CP.Begin()

Coding Club, IIT Guwahati

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

int main()
{
    vector<string> CP = {"Week 1", "Week 2", "Week 3"};
    CP.push_back("Contest");

    auto it = CP.begin();
    it++;

    cout << *it ;
    return 0;
}
```

>>> Week 2 <<<

CONTENTS

Note:

- The articles mentioned contain theoretical concepts and should be read by everyone. The codes might be given in C++, but we can correspondingly develop the python code from the theory given.
- The articles from USACO Guide can also be accessed in both python and C++ by changing the language using an option on the webpage

You can checkout CPModule-IITG/CP.Begin() which contains the code of some useful functions to be covered ahead for both python and cpp users .

SORTING

(Expected Time: 1-2 days)

Sorting means arranging elements (in a data structure) in a particular order.
There are several sorting algorithms. Some basic sorting algorithms are as following to give you an idea of sorting techniques:

1. **Insertion Sort**
   a. **Article**
   b. **Video**

2. **Merge Sort**
   a. **Article**
   b. **Video**

But, why so many algorithms!!? **Read here**

While doing CP problems, we don't write any sorting algorithm. C++ STL provides a function to sort elements in any order in **O(N log N)** time.

SYNTAX - C++

Array : *sort(arr, arr+arr.size());*
Vector : *sort(vect.begin(), vect.end());*

Formally, *sort(it1, it2)* will sort elements in ascending order in the range [it1, it2) . Please note that this is even true for array {arr is a pointer} .

Now assume if we have to sort a vector of pair based on the second element, we will have to use **custom sort.** We can write our own comparator function based on how we wish to sort the elements .
Read **Custom sort** (Ignore *qsort*)

Further Read (Optional)

- **Sorting a vector of pairs - 1**
- **Sorting a vector of pairs - 2**

SYNTAX - Python

List : *list.sort(reverse = True/False)*
Dictionary/Set : *sorted(iterable,reverse=True/False)*

Sorting a list of pairs can be done by the normal sort function itself. Let's say a list, a= [(4,5),(2,3),(2,1),(4,4)] then simply calling ***a.sort()*** will give a = [(2,1),(2,3),(4,4),(4,5)] (sorting based on the first element of each tuple). You can learn more about **custom sort** using lambda function.

Practice Problems

- **Smallest Pair**
- **Bear and Extra Number**
- **Incinerate**
- **Swiss System Tournament**
- **Stick Lengths**
- **Divisibility by 2^n**

Additional Questions

- **Nested Ranges Check**
- **Blue Red Permutation**

GREEDY ALGORITHM *(Expected Time: 0-1 days)*

A greedy algorithm, as the name suggests, always makes the choice that seems to be the best at that moment. This means that it makes a locally-optimal choice in the hope that this choice will lead to a globally-optimal solution.

Resources

- **Watch this lecture on Algomaniacs**

Practice Problems

- **Different Differences**
- **Kathmandu**
- **Berland Music**
- **Closing the Gap**
- **Paprika and Permutation**

Additional Questions

- **Problems by Algomaniacs**

NUMBER THEORY

(Expected Time: 2-3 days)

Another important part of CP is number theory, and recently questions based on maths and number theory have become quite frequent. A basic grasp on these topics will be very useful.

CALCULATING GCD

- GCD of two numbers can be calculated using the Euclidean Algorithm, i.e., **gcd(a,b)=gcd(a, a-b)**.
- This can be further optimized as we can reduce repeated subtractions. Hence, **gcd(a,b)=gcd(a, a%b)**.
- Its time complexity is **O(logn)**.
- You can also use the inbuilt functions

  C++ : ***__gcd(a,b)*** (on some compiler __gcd(0, 0) gives exception so while using __gcd we must carefully handle __gcd(0,0) case)

  Python(use import math) : ***math.gcd(a,b)***

Resources

- CPH, page **200-201**
- Euclidean algorithm (Ignore Binary GCD)
- *Extended Euclidean Algorithm

PRIME NUMBERS AND THEIR TESTS

- To check whether a number is prime or not, we need not check its divisibility by all numbers from 1 to n.
- We can just check for numbers **<= sqrt(n)** because if there exists a factor greater then sqrt(n), there will also exist a factor less than sqrt(n).
- It's time complexity is **O(sqrt(n))**

**Optional topics ! Study only if you have time .*

Resources

- CPH, page **199**
- **Primality test: algorithm**
- ***Extended Primality Tests**

PRE PROCESSING PRIMES TILL INTEGER

- When we need to check for primality of multiple numbers in the same code, it can be better to preprocess and store whether a number is prime or not.
- This can be done in **O(n(log(log(n)))** time for preprocessing of **Sieve of Eratosthenes**. Then we can access whether a number is prime or not later in O(1) time. {Might be useful if you have queries of numbers .
- The problem with this is that it may **not** be possible for large n (>=10^7)
- CPH, page 200
- **Sieve of Eratosthenes** (Ignore Segmented Sieve)

INTEGER FACTORIZATION

- **CPH, page 197-199** (Give it a quick read, it mostly contains mathematics that you must have studied before like number of factors of a number etc)
- Prime Factorization of a number can be done using trial division method in **O(sqrt(n))** time.
- However, again for multiple queries, it may not be optimal. So we can precompute the smallest prime factor **(spf)** of every number from 1 to n. This takes **O(nloglogn)** time.
- Suppose we have a number x, one of its prime factors will be spf(x). And rest of the prime factors will be contained in x/spf(x). We can do this recursively to find all prime factors in O(logn) time for each query.
