PrepBytes

50 Must Do Coding Questions

for Product Based Companies Solved

1. Maximum Sub-Array Sum

Problem Statement

PrepBuddy has an array of size N containing integers.
He wants to find the maximum sum sub-array among all possible sub-arrays and print its sum.

Example

Suppose arr=[20 0 19 2 -20 -3 4 -14] we select the subarray = 20 0 19 2 and its sum is 41.
Hence, we return 41.

Explanation

In the given test case 4, 2 from the first stack and 2, 1 from the second stack can be removed to achieve a sum of 9 which is less than 10. So, the total score of Ritika becomes 4.

Logic

The idea is to use two variables to store the current and global maximum of the subarrays, we keep track of the current subarray sum in the current sum and we take the maximum of the current sum and the global maximum. We then just return this global sum. We initialize the current sum and global sum to INT_MIN or the first element of the array. We start iterating from the second element of the array and store the current maximum and the global sum to store the maximum of the subarray
sums in the global sum as :
curr_sum = max(nums[i],curr_sum+nums[i])
global_sum = max(global_sum,curr_sum)

Code

```
#include <bits/stdc++.h>
#define ll long long
using namespace std;
ll kadanes(vector<ll> nums){
  ll n = nums.size();
  ll global_sum = INT_MIN, curr_sum = INT_MIN;
  for(ll i = 0;i<n;i++)
```

```
  {
    curr_sum =  max(nums[i],curr_sum+nums[i]);
    global_sum = max(global_sum,curr_sum);
  }
  return global_sum;
#include <bits/stdc++.h>
#define ll long long
using namespace std;
ll kadanes(vector<ll> nums){
ll n = nums.size();
  ll global_sum = INT_MIN, curr_sum = INT_MIN;
  for(ll i = 0;i<n;i++)
  {
    curr_sum =  max(nums[i],curr_sum+nums[i]);
    global_sum = max(global_sum,curr_sum);
  }
  return global_sum;
```

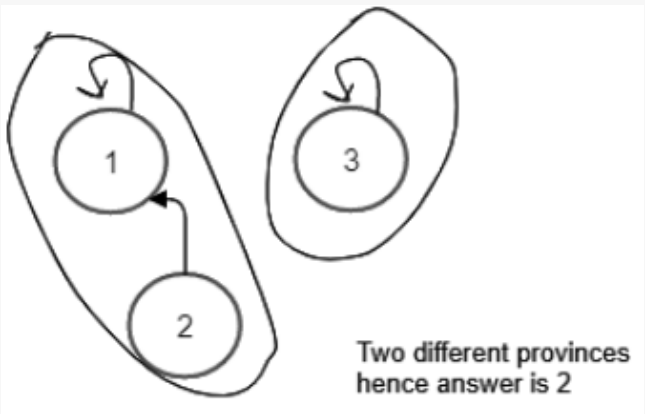
2. Count the Provinces

Problem Statement

There are N cities in the country of Prepland. Some of them are connected, while some are not. If city X is directly connected with city Y, and city Y is connected directly with city Z, then city X is indirectly connected with city Z.
A group of cities that are connected directly or indirectly with each other are known as Province. Your task is to count the total number of provinces in the Prepland.

Example

Let suppose we have 3 cities, denoted as 1,2, and 3. There is an edge from 1 to 2 and hence 1 and 2 are connected, this makes 2 provinces.



Logic

The idea is to take a parent array that stores the root of each of the cities. Initially, each city will be the parent of itself which is represented by -1. Now when we add an edge, we will see that if they are in different groups then unite them by making parent of one node parent of another also. So, if 1 is connected to 2 then the parent of 2 is changed to 1 and this goes on for every edge. In the end, we traverse the parent array and count the number of -1, which represents the number of separate groups. This will also be our final answer.

Code

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

void addEdge(vector<int> adj[], int u, int v)
{
    adj[u].push_back(v);
    adj[v].push_back(u);
}
void DFSRecur(vector<int> adj[], vector<bool> &visited,int u)
{

    visited[u]=true;
    for(int i=0;i<adj[u].size();i++)
    {
        if(visited[adj[u][i]]==false)
            DFSRecur(adj,visited,adj[u][i]);
    }
}
void DFS(vector<int> adj[], int n)
{
    vector<bool> visited(n,false);
    int count=0;
for(int i=0;i<n;i++)
    {
        if(visited[i]==false)
        {
            DFSRecur(adj,visited,i);
            count+=1;
        }
    }
    cout<<count<<"\n";

}
```

```
int main() {
    int t;
    cin>>t;
    while(t--) {
        int nodes, edges;
        cin >> nodes >> edges;

        vector<int> adj[nodes];
        for (int i = 0; i < edges; i++) {
            int u, v;
            cin >> u >> v;
            u--;
            v--;
            addEdge(adj, u, v);
        }
        DFS(adj, nodes);
    }
    return 0;
}
```

3. Minimum Number of Conference Rooms

Problem Statement

You are given the time intervals of n meetings. Each time interval consist of the start time x and end time Y. The task is to determine the minimum number of conference rooms required to hold all the meetings.

Example

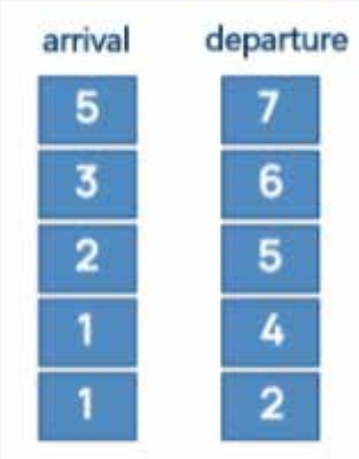
In this problem, we are given the start time and end times of the meetings and are asked to find the minimum number of rooms that will be required so that no two meetings are organized in the same room. Suppose we have our starting and ending time arrays as –

5 7
2 5
3 4
1 6
1 2
Then the minimum number of rooms that we will require is 3.
For the first test case:
Room 1: 5th meeting→ 2nd meeting
Room 2: 4th meeting
Room 3: 1st meeting
ut
2 4 5

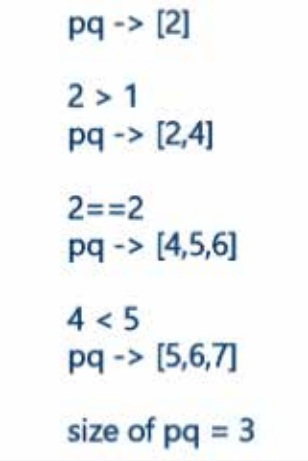
Output
1 10 11 100
1 10 11 100 101

Logic

The idea is we sort the arrival and departure times in a non-decreasing fashion and then check how many overlapping time intervals we are getting. The maximum of these numbers will give me the final answer.



The idea is if a meeting has already started then if any other meeting starts before the current meeting end time then those will overlap.



Code

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

int main()
{
  //write your code here
```

```
  int t;
  cin>>t;
  while(t--){
    int n;
    cin>>n;
    vector<pair<int,int> > vp;
    for(int i = 0;i<n;i++){
int x,y;
      cin>>x>>y;
      vp.push_back(make_pair(x,y));
    }

    priority_queue<int,vector<int>,greater<int>
> pq;
    sort(vp.begin(), vp.end());
    for(auto i : vp){
        pq.push(i.second);
int x,y;
      cin>>x>>y;
      vp.push_back(make_pair(x,y));
    }

    priority_queue<int,vector<int>,greater<int>
> pq;
    sort(vp.begin(), vp.end());
    for(auto i : vp){
        pq.push(i.second);
```

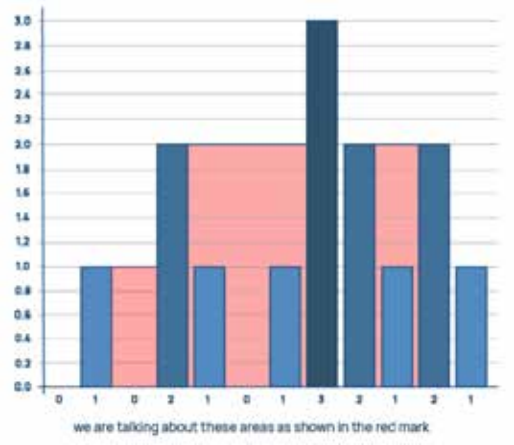
4. Trapping Rain Water

Problem Statement

You are given an array A of N non-negative integers, representing an elevation map where the width of each bar is 1. The task is to compute how much water it can trap after raining.

Example

For example, if the elevation map with heights arr = 0,1,0,2,1,0,1,3,2,1,2,1, the total water trapped will be 6 units. In the above example for bar 1, we have contribution[1] = min(3,2)- 1=1, hence the water column above 1 will contain 1 unit of water. Like this, if we calculate the water columns for each bar then the summation of all these contributions will give the maximum water trapped in the bars.



Logic

The most naïve way to calculate these left and right maximum heights is using another loop to search in the left and search in the right of the current node. The idea is to iterate to the left and right of the current element to find the left maximum and right maximum and then add the resultant contribution to our answer

Code

```
C++
#include <iostream>
using namespace std;

int calculateWater(int height[], int n)
{
      int result = 0;

      int left_max = height[0], right_max = height[n-1],
left = 1, right = n-2;

      while (left <= right) {
            if (left_max <= right_max) {
                  if (height[left] < left_max)
                        result += left_max - height[left];
                  else
                        left_max = height[left];
```

```
                left++;
            }
            else {
                if (height[right] < right_max)
                    result += right_max - height[right];
                else
                    right_max = height[right];
                right--;
            }
        }
        return result;
}
```

5. Longest Palindromic Substring

Problem Statement

You are given string str, find the longest palindromic substring in str.
Longest Palindromic Substring of string strstr: LPS[i...j] where 0<=i<=j<len(LPS)

Example

Palindrome string: LPS is a palindrome if reverse(LPS) = LPS.
If multiple LPS is the same then return the Longest Palindromic Substring which occurs first (with the least starting index)
Input : "bbbbbabbb"
Output : "bbbabbb"

Logic

The idea is quite simple and I call it the spreading hand technique. :) Well, what we know about a palindrome is, if we reverse the next half of the string then it will be equal to the first half for a palindrome.

We keep a variable longest which stores the current longest palindrome seen.
We start iterating from 1 till the end –

1) For each character in this iteration, we have two options either search for a palindrome taking the ith character as second middle or only middle.

2) This means, when a palindrome with even length like “baab”, we have two middles as a whereas for palindromes with odd length like “bab”, we do not consider the middle a for the mismatch.

3) We have two options to check as given above. For both the options, take the middle character and start expanding the pointers towards the opposite direction till we reach the end or different characters.

4) If both the characters keep on matching then we can say that there is a palindrome in between these pointers and we keep track of this length and start position.

