MESA

> StartBig

First Practical

2023

WHO IS THIS GUY?

Giovanni Manfredi

- MESA active Member since October 2022
- Participant at IT Sprint 2023 in Skopje, North Macedonia
- HO and creator of StartBig
- Tech savvy since I was 6 years old

AGENDA

- **MESA** - What is MESA?
- What **does** MESA do?
- How do I **join**?
- What about **StartBig**?

What is MESA?

Local Association

Milan **E**ngineering
Student **A**ssociation

European Association

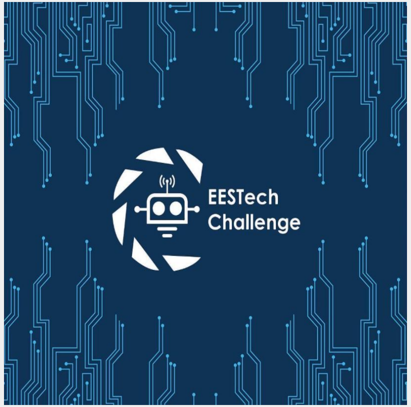
Electrical **E**ngineering **ST**udent
European Asso**C**iation

What does MESA do?

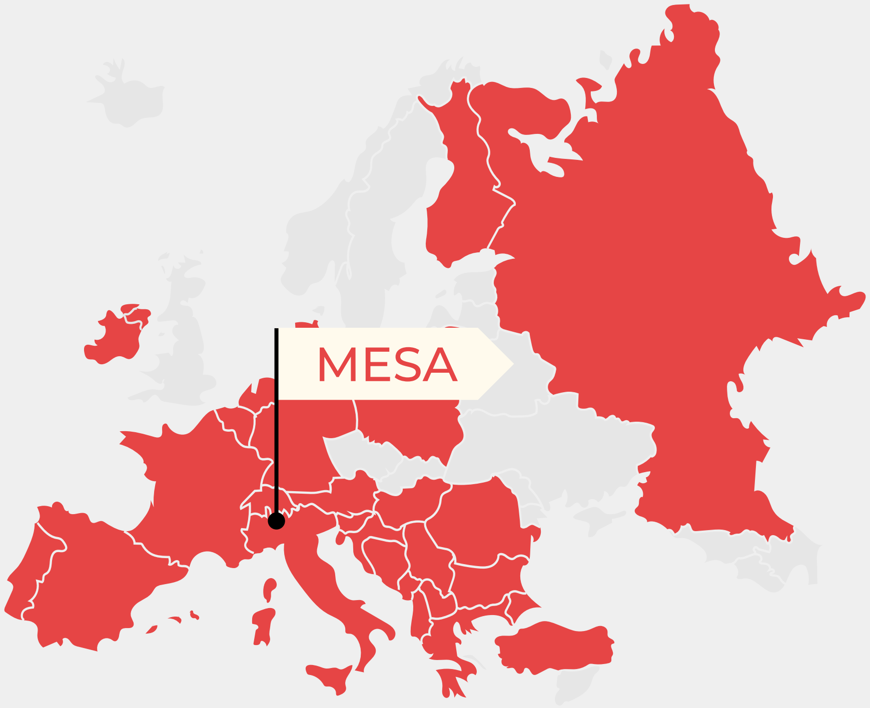
EVENTS!

Hard Skills

Soft Skills



An International Network



24 countries

43 universities

5000+ people

5 regions

How do I join?

Scan the QR and
click on the section Join us

What about StartBig?

CareerService session

How do we get to do a coding interview?

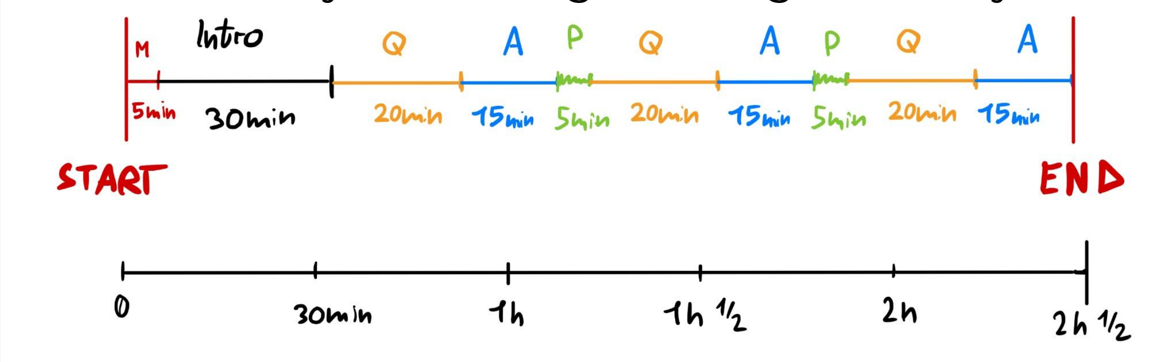
CareerService is here to help us find that out!

