L3

So after introduction to binary search. What we need is practice of questions related to arrays and basic recursion, which can help us solve almost any problem (though not in time limit), but it will still solve the problem.

There are some other topics as well that should be discussed along side, i.e.

1. Kadane's Algorithm

Kadane's Algorithm

Question: Given an array, and we have to find the maximum sum of a continous subsegment (or subarray)

Explanation to the question:

Consider an array $10, 22, -20, 24$
Now there are following continous subsegment possible

1. 10
2. $10 + 22 = 32$
3. $10 + 22 - 20 = 12$
4. $10 + 22 - 20 + 24 = 36$
5. 22
6. $22 - 20 = 2$
7. $22 - 20 + 24 = 26$
8. -20
9. $-20 + 24 = 4$
10. 24

    Out of the these continous subsegment for index $(0, 3)$ give the maximum sum;
    But note that it note always the whole array always

Consider this array: $10, -20, 15, -5, 16, -44$
Here maximum sum is 26 from subArray $\{15, -5, 16\}$

Now its naive algorithm take $O(n^2)$ time but with Kadane we can do it in $O(n)$

Algorithm:

1. Set Sum = 0, sum_temp = 0
2. Iterate over array
3. Add element to the sum_temp

4. if sum_temp < 0
   4.1. Set sum_temp = 0
5. if sum_temp > Sum
   5.1 set Sum = sum_temp
6. END

C++ Implemenation:

```
int maximumSum(vector<int> arr){
    int sum = 0;
    int sum_temp = 0;
    for(int i = 0; i < arr.size(); ++i){
        sum_temp = 0;
        sum_temp = max(0,sum_temp);
        sum = max(sum_temp,sum);
    }
    return sum;
}
```

Recursion

Now question arises, why recursive and not iterative?
Because recursion is easier

Now can we see recursion?
Let's say I ask you to find factorial of x, but you are too lazy for whole calculation, so you ask someone else for $(x-1)!$ and that person does so for $(x-2)!$ and so on

So, When that person tells you $(x-1)!$ you just multiply it by x to get $x!$

That is recursion in **bottom-up** methodology

Like we had 1 important requesite for binary search, we also have a very important requesite for Recursion
A problem should be divisible into simpler subproblems

Now, lets see places where we can use recursion as brute force.

Note that these solution won't be accepted in an online judge because they might be too slow, but our aim right now is to understand recursion

1. **Generate all subsets of an array**
   Now what is the subproblem here?
   If we consider array $1, 2, 3$
   then let's say if find subset of $2, 3$ and then add 1 to them, I will get an answer
   Let me explain
   Subset of $\{3\} = \{\Phi, \{3\}\}$

Subset of $\{2,3\} = \{\Phi,\{3\}\} \times \{\Phi,2\}$ = $\{\Phi,\{3\},\{2\},\{2,3\}\}$
Subset of $\{1,2,3\}$ =Subset $\{2,3\}\times\{\Phi,1\}$ =
$\{\Phi,\{1\},\{2\},\{3\},\{1,2\},\{1,3\},\{2,3\},\{2,3,4\}\}$
In other words, either an element can be in a subset of it can't be

```
void generateSubset(int i, vector<int> arr, vector<int> sub){
    if(i == arr.size()){
        if(sub.size() == 0)cout << "PHI\n";
        else{
            for(auto element : sub)cout << element << " ";
            cout << "\n";
        }
        return;
    }
    generateSubset(i+1,arr,sub);
    sub.push_back(arr[i];)
    generateSubset(i+1,arr,sub);
}
```

2. **Fibonacci Series**

   Fibonacci can be expressed as

$$F(n) = F(n-1) + F(n-2), F(1) = 1, F(0) = 0$$

   So we can easily divide it into subproblems which asks to solve $F(n-1)$ and $F(n-2)$

```
int fibonacci(int n){
    if(n == 1)return 1;
    if(n == 0)return 0;
    return fibonacci(n-1) + fibonacci(n-2);
}
```

3. **Longest Increasing Subsequence**

   Main Question: Given an array, find the length of longest subsequence such that $a_i \leq a_j$ for any $i \leq j$

   Example:

   In the array $1, 10, 3, 4, 15, 16, 5, 6, 7$

   Here, for indices $0, 2, 3, 6, 7, 8$ constitute an increasing subsequence thus answer is 6

   Note: Even indices $0, 1, 4, 5$ was creating an increasing sub-sequence but its size if 4 thus, answer is 6

   This is a very important classical algorithm question. Using recursion it can be solved in $O(2^n)$ complexity

   Using iterative method it can be solved in $O(n^3)$ complexity

   Using dynamic programming it can be solved in $O(n^2)$ complexity*

and using binary search, we further reduce it $O(nlogn)$ time complexity.
But right now we are only focusing on using recursion

So let's divide it into subproblems:

1. We can either include a number or not
2. So if we include a number, it become the maximum of the current series and this it should be strictly greater than previous number. If it is not possible then, it cannot be included.

```
int length_ans = 0;
void longestIncreasingSubsequence(vector<int> arr, int index, int maximum, int leng
    if(index == arr.size()){
        length_ans = max(length,length_ans);
        return;
    }
    if(arr[index] > maximum){
        longestIncreasingSubsequence(arr,index+1,arr[i], length+1);
    }
    int l2 = longestIncreasingSubsequence(arr,index+1,maximum, length);
}
```

4. Majority of Dynamic Programming Questions can be solved in brute force via recursion, LIS was just an example to show. Though recursion may not be the best technique to solve it even in brute force but right now we don't care we just are learning recursion.

Stacks and Queues

These are important Data Structures. Infact we will be using them a lot in the future

What is a stack?

Stack is a data structure in which the array entered last is the first one to be access.
For example, we have to simulate the process of notebooks being checked.
A notebook is taken from top and checked, meanwhile other students keep piling their notebooks above the pile of their notesbooks. So next notebook to be checked is the one to be checked at top of file. This can easily be simulated with the help of stack

What are queues?

Queue is a data structure where, first entered data is accessed first. For example, in a movie queue people keep adding but the person who goes to counter next is the person who joined afer the current user only

How to Use stacks and queues in C++?

STL provides great resources for conventional data structures such as tree, red-black tree, hash-maps, heaps and many more. On top of that there are even many methods(functions) provided with them to make many tasks easier
To use stack in C++, we do the following

- **Declaring a stack or queue**

```
stack<data_type> stack_name;
```

example

```
stack<int> s;
```

- **Inserting into a stack or queue**

```
stack_name.push(variable_name);
```

example

```
int a = 10;
s.push(a);
```

- **Accessing the value at top of stack**

```
stack_name.top();
```

example

```
int a = s.top();
OR
cout << s.top();
```

- **Accessing the value at front of queue**

```
queue_name.front();
```

example

```
int a = s.front();
OR
cout << s.front();
```

- **Removing value from top or queue**

```
stack_name.pop();
```

example

```
s.pop()
```

- **Checking if a stack/queue is empty or not**

```
if(stack_name.empty())
```

example

```
if(s.empty())
```

- **Checking size of a stack or queue**

```
stack_name.size();
```

example

```
int sz = s.size();
```

Problems

1. Coding Skills(Kadane's Algorithm)
2. Kadane's Algorithm (geeksforgeeks)
3. Frustated Coders
4. Stack Operations
5. Stack and Queue(Prefix Sum Array)
6. Monk and Prisoner of Azkaban
7. Monk and Chamber of Secrets(Queues)
8. Equal Stacks
9. Queue Using two stacks
10. Rails (UVa 00514)