- **Precomputing SPF**
- ***Integer factorization**

**Optional topics ! Study only if you have time .*

MODULAR ARITHMETIC

An essential part of CP is working with remainders of integers with a particular number, instead of working with integers themselves. This is done to prevent integer overflows in built-in data types.

A complete introduction to modular arithmetic along with sufficient practice problems can be found here :

- CPH, page 201-203
- **Modular Arithmetic**
- **Binary Exponentiation** is a trick using which we can calculate **a^n** in **O(logn)** time instead of **O(n)** time required by the naïve approach .
- **Modular Inverse**

Note: In C++, -8%7=-1 and not 6. You have to add 7 at the end and take mod again. In general, n mod m= (n%m+m)%m.

BONUS TIP FOR PYTHON USERS

The python inbuilt function "**pow(a,b,mod)**" calculates (a^b)%mod in **O(logb)** complexity itself. So python users can directly use this function instead of writing the code for binary exponentiation, **however, it is recommended that you know how to do the binary exponentiation manually.**

*ADDITIONAL READ

- **Essentials of Number Theory** this will help in developing an intuitive feeling for elementary number theory .

**Optional topics ! Study only if you have time .*

Practice Problems

- Exponentiation
- Counting Divisors
- Kill Demodogs
- Divisor Analysis
- All are Same
- Exponentiation II
- Divisible Numbers

Additional Questions

- Strange Function
- Integers Have Friends
- Half of Same
- Plus and Multiply
- X-Magic Pair
- Taxes

TWO POINTERS

(Expected Time: 1-2 days)

The name “Two Pointer” suggests the use of two different pointers, and it is exactly that, but without actually using C++ pointers (Well, you can use, but why would you want to?). It’s just a fancy name for a technique which uses two variables in a single loop. The variables are generally used to keep track of indices of an array or string, and generally in a sorted array.

This technique is all about iterating two monotonic pointers across an array to search for a pair of indices satisfying a given condition in linear time. This technique comes handy in other problem types like **sliding-window** and sub-array problems.

The best place to learn this technique is from **Codeforces EDU** **Two-pointers-CF**

Some other resources

- **USACO guide**
- **CPH**

Practice Problems

- **Diamond Collector**
- **Sum of Two Values**
- **Sum of Three Values**
- **Maximum Subarray Sum**
- **Balanced Team**

Codeforces

- **Codeforces edu** Two Pointers - Step 1

Additional Questions

- **Codeforces edu** Two Pointers - Step 2

COMBINATORICS

(Expected Time: 0-1 days)

Remember your JEE days? Well guess what... the stuff you have learnt for JEE doesn't go for waste, it has its own importance in CP too!

CALCULATING ${}^{n}C_{r}$

Naïve method

You must have learnt the formula ${}^{n}C_{r} = {}^{n-1}C_{r} + {}^{n-1}C_{r-1}$. This essentially means that you could recursively calculate the value of ${}^{n}C_{r}$ through the previously calculated ${}^{n-1}C_{r}$ and ${}^{n-1}C_{r-1}$. The code for this part can be accessed in the **USACO** guide. However, the complexity of this naïve approach is **O(2^n)** and this can be **optimized** to **O(n^2)** using a technique called **dynamic programing** that we will go through in detail in the next week.

Optimal method

You must have also learnt an alternate formula for nCr which is **n!/(r!*(n-r)!)**. This can be used to find out the binomial coefficients too.

Also we know that factorials tend to be pretty large which can overflow the integer size (and in python even though you can store integers of very long size, it takes up a lot of time giving TLE), so we generally calculate it modulo some number (which is generally prime). This can be done in **O(n+log(mod))** complexity by precomputing the factorials and inverse factorials individually and then using them in the problem as required. Code is available at **USACO**.

Practice Problems

- **Binomial Coefficients**
- **Creating Strings II**
- **Binary Strings are Fun**
- **Close Tuples**

Additional Questions

- **USACO Problemset**

BIT OPERATIONS

(Expected Time: 0-1 days)

In Competitive Programming, writing numbers in their binary form might help you. It's really necessary to know different properties of binary representation of numbers and how to perform binary operations.

While we see integers in decimal system, they are still stored in binary form. We can apply operations like and, or, xor etc to each bit of a number and use it for our benefit.

For example, you might know that a set of n elements (array of n elements for our purpose) has 2^n subsets. All of these can be easily accessed using binary representation as 0 or 1 at a bit position can indicate the presence or absence of that element in the subset.

- **CPH (pg 96 to 99):** This covers the essentials of how bit operations work and how they can be useful. This is a **must read**.
- It is fine if you find the concepts difficult to grasp, the following video can be helpful- **Bitwise Operations tutorial #1 | XOR, Shift, Subsets**
- **Bitwise Operators in Python** (Uses the same idea as the above, just that the code will be in py).

Practice Problems

- **Petr and a Combination Lock**
- **Absolute Maximization**
- **NIT orz!**

Additional Questions

- **Even-Odd XOR**
- **Orray**
- **Odd Topic**