Group 3 Manual

MENTORED BY
Anshul, Ujjwal and Manas

Compiled on 06/04/2021 at 01:13:00

This document will be updated regularly so make sure you are viewing the latest copy.

Contents

Chapter 1

Assignment 1

1.1 System Setup

- Install an editor/IDE
- Choose a Language
- Install a compiler

Chapter 2

Switching from C to C++

2.1 Why so serious?

```c
#include <stdio.h>

int main() {

        int n;
        long long m;
        char s[1000];

        scanf("%d%lld%s", &n, &m, &s);

        printf("%d %lld %s\n", n, s);

        return 0;
}
```

becomes

```cpp
#include <bits/stdc++.h>
using namespace std;

int main() {

        int n;
        long long m;
        char s[1000];

        // a new data structure for strings which makes life easy
        string ss;

        cin >> n >> m >> s >> ss;

        cout << n << ' ' << m << ' ' << s << ' ' << ss << '\n';

        return 0;
}
```

2.2   C++ style strings

In C++, the **string** data type is very similar to character arrays, the only thing is here we do not have a NULL terminator. You get the following new functionalities:

```
string s = "Hello", s1 = "abcd", s2 = "abcd";
```

- append characters :

  ```
  s += '2';
  ```

- append string :

  ```
  s += ", how are you?"
  ```

- compare strings :

  ```
  if(s1 == s2) cout << "Strings are equal!!\n";
  ```

- indexing operator[ ] :

  ```
  for(int i = 0; i < s.length(); i++) { cout << s[i]; } cout << '\n';
  ```

  will print all the characters of the string. But that is not how you print a string, you directly do

  ```
  cout << s << '\n';
  ```

Chapter 3

Assigment 2

3.1 How much do you know?

The aim is to gauge your level of expertise so that we can provide everyone with what they want without wasting anybody's time. Don't cheat or lie, it will defeat the purpose of the assignment.

3.1.1 Level 1

1. Fabonacci (Editorial)

2. Watermelon (Editorial)

3.1.2 Level 2

1. Zeroes to the end If you do it in $O(n^2)$ we won't mind, but leetcode might give a TLE.

   Editorial: in $O(n^2)$: Just like insertion sort, pick every non-zero element and put it at the correct position.

2. Count Primes (Editorial)

3.1.3 Level 3

1. Zeroes to the end IMP: In a single linear $O(n)$ traversal, Editorial : See Two Pointers Chapter

2. Convert to base 2 (Editorial)

Adding the entry to the google sheet
If you solved 1 from L1; 1, 2 from L2 and none from L3, you write it as **"10 11 00"**.
Someone who solves all will write **"11 11 11"** and will color his box green.
Partial solvers color their box blue.
Those who tried but solved none will color their's red.

This will be followed for all the future assignments. This assignment needs to be done within next 4 days, and then we will have a little discussion if required.
Indenting the code is very important, I personally follow the K&R style but you can choose Google, LLVM or clang. Use your left pinky finger to hit the $\boxed{\rightarrow}$ key.

Chapter 4

Baby Steps

4.1 Making I/O fast in C++

These two lines of code can make your C++ runtimes faster by 50%, you only notice the difference on large testcases.

```
ios_base::sync_with_stdio(false);
cin.tie(NULL), cout.tie(NULL);
```

4.2 A basic template file

A template is a piece of code you can copy everytime you solve some problem.

If the problem has no test cases:

```cpp
#include <bits/stdc++.h>
using namespace std;

#define fastIO ios_base::sync_with_stdio(0 && cin.tie(0) && cout.tie(0));

typedef long long ll;

int main() {
        fastIO;

        // insert your code here ....

        return 0;
}
```

If the problem has test cases:

```cpp
#include <bits/stdc++.h>
using namespace std;

#define fastIO ios_base::sync_with_stdio(0 && cin.tie(0) && cout.tie(0));

typedef long long ll;

// this solve() function can be treated exactly like main()
int solve() {
```

```
10
11          // insert your code here
12
13          return 0;
14  }
15
16  int main() {
17          fastIO;
18
19          int t;
20          cin >> t;
21          while(t--) {
22                  solve();
23          }
24
25          return 0;
26  }
```

4.3 Frequently seen numbers

There are a few numbers like 10^7, $10^9 + 7$ (1000000007) , etc. which you will frequently find in problems, we will discuss what they mean over here.

