What is DSA? CP vs DSA?

DSA stands for **Data Structures and Algorithms**.

Data Structures means the way some data is stored. For example, we can store data as lists or in a stack or as a tree. All of these are examples of data structures.

Algorithms is a set of steps you need to follow to solve some problem. Either the set of steps is known (example, sorting, DFS etc) or you derive the set of steps using first principles and logical thinking.

CP stands for **Competitive Programming.** It is a **sport** where competitors solve DSA problems. They are either online or onsite. CodeChef, Codeforces are examples of online platforms where CP contests are organised. You can participate in these sitting from home. ICPC, IOI, CodeJam, local contests in tech fests etc are example of onsite contests.

CP vs DSA

▼ How is CP and DSA different?

Technically, CP is a sport version of DSA.

However, we often use “DSA” to refer to the action of people learning DSA for interview preparation or college exams whereas the term “CP” is used to refer to the action of people participating in the coding competitions / sport.

▼ So what will we be focussing on in this Cohort?

Our main focus will be “Interview preparation”. However this does not mean mugging up Leetcode or solving 1000s of problems there. Rather, we will be learning the basics and trying to master the **art of problem solving**. Also, we will be covering the popular data structures and algorithms as well. However, as of now, we will not be focussing on advanced concepts that are generally used in competitive programming and not super relevant for interviews.

Choice of Language

All of us wants to stay in our comfort zone. But it's very difficult to write code in every language or more than 1 language in a live class.

We will try keeping most of our classes **language agnostic.** We will focus more on how to think about the problem and solve it than writing the code. However, in some cases I will be writing some code - we can decide on C++ / JS or do alternatively. However, you will also get the code in all other languages in the reading materials.

▼ **My belief**

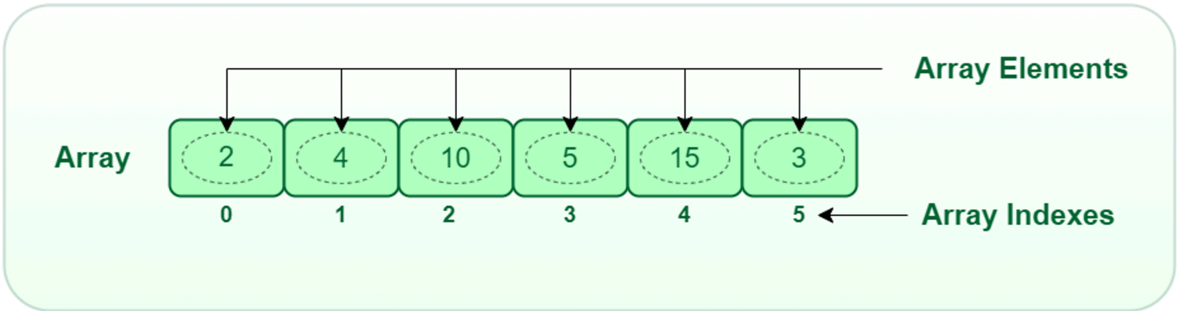
You are better off if I am not using the language you are comfortable in

▼ Why?

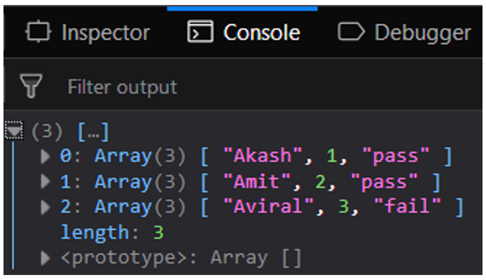
1. You are learning a new language which is always advantageous without additional (or very minimal) effort on your end
2. You get to write the code yourself in your favourite language without my help. Your implementation strength increases.

Arrays / Lists

Collection of elements of some (any) type is called an array / list



As mentioned above, the type of elements can be fixed or different depending on the language. For strongly types languages like C++, Java etc you need to specify the type of element the array will be holding. However for some languages (generally weakly typed ones) allow you to have heterogeneous arrays i.e a single array can hold different values (ex - Python, JS)



```
(3) […]
  0: Array(3) [ "Akash", 1, "pass" ]
  1: Array(3) [ "Amit", 2, "pass" ]
  2: Array(3) [ "Aviral", 3, "fail" ]
  length: 3
  <prototype>: Array []
```

💡 Python is not weakly typed. It does determine types and runtime and restrict some operations like 10 + "20". However, it does support heterogeneous arrays.

Operations

▼ Creating an array / list

- For strongly typed languages you need to provide a type
- You can also specify the size of the array / list
- If you don't specify the size, it creates a dynamic array

▼ C++ code

```
int arr[10]; // Fixed size

int arr[n]; // Dynamic size. Determined at runtime.

int *arr = new int[n]; // Dynamic size. Determined at ru
```

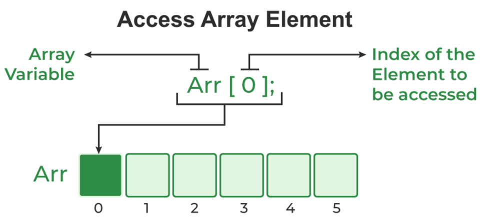
Copy

```
vector<int> arr; // Completely dynamic. You can add / re
```

▼ JS code

```
const arr = []
const arr = [1, 2, 3, 4]
const arr = new Array(5) // Not fixed size
```

▼ Access array element using index



▼ Iterating on the elements

You can choose to iterate through all the elements of the array. Generally you go left to right or right to left, one element at a time.

▼ Update the value at any index

Using an index, you can update the array value at that index. You can only do this for mutable arrays / lists however.