1) if high - low + 1 > longest then start=low and longest = high - low +1

Code

```
C++
#include <iostream>
using namespace std;
int main() {
    int t;
    cin>>t;
    while(t--){
        string st;
        cin>>st;
        int longest = 1;
        int start=0;
        int len = st.length();
        int low,high;
        for(int i=1;i<len;i++){
            low = i-1;
            high = i;
            while(low>=0 && high<len && st[low]==st[high]){
                if(high - low + 1 > longest){
                    start=low;
                    longest = high - low +1;
                }
                low--;
                high++;
            }
            low = i-1;
high = i+1;
            while(low>=0 && high<len && st[low] == st[high]){
                if(high - low +1 > longest){
                    start = low;
                    longest =  high - low +1;
```

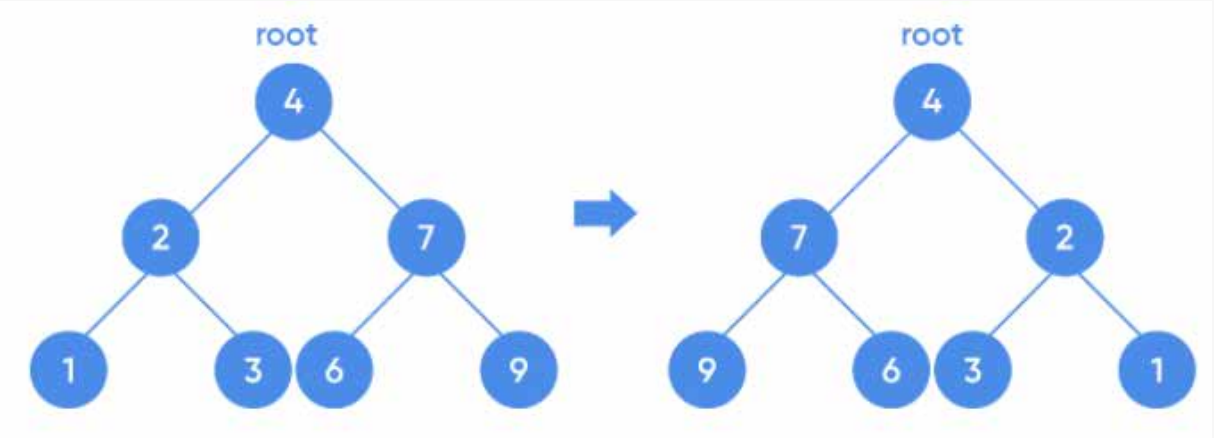
```
                }
                low --;
                high++;
            }
        }
        for(int i=start; i<=start+longest-1;i++)
            cout<<st[i]<<"";
        cout<<"\n";

    }
    return 0;
```

6. Invert the Binary Tree

Problem Statement

You have been given the root of a binary tree. Your task is to invert this binary tree. In other words, you should swap every left node in the tree for its corresponding right node.

Example



Logic

The idea of recursion is standing at a current node, we will assume that our recursive function has already brought us the inverted left subtree and right subtree. We just have to swap the left and the right subtree and return this node.

- The swap of the left and right subtree root can be easily done with the swap function.
- Use the pre-order traversal technique.
- Swap the left and right subtree nodes.
- Recursively do this for root→left and root→right.

Code

```
C++
void invertBinartTree(node *t)
{
    if(t == nullptr)
        return;
    swap(t->left,t->right);
    invertBinartTree(t->left);
    invertBinartTree(t->right);

}
node* invertBinaryTree(node *t)
{
    invertBinartTree(t);
    return t;
}
```

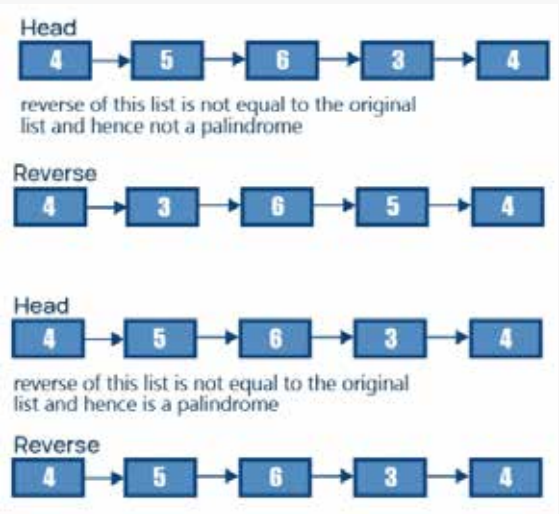
Note: Only Function is implemented

7. Palindrome List

Problem Statement

You must have already solved a few problems on palindromes by now. You must have checked whether a number is a palindrome or not or a string is a palindrome or not. Now it's time to check whether the given linked list is a palindrome or not.

Example



Logic

The idea is to reverse half of the list. Intuitively the first half of the list should be a mirror image of the second half or reverse. Hence, if we reverse only the second half of the list and then check whether the two lists are the same or not then also, we will get the same result. The reverse can be done in $O(n/2)$ time after finding the middle of the linked list. In finding the middle of the list, we will use two pointers - slow and fast. The slow pointer will move one step in one iteration while the fast pointer will move twice the same. So, when the fast pointer will reach the end of the list then the slow pointer will be in the middle. Now, we just have to reverse the linked list after this middle node.

Code

```
SinglyLinkedListNode* reverseList(SinglyLinkedListNode* head)
{
    SinglyLinkedListNode* prev = NULL;
    SinglyLinkedListNode* curr = head;
    while(curr != NULL){
        SinglyLinkedListNode* next = curr->next;
        curr->next = prev;
        prev = curr;
        curr = next;
    }
    return prev;
}
SinglyLinkedListNode* findMid(SinglyLinkedListNode* head){
    SinglyLinkedListNode* fast = head;
SinglyLinkedListNode* slow = head;
    while(fast->next != NULL && fast->next->next != NULL){
        fast = fast->next->next;
        slow = slow->next;
    }
return slow;
}
bool palindromeLlist(SinglyLinkedListNode *head)
{
//write your code here
    SinglyLinkedListNode *mid = findMid(head);
    SinglyLinkedListNode *head1 = reverseList(mid->next);
    bool result = true;
    while(result && head1 != NULL){
        if(head->data != head1->data)
            result = false;
        head = head->next;
        head1 =  head1->next;

    }
    return result;
}
```

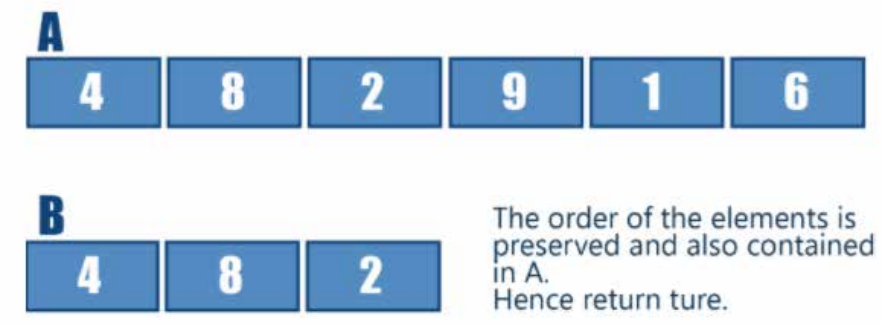
8. Validate Subsequence

Problem Statement

You have been given two arrays A and B consisting of integers only. Your task is to determine whether array B is a subsequence of array A. A subsequence of the array is a set of numbers that are not necessarily adjacent in the array but that are in the same order as they appear in the array.

Example

Let suppose we have A=[4 8 2 9 1 6] and B=[8 1 6], then we can see that B is contained in A as a subsequence. Hence, we will return “YES” else “NO”.



Logic

The idea is to compare each element and change the iterating variables to others according to their compare values. If an element in A matches with an element with B then we move both the pointers and if they do not then we move the pointer in A by 1 and repeat this until we reach the end of either of the array.

1.) Initialize two pointers at the beginning or take the last elements of the array as a pointer in the recursive function.

2.) If the last elements are equal then move both n=n-1 and m=m-1 and call the recursive function again.

3.) The base case stands as if we have completely traversed the array B then we return true else return false.

Code

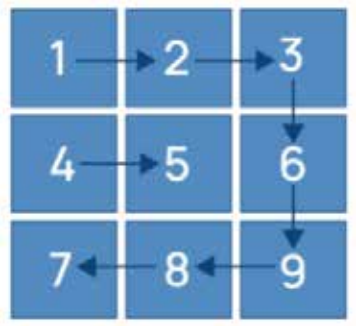
```
#include <bits/stdc++.h>
using namespace std;

int main() {
  int t;
  cin>>t;
  while(t--){
    int n,m;
    cin>>n;
    int arr[n];
    for(int i=0;i<n;i++)
        cin>>arr[i];
    cin>>m;
    int  brr[m];
    for(int i=0;i<m;i++)
        cin>>brr[i];
    int i=0;
    int j=0;
    while(i<n && j < m){
        if(arr[i] == brr[j])
            j++;
        i++;
    }
    if(j==m)
        cout<<"YES\n";
    else
        cout<<"NO\n";
  }
    return 0;
}
```

9. Spiral Matrix

Problem Statement

You are given a matrix A of N rows and M columns. The task is to print all the elements of the matrix in spiral order as shown in the figure below.

Example

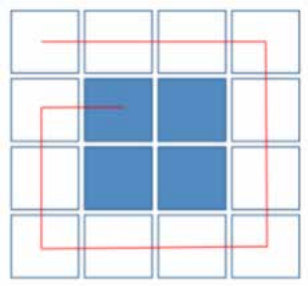


So, for the given matrix above we should print

-1 2 3 6 9 8 7 4 5

Logic

The idea is to keep some flags to change the direction after each iteration of a row or column. We can also break this problem into the smaller problem now. Each circular round can be considered as one iteration and after one complete circular the same process can be followed for the next submatrix.



For this matrix, the red mark shows the first iteration. Now, when it reaches the first blue cell, you can see the 4 blue cells become the new submatrix to be printed in a spiral fashion. Hence, we apply the same algorithm again till all the elements of the matrix are visited.

Code

```
#include<bits/stdc++.h>
using namespace std;
int main(){
    int t;
    cin>>t;
    while(t--){
    int n,m;
    cin>>n>>m;
    int arr[n][m];
    for(int i=0;i<n;i++)
        for(int j=0;j<m;j++)
            cin>>arr[i][j];

    int p=0,q=0;
    while(p<n && q<m){
        for(int i=q;i<m;i++)
            cout<<arr[p][i]<<" ";
        p++;
        for(int i=p;i<n;i++)
            cout<<arr[i][m-1]<<" ";
        m--;
        if(p<n){
            for(int i=m-1;i>=q;i--){
                cout<<arr[n-1][i]<<" ";
            }
            n--;
```

```
        }
        if(q<m){
            for(int i=n-1;i>=p;i--){
                cout<<arr[i][q]<<" ";
            }
            q++;
        }
    }
    }
    return 0;
}
```

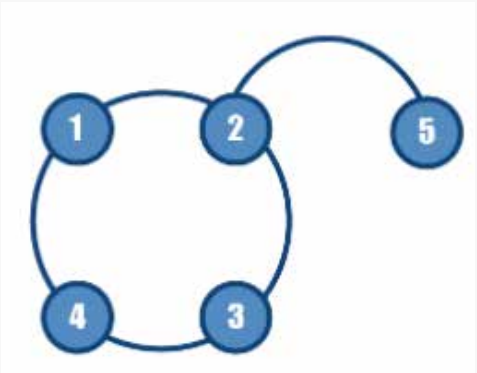
10. Detect Cycle

Problem Statement

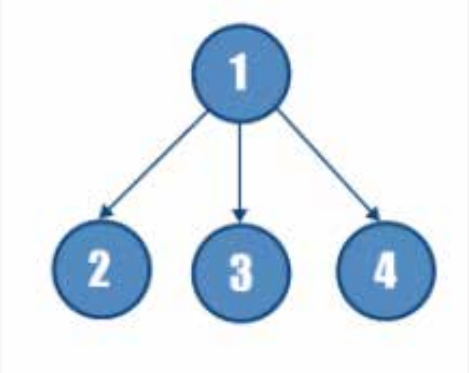
Given an undirected graph, print Yes if a cycle is present in the graph else print No.

Example

Undirected Graph with
Cycle prints Yes



Undirected Graph without
Cycle prints No



Logic

The idea is to run a DFS from every unvisited node. For every visited vertex v, if there is an adjacent u such that u is already visited and u is not a parent of v, then there exists a cycle in the graph. If we don't find any such an adjacent for any vertex, we can say that there exists no cycle in the graph. This approach assumes that there are no parallel edges between any two vertices. We use a parent array to keep track of the parent vertex so that we do not consider the visited parent as a cycle.