- You can run $10^{7 \text{ to } 8}$ iterations in around one second, so mostly you will see that they give size n of array equal to 10^5 and number of test cases of the order of 10^3 or 10^2.

- $10^9 + 7$ (1000000007) is provided as a huge prime number you will need to take a modulo with of your answers in combinatorics type problems, where the answer can be so large that it will overflow the 64 bit integer (long long) too! You will need to know the modulo arithmetic rules for solving such problems. 998244353 is another prime number which might be found in a few problems.

- $10^{-9} \leq x \leq 10^9$, means you need to use **int** for storing x, anything larger than that goes inside the 64 bit **long long**, which will mostly be mentioned as $x \leq 10^{18}$.

4.4 Common Terminology

I have added the ones which always confuse me, will add more as they come to my mind.

- **substring:** A continuous piece cut from a string. For e.g. "abcd" is a substring of "aaabcdddeee" but "abcde" is not

- **subsequence:** A discontinuous piece of an array or string. In the above example "abcde" is a subsequence of "aaabcdddeee".

- **subarray:** It is the **array** version of **substring**.

- **segment:** It is another name for a subarray.

Chapter 5

Assignment 3 : Two Pointers

5.1 Level 1

- Maximum Ascending Sum
- Sereja and Dima

5.2 Level 2

5.2.1 Editorial for Zeroes problem

```cpp
class Solution {
public:
    void moveZeroes(vector<int>& nums) {

        int n = nums.size();

        int src = 0, dest = 0;

        // src is the pointer which will always point to
        // non zero numbers, whereas dest will point to
        // the first occurence of zero at any moment
        // we pick the non-zero number from src and swap it with dest

        while(1) {

            // increment dest until the numbers are non-zero
            while(dest < n && nums[dest] != 0)
                dest++;

            // this is common sense that src must always be greater than dest
            if(src < dest) src = dest + 1;

            // increment src until the it is zero
            while(src < n && nums[src] == 0) src++;

            // a check to ensure that src is within bounds
            // this is also the place where the code will break
            if(src >= n) break;

```

```
30              // this is the inbuilt swap function of c++
31              swap(nums[src], nums[dest]);
32
33              dest++, src++;
34
35          }
36
37          return;
38      }
39  };
```

You might feel that it is not O(n) but O(n²), so I say make a mini test case and simulate it using the above code on paper.

- Favorite Sequence

5.3 Level 3

- Long Pressed Name
- Max number of k sum pairs

5.4 Level 4 - (HARD)

- Trapping Rain Water This problem also has a stack solution.

It is recommended you spend enough time with the Level 3 and 4 problems, because they are going to stay for a long time and are a key to many problems and logics.

NOTE:

This is how your use the inbuilt sort function for array **a** of size **n**.

```
sort(a, a + n);
```

Sorting in reverse. Use:

```
sort(a, a + n, greater<int>());
```

or

```
sort(a, a + n); reverse(a, a + n);
```

A slight difference for sorting a vector **v**:

```
sort(v.begin(), v.end());
```

Chapter 6

std::map

6.1   What is dynamic programming?

"If you forget the past, you are doomed to repeat it."
But what are those problems called where you get the answer right without worrying about the future or the past? **answer: greedy problems**
Maps be it simple frequency arrays or std::map, help you save your past, so that you can always look back.