3 Practical sessions

How can we prepare for a coding interview?

Content of the sessions:

- Introduction to coding interview questions - held by me
- Trees & Graphs (DSF & BSF algorithms) - held by researcher Davide Yi Xian Hu
- What is Dynamic Programming? - held by researcher Nicolò Felicioni



Company session

How is the true experience behind coding interviews?

Andrea from **Oracle** will help us understand that!

There will be a small ice-breaker interview and then open questions from you!

STAGE TO …

Giovanni Manfredi

Yes, that’s still me

MESA

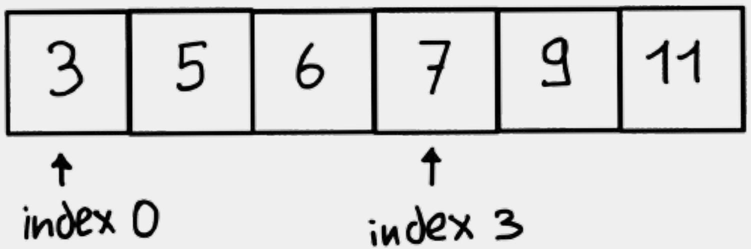
Tutors here to help!

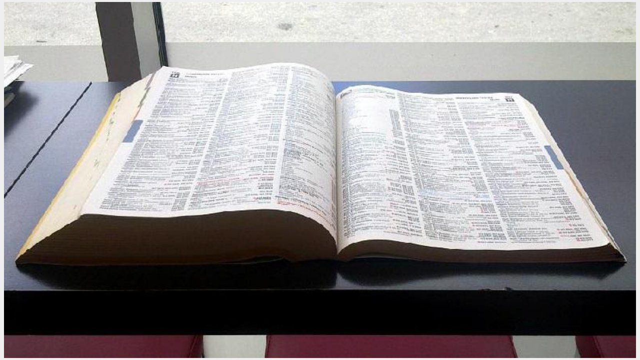
First Practical Meeting AGENDA

1. **How** do I **solve** a **coding interview** question?
2. **How** can I **optimise** my solution?
3. What's **LeetCode** and how does it work?
4. Let's **start coding**!
5. Let's see a **solution** together
6. **Repeat** 4. three times
7. **End** of the session!

Draw out the problem!

Example: Search in an ordered Array

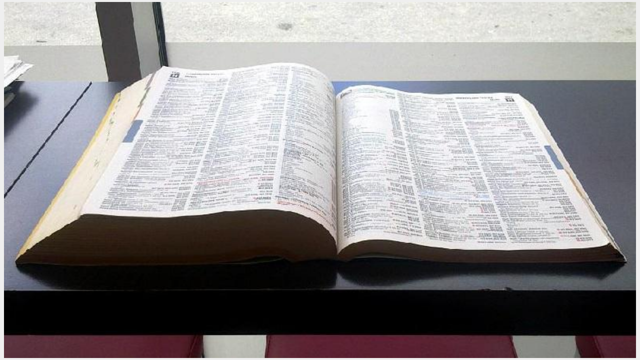




How would you solve it by hand?

Example: How do you search into a address book? You use the index!

If you open the book at random, you will turn the pages right or left **depending** on how the letter you're searching for **compares** to the one present in the address book!



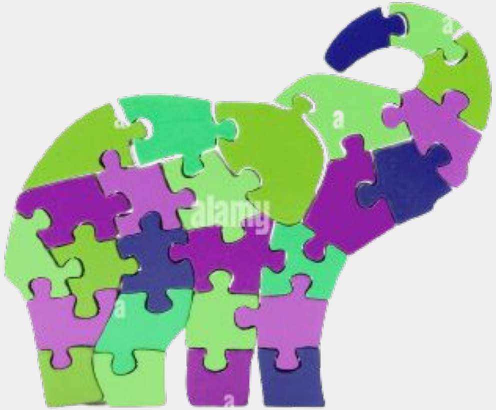
What are some more examples?