Code

```
#include <bits/stdc++.h>
using namespace std;
void addEdge(vector<int> adj[], int u, int v){
    adj[u].push_back(v);
    adj[v].push_back(u);
}

bool isCyclicConnected(vector<int> adj[], int s, int n, vector<bool>& visited){
    vector<int> parent(n,-1);
    queue<int> que;
    visited[s]=true;
    que.push(s);
    while(!que.empty()){
        int u = que.front();
        que.pop();
        for(auto v : adj[u]){
            if(!visited[v]){
                visited[v] = true;
                que.push(v);
                parent[v] = u;

            }
            else if(parent[u]!=v){
                return true;
            }
        }
    }
    return false;
}
int main()
{
```

```
    int n,m;
    cin>>n>>m;
    vector<int> adj[n];
    for(int i=0;i<m;i++)
    {
        int u,v;
        cin>>u>>v;
        addEdge(adj,u,v);
    }
    bool flag=false;
    vector<bool> visited(n, false);
    for(int i=0;i<n;i++){
        if(!visited[i] && isCyclicConnected(adj,i,n,visited))
        {
            flag=true;
            break;
        }
    }
   if(flag)
       cout<<"Yes\n";
   else
       cout<<"No\n";


    return 0;
}
```

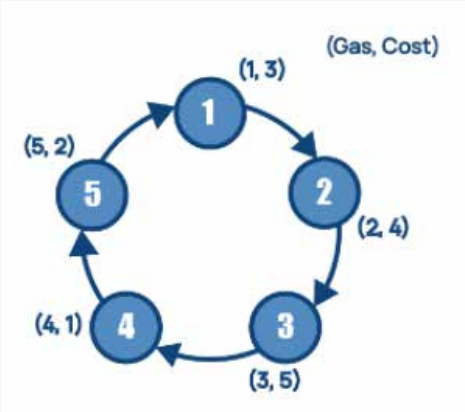
11. Truck and Circular Route

Problem Statement

You have a truck that has to cover a circular route and along this route are N gas stations and the amount of gas at each station is gas[i].
You are also given a cost array where cost[i] denotes the amount of gas required to travel from station ii to station i+1. The tank of the truck has unlimited capacity to store gas but if at any point the amount of gas in the truck is less than the cost[i], then the truck can not move further.
Consider gas stations are from index 0 to N−1 and you can start the ride from any index such that the truck completes the ride along the circular route once in the clockwise direction.
Print the index from where the truck can start if there is no such route possible print −1.

Example

Here the final cost will be 3.



Logic

The idea is to iterate over the arrays of gas and cost. We will keep the measures of current tank value, current total value, and the starting index as the first index. When we start iterating over the arrays, we will find the current value by subtracting $cost_i$ from gas_i. We will add this current value to the total as well as the tank valve.

But if the value of the tank is not positive, that means we have to change our start index to the very next index and will make empty the tank. After the iteration over the arrays, if the value of total is negative, that means there exists no index from where we can start and complete the whole circular route clockwise and if the value of total is positive, we will simply return the starting index.

Code

```
#include<iostream>
#include<vector>

using namespace std;

int canCompleteCircuit(vector<int>& gas, vector<int>& cost)
{
    int tank=0;
    int total=0;
    int start=0;

    for(int i=0; i<gas.size(); i++)
    {
        int current = gas[i]-cost[i];
```

```
            tank += current;
            total += current;
    if(tank < 0)
            {
                start = i+1;
                tank = 0;
            }
        }

        return total < 0 ? -1 : start;
    }

    int main()
    {
        int n;

        cin >> n;

        vector<int> gas(n);
        vector<int> cost(n);

        for (int i = 0; i < n; ++i)
```

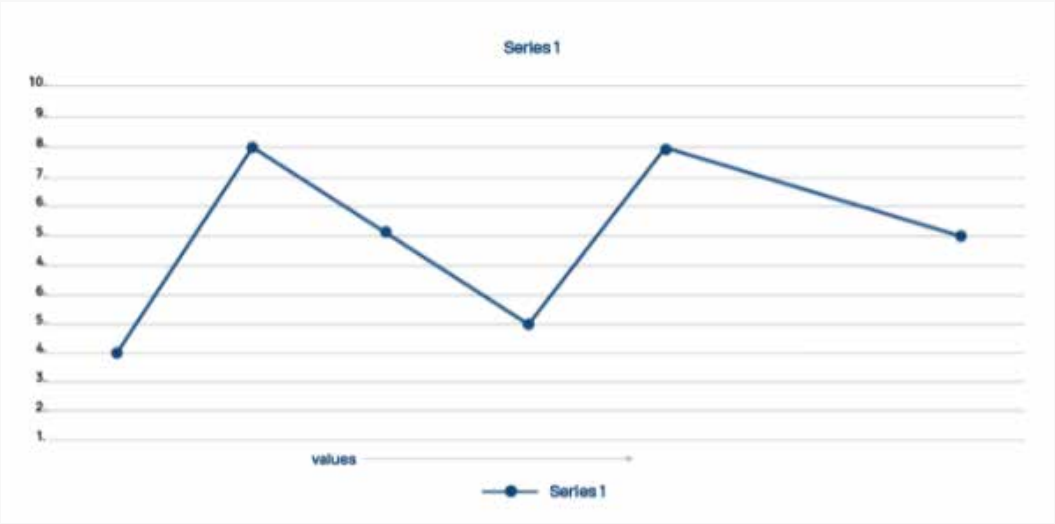
12. Delete Digit

Problem Statement

You are given an integer N. You need to do K operations to the integer so that the resulting integer is minimum possible. In one operation you can remove any digit from the given integer.

Example

Suppose we have our N = 296383 and k = 2, to remove 1 digit, we can choose to wither 9 or 8 in the above peaks to make the curve flat. This is because we need a non-decreasing order of digits. So, we first remove the 8 we get 29633, and after removing 9 we get 26383, which is smaller? We need to remove the peaks from the MSB first then we will get the smallest value. This whole approach is based on this fact.
Whenever the graph dips and we have some operations left then we can remove the peak. We will use a stack for this purpose.



Logic

Take a stack of characters

i.) Push the first character

ii.) Compare the current element with the stack top for any dips. If stack top > current element then a dip occurs at the stack top. We need to remove it if we have any k operations left and k = k-1.

iii.) In the end, we will check if there are any more elements left in the stack and pop and append them to our answer.

iv.) Keep in mind that, we have traversed the N from the left side, so the actual number in the answer will be the reverse of what we have got. So, reverse the answer.

Code

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

int main() {
    int t;
    cin>>t;
    while(t--){
        string s;
        cin>>s;
        int k;
        cin>>k;
stack<char>st;
        int n=s.size();
        for(int i=0;i<n;i++){
            if(st.empty()||st.top()<s[i]||k<=0){
                st.push(s[i]);
            }
```

```
            else{
                while(!st.empty()&&st.top()>s[i]&&k>0){
                    st.pop();
                    k--;
                }
                st.push(s[i]);
            }
        }
        while(!st.empty()&&k>0){
            st.pop();
            k--;
        }
        string ans;
        while(!st.empty()){
            ans+=(st.top());
            st.pop();
        }
        reverse(ans.begin(),ans.end());
        int index=-1;
        for(int i=0;i<ans.size();i++){
            if(ans[i]!='0'){
                break;
            }
            index=i;
        }
        if(index!=-1){
            ans=ans.substr(index+1);
        }
        if(ans.empty()){
           cout<<"0\n";
        }
        else
            cout<<ans<<"\n";

    }
    return 0;
}
```

13. Mike and Binary Number

Problem Statement

Mike loves solving Binary number problems. So he gives you a number N and asks you to find the binary representation of N. But he thinks this is easy for you, so he added one more condition to this problem. Now he asks to find all possible permutations of the binary representation of number N.

Example

Suppose we have N=5, so its binary representation is "101", and all the permutations of this string are 011, 101, 110. We have to return this string of permutations.

Logic

The idea is to count all the 1s and 0s in the binary representation of N and then we can have a recursive function to find all the permutations of this number. In the recursive function, if there are no zeroes then we will append all ones in the new string and return it or if there are no ones then append all 0s and return it for the base case, and for the recursive case we will once call the recursive function which appends 1, and later 0 and with these, all the values of zeroes and ones are reduced correspondingly.

Code

```
#include <bits/stdc++.h>
using namespace std;
void permutation(int no_ones, int no_zeroes, string accum,vector<string>&perm){
    if(no_ones == 0){
        for(int i=0;i<no_zeroes;i++){
            accum += "0";
        }
        perm.push_back(accum);
        return;
    }
else if(no_zeroes == 0){
        for(int j=0;j<no_ones;j++){
            accum += "1";
        }
        perm.push_back(accum);
        return;
    }

    permutation (no_ones - 1, no_zeroes, accum + "1",perm);
    permutation (no_ones , no_zeroes - 1, accum + "0",perm);
}

int main(){
    int t;
    cin>>t;
    while(t--){
    int n,ones=0,zeros=0;
    cin>>n;
    while(n>0)
    {
```

```
        if(n&1)
            ones++;
        else
         zeros++;
       n =n>>1;
    }
    string append = "";
    vector<string>perm;

    permutation(ones, zeros, append,perm);
    sort(perm.begin(),perm.end());
    for(int i=0;i<perm.size();i++)
      cout<<perm[i]<<" ";
    }
}
```

14. Sum Subarray

Problem Statement

You are given an array A of size N. You have to print the length of the smallest subarray that has a sum at least equal to K.

Example

Suppose we have our array as [1 2 3 4 2 1] and k = 7, then we have many subarrays whose sum is at least equal to 7 like – [4 2 1], [1 2 3 4], [1 2 3 4 2],..[3 4],...[3 4 2], etc, but out of these many subarrays, we have only [3 4] whose size is the minimum. Hence, we return its length 2.

Logic

Well, the idea here is to use a data structure that can store our subarray which has the sum of more than ‘k’ only. If we have a subarray such in our ds then we will evict any of the elements and try to minimize the size keeping the sum more thank. This can be done using a map or queue. We will try to implement this logic using a queue. Take a sum variable that will keep on adding the elements to it and when the sum is greater than k we will check if this subarray is the smallest we can get. For the next elements when the sum is greater than k, we evict the element from the front of the queue and also update our sum. We keep on doing this till we reach the end of the array. In this, we also keep track of the minimum length subarray when the sum becomes greater than k.

Code

```
#pragma GCC optimize("Ofast")
#include<bits/stdc++.h>
using namespace std;
int32_t main()
{
      int n,k;
      cin>>n>>k;
      int A[n];
      for(int i=0;i<n;i++)
      {
            cin>>A[i];
      }
      int sum=0;
      int i=0;
      queue<int>q;
      int ans=INT_MAX;
      while(i<n)
      {
            q.push(A[i]);
            sum+=A[i];
            if(sum>=k)
            {
                  while(sum>=k)
                  {
                        if(sum>=k)
                        {
                              int temp=q.size();
                              ans=min(ans,temp);
                        }
                        sum-=q.front();
                        q.pop();
                  }
            }
            i++;
      }
      cout<<ans<<endl;
}
```

15. Sunset View

Problem Statement

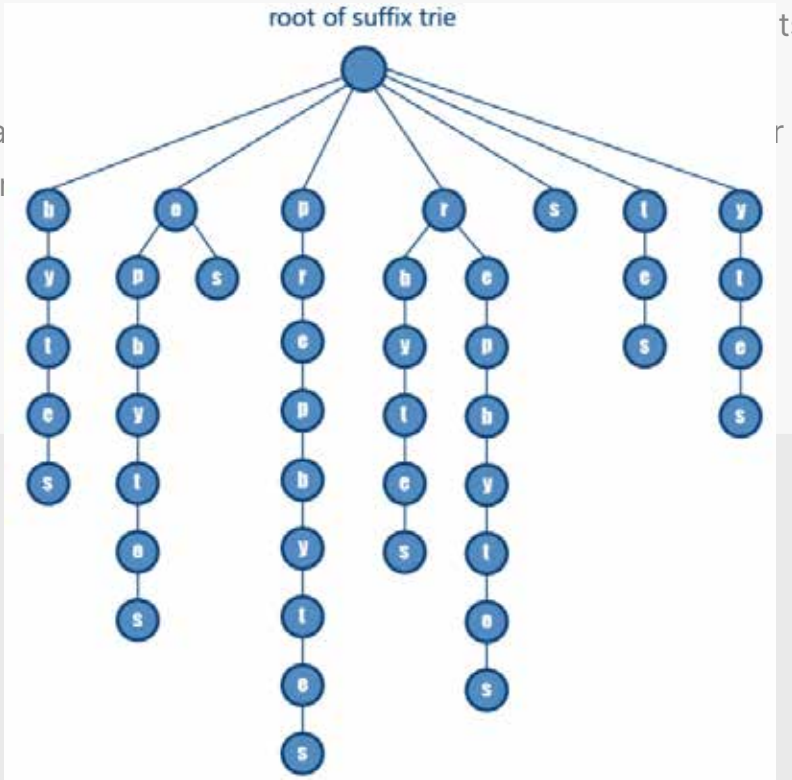
You have been given an array B of buildings and a direction that all of the buildings face. You have to find an array of the indices of the buildings that can see the sunset. A building can see the sunset if it's strictly taller than all of the buildings that come after it in the direction it faces. All of the buildings face the same direction, and this direction is either east or west, denoting by the input string named direction, which will always be equal to either EAST or WEST.

Example

Suppose the given array B = 4 6 5 4 5 2, and EAST then output is 1 4 5

Logic

The idea is to make use of stack data structure for an appropriate and efficient solution to this problem. Take the input of the building heights and store them in an array. Take the input direction, whether it is EAST or WEST. Iterate ts into the stack appropriately. If the top of r push it into the stack. Else pop out of sta r than the current element. Using this appr have a sunset view.



Code

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

int main()
{
    int t;
    cin>>t;
    while(t--){
    int n;
    cin >> n;
    int arr[n];
    for (int i = 0; i < n; i++)
```

```
        cin >> arr[i];
    string str;
    cin >> str;
    stack<int> st;
    int i = 0;
    int j = 1;
    if (str == "WEST")
    {
        i = n - 1;
        j = -1;
    }
    while (i >= 0 && i < n)
    {
        if (st.empty() || arr[i] < arr[st.top()])
            st.push(i);
        else
        {
            while (!st.empty() && arr[i] >= arr[st.top()])
            {
                st.pop();
            }

            st.push(i);
        }

        i += j;
    }
    vector<int> res;
    while (!st.empty())
    {
        res.push_back(st.top());
        st.pop();
    }
    if (str == "EAST")
        reverse(res.begin(), res.end());

    for (auto ele : res)
        cout << ele << " ";
    cout << "\n";
    }
    return 0;
}
```