6.2   K Sum Pair Problem

Consider the Max number of k sum pairs problem where you had to count the number of pairs in the array with sum k. You followed a two pointer approach where you sorted the array and then took two pointers which were moved from both sides towards the middle, until they meet.
Time Complexity $= \mathbf{O(n * log2(n))}$ for sorting and $\mathbf{O(n)}$ for the two pointer iteration which makes it $\mathbf{O(n * log2(n) + n)} \approx \mathbf{O(n\ log2(n))}$ simply written as $\mathbf{O(n\ lg\ n)}$.

Now what would have been the brute force approach?

```
// the array is not sorted
// Time O(n^2)


int number_of_pairs = 0;


int ans = 0;
for(int i = 1; i < n; i++) {


        if(a[j] > k) continue;


        // for each element see how many elements
        // before it if added to itself sum up to K
        for(int j = 0; j < i; j++) {
                if(a[i] + a[j] == k)
                        ans++;
        }
}
cout << ans << '\n';
```

Now what if, as we move ahead in the array, we keep saving the frequencies for each element a[i], in a kind of **frequency array**. A frequency array is an array of size $\geq \mathbf{max(a[i])}$ and is filled with zeroes initially.

```
1   // max a[i] is 10^6
2   freq[(int)1e6] = {0};
3
4   // you could also do for older C++ versions
5   // memset(freq, 0, sizeof(freq));
6
7   int ans = 0;
8   for(int i = 0; i < n; i++) {
9           if(a[i] > k) continue;
10
11          // understand the next two statements properly!!!!!!!
12
13          // for every complimentary element k - a[i] occurring before
14          // we have a pair with a[i] hence,
15          ans += freq[k - a[i]];
16
17          // now save the current element's frequency
18          // We do it in O(1)
19          freq[a[i]]++;
20  }
```

The time complexity of the above code is **O(min(n, max a[i] value)**.
Now we know that 10^7 iterations usually run in a second or two, but what if a[i] is very very large of the order 10^{18}? This is where the **std::map** comes to rescue.
You will learn about Red-Black Trees in the DSA course (no you won't!). But for now, imagine they are simple frequency arrays. The time complexity to saving a new element comes down to **O(lg n)** but you now get to save huge elements.

```
1   // <key, value> : a key can be an object with a comparator only, and for value we
2   // can have anything, a counter, or an address to some array anything!!!
3   map<int, int> mp;
4
5   mp.clear(); // this statement clears the map, use the same map for each test case
6   // and clear like this right at the beginning of solve()
7
8   int ans = 0;
9   for(int i = 0; i < n; i++) {
10          if(a[i] > k) continue;
11
12          // compare this code to the one above
13
14          auto temp = mp.find(k - a[i]); // returns a map iterator to the pair <k - a[i], counter>
15          // actually it must be
16          // std::map<int, int>::iterator temp = mp.find(k - a[i]);
17          // but C++ compilers are smart and auto will write that shit for you post-compilation
18
19          // is temp a key in the map or not?
20          // or does find give us the end of the map?
21          if(temp != mp.end()) {
22                  ans += *temp.second; // pair<int, int> is the return type,
```

```
23              // use first and second to access values
24              // also written as
25              // ans += temp->second;
26          }
27
28          // now save the current element's frequency
29          // We do it in O(log n)
30          mp[a[i]]++; // Like an array, map too provides
31          // the operator[](), but here it works in O(log n)
32
33          // IMP : If you try to access an element not present in the map
34          // using int value = mp[x].second; then you will get 0
35          // the downside is it will add <x, 0> to the map too!
36          // that is why you must find x first using mp.find() and then
37          // change values, but if you are confident that an element x is
38          // definitely present, then go ahead, nothing is stopping you
39          // you can also use mp.count() > 0, read more on the internet
40  }
```

6.3   Read the following before attempting assignment 4

- Prefix sum arrays

- Finding range sum using prefix arrays

- Unordered maps, and why we must avoid them sometimes

- Side Mission : Learn the Kandane's Algorithm from GeeksForGeeks

Also, it is not always necessary to use std::map, for small values frequency arrays also work.

Chapter 7

Assignment 4 on std::map

7.0.1 Easy

7.0.2 Medium

7.0.3 Hard