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();

    string out = *it;
    out[5] = '3';
    cout << out ;

    return 0;
}
```

>>> Week 3 <<<

CONTENTS

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 .

INTERACTIVE PROBLEMS

(Expected Time: 0-1 days)

Interactive Problems are those problems in which our solution or code interacts with the judge in real time. When we develop a solution for an Interactive Problem , the input data given to our solution may not be predetermined but is built for that problem specifically. The solution performs a series of exchange of data with the judge and at the end of the conversation the judge decides whether our solution was correct or not. You can refer to **this blog** to get a better idea.
You have to use special flush operation each time you output some data.

C++

```
cout << flush ;
```

PYTHON

```
import sys
sys.stdout.flush()
```

A sample solution for the problem **Lost Numbers** has been uploaded on our **github** .

Practice Problems

- **GCD Queries**
- **Interactive Factorial Guessing**
- **AquaMoon and Stolen String**
- **Guess the Array**

BINARY SEARCH

(Expected Time: 1-2 days)

This is one of the most useful and important algorithms you will ever come across. It is used to solve problems which have **some monotonic features.**

The main idea of the algorithm is to **divide the problem into half at each stage**.
Let's take an example of a classical problem. Suppose you are given an array of elements whose elements are sorted in increasing order and you are asked the smallest number greater than k which is present in the array.

Now take some time and think of some strategies.

One possible strategy can be:
Iterate over the array from 1 to N, once you get to a position whose value is greater than k then this is the answer.

Optimal Strategy: **Binary search**

You can divide the array into two halves **1 to N/2** and **N/2 to N**. Now let's check in which half is our answer present. If it's in the left half then a[N/2] has to be greater than k. If not, then it has to be in the right half.

Thus, just by checking the value of **a[N/2]** we can get rid of one half.

An important point to note is that after every query the length of the array in which we search becomes half.
So **N -> N/2 -> N/4 -> N/8 ..... 1** , and we only need to ask **log(N)** queries.

So, Binary search is only this much (just think about removing one half).

Here are some great resources which will teach you how to think about binary search and how to implement it:-

- **Binary Search tutorial (C++ and Python)**
- First 3 steps of **Codeforces Edu Binary Search**

STL STUFF RELATED TO BINARY SEARCH

In almost all Containers in C++ you can use inbuilt binary search which works in **O(log(n))**.

These two you will use most frequently.

- **Vectors** :- You need to first sort the vector for these to work correctly.
  - **Binary Search**
  - **Lower Bound**
  - **Upper Bound**
- **Sets / Multisets** :-
  - **Find** (works like binary search)
  - **Lower Bound**
  - **Upper Bound**

Note that both have different syntax. If you use the syntax for vectors in Set then its time complexity won't be **O(log(n))**, it will be **O(n)**, and will probably give TLE. You have to be very careful about the syntax.

PYTHON

These two you will use most frequently.

- **Lists** :- You need to first sort the list for these to work correctly.
  Bisect module:
  The bisect module (as discussed in week 1), is an inbuilt library that allows you to apply binary search over a list in **O(log(n))** time.
  These modules are a little uncommon among python users so the best place to learn them is from their **official documentation**

- **Sets** :- Unfortunately, Sets in python don't work the same way as sets in cpp. So it becomes really complex to apply binary search over sets in python. It can be done by using a separate library called **Itertools** which has an attribute '**iter'** that can be used to iterate over a set. However, this method is extremely tedious. The best method would be to make your own class object "set" using Red black trees which is kind of complex at this stage. Read **this** for more information

- **Multisets** :- Like cpp, python has **multisets** as well and just like python sets, it isn't possible to apply binary search over python multisets. So the best method is to create your own Class object "Multiset" using **Red black trees** as suggested above.

Practice Problems

- **Maximum Median**
- **Codeforces EDU (Step 1,2,3)**
- **Guessing the Median**
- **Alyona and a Narrow Fridge**

Additional Questions

- **Fixed Point Guessing**
- **Sagheer and Nubian Market**
- **K-th Not Divisible by n**
- **Valhalla Siege**
- **The Doctor Meets Vader**
- **Concert Tickets (Imp)**
- **Cellular Network**
- **Sum of Three Values**
- **Array Division (Tough)**
- **Elemental Decompress**

In week 2 you learnt Binary exponentiation. You can relate it to binary search also. First try to write a code yourself to calculate a to the power of b. If you can't come up with a solution, you may refer to this video **Binary Exponentiation**. I advise you to devote, at the very least, half an hour before watching the video.

INTRO TO DP

(Expected Time: 2-3 days)

No doubt this is the widest topic in Competitive Programming, but it's not something which cannot be taught.

Few resources:-

- **CPH** (pg 65 to 69) (If you don't understand in one read, read it again)
- Errichto (**Lecture 1** and **Lecture 2**)

If you didn't understand even after referring to the resources, don't worry it's normal, just jump to solving problems.

PYTHON

Dp in python is the same as that in cpp. Here you use lists to memorize instead of arrays in cpp. You can refer **here** to learn how to use python for dp.

