Introduction

When it comes to divide and conquer, the purpose of these questions is to "eliminate redundancies in process". These redundancies are removed by splitting the variables of a problem into smaller more manageable chunks. Redundancies in these cases are removing the additional repeated steps in finding your answer. This is left semi-vague for unlike the other topics, there are not necessarily a clear step by step approach to tackling these problems.

Divide and Conquer

What is it

As mentioned, divide and conquer is the process of splitting a problem into smaller manageable space in hopes to remove redundancy in certain processing. However, the question then becomes:

- How do I split the data?
- What sort of redundancy?

Typically, the algorithm which exist for finding the solution to a D&C problem is quite obvious. However, the challenge is to re-work the algorithm in such a way that you can find the solution much faster. There are no hard steps of finding this, but a good general mindset to have is:

1. What is the obvious naive method?
2. What is the naive algorithm doing that is completely unnecessary/repeated?
3. How do I remove the repetition by dividing the space?

The celebrity search

```
You are an foreign exchange student who was just invited to a bitch'n party and
you are excited as you heard from a friend that a celebrity might be there and
you want to meet them. At the party, you notice a lot of people but forget those
nerds, you want to meet the celebrity.
```

```
The criteria for being a celebrity (both need to be satisfied):
- Everybody knows the Celebrity
- The Celebrity knows nobody there
```

```
Since your english is absolute trash, you cannot ask the obvious questions: "Who is the celebrity?" or "Are you the celebrity?". The only thing you know how to ask is "Do you (Person A) know (Person B)?"
```

What is the fastest way to find whether a celebrity is at the party and if so who it is?

Like all divide and conquer question. Let's first figure out the naive method.

Naive Method

Suppose there were 5 people in the party: A, B, C, D, E. The most obvious way is to slowly build up a table of connection and see who knows who and who doesn't. Let's start with person A, you ask them:

- "Does A know B?"
- "Does A know C?"
- "Does A know D?"
- "Does A know E?"
  Then you move on to the next person, Person B and ask them for their connection with everybody else. Now this is a very valid algorithm and it will always work.

So since you are asking 5 people 4 question each time, the time complexity is hence O(20), if we are generalising, the algorithm has a O(n(n-1)) time complexity or O(n^2) once we simplified.

Redundancies

Now can you spot what sort of questions were unnecessary? Consider the potential answer to those questions: Does X know Y? Yes or No. Remember, that a celebrity knows nobody but everybody know them.

Can you figure the elimination you can carry out?

If person A answer Yes to the question, recognise that they have actually violated one of the properties of being a celebrity: "The Celebrity knows nobody there". Hence you can eliminate A as a potential celebrity candidate!

Same goes for if they answer No to the question, recognise that the person who you ask if the Person A have a connection to have themselves violated the other property of being a celebrity: "Everybody knows the Celebrity"!

Eliminating Redundancies

As mentioned since you are removing 1 person from the pool of potential candidate per question asked, you can find the mostt likely candidate by asking n-1 questions.

However, we haven't satisfy the criterias ust yet. While we have figured out who is the most likely candidate, you need to ensure you satisfy the criteria that they know *nobody* and *everybody* knows them!. To do so you need to ask everybody if they know tthe celebrity, which is n - 1 questions and the potential celebrity if they know anybody which is an additional n - 1 questions.

Hence the final time complexity of this algorithm is O((n-1) + (n-1) + (n-1)) = O(3n - 3) which is in turn O(n). Significantly less time than the original O(n^2) method!