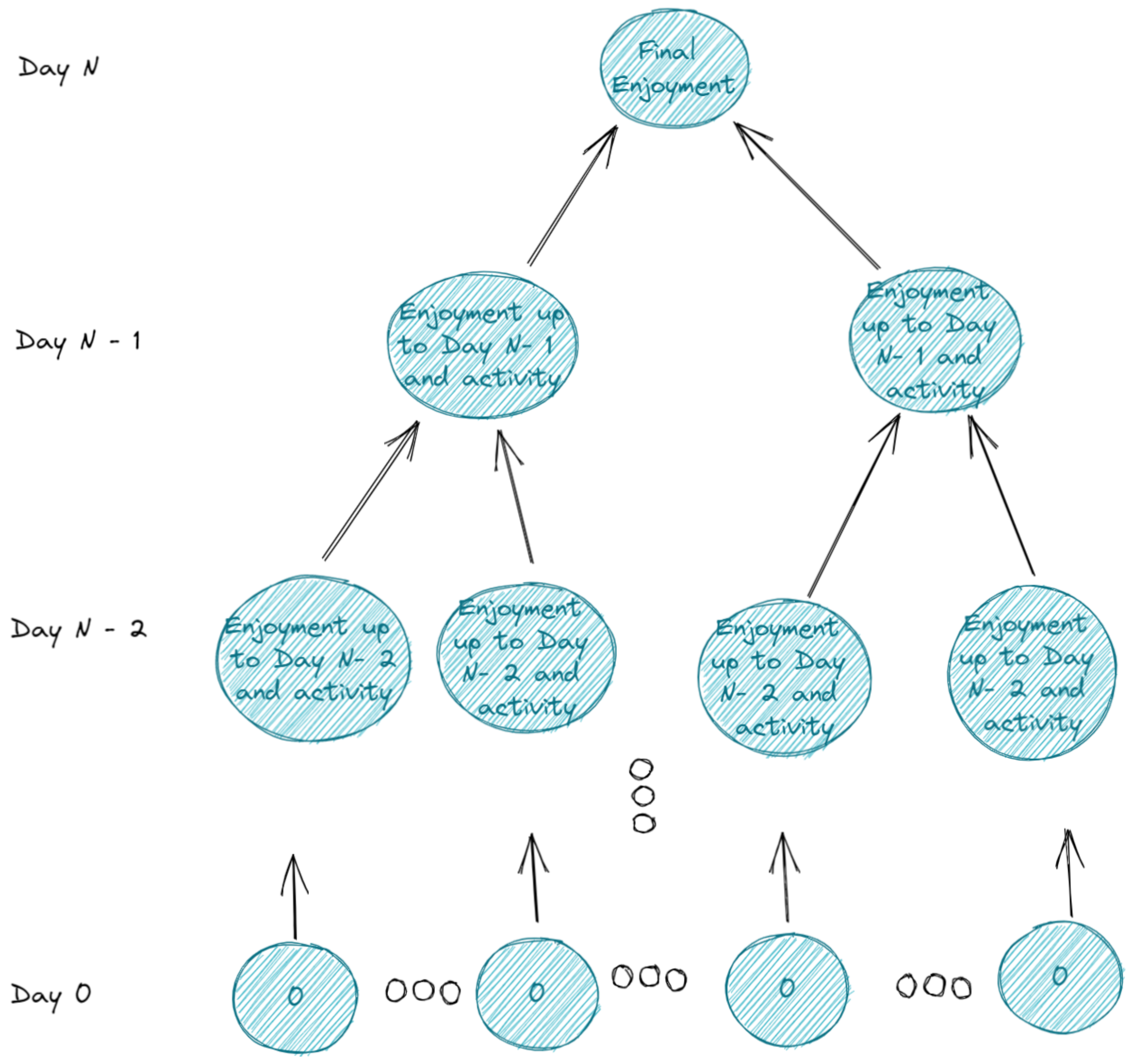
Recursive case

Let's think about this for a second, given that you have a fix number of activities, you can also define which activities can be done after another activities. This would mean that for 3 activities, you know that:

- Activity 1 is compatible with Activity 2 and Activity 3
- Activity 2 is compatible with Activity 1 and Activity 3
- Activity 3 is compatible with Activity 1 and Activity 2

You can think of this as your set of decisions when choosing which activities to do. This meant that for every day, you will be selecting the higher of the two resulting optimal solution based on the two selection.