16. Zip Linked List

Problem Statement

You are given the head of the singly linked list of arbitrary length K. Your task is to zip the linked list in place (i.e., doesn't create a brand new linked list) and return its head.

A linked list is zipped if it's nodes are in the following order, where KK is the length of the linked list:

1st node → Kth node → 2nd node → (k-1)th node → 3rd node → (k-2)th node → ....

Example

Initial Linked List

Final Linked List

We have to reorder the linked list in place i.e. extra space cannot be used. We have to complete the function zipLinkedList() and return the head of the new linked list.

Logic

The idea is to first find the mid of the linked list and break it into two halves. Then we have to reverse the second half obtained and attach nodes from both parts one by one. No extra space will be used in this way.

For example, if the linked list is: 1 → 2 → 3 → 4 → 5, We break it into two halves as - 1 → 2 → 3 and 4 → 5. After reversing the second half, we get - 1 → 2 → 3 and 5 → 4.

Finally we can take nodes one by one from both linked lists and attach them as - 1 → 5 → 2 → 4 → 3.

Code

```
C++
SinglyLinkedListNode* reverseLinkedList(SinglyLinkedListNode* head){
    SinglyLinkedListNode * prev=  NULL, *next = NULL;
    SinglyLinkedListNode *curr = head;
    while(curr!=NULL){
        next = curr->next;
        curr->next = prev;
        prev = curr;
        curr = next;
    }
    return prev;
}
SinglyLinkedListNode * findMid(SinglyLinkedListNode *head){
    SinglyLinkedListNode* fast = head;
    SinglyLinkedListNode* slow = head;
    while(fast->next != NULL && fast->next->next != NULL){
        slow = slow->next;
        fast = fast->next->next;
    }
    return slow;
}
SinglyLinkedListNode *zipLinkedList(SinglyLinkedListNode *head)
{
    if(head->next == nullptr || head->next->next== nullptr)
```

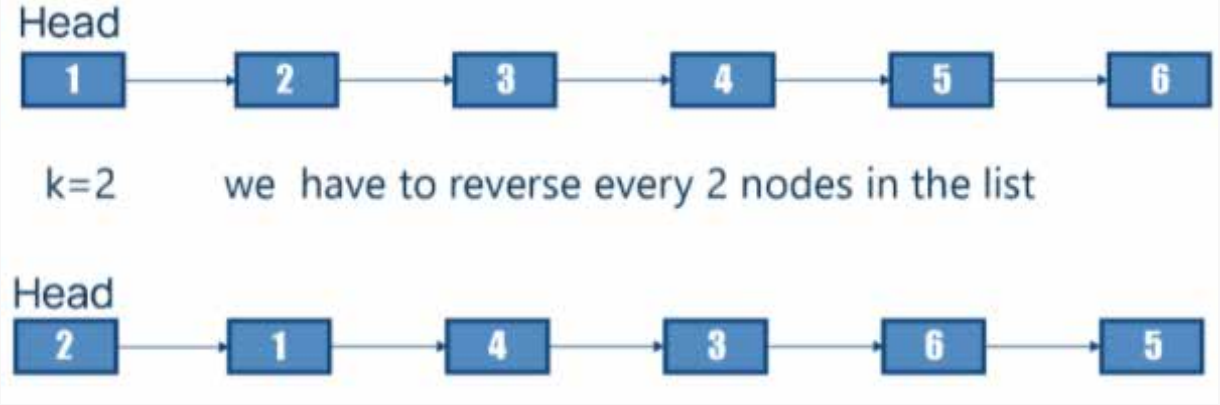
```
        return head;
    SinglyLinkedListNode* mid = findMid(head);

    SinglyLinkedListNode* temp = mid->next;
    mid->next = nullptr;
    SinglyLinkedListNode* head2 = reverseLinkedList(temp);

    SinglyLinkedListNode* t1 = head;
    SinglyLinkedListNode* t2 = head2;
    while(t1!= nullptr && t2!= nullptr){
        SinglyLinkedListNode* temp1 = t1->next;
        SinglyLinkedListNode* temp2 = t2->next;
        t1->next = t2;
        t2->next = temp1;

        t1 = temp1;
        t2 = temp2;
    }
    return head;
}
```

17. Reverse in a group

Problem Statement

This is another reverse-the-list problem. You must be thinking, "It is easy, I have already Done it!" But wait, there is a twist. You have to reverse the list in a group of KK nodes. If the list is HEAD→2→3→7→4→1→6→NULL and K is 3, your modified list should be HEAD→7→3→2→6→1→4→NULL

Example



Logic

We will take three-pointers like current, previous, and next to where the current is pointed to the head. We will also take a count by which we will have the count of nodes to reverse at a time in a group. For one iteration or group of k, we will use the normal algorithm to reverse the group. In the end, we assign the head to the new previous.
Assign three-node pointers curr = head, & prev, next to null.
- For every iteration till count is less than k,
a.) Next = curr→next
b.) Curr→next = prev
c.) Prev = curr
d.) Curr = next
e.) In the end, we assign the new head to the prev as prev is the new head.

Code

```
C++
SinglyLinkedListNode *reverseKnodes(SinglyLinkedListNode *head,int k)
{
//write your code here
SinglyLinkedListNode* current = head;
   SinglyLinkedListNode* next = NULL;
  SinglyLinkedListNode* prev = NULL;
    int count = 0;
    while (current != NULL && count < k)
    {
        next  = current->next;
        current->next = prev;
        prev = current;
        current = next;
        count++;
    }

    if (next !=  NULL)
       head->next = reverseKnodes(next, k);
    return prev;
}
```

18. Painter Partition

Problem Statement

PrepBuddy has N boards of length A1, A2, A3...AN. He wants to paint these boards so he hires P painters. Each painter takes 1 unit of time to paint 1 unit of the board. He is short on time, so he asks for your help in determining the minimum time to complete the task with the constraint that any painter can only paint continuous sections of boards.

Example

For example if the length's are [1,2,3,4] then a painter can be assigned [1,2], [1,2,3], [1,2,3,4], [2,3] etc, but not [1,2,4],[2,4] etc. Another example, let the board array be 9 6 7 10 5 and the number of painters is 3. So the 3 painters can be assigned subarrays as [9,6], [7], [10,5]. They will work simultaneously, so the minimum time required to paint all the boards will be 15 units.

Logic

Take input and calculate the upper and lower bounds required for binary search.
In the binary search, we run a while loop till $lo \lt hi$:
- We calculate mid as lo + (hi - lo) / 2.
- Now we have to check for this mid-value if it is possible to paint all boards within this time, with the given number of painters.
- If it is possible, we check for values in the left half, if we can find a lesser time than this. So we update hi-value with mid.
- Else we search in the right part and update the low value with mid+1.

In the end, we return lo from the function which will give us our answer.

Code

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

long long getMax(int arr[], int n)
{
    long long max = INT_MIN;
    for (int i = 0; i < n; i++)
        if (arr[i] > max)
            max = arr[i];
    return max;
}

long long getSum(int arr[], int n)
{
    long long total = 0;
    for (int i = 0; i < n; i++)
        total += arr[i];
    return total;
}

long long numberOfPainters(int arr[], int n, int maxLen)
{
    long long total = 0, numPainters = 1;

    for (int i = 0; i < n; i++)
    {
        total += arr[i];

        if (total > maxLen)
        {

            total = arr[i];
            numPainters++;
        }
    }

    return numPainters;
}
long long partition(int arr[], int n, int k)
{
    long long lo = getMax(arr, n);
    long long hi = getSum(arr, n);
```

```
while (lo < hi)
    {
        long long mid = lo + (hi - lo) / 2;
        long long requiredPainters = numberOfPainters(arr, n, mid);
        if (requiredPainters <= k)
        {
            hi = mid;
        }
        else
            lo = mid + 1;
    }
    return lo;
}

int main()
{
    int t;
    cin >> t;
    while (t--)
    {
        int n, p;
        cin >> n >> p;
        int arr[n];
        for (int i = 0; i < n; i++)
        {
           cin >> arr[i];
        }
        cout << partition(arr, n, p) << endl;
    }

    return 0;
}
```

19. Sudoku Solver

Problem Statement

As we all know, you are quite interested in the politics of Prepland since childhood, here comes the best opportunity for you as the contest for becoming Mayor of the city is just going to start. In the contest, you will be given a partially filled sudoku. A sudoku solution must satisfy all of the following rules

Each of the digits 1 - 9 must occur exactly once in each row

Each of the digits 1 - 9 must occur exactly once in each column

Each of the digits 1 - 9 must occur exactly once in each of the 9 3 × 3 sub-boxes of the grid.

Example

...26.7.1
68..7..9.
19...45..
82.1...4.
..46.29..
.5...3.28
..93...74
.4..5..36
7.3.18...
Is the matrix then the output
435269781
682571493
197834562
826195347
374682915
951743628
519326874
248957136
763418259

Logic

The basic idea here is to try to fill the empty cells one by one using numbers 1 to 9 and check whether placing a number at an empty cell violates the above rules or not. If placing a number in the range of 1 to 9 does not violate the above rule then do the same for the next empty cell and if it violates the above-mentioned rules then try filling the empty cell with the next number in the range 1 to 9. Do it until the grid is filled. If the grid gets filled return the grid.

Code

```
#include <bits/stdc++.h>
using namespace std;
bool isValid(vector<vector<char>>& board, int a, int i, int j){

    for(int c=0;c<9;++c)
        if(board[i][c]-'0' == a)
            return 0;

    for(int r=0;r<9;++r)
        if(board[r][j]-'0' == a)
            return 0;

    int br = i/3, bc = j/3;
    for(int r=br*3; r<br*3+3;++r)
        for(int c=bc*3; c<bc*3+3;++c)
            if(board[r][c]-'0' == a)
                return 0;
    return 1;
}

bool sudoku(vector<vector<char>>& board, int i, int j){
    if(i == 9) return 1;
    if(j == 9) return sudoku(board,i+1,0);
    if(board[i][j] != '.') return sudoku(board,i,j+1);
    for(int a=1;a<=9;++a){
        if(isValid(board,a,i,j)){
            board[i][j] = a+'0';
            if(sudoku(board,i,j+1))
                return 1;
            board[i][j] = '.';
        }
    }
    return 0;
}
int main() {
    vector<vector<char>> board;
    for(int i=0;i<9;i++){
        vector<char>row;
        for(int j=0;j<9;j++){
            char a;
            cin>>a;
            row.push_back(a);
        }
        board.push_back(row);