Example: think of other inputs to our program, does it still work with negative numbers? With zero numbers?

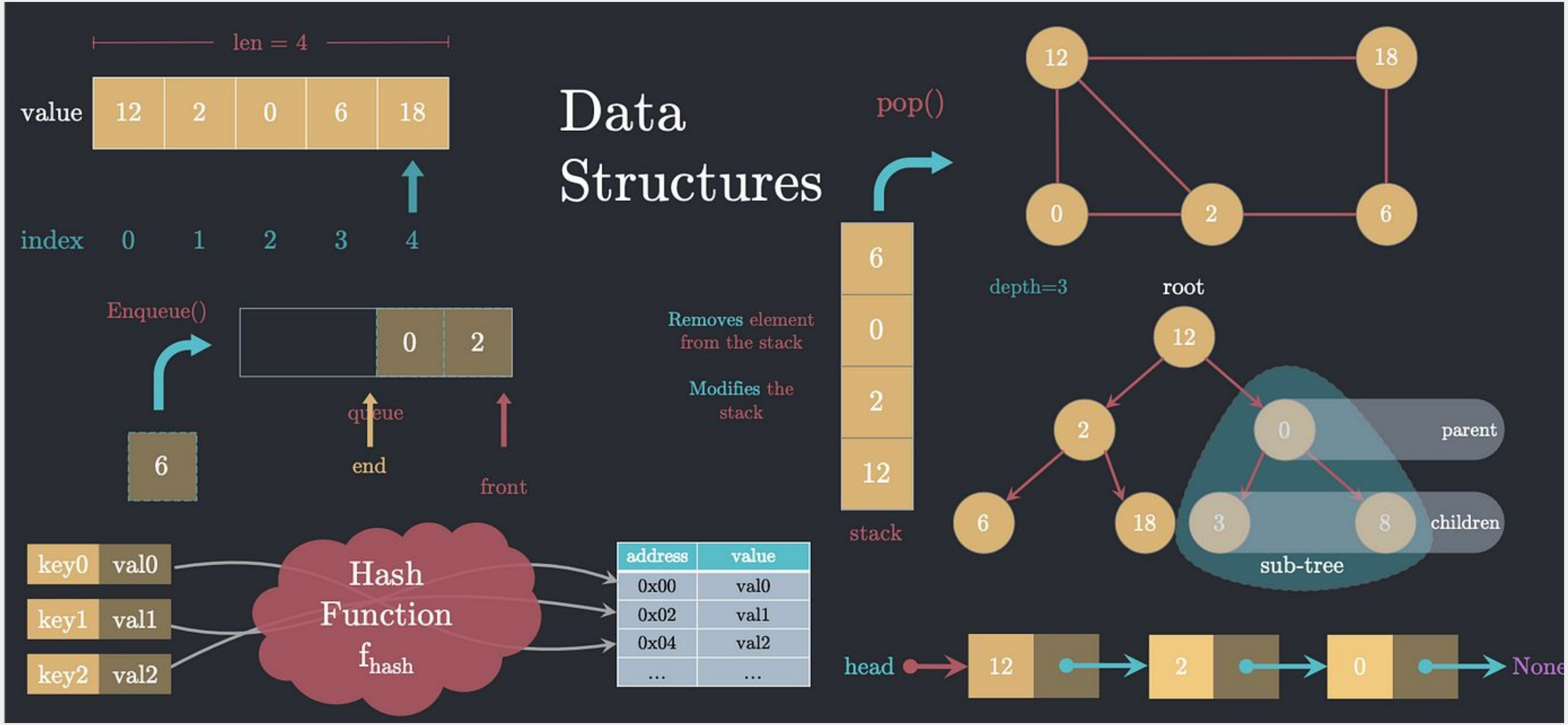
This also greatly helps with **test cases**!