Strings

Array / List of characters is called a string. It is a collection of alphabets, words or other characters

Example:

```
"Hello world"
"Hello"
"Hello, I am 2020 graduate"
```

As it is a list of characters, you can perform similar operations on strings like arrays. Also the language often provides you with more capabilities for string apart from the array operations like splitting a string, searching for a substring within a string etc.

💡 In some languages (ex: Python, Java), strings are immutable even if arrays are not immutable.

💡 Whenever you get a string problem, try determining the possible set of characters. For example, if it is only lowercase or uppercase english alphabets, there are only 26 of them. If all characters (Extended ASCII) are allowed there are 256 of them.

Problem 1: Good Pairs

Problem Link: https://leetcode.com/problems/number-of-good-pairs/description/

Given an array of integers nums, return the number of good pairs.

A pair (i, j) is called good if `nums[i] == nums[j]` and `i < j` .

Example 1:

```
Input: nums = [1,2,3,1,1,3]
Output: 4
```

Copy

```
Explanation: There are 4 good pairs (0,3), (0,4), (3,4), (2,5) 0-indexed.
```

Example 2:

```
Input: nums = [1,1,1,1]
Output: 6
Explanation: Each pair in the array aregood.
```

Example 3:

```
Input: nums = [1,2,3]
Output: 0
```

Solution

▼ Initial approach

Let's consider a different problem. Given a part of the array [0...i] where i can be anything, can we find how many elements are there that are equal to arr[i] in this part of the array?

We can run another loop j from 0 to i-1 and check if `arr[j] = arr[i]` and increase count.

Now do the above logic for all i from 1 to n-1.

💡 Lesser no of nested loops, the better the solution is

▼ Optimal solution

Instead of running 2 loops, let's see if we can do with one loop. We are running the loop of i. What do we need? No. of elements on the left that are equal to arr[i] right? And what are the elements on the left? It is same as the elements that we have visited in the loop till now, isn't it 🙂 ?

So if we can maintain a frequency for all the elements that we have visited till now, in an array called `freq` , then `freq[arr[i]]` can be added to the ans.

Extension: Two Sum

Problem Link: https://leetcode.com/problems/two-sum/description/

▼ Solution

The problem reduces to the above problem, instead of finding frequency of arr[i] we just need to find `target - arr[i]` .

Think: How do we handle negatives? We can have negative array values as well, how to handle them?

Problem 2: Chef and Happy Strings

Problem Link: https://www.codechef.com/practice/course/strings/STRINGS/problems/HAPPYSTR

Chef has a string *S* with him. Chef is happy if the string contains a **contiguous substring** of length **strictly greater** than 2 in which all its characters are vowels.

Determine whether Chef is happy or not.

Note that, in english alphabet, vowels are `a`, `e`, `i`, `o`, and `u`.

Example 1:

```
Input: aeiou
Output: "Happy"
Explanation: "aei" is one contiguous substring with length strictly greater t
```

Copy

Example 2:

```
Input: abxy
Output: "Sad"
Explanation: There is no substring of length > 2 containing all vowels
```

Copy

Solution

▼ Brute force

Generate all substrings with length > 2 and check if anyone of them is completely made of only vowels.

```
bool checkOnlyVowels(string s) {
  // Returns true if only vowels in s, false otherwise
}

for(int i=0; i<n-2; i++) {
   for(int j=i+2, j++) {
     if (checkOnlyVowels(s.substr(i, j)) return true;
   }
}
```

Copy

```
return false;
```

▼ Optimisation

Observation: If a substring of length 4 or more has only vowels, then any substring of that substring of length 3 will also have only vowels. Thus, we can only focus on substrings of length 3.

Problem 3

Q - You are given a string containing lower-case english alphabets. You need to perform the following operation as many times as possible:

Take the first occurrence of each alphabet if it exists and remove it from the string. Eventually the string becomes empty.

For example, let's say the string is "ababcba".

In the first move, we need to remove the first occurrence of each character 'a', 'b' and 'c'. It becomes

"ababcba" → "abba"

Again, we repeat unless the string becomes empty

"abba" → "ba"

"ba" → ""

Find the last non-empty value for the string.

Example 1:

```
Input: s = "ababcba"                    Copy
Output: "ba"
Explanation:
```

Solution

▼ **Brute force**

We can do the moves iteratively, till we reach empty string. Also, before we do a move we need to maintain the last string value as well. Once we reach empty string, the ans is the last string value.

```
// Pseudo code                                                    Copy
function make_move(s) {
   // remove first occurence of each character of s and return r
}

last_val = ""
while (s is not empty) {
    last_val = s
    s = make_move(s)
}
```

▼ Optimisation

💡 Whenever there is some moves involved, generally there is some pattern that is followed that can help us quickly determine the final state

Let's answer these 2 questions:

1. What will be the final set of characters?
2. Which occurrence of these final set of characters are remaining?

Focus on this: We are removing **one character** of each type in every move. What will be left? Obviously, the characters with maximum frequency. We can quickly determine that running 2 loops right? One for frequency and one for getting the characters with max frequency.

Now for 2, focus on this: Take the **first occurrence.** So essentially what will be pending is the last occurrence of these characters right?

So final answer will be, the last occurrence of the maximum occurring characters.

Now we have everything, but can we write a cleaner code? Yes. What is the last occurrence of a character? It is the first occurrence in reverse right? Now you want to generate a string containing only **last occurrences** of **some** characters. How to go about it? Take the first occurrence in reverse order. But the final string is reversed as we travelled in reverse order. So the answer will be reverse of derived string.