    }
```

```
    sudoku(board,0,0);
    for(int i=0;i<9;i++){
        for(int j=0;j<9;j++){
            cout<<board[i][j];
        }
        cout<<"\n";

    }

    return 0;
}
```

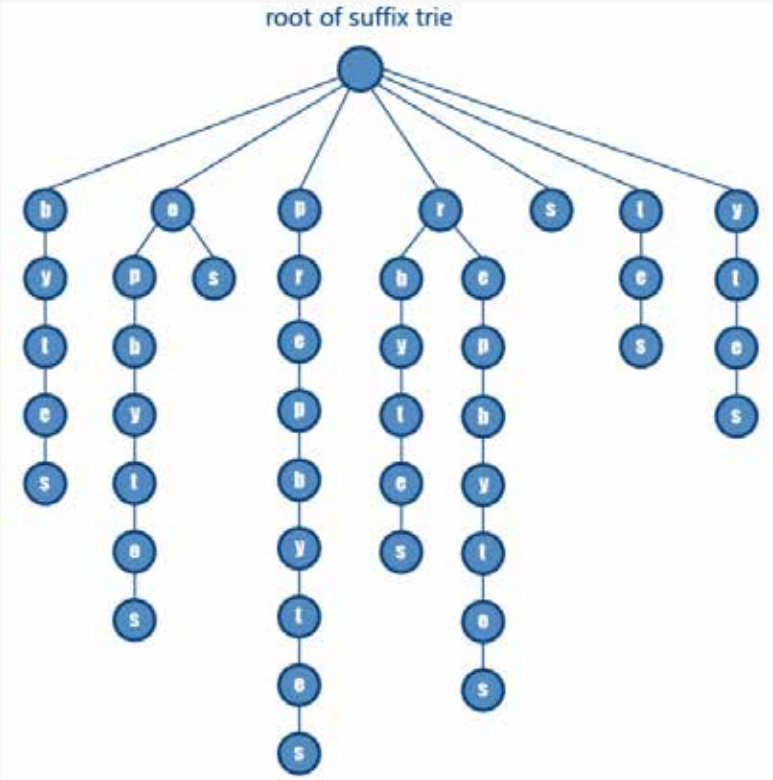
20. Suffix Trie Construction

Problem Statement

You are given two strings, one is a text string SS and the other is a pattern string PP. The task is to print the starting indexes of all the occurrences of pattern string in the text string. For printing, starting index of a string should be taken as 0. If the pattern string cannot be found, print -1.

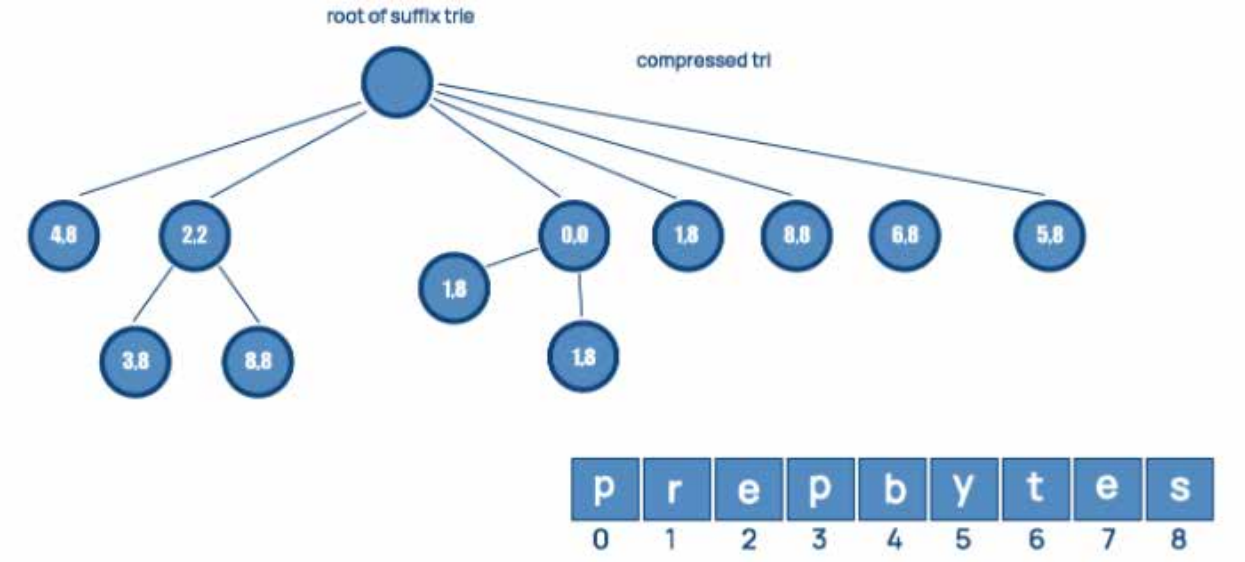
Example

Suppose our S = “prepbytes”



S can be represented as suffix trie consisting of suffix words as –
s
es
tes
ytes
bytes
pbytes
epbytes
repbytes
prepbytes

And this is then compressed as follows –



Logic

Building this trie –

1) Starting from the first character of the pattern and root of the Suffix Tree, do the following for every character.

a) For the current character of the pattern if there is an edge from the current node of the suffix tree, follow the edge.

b) If there is no edge, print “pattern doesn’t exist in-text” and return.

2) If all characters of the pattern have been processed, i.e., there is a path from the root for characters of the given pattern. To store indexes, we use a list with every node that stores indexes of suffixes starting at the node.

Code

```
#include <bits/stdc++.h>
using namespace std;
const int size=26;
struct SuffixTrie
{
    struct SuffixTrie *children[size];
    bool endofWord;
    list<int> nodeIndex;
};

struct SuffixTrie* createSuffixTrieNode()
{
    struct SuffixTrie * temp= new SuffixTrie;
    temp->endofWord = true;
    for(int i=0;i<26;i++)
        temp->children[i]= nullptr;
    return temp;

}

void insert(struct SuffixTrie *root, string key, int offset)
{
    int keyIndex;
    int keyLength = key.length();
    int trieIndex;
    struct SuffixTrie *temp = root;
    for(keyIndex=0;keyIndex<keyLength;keyIndex++)
    {
        trieIndex = key[keyIndex]-'a';
        if(temp->children[trieIndex]== nullptr)
        {
            temp->children[trieIndex]= createSuffixTrieNode();

        }
        temp = temp->children[trieIndex];
        temp->nodeIndex.emplace_back(keyIndex+offset);
    }
    temp->endofWord=true;
}

void matchPattern(struct SuffixTrie *root, string pattern)
{
    if(pattern.empty())
        return;
    struct SuffixTrie * currentNode = root;
    int i=0;
    while(i<pattern.length())
    {
```

```
        if(currentNode== nullptr) break;
        currentNode = currentNode->children[pattern[i]-'a'];
        i+=1;
    }
    if(i<pattern.length() || currentNode==NULL)
        cout<<"-1"<<"\n";
    else
    {
        list<int> :: iterator it;
        for(it=currentNode->nodeIndex.begin();it!=currentNode->nodeIndex.end();it++)
        {
            cout<<*it-(pattern.length()-1)<<" ";
        }
        cout<<"\n";
    }
}
int main() {
    int t;
    cin>>t;
    while(t-->0) {
        string text,pattern;
        cin>>text>>pattern;
        struct SuffixTrie *root = createSuffixTrieNode();
        for (int i = 0; i < text.length(); i++)
            insert(root, text.substr(i, i + text.length()), i);
        matchPattern(root, pattern);
    }
    return 0;
}
```

21. Playing With Balloons

Problem Statement

Claudia and Arif are playing with balloons. Initially, she has n balloons and she marked each balloon with a positive integer. The integer of the ith balloon is represented as A[i] in the array A. Now she asked Arif to burst all the balloons. If Arif bursts a balloon then his score gets incremented with an integer that is obtained by multiplying the values of adjacent balloons with that of the present balloon. In other words, if Arif Brust ith balloon then his score gets incremented by A[i-1]*Ai*A[i+1]. After the burst, the (i-1)th and (i+1)th balloon will become adjacent to each other. Help Arif to calculate his maximum score.

Example

Suppose we have array as arr=[2 1 4 7], then [2, 1, 4, 7] -- > [2, 4, 7] -- > [2, 7] -- > [7] -- > [] score = 2 * 1 * 4 + 2 * 4 * 7 + 1 * 2 * 7 + 1 * 7 * 1 = 8 + 56 + 14 + 7 = 85

Logic

We can solve this problem using dynamic programming. First, consider a sub-array from indices Left to Right(inclusive). If we assume the balloon at index Last to be the last balloon to be burst in this sub-array, we would say the coined gained to be-A[left-1]*A[last]*A[right+1]. Also, the total Coin Gained would be this value, plus dp[left][last – 1] + dp[last + 1][right], where dp[i][j] means maximum coin gained for sub-array with indices i, j. At the end we will return dp[1][N].

Code

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

int main() {
    int n;
    cin>>n;
    int arr[n+2];
    arr[0]=1;
    arr[n+1]=1;
    for(int i=1;i<=n;i++)
        cin>>arr[i];
    int dp[n+2][n+2];
    memset(dp,0,sizeof dp);
    for(int winLen=1;winLen<=n;winLen++){
        for(int lb=1;lb<=n-winLen+1;lb++){
            int ub = lb+winLen-1;
            for(int i=lb;i<=ub;i++){
                int left = dp[lb][i-1];
                int right = dp[i+1][ub];
                int current = arr[lb-1]*arr[i]*arr[ub+1];
                dp[lb][ub] = max(dp[lb][ub], left+right+current);
            }
        }
    }
    cout<<dp[1][n]<<"\n";
    return 0;
}
```

22. K Closest Points to Origin

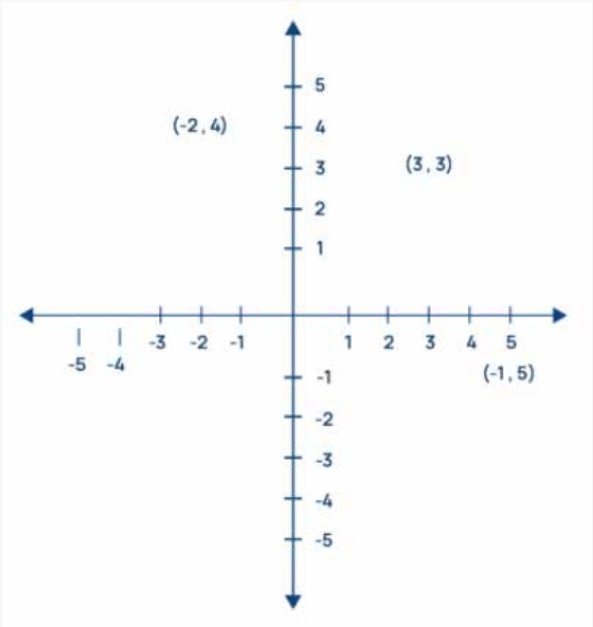
Problem Statement

You have been given N points placed in the X–Y plane and an integer K. You have to find the KK closest points which are close to the origin i.e., (0,0) out of the given N points. Note -You have to output these K–points after sorting them in the ascending order first with respect to X–coordinate and the according to Y–coordinate.

The distance between two points on the X-Y plane is the Euclidean distance as

$$\sqrt{(x1-x2)^2+(y1-y2)^2}$$

Example



Output: - 2 4

3 3

Logic

The idea is to calculate the Euclidean distance of the given points from the origin and store the distances in an array. Now make a copy of the array and sort the array in ascending order. Now we can see the orginal index of the first k minimum distances of the sorted array in the original array of distances and print the first k closest points from the list.

