SoC Video

Arnav Baranwal
Roll Number: 24B1099

July 23, 2025

Introduction

Myself Arnav Baranwal, and I am now going to present my SoC. The topic of my SoC was "Mastering Competitive Programming". Its goal was to help the mentees improve their competitive programming skills by strengthening core concepts, learning new algorithms and techniques, practicing questions, and giving contests frequently and discussing about them.
Now I will week wise discuss about what I learnt :

Week 1

In this starting week, we were asked to read first 4 chapters of CP handbook, and given questions to solve. This week covered the fundamental concepts of cp, which are essential to solve any question.
I learnt about time complexity, sorting, binary search and data structures like sets, maps, vectors, strings with their commonly used functions and methods.
We were also asked to make template, which is shown below :

```
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
#define fori(n) for(int i = 0; i < n; ++i)
#define forj(n) for(int j = 0; j < n; ++j)
#define all(v) v.begin(),v.end()
ll mod=1e9+7;

int main(){
```

```
    ios::sync_with_stdio(false);
    cin.tie(NULL);
    cout.tie(NULL);
    int t;
    cin>>t;
    while(t--){

    }
}
```

A specific improvement I saw in myself is that I got more comfortable with binary searching, which is often essential for efficiency of code.

Week 2

The 2nd week focused on Greedy Algorithms and Bit manipulation. I read about them in cp handbook and solved questions given by mentor.
Learnt about many different greedy strategies and improved upon the intution as to when can it be used through solving questions.
I knew bit manipulation theoretically but never applied it in coding. By reading cph and cf blogs, and solving questions, I learnt how to use it in code to perform certain operations efficiently, represent subsets, etc. Using bit representation of numbers sometimes turn out to be the optimal way to solving certain questions.

Week 3

Week 3 covered the crucial mathematical concepts needed in competitive programming like number theory, combinatorics, probability.

- Read CP handbook for number theory, combinatorics and probability. Following is a template to calculate combinatoric coefficients modulo a given number :

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;

const int MAX = 1e6 + 5;
const ll MOD = 1e9 + 7;
```

```
ll fact[MAX], invFact[MAX];

ll binpow(ll a, ll b) {
    ll res = 1;
    a %= MOD;
    while (b > 0) {
        if (b & 1) res = res * a % MOD;
        a = a * a % MOD;
        b >>= 1;
    }
    return res;
}

void func() {
    fact[0] = 1;
    for (int i = 1; i < MAX; i++)
        fact[i] = fact[i - 1] * i % MOD;

    invFact[MAX - 1] = binpow(fact[MAX - 1], MOD - 2);
    for (int i = MAX - 2; i >= 0; i--)
        invFact[i] = invFact[i + 1] * (i + 1) % MOD;
}

ll binom(int a, int b) {
    if (b < 0 || b > a) return 0;
    return fact[a] * invFact[b] % MOD * invFact[a - b] % MOD;
}
```

- Read about modular arithmetic and solved questions on it, from US-ACO guide.
- Learnt about Binary exponentiation from cp algorithms.

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;

const ll MOD = 1e9 + 7;
```

```
ll pow(ll base, ll exp, ll mod) {
    ll res = 1;
    base %= mod;
    while (exp > 0) {
        if (exp % 2) res = res * base % mod;
        base = base * base % mod;
        exp /= 2;
    }
    return res;
}
```

- Learnt about Sieve of Eratosthenes from cp algorithms. Following is a function that uses it to count number of primes till a given number:

```
int countPrimes(int n) {
    int count=0;
    vector<bool>prime(n+1,true);
    for(int i=2;i<n;i++){
        if(prime[i]){
            count++;
            for(int j=2*i;j<n;j+=i){
                prime[j]=false;
            }
        }
    }
    return count;
}
```

- Learnt about mobius function from USACO guide.
  The Mobius function is a multiplicative function that comes in handy when dealing with inclusion-exclusion technique and divisors-related problems. It has values in $\{-1, 0, 1\}$ depending on number's factorization.

$$\mu(n) = \begin{cases} 1 & \text{if } n \text{ is 1,} \\ 0 & \text{if } n \text{ has a squared prime factor,} \\ (-1)^k & \text{if } n \text{ is a product of } k \text{ distinct prime factors.} \end{cases}$$

  Following is its template :

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

```
typedef long long ll;

const int MAX = 1e7 + 1;
vector<int> mobius(MAX, 1);
vector<bool> isPrime(MAX, true);

void compute_mobius() {
    for (int i = 2; i < MAX; ++i) {
        if (isPrime[i]) {
            for (int j = i; j < MAX; j += i) {
                mobius[j] *= -1;
                isPrime[j] = false;
            }
            for (ll j = 1LL * i * i; j < MAX; j += 1LL * i * i){
               mobius[j] = 0;
            }
        }
    }
}
```

- Learnt Chinese Remainder Theorem and Euler's Totient function from cp algorithms.

Week 4

Week 4 focused on Dynamic Programming. After reading the chapter on dynamic programming from cp handbook, I did questions given by mentor through which I learnt the different methods and conditions in which we can efficiently solve the problem through dp, how to plan states, transitions, base cases and lookout for memory related issues if the number of states get too high.
Also learnt about difference between recursive and iterative dp, which one to use to solve a question efficiently.
Also a useful point is that in a question, in case of multiple queries over same input data, we can just process for 1st query, store the states till then, and process the next query from these stored states if not already stored, which saves us from recomputing from base case very time. Thus we essentially just compute states till the largest query, whether offline or online, which saves a lot of time than processing from base case each time.

This is also called memoization.

Week 5

The last week introduced us to the world of graphs and trees. I started by reading chapters 11-19 of cph to get familiar with the concepts, how to store graphs as adjacency lists, graph traversals, bfs and dfs, and advanced algorithms on graphs and trees. Then I solved questions, mainly from CSES, on the different algorithms.

References

Here I enlist the references used during the SoC :

- Codeforces questions and some blogs
- CSES Problemset for questions
- LeetCode for questions
- USACO Guide for articles and few questions
- CP algorithms for articles on algorithms