Again, you should be able to represent most of the solutions, given their recursive nature as trees. Where the child of each node is the choice between two activities. The root node being the maximum amount of enjoyment and each previous node being the most optimal amount of enjoyment including their respective days



Day N
Final Enjoyment
Day N - 1
Enjoyment up to Day N- 1 and activity
Enjoyment up to Day N- 1 and activity
Day N - 2
Enjoyment up to Day N- 2 and activity
Enjoyment up to Day N- 2 and activity
Enjoyment up to Day N- 2 and activity
Enjoyment up to Day N- 2 and activity
Day 0
0
0
0
0

Knowing this, we can say that the for each day:

```
The most optimal decision for each day is to pick the highest of the two valid
activities summed to the optimal enjoyment of the previous day.
```

Or mathematically:

$$\text{opt}(k,t) = e(k,t) + \max\{(opt(k-1,s)) : s \text{ is compatible with } t\}$$

To find the order in which they are done, the activity which is selected should be recorded in order of how the recursive call is done back to the root/last day.

DP 2 - Shady Business

(I like to preface that this question is really weird nor did I come up with the question)

Question

```
You are handed the resposibility to run a business for the next N days. You have
K illegal workers.

At the beginning of each day, you may hire an illegal worker, keep the same
number of illegal worker or fire an illegal worker. (Only 1 worker to hire or
fire per day)

At the end of everyday, there is an inspection that you have between l_i and r_i
illegal workers, *i* being the day. If you do not fall between the values, you
will fail the inspection.

Minimise the number of inspection failed via firing and hiring illegal employees
```

Answer

Yes I know it's a weird question and I do not know why they picked such a weird scenario but lets look past that and just focus on optimising the immoral practice of firing and hiring of overworked illegal immigrants shall we?

Like any Dynamic Programming problem, you go through the 3 steps

Subproblem

What is the minimum number of inspection we can fail up to each day i given that we have j many illegal workers?

Base case

Well, if we think about it, since we would not have failed any inspection prior to starting the inspection, the base case would be 0.

Recursive case

Let's look at what options we have first:

- Firing an illegal worker
- Hiring an illegal worker
- Doing neither

Hence each day is determined by whether which of the 3 decisions we made in the prior day and since the question require minimisation of failed inspection, we want to choose the decision which minimises the the failure for the next day. Remember that we start with K illegal workers.

The number of workers will be defined as *j* which will bounded by the maximum and minimum number of hiring and firing we can do (Where we wither only fired or only hired everyday)

```
(K - i) <= j <= (K + i)
```

Let's also suppose that the inspection can be defined as a function that returned either 1 or 0 given the day and the number of workers.

```
inspectFailed( i , j )
```

Lets look at each decision in a vacuum for now:

Hiring an illegal worker:

Given that we are hiring an illegal worker, it stands to reason that on the previous day, we would have 1 less worker. Hence optimal function can be defined as:

```
opt(i - 1, j - 1)
```

Firing an illegal worker:

Given that we are hiring an illegal worker, it stands to reason that on the previous day, we would have 1 more worker. Hence optimal function can be defined as:

```
opt(i - 1, j + 1)
```

Staying the same:

Since we are doing neither, the worker represented by *j* will remain the same. Hence the optimal function can be defined as:

```
opt(i - 1, j)
```

For completeness sake, let's assume that any number of workers, *j*, that falls outside of the legal possible hiring and firing bound is defined as infinity (because we never want it selected in our minimisation function)

Hence we can say that the optimal decision is:

```
Given the day before, pick the lowest of the 3 decisions such that when added to
the result of the inspection, the value is the minimum
```

Mathematically, we can represent the decision as this:

$$\mathrm{opt}(i,j) = \mathrm{failed}(i,j) + \min \begin{cases} \begin{cases} \mathrm{opt}(i-1, j-1) & \text{if } j-1 \geq \max(0, K-(i-1)), \\ \infty & \text{if } j-1 < \max(0, K-(i-1)) \end{cases} \\ \mathrm{opt}(i-1, j), \\ \begin{cases} \mathrm{opt}(i-1, j+1) & \text{if } j+1 \leq K+i-1 \\ \infty & \text{if } j+1 > K+i-1 \end{cases} \end{cases}$$