Code

```
C++
#include <bits/stdc++.h>
using namespace std;
int main() {
    int n,k;
    cin>>n>>k;
    vector<vector<long long>> arr(n, vector<long long> (2));

    int dist[n];
    for(int i = 0; i < n; i++)
    {
        int x, y;
        cin>>x>>y;
        arr[i][0] = x;
        arr[i][1] = y;
        dist[i]=x*x + y*y;
    }
    priority_queue<pair<int, int>>pq;
    for(int i=0;i<n;i++){
        pq.push(make_pair(dist[i],i));
        if(pq.size()>k)
            pq.pop();
    }
    vector<vector<long long>> ans(k, vector<long long> (2));
    int p=0;
    while(!pq.empty()){
        int index = pq.top().second;
        pq.pop();
        ans[p][0]= arr[index][0];
        ans[p][1] = arr[index][1];
        p++;
    }
    sort(ans.begin(),ans.end());
    for(int i=0;i<k;i++)
        cout<<ans[i][0]<<" "<<ans[i][1]<<"\n";
    return 0;
}
```

23. Longest increasing subsequence

Problem Statement

You are given an array of N non-negative integers. The task is to find the length of the longest increasing subsequence. The longest increasing subsequence is an increasing subsequence such that all elements of the subsequence are sorted in increasing order.

Example

Suppose we have an array as arr=[1 8 3 14 5] then the length of longest increasing subsequence is 3 which includes - [1 3 5]. We need to return the length of the subsequence.

Logic

We can start with a recursive type solution. We create a recursive function that returns the length of the LIS possible from the current element onwards (including the current element). Inside each function call, we consider two cases

The current element is larger than the previous element included in the LIS. In this case, we can include the current element in the LIS. Thus, we find out the length of the LIS obtained by including it. Further, we also find out the length of LIS possible by not including the current element in the LIS. The value returned by the current function call is, thus, the maximum out of the two lengths.

The current element is smaller than the previous element included in the LIS. In this case, we can't include the current element in the LIS. Thus, we find out only the length of the LIS possible by not including the current element in the LIS, which is returned by the current function call.

Let arr[0..n-1] be the input array and L(i) be the length of the LIS ending at index i such that arr[i] is the last element of the LIS.

Then, L(i) can be recursively written as:

L(i) = 1 + max(L(j)) where 0 < j < i and arr[j] < arr[i] or

L(i) = 1, if no such j exists.

Code

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

int main() {
    int t;
    cin>>t;
    while(t--){
        int n;
        cin>>n;
        int arr[n];
        for(int i=0;i<n;i++)
            cin>>arr[i];

        int maximum = 0;
        int dp[n];
        dp[0]=1;
        for(int i=1;i<n;i++){
            dp[i]=1;
            for(int j=0;j<i;j++){
                if(arr[i]>arr[j] && dp[i]<dp[j]+1){
                    dp[i] = dp[j]+1;
                    maximum = max(maximum, dp[i]);
                }
            }
        }
        cout<<maximum<<"\n";
    }
    return 0;
}
```

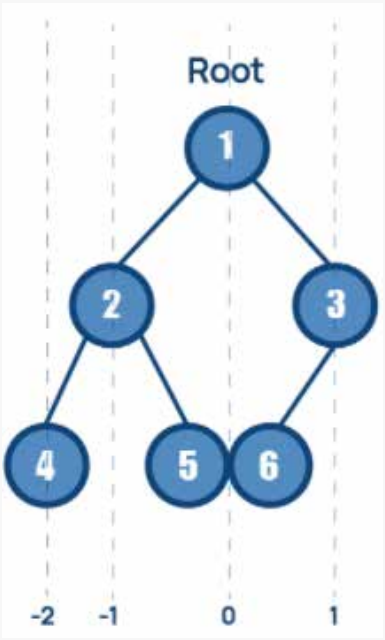
24. Vertical Order Traversal of a Binary tree

Problem Statement (Simplified)

You have been given the root of a binary tree. Your task is to find the Vertical order traversal of its node's values. The vertical order traversal of a binary tree is a list of top-to-bottom orderings for each column index starting from the leftmost column and ending on the right-most column.

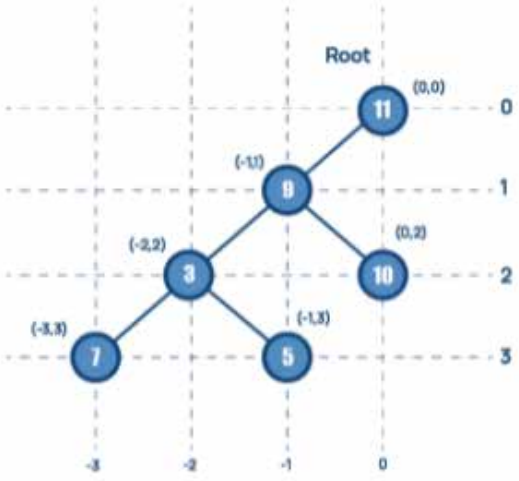
Example

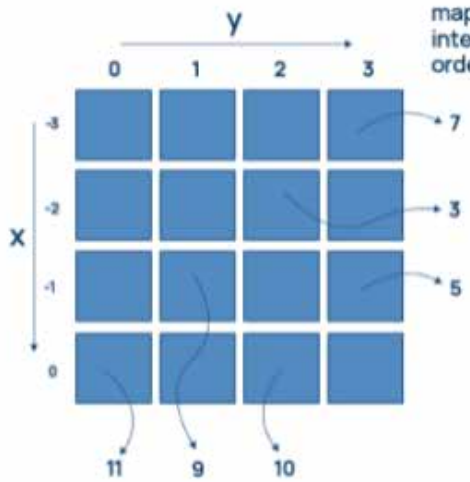
The order will now be given as 4 2 1 5 6 3



Logic

The idea is to use a coordinate system to assign the position to the nodes. We will use the lines shown above as a reference, all the nodes to the left of the root will keep on decreasing this vertical line value and increases when we go to the right of the root.
We will store the elements having the same vertical line number in the same list assuming the vertical line number for the root node is 0. We will store this on
a map.





map of map of int and set of integers to store the list in sorted order in a particular position list

at last we traverse this map of map and print the elements present in them, as it is a set for the same position the list will already be stored.

we traverse this matrix and get 7 3 9 5 11 10

Now this is fine, but we will need to modify this a bit as there are more conditions like – There may be multiple nodes in the same row and same column. In such a case, sort these nodes by their values. For this, we need to keep track of both the x and y coordinates level-wise. If two nodes at the same level have the same x,y then we print them in sorted order.
We will require the x coordinate also let's draw that on our sample test case.

We assign the position of x and y coordinates by traversing the nodes using simple DFS. For moving to left child col = col -1 and row = row+1 and for the right child we have col=col+1 and row=row+1.
We store the values of nodes as a list in the map of sets. Each map for one type of coordinate.

Code

```
void dfs(node* node, int hd, int vd, unordered_map<int, vector<pair<int, int>>>&
coordinates, int& minDistance, int& maxDistance){

    if(node == nullptr)
        return;

    if(coordinates.find(hd) != coordinates.end())
        coordinates[hd].push_back({vd, node->value});
    else
        coordinates.insert({hd, {{vd, node->value}}});

    minDistance = min(minDistance, hd);
    maxDistance = max(maxDistance, hd);

    dfs(node->left, hd-1, vd+1,  coordinates, minDistance, maxDistance);
    dfs(node->right, hd+1, vd+1, coordinates, minDistance, maxDistance);
}


vector<vector<int>> verticalTrav(node* root) {

    vector<vector<int>> result;
if(root == NULL)
        return result;
    unordered_map<int, vector<pair<int, int>>> coordinates;

    int minDistance = 0, maxDistance = 0;

    dfs(root, 0, 0, coordinates, minDistance, maxDistance);
```

```
    for(int i = minDistance; i < maxDistance+1; i++){
        sort(coordinates[i].begin(), coordinates[i].end(), [](pair<int, int>& p1,
pair<int, int>& p2){
            return (p1.first < p2.first) || ((p1.first == p2.first) && (p1.second
< p2.second));
        });

        vector<int> vertical;

        for(pair<int, int>& p: coordinates[i])
            vertical.push_back(p.second);

        result.push_back(vertical);
    }
    return result;
}
void verticalTraversal(node *t)
{
    vector<vector<int>>res =verticalTrav(t);
    for(auto i:res){
        for (auto j:i)
            cout<<j<<" ";
    }
}
```

25. Power Game

Problem Statement (Simplified)

Today, Bunny's father is doing work from home. But, Bunny is constantly annoying him and makes it hard for him to complete his work. So, Bunny's father gave him two integers X and N, and order him to find the value of XN. As the value of XN could be very large, he also orders to mod the result by (109+7). Bunny is not so good with mathematical problems, so he asks you for help. As a good friend of bunny, now it's your task to solve this problem.

Example

for the testcase, 25= 2*2*2*2*2 = 32 and 32 % 109+7 = 32.

Logic

The idea is to use an iterative approach to find the product. Here we will use a trick, suppose the power is even that will mean x4 can be broken into x2 and x2 and multiply these two rather than continuously multiplying x. This will reduce the time complexity from O(n) to O(logn).
Initialize our result = 1
If n is odd
- res = (res * x)%1000000007
If n is even
- n = n / 2;
- x = (x * x)%1000000007;

Code

C++

```
#include <iostream>
using namespace std;
long long binaryExponentiation(long long a, long long b,long long m)
{
   long long result=1;
   while(b>0)
   {
       if(b&1)
       {
           result = (result*a)%m;
       }
       a=(a*a)%m;
       b=b>>1;
   }
    return result;
}
int main() {
    int t;
    cin>>t;
    while(t-->0) {
        long long a, b, m;
        cin >> a >> b;
        m = 1000000007;
        cout << binaryExponentiation(a, b, m)<<"\n";
    }
    return 0;
}
```

26. Decode

Problem Statement (Simplified)

Arnab is given two sorted arrays of length nn and length mm. Help Arnab to find the median of two arrays after they are merged. The catch is that it is not allowed to use extra space or merge the two arrays into one single array and find the median.
Expected Complexity: O(log(m+n))

Example

Array1 = [1,2]
Array2 = [3,4,5]
Merged array would be [1,2,3,4,5]
Median = 3

Logic

Now, one intuition is both are sorted array and we have to use some modification of Binary Search. The idea is we start from middle of both the arrays and now in the middle we have two options either left of partition or right of partition and now we have to observe these four elements, we say the first array as X and next array as Y.
We assign four pointers maxLeftX = 0, maxLeftY = 0, minRightX = 0, minRightY = 0
if maxLeftX <= minRightY ans maxLeftY <= minRightX then if the sum of x and y is even then median = (max (maxLeftX, maxLeftY) + min (minRightX, minRightY)) / 2.0;
Else median = max maxLeftX, maxLeftY
        else if maxLeftX > minRightY then r = partX - 1
 else l = partX + 1;
Return the median obtained.

Code

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

  double findMedian(vector<int> x, vector<int> y) {
    if(x.size()>y.size()) {
      return findMedian(y,x);
    }
    int n=x.size(),m=y.size();
    int low=0;
    int high=n;
    while(low<=high) {
      int partx=(low+high)/2;
      int party=(n+m+1)/2-partx;

      int maxleftx=(partx==0) ? INT_MIN : x[partx-1];
      int minrightx=(partx==n) ? INT_MAX : x[partx];

      int maxlefty=(party==0) ? INT_MIN : y[party-1];
      int minrighty=(party==m) ? INT_MAX : y[party];

      if(maxleftx<=minrighty&&maxlefty<=minrightx) {
        if((n+m)%2==0) {
          return  (double)(max(maxleftx,maxlefty)+min(minrighty,minrightx))/2;
        }
        return (double)max(maxleftx,maxlefty);
      }
      else if(maxleftx>minrighty) {
        high=partx-1;
      }
      else {
        low=partx+1;
      }
    }
  }

  int main()
  {
    int t;
    cin>>t;
    while(t--)
    {
      int n,m;
      cin>>n>>m;
      vector<int> X,Y;
      for(int i=0; i<n; i++) {
```

```
        int a;
        cin>>a;
        X.push_back(a);
      }
      for(int i=0; i<m; i++) {
        int a;
        cin>>a;
        Y.push_back(a);
      }
      cout<<findMedian(X,Y)<<endl;
    }
    return 0;
  }
```

27. Find the Longest Peak

Problem Statement (Simplified)

You have been given an array A which consists of integers only and your task is to find the length of the longest peak in this array. A peak is defined as adjacent integers in the array that are strictly increasing until they reach a tip(the highest value in the peak), at which point they become strictly decreasing.
Note - Atleast 3 integers are required to form a peak.