Cut the elephant into pieces

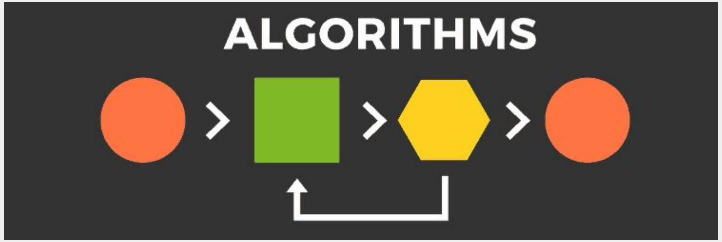
Divide your main **problem into** (ideally independent) **subproblems**



Remember and use your tools



Remember and use your tools



Common algorithms and approaches

- Sorting
- Binary Search
- Sliding window technique
- Two pointers approach
- Union find
- Breadth First Search (2° Practical meeting topic)
- Depth First Search (2° Practical meeting topic)
- Topological sorting

Optimising your solution

1. **Time complexity**
   the time complexity is the **computational complexity** that describes the **amount of** computer **time** it takes to run an algorithm
   a. Typically **more relevant** than space complexity in a coding interview
   b. We will use the **Big-O notation** → O(N), etc.
2. **Space complexity**
   The space complexity of an algorithm or a computer program is the **amount of memory** space required to solve an instance of the computational problemas a function of characteristics of the input. It is the **memory required** by an algorithm until it executes **completely**
   a. Typically **less relevant** than time complexity in a coding interview
   b. We will use the **Big-O notation** → O(N), etc.

Time complexity

We need to identify the **Best Theoretical Time Complexity** (from now on, **BTTC**) of the solution.
The **BTTC** is the the **time complexity that you cannot beat**.

Example:
The **BTTC** of finding the **sum of numbers** in array is O(N) because you have to l**ook at every value** in the array at least once.

NB! The **BTTC doesn't always correspond to the total number of elements in a data structure**. Think at the **Binary search** that uses the fact that the set is ordered to not look at every single element (time complexity O(logN)).

Where am I losing time?

1. **Identify overlapping and repeated computation**
   If your algorithm is doing something **repetitive**, that **you wouldn’t do** if you were to solve it by hand, think of another approach that it’s faster and doesn’t waste that time.

2. **Try different data structures**
   Knowing you data structures means to be able to understand when you need them and when you don’t. **Example**: if you’re struggling with lookup times, you might want to use an **hashMap**

Space complexity

1. **Changing data in-place/overwriting input data**
   If your solution creates a **support data structure** for the input, you can save some space by instead of creating it, **modifying directly the input**. This is <u>discouraged in software engineering</u> (hard to maintain), but **can be used in coding interviews** to reduce space complexity.

2. **Change the data structure**
   As for time complexity, also for space complexity **selecting the correct data structure is crucial** to reduce the complexity to the minimum.

LeetCode, your new best friend

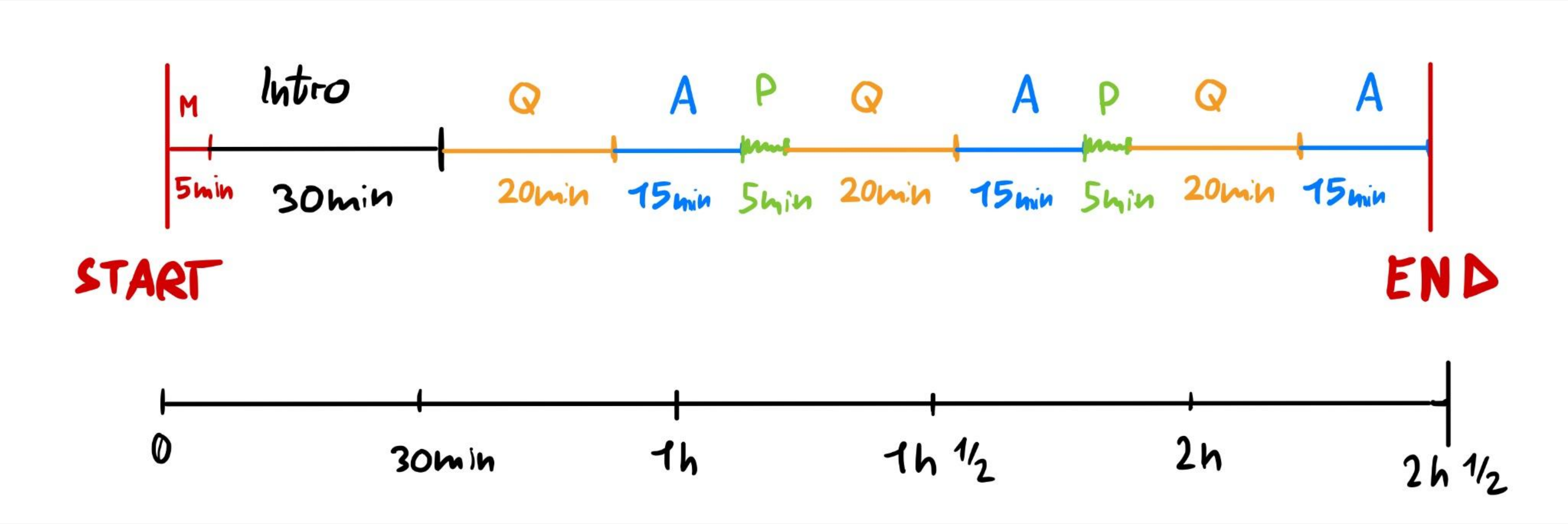
A perfect platform for **practicing coding problems** and to master coding interviews!