Note :-

There is one problem with python when it comes to dp though. Python functions take up more time and space when compared to C++ counterparts . This means that, if you happen to submit a recursive code in python, it's highly probable that you will encounter a **runtime error**(RE) or **Time limit exceeded error**. (TLE)

If you are getting a RE, it's probably because of two reasons :-

Maximum Recursion Depth exceeded (MRD)

Problem: In the backend, all recursion calls are stored in a stack. When the number of recursion calls exceeds the stack size, you get this error.

Solution: Import the sys module and use the **set recursion limit** attribute to increase the recursion limit to a value that doesn't give RE (a matter of trial and error). A general value that works in many problems is 3* 10^5.

Memory Limit Exceeded (MLE)

Problem : Every problem has a particular memory limit that you shouldn't exceed. But when your stack size increases, it uses more memory giving out MLE.

Solution : Like the soln above, you could reduce the stack size using the set recursion limit attribute but then again it would give you a MRD. So the best thing would be to avoid recursive calls at all. This can be done by writing the iterative version of the recursive logic you used.

Things to keep in mind:

So always remember that, if you are using python to solve a dp problem, it is highly advised to submit an iterative logic rather than a recursive logic. If the iterative logic gives TLE even if your complexity is well within the Intended complexity of the problem, switch to cpp for solving problems of dynamic programming.

This might be a little bit demotivating for python users but python is not very good with programs involving function calls. So you could switch your language to cpp just in programs that use functions (mainly **dp and dfs/bfs**,etc). This doesn't make python a bad language either, it just has its own pros and cons...(like python has no integer overflows and has simpler syntaxes than cpp and is very user-friendly!)

Practice Problems

- **Dice Combinations**
- **Minimizing Coins**
- **Frog 1**
- **Frog 2**
- **Vacation**
- **Knapsack 1** (Knapsack Problem, Very Important, try it first if you have no clue how to solve it then refer **this** or **this**)

Additional Questions

- **Coin Combinations I**
- **Coin Combinations II**
- **Array Description**
- **Road Optimization**
- **New Year and the Permutation Concatenation**

*In future if you want to study more things in DP then you can refer **Atcoder DP Contest**(It covers almost all DP techniques) and for reading more **USACO Guide**

Note for **Concert Tickets** in Binary Search Section :-

- If you haven't tried it yet, stop reading from this point onwards.
- If you are still reading, you should have devoted, at the very least, half an hour on the problem. If not, try to solve it sincerely first.
- You will not be able to solve the problem with Binary Search. You will have to use the set data structure for solving it. You should still try solving it using Binary search though, since it is important to learn about its limitations.

GRAPH THEORY

(Expected Time: 1-2 days)

Graphs are a way to formally represent a network, which is basically just a collection of objects that are all interconnected

GRAPH REPRESENTATION

The following two are the most commonly used representations of a graph.

1. **Adjacency Matrix** : Adjacency Matrix is a 2D array of size V x V where V is the number of vertices in a graph. Let the 2D array be adj[][], a slot adj[i][j] = 1 indicates that there is an edge from vertex i to vertex j. Adjacency matrix for undirected graph is always symmetric.
2. **Adjacency List** : An array of lists is used. The size of the array is equal to the number of vertices. Let the array be an array[]. For every index i, we store the no of vertices adjacent to vertex-i in array[i]. This method is more preferrable .

You can read more about this **here**.

GRAPH TRAVERSALS

The two most important graph algorithms are **depth-first search** and **breadth-first search**. These are graph traversal algorithms which begin at a starting node and visit all the nodes that can be reached from the starting node. The only difference between depth-first search and breadth-first search is in the order in which they visit the nodes.

Resources to get a better idea of these algorithms :

- **CPH** **(Chapter 12)**
- **Depth-first search**
- **Breadth-first search**
- **CP for Graphs** by **Utkarsh Gupta**

Practice Problems

- Labyrinth
- Building Roads
- Building Teams
- Railway System
- Game Master
- Round Trip

SHORTEST PATH ALGORITHMS

DJIKSTRA'S ALGORITHM

Dijkstra's Algorithm is an algorithm that is used for finding the shortest distance, or path, from starting node to target node in a weighted graph.

Dijkstra's algorithm makes use of weights of the edges for finding the path that minimizes the total distance (weight) among the source node and all other nodes. This algorithm is also known as the single-source shortest path algorithm.

These resources will help you to get a better idea on this algorithm : -

- CPH (pg 126-129)
- Dijkstra's algorithm
- Dijkstra Algorithm - Single Source Shortest Path

Practice Problems

- Moving Both Hands
- Shortest Routes I
- Flight Discount

Additional Questions

- Jzzhu and Cities
- High Score

ADDITIONAL READ - DFS AND BFS TREES

DFS/BFS Trees can be generated by marking the edges followed during a dfs/bfs traversal. This can prove to be a useful technique while solving some tough graph questions.
You can read the DFS Tree section of this **blog** to know more.

Try this question **Two Spanning Trees** . The **editorial** to the given problem is a good intro to dfs/bfs trees and how you can use it in CP.