Example

For example, if the input array is: -1 3 2 4 5 6 10 -5 9 1, it has the following peaks -
- -1 3 2
- 2 4 5 6 10 -5
- -5 9 1

Logic

The idea is to identify all peaks and display the length of the longest one among them. Atleast 3 elements are required to form a peak, so we just need to compare the current element with its two neighbours. If it is greater than its neighbours, a peak is formed and now we need to find the length of this peak. To find the length of the peak -

Let our peak element be at index i. We take two pointers and initialize them to its neighbours as left = i-1 and right = i+1. Then we keep decrementing left till the value at index left is less than the value at index left+1. Similarly, we keep incrementing right till the value at index right is less than the value at index right-1. After finding the two extreme bounds, we can find the length of the peak as right-left+1 (number of elements between left and right). For the next iteration, we can start checking for peaks directly from index right as all elements between i and right are in decreasing order. So no peak will exist in that range.

Finally, we display the length of the longest peak we get, at the end of the array.

Code

```
#include <bits/stdc++.h>
using namespace std;
int main() {
    int t;
    cin>>t;
    while(t--){
    int n;
    cin>>n;
    int arr[n];
    for(int i=0;i<n;i++)
        cin>>arr[i];
    int longestPeak = 0;
    int i=1;
    while(i < n-1){
        bool isPeak = arr[i-1]<arr[i] && arr[i] > arr[i+1];
        if(!isPeak){
            i+=1;
            continue;
        }
        int leftIdx = i-2;
        while(leftIdx >=0 && arr[leftIdx] < arr[leftIdx+1])
            leftIdx -=1;
        int rightIdx = i+2;
        while(rightIdx < n && arr[rightIdx]< arr[rightIdx-1])
            rightIdx +=1;
        int currentPeak = rightIdx- leftIdx-1;
        longestPeak = max(longestPeak,currentPeak);
        i = rightIdx;
    }
    cout<<longestPeak<<"\n";
    }
    return 0;
}
```

28. Longest Unique

Problem Statement (Simplified)

A problem statement without any story. Given a string S, find the length of the longest substring with all unique characters.

Example

S = prepbytes

The longest substring with unique characters are 'pbytes' and 'repbyt' both have length=6.

Logic

As we saw above, multiple loops take a longer time to solve the problem. So, we use Window Sliding Method to solve this problem.

Where a window contains substring of the given string having all unique characters.

We maintain an array that saves the last position of a character in the string, initially all characters are set to lie at -1 index. We update this array, as we move further in the string.

Two pointers of window i.e start and end are initialised at 0th index.

We move end pointer to the next character in each iteration.

We also move start pointer to the last position of character at end pointer if the last position of character at end pointer lies in the current window. Hence the window will contain unique characters again. We store the window length if it is greater than the current maximum length of the window.

After end pointer crosses the string, we print the maximum length achieved during traversal.

Code

```
C++
#include <bits/stdc++.h>
using namespace std;
#define NO_OF_CHARS 256

int longestUniqueSubsttr(string str)
{
    int n = str.size();
    int cur_len = 1;
    int max_len = 1;
    int prev_index;

    int* visited = new int[sizeof(int) * NO_OF_CHARS];
    for (int i = 0; i < NO_OF_CHARS; i++)
        visited[i] = -1;

    visited[str[0]] = 0;

    for (int i = 1; i < n; i++) {
        prev_index = visited[str[i]];

        if (prev_index == -1 || i - cur_len > prev_index)
            cur_len++;

        else {
            if (cur_len > max_len)
                max_len = cur_len;

            cur_len = i - prev_index;
        }
        visited[str[i]] = i;
    }


    if (cur_len > max_len)
        max_len = cur_len;

    free(visited);
    return max_len;
}



int main()
{
    int test_case;
    cin>>test_case;
    for(int i=0;i<test_case;i++){
    string str;
    cin>>str;
    int len = longestUniqueSubsttr(str);
    cout<<len<<endl;
    }

    return 0;
}
```

29. BLK Numbers

Problem Statement

Playing with numbers is always fun! In this problem, we are given a number and we are asked to check if it is a special kind of number or not. If n be our number and if it cannot be represented in a form x + countSetBits(x). Here, countSetBits () is a function that returns the number of set bits of argument number.

Example

N = 3, we can write 3 = 2 + countSetBits (2), hence 3 is not a BLK number.

N = 4, we cannot represent in our form, hence it is a BLK number.

Logic

We traverse from 1 to less than that number n and check for each x if it can be used to form our number n. This is supposed to work in linear time. Can we reduce this time complexity? Here comes the fun part. Trick -Idea is we do not need to search for all the elements from 1 to n-1, rather the largest count of set bits in any number smaller than n cannot exceed ceiling of Log2n. Hence, we check for numbers in range n – ceilingLog2(n).

Code

```
#include <bits/stdc++.h>
using namespace std;
int countSetBits(int n){
  int count = 0;
  while(n){
    if(n&1)count++;
    n >>= 1;
  }
  return count;
}
int ceillog(int n)  //calculating ceil log base 2
{
```

```
    int count = 0;
    n--;
    while (n > 0) {
        n >>= 1;
        count++;
    }
    return count;
}
bool blknumber(int n){
  for(int i = n - ceillog(n);i<n;i++){
    if(i + countSetBits(i) == n)
      return false;
  }
  return true;
}
int main()
{
//write your code here
  int n;
  cin>>n;
  if(blknumber(n))cout<<"Yes\n";
  else cout<<"No\n";
  return 0;
}
```

30. Water and Jug

Problem Statement

You are at the side of a river. You are given two jugs of M and N liters. Both the jugs are initially empty. The jugs don't have markings. You need to determine the minimum number of operations to be performed to obtain D liters of water in one of the jug.

The operations you can perform are:

a.) Empty a Jug

b.) Fill a Jug

c.) Pour water from one jug to the other until one of the jugs is either empty or full.

Example

By these operations, we have to measure quantity d from these two jugs. Suppose we have n=3 and m=5 with d=4 then we know the answer is 6 as follows –

J1,J2=(0,0)→(0,5)→(3,2)→(0,2)→(2,0)→(2,5)→(3,4)

Logic

The idea comes from a very popular mathematical equation – Diophantine Equation which states as we can have solutions for the equation: nx+my=z, only when gcd(n,m) divides z. We just need to come up with a certain value of x, y for the current z value. In this problem also we face a similar situation where for n, m we have to find whether it is possible to get d or not.

Code

```
#include <bits/stdc++.h>
using namespace std;
int gcd(int a,int b){
  if(b == 0)return a;
  return gcd(b,a%b);
}
int pourstep(int maxfrom,int maxto,int d){
  int step = 1;
  int from = maxfrom;
  int to = 0;
  while(from != d && to != d){
    int maxfill = min(from,maxto - to);
    from -= maxfill;
    to += maxfill;
    step++;
    if(from == d || to == d)return step;
    if(from == 0){
      from = maxfrom;
      step++;
    }
if(to == maxto){
      to = 0;
      step++;
    }
  }
  return step;
}
int solve(int n,int m,int d){
  if(n > m)swap(n,m);
  if(d > m)return -1;
  if(d%gcd(n,m) != 0)return -1;
  return min(pourstep(n,m,d),pourstep(m,n,d));
}
int main()
{
  //write your code here
  int t;
  cin>>t;
  while(t--){
    int n,m,d;
    cin>>n>>m>>d;
    cout<<solve(n,m,d)<<endl;
  }
  return 0;
}
```

31. Finding Path

Problem Statement

In this problem, we are given a 2D matrix and some conditions of the elements present in it. The matrix may contain

0 - means obstacle

1 - means source

2 - means destination

3 - means empty cell

We can travel the matrix in 4 directions top, right, left, bottom. If we get a 0 it means we cannot travel that cell. We can only travel through empty cells. Now it is asked to find whether we can find a path from source 1 to destination 2 in the given matrix.

We can travel in

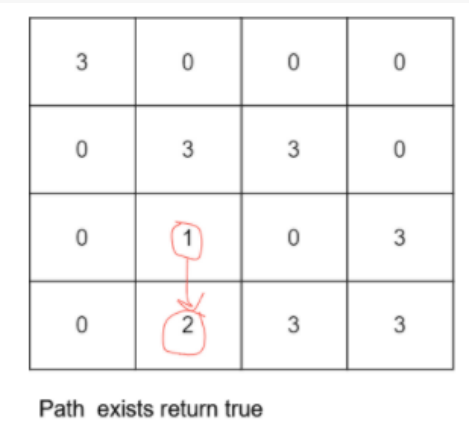
(i,j)→(i,j+1) right

(i,j)→(i,j-1) left

(i,j)→(i+1,j) down

(i,j)→(i-1,j) up

Example



Logic

The idea is to solve this recursively by visiting each cell and if it is a valid cell, we mark it as visited. In this way, if we can reach a cell that contains 2 from 1 then we can return true for this problem.

We keep track of visited cells using a visited matrix.

Create a valid function that checks whether the current cell is valid not.

Recursively visit the cells till we find 2(destination cell).

Code

```
CPP
#include <bits/stdc++.h>
using namespace std;
bool isValid(int **mat,int i,int j,int n){
  if(i<0 || i>=n || j<0 || j>=n || mat[i][j] == 0)
    return false;
return true;
}
bool hasPath(int **mat,int i,int j,int **visited,int n){
 if(isValid(mat,i,j,n) && !visited[i][j]){
     visited[i][j] = 1;
     if(mat[i][j] == 2)return true;
     bool top = hasPath(mat,i-1,j,visited,n);
     if(top)return true;
     bool bottom =hasPath(mat,i+1,j,visited,n);
     if(bottom)return true;
     bool left =hasPath(mat,i,j-1,visited,n);
     if(left)return true;
     bool right =hasPath(mat,i,j+1,visited,n);
     if(right)return true;
 }
    return false;
}
void checkPathPresence(int **mat,int n){
 bool ans = false;
 int **visited = new int*[n];
 for(int i = 0;i<n;i++)
   {
       visited[i] = new int[n];
       for(int j = 0;j<n;j++)
       visited[i][j] = 0;
   }
for(int i = 0;i<n;i++)
    for(int j = 0;j<n;j++){
        if(mat[i][j] == 1 && !visited[i][j]){
            if(hasPath(mat,i,j,visited,n))
                {
                    ans = true;
                    break;
                }
        }
    }

   if(ans)cout<<"Yes";
   else cout<<"No";
```

```
}
int main()
{
  //write your code here
  int n;
  cin>>n;
  int **mat = new int*[n];
  for(int i = 0;i<n;i++){
    mat[i] = new int[n];
    for(int j = 0;j<n;j++)
      cin>>mat[i][j];
  }
  checkPathPresence(mat,n);

  return 0;
}
```

32. Tasty Triplets

Problem Statement

In this problem we are given an array and we are asked to find the number of triplets count (A[i], A[j], A[k]) such that A[i]> A[i]> A[i] for every i<j<k. This is a modification of a very popular problem called Inversion Count.

Example

Suppose we have an array –
A = 10 8 3 1
Then the count for such triplets is 4 – 1,3,8 1,3,10 1,8,6 3,8,10

Logic

We will be using Binary Indexed Tree in this approach. The idea is to have two arrays –
1.) Left_greater array: which will contain the count of all the numbers which are greater than ith element and to the left of arr[i] i.e., from 0 to i-1th index.
2.) Right_smaller array: which will contain the count of all the numbers which are smaller than ith element and to the right of arr[i] i.e., from i+1 to n-1th index.
3.) So for every index, if we can get this we just need to multiply both for our resul tant count.

Code

```
C++
#include <trees/stdc++.h>
using namespace std;
void updtree(int TREE[], int n, int i, int val)
{
    for (; i <= n; i += (i & -i)) {
        TREE[i] += val;
    }
}
int sumtree(int TREE[], int i)
{
    int sum = 0;
    for (; i > 0; i -= (i & -i)) {
        sum += TREE[i];
    }
    return sum;
}

void Change(int *arr, int n)
{
    int temp[n];
    for (int i = 0; i < n; i++) {
        temp[i] = arr[i];
    }
    sort(temp, temp + n);

    for (int i = 0; i < n; i++) {
    arr[i] = lower_bound(temp, temp + n, arr[i]) - temp + 1;
    }
}

int getCount(int *arr, int n)
{

    Change(arr, n);

    int TREE[n + 1] = { 0 };
    int right_smaller[n + 1] = { 0 };
    int left_greater[n + 1] = { 0 };
    for (int i = n - 1; i >= 0; i--) {
        right_smaller[i] = sumtree(TREE, arr[i]-1);
        updtree(TREE, n, arr[i], 1);
    }

    for (int i = 0; i <= n; i++) {
        TREE[i] = 0;
    }
```

```
    for (int i = 0; i < n; i++) {
        left_greater[i] = i - sumtree(TREE, arr[i]);
        updtree(TREE, n, arr[i], 1);
    }


    int ans = 0;
    for (int i = 0; i < n; i++) {
        ans += left_greater[i] * right_smaller[i];
    }
    return ans;
}
int main()
{
  //write your code here
  int n;
  cin>>n;
  int *arr = new int[n];
  for(int i = 0;i<n;i++)
    cin>>arr[i];
  cout<<getCount(arr,n)<<endl;
  return 0;
}
```

33. Subarray of all One

Problem Statement

You are given a binary array of size N. i.e., the Binary array only contains 00 and 11. Can you find the length of the longest subarray consisting of all ones? But there is a cache, You have to remove exactly one element from the array after that print the length of the longest sub-array of all ones.

Example

Suppose we have arr=[1 1 0 1 1], then our longest subarray after removing the middle 0 is 4 which is our answer.

Logic

We need to find a subarray with one zero such that after its removal we can get a maximum length of ones. So basically, we can say that if we find such a subarray and change that zero to one then we will have a maximum length subarray of ones and hence our answer will be this max length – 1

Code

```
CPP
#include <bits/stdc++.h>
using namespace std;
int longestOnes(int *arr,int n){
  int cnt = 0;//zero count
  int max_len = 0;
  int l = 0;
  for(int i = 0;i<n;i++){
    if(arr[i] == 0)cnt++;
    while(cnt>1){
      if(arr[l] == 0)
        cnt--;
      l++;
    }

    max_len = max(max_len,i-l+1);
  }
  return max_len;
}
int main()
{
  //write your code here
  int n;
  cin>>n;
  int *arr = new int[n];
  for(int i = 0;i<n;i++)
    cin>>arr[i];
  cout<<longestOnes(arr,n)-1<<endl;
  return 0;
}
```

34. Single Number III

Problem Statement

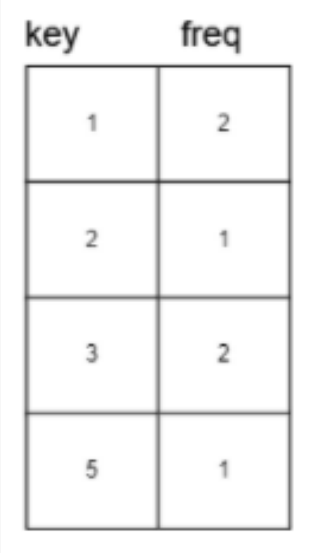
An array A of N integers is given where every element appears twice except for two elements, which appear exactly one time. Find those single numbers.

Example

Let's say we have our sample test case array arr=[1 2 1 3 3 5] then we have to return 2, 5.

Logic

We use a HashMap to store the frequencies of each element we encounter in the array. Again, we iterate through the map and find out the elements having a frequency of 1. We print these elements.



key	freq
1	2
2	1
3	2
5	1

The map will look like this, we can see the elements whose frequency is 1.

1.) Create a HashMap

2.) Iterate through all the elements in the array and keep increasing their frequencies.

3.) Iterate through the map if we find a key whose frequency is equal to 1 then we print it.

Code

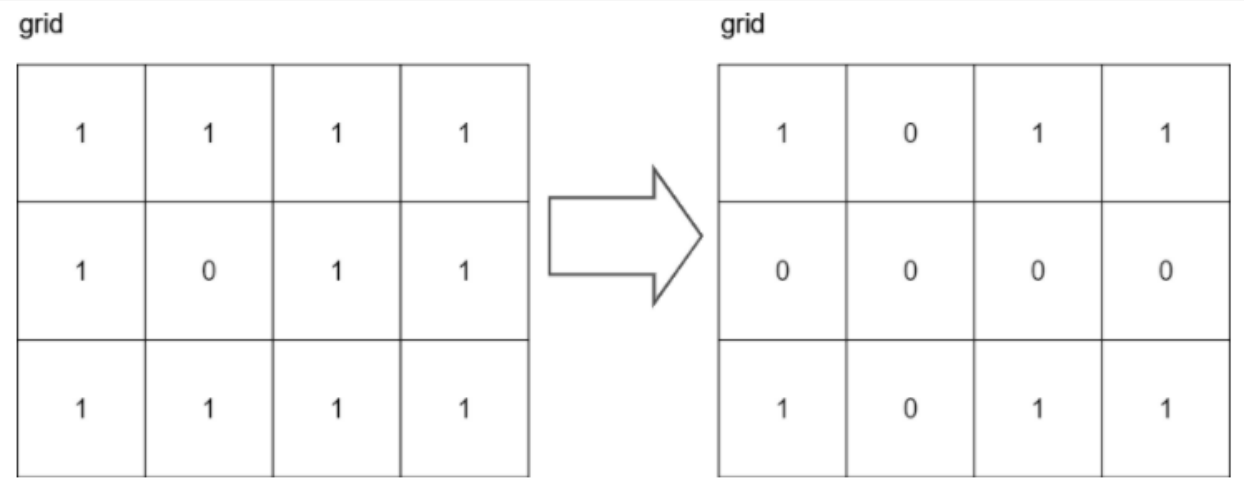
```
CPP
#include <bits/stdc++.h>
using namespace std;
void solve(int *arr,int n){
  map<int,int> m;
  for(int i = 0;i<n;i++)
      m[arr[i]]++;
  for(auto ele : m){
    if(ele.second == 1)
      cout<<ele.first<<" ";  }
  cout<<endl;}
int main()
{
  int n;
  cin>>n;
  int *arr = new int[n];
  for(int i = 0;i<n;i++)
    cin>>arr[i];
  solve(arr,n);
  return 0;
}
```

35. Signal

Problem Statement

Prepland town is getting towers installed for TV signals. The town can be represented as a matrix of rows and columns. The cells where towers are installed are represented as '0' and the rest are represented as '1'. The tower can generate signals in vertical and horizontal directions i.e the cells with common rows or columns with that of the tower will be getting signal. You have to mark those cells with '0' where the signal is available.

Example



Logic

The idea is we need to store the rows and columns where we get a zero to mark those columns and rows as zero. Well, we can use the first cells of each column and rows to store this information. If along a row if there is any zero then the whole row would become zero and the same goes for columns. So, we use this idea to remove our extra space.

We still have a small problem which is about the first cell i.e., grid0[0]. This because supposes there is a cell in the first column where we have a 0 this will essentially mean that the whole first column will become 0 but our previous method will make all cells in the first row also zero as they have the same value if coordinates. So, we just need another variable say col zero to store this information.

Code

```
CPP
#include <bits/stdc++.h>
using namespace std;
void signal(int **grid,int n,int m){
  bool colzero = false;
  for(int i = 0;i<n;i++){
    if(grid[i][0] == 0)
      colzero = true;
    for(int j = 1;j<m;j++){
      if(grid[i][j] == 0){
        grid[i][0] = 0;
        grid[0][j] = 0;
      }
```

```
    }
  }

  for(int i=1;i<n;i++)
    for(int j = 1;j<m;j++){
      if(grid[i][0] == 0 || grid[0][j] == 0)
        grid[i][j] = 0;
    }
  if(grid[0][0] == 0)
    for(int j = 0;j<m;j++)
      grid[0][j] =0;
  if(colzero){
    for(int i =0;i<n;i++)
      grid[i][0] = 0;
  }
}
int main()
{
  int n,m;
  cin>>n>>m;
  int **grid = new int*[n];
  for(int i= 0;i<n;i++)
    grid[i] = new int[m];
  for(int i = 0;i<n;i++)
    for(int j = 0;j<m;j++)
      cin>>grid[i][j];
  signal(grid,n,m);
  for(int i =0 ;i<n;i++)
    {
      for(int j = 0;j<m;j++)
        cout<<grid[i][j]<<" ";
      cout<<endl;
    }

  return 0;
}
```

36. Claudia and Number Break

Problem Statement

Claudia is having an integer N. She wants to break it into the sum of at least two positive integers and maximize the product of those integers. Help Claudia to find the maximum product for a given integer.

Example

n = 10, 4 + 3 + 3 = 10, such that 4*3*3 = 36

Logic

Let's think of a simple case, if n = 2, then the only possible way we can break it for the maximum product is 1,1. Again for n=3, we can break it in 2,1 and the product is 2. Now, if we need to break it a maximum number of times then we can decompose it into powers of 2 and 3. I am saying if n is the number then we can have a value x in which we can divide it into x such that the product will become xnx.

Now mathematically we need to maximize this product.

let p= xnx

We take the derivative of this p w.r.t x to find x for maximum p and make this derivative to 0.

So, 2<e<3 hence all numbers should be broken into 2 and 3s to get the maximum product.

Define the base cases for n=2 and n=3, as 1 and 2.

Otherwise, we will break the number for 3 remainders either when remainder –

1.) When rem = 0, we can decompose the number in a power of 2.

2.) When rem = 1, we can decompose the number in terms of 4*powers of 3.

3.) When rem = 2, we can decompose the number in terms of 2*powers of 3.

Code

```
CPP

#include <bits/stdc++.h>
using namespace std;
long long int power(long long int x, long long int p)
{
   long long int res = 1;
    while (p) {
        if (p & 1)
            res = res * x;
        x = x * x;
        p >>= 1;
    }
    return res;
}
long long int maxprod(long long int N)
```

```
{
    if (N == 2)
        return 1;

    if (N == 3)
        return 2;

    long long int prod;

    if(N%3 == 0){prod = power(3,N/3);}
    else if(N%3 == 1)prod = 2*2*power(3,(N/3)-1);
    else if(N%3 == 2)prod = 2*power(3,N/3);

    return prod;
}
int main()
{
  //write your code here
  int t;
  cin>>t;
  while(t--){
    long long int n;
    cin>>n;
    cout<<maxprod(n)<<endl;
  }
  return 0;
}
```

37. Minimum days to make juices

Problem Statement

In this problem, we are given an array that consists of the availability of squash on a particular day. Rather only on that day, a particular squash will be available. Now, we have to make X types of juices or simply X juices using K contiguous fruit squash. For this we have to return the minimum number of days we will require to wait to prepare X juices.

Example

Suppose arr[]=[1 1 10 2 2], X = 2, K=2

So, after 1 day our arr will seem like – [1 1 _ _ _], hence we can use these squashes to make a juice.

For that 2 - [1 1 _ 2 2], with 4th and 5th squash we can make another juice. Hence, we will require 2 days to make 2 types of juices.

Logic

The intuition is that we have a fixed range of answers which is from the minimum element in the array to the maximum element in the array. We can use a juice only once, hence if X*K>N then we won't be able to make the required number of juices. Otherwise, we can simply have a solution but yet not the minimum one.

So, when we have a problem like this where we have an answer in our range, we try to think of a binary search solution. We check for every value in range 1 to 1e9.

Code

C++

```
#include <bits/stdc++.h>
using namespace std;
int minDays(vector<int>& arr, int x, int k) {
        int n = arr.size(), left = 1, right = 1e9;
        if (x * k > n) return -1;