Problem difficulty:
As in a videogame, start easy, than go to higher difficulties

General advices

1. Try to **find a solution** even if not efficient
2. **Don’t look at solutions** straight away
3. **Tips** only **after trying**
4. <u>Practice makes perfect</u>
5. <u>Learn from your mistakes</u>
6. It’s more **fun with friends**

Let’s do some questions together!



3, 2, … 1, CODE!

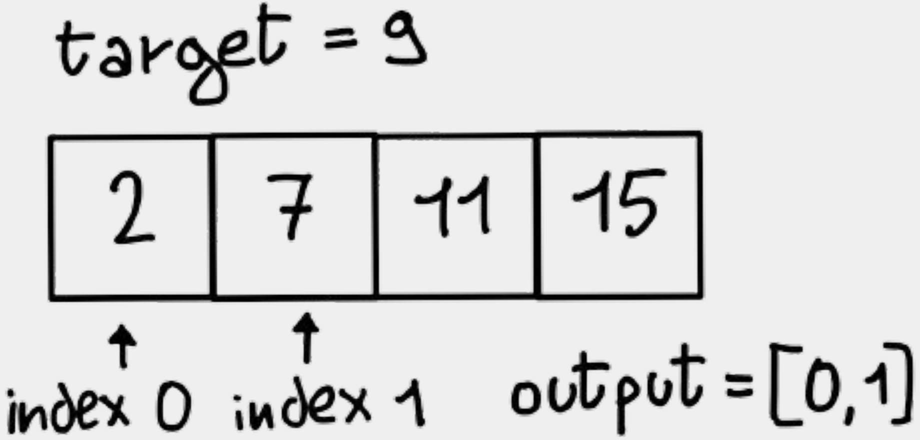
Your first exercise is **TWO SUM** (See GitHub repo)

GitHub

Start coding now.

Two sum solution

1. Visualise the problem: draw the array



2. Solving it by hand: comparison between one element and each other element

First implementation

- Time complexity: **O(N²)**
- Space complexity: **O(1)**

```c
int* twoSum(int* nums, int numsSize, int target, int* returnSize) {
    int *arr = malloc(2*sizeof(int));
    *returnSize = 2;
    for(int i=0; i < numsSize - 1; ++i) {
        for(int j=i+1; j < numsSize; ++j) {
            if(nums[i] + nums[j] == target) {
                arr[0] = i;
                arr[1] = j;
                return arr;
            }
        }
    }
    return arr;
}
```

Language: C

Two sum solution

3. <u>Some more examples</u>: let us make other sample arrays
4. <u>Cut elephant into pieces</u>: here we could think to **divide the lookup from the comparison part**
5. <u>Remember and use you tools</u>: the lookup part in the previous implementation took a lot of time. Maybe we could use a **HashMap** to reduce the time complexity

Second implementation

- Time complexity: **O(N)**
- Space complexity: **O(N)**

```java
class Solution {
   public int[] twoSum(int[] nums, int target) {
       Map<Integer, Integer> numToIndex = new HashMap<>();
       // For each number we verify if in the HashMap there is
       // its “complement” to get to target
       for (int i = 0; i < nums.length; i++) {
           if (numToIndex.containsKey(target - nums[i])) {
               return new int[] {numToIndex.get(target - nums[i]), i};
           }
           numToIndex.put(nums[i], i);
       }
       return new int[] {};
   }
}
```

Language: JAVA

3, 2, ... 1, CODE!

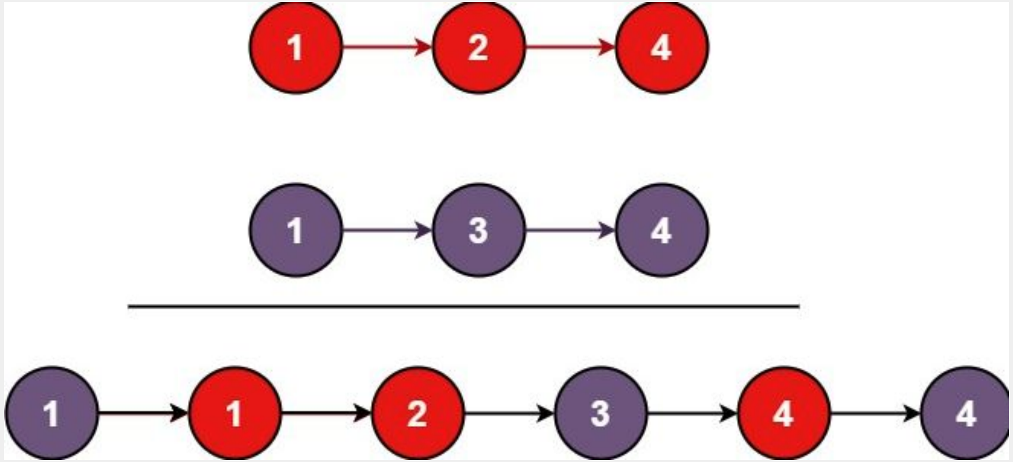
Your first exercise is **Merge two sorted lists** (See GitHub repo)

GitHub

Start coding now.

Merge two sorted lists solution

1. Visualise the problem: draw the two linked lists



2. Solving it by hand: going through each element of the two lists I would take the lowest for the new list

First implementation

- Time complexity: **O(N+M)**
- Space complexity: **O(1)**

```c
struct ListNode* mergeTwoLists(struct ListNode* list1, struct ListNode* list2) {
    if(list1 == NULL) {
        return list2;
    }
    if(list2 == NULL) {
        return list1;
    }
    if(list1 -> val <= list2 -> val) {
        list1 -> next = mergeTwoLists(list1 -> next, list2);
        return list1;
    } else {
        list2 -> next = mergeTwoLists(list1, list2 -> next);
        return list2;
    }
}
```

Language: C

Merge two sorted lists solution

3. Some more examples: let us make other sample lists
4. Cut elephant into pieces: here we could think to **keep track of the element of the two lists by having two pointers**, one per each list
5. Remember and use you tools: by using two pointers (references) instead of one we are much faster

Second implementation

- Time complexity: **O(N)**
- Space complexity: **O(1)**

```
class Solution:
   def mergeTwoLists(self, list1: Optional[ListNode], list2: Optional[ListNode])
-> Optional[ListNode]:
       cur = dummy = ListNode()
       while list1 and list2:
           if list1.val < list2.val:
               cur.next = list1
               list1, cur = list1.next, list1
           else:
               cur.next = list2
               list2, cur = list2.next, list2
       if list1 or list2:
           cur.next = list1 if list1 else list2
       return dummy.next
```

Language: Python

3, 2, … 1, CODE!

Your first exercise is **Top K frequent elements** (See GitHub repo)

GitHub

Start coding now.

Top K frequent elements solution

1. Visualise the problem: draw the array
2. Solving it by hand: writing down on a sheet of paper how many times a number appears and at the end of the array get our best K
3. Some more examples: let us make other sample arrays
4. Cut elephant into pieces: here we could think to **first go through the array and saving the frequencies**, then **calculate the top K**
5. Remember and use you tools: the best tool for easy access is a **HashMap** (so we can use it to save the information), then we can do a **Heap** to make the top K. Then the element in the heap will be our solution

First implementation

- Time complexity: **O(N log K)**
- Space complexity: **O(1)**

```python
class Solution:
    def topKFrequent(self, nums: List[int], k: int) -> List[int]:
        freq = {}
        for num in nums:
            freq[num] = freq.get(num, 0) + 1
        heap = []
        for num, count in freq.items():
            if len(heap) < k:
                heapq.heappush(heap, (count, num))
            elif count > heap[0][0]:
                heapq.heappushpop(heap, (count, num))
        return [num for count, num in heap]
```

Language: Python

T.HANKS everyone!

T.HANKS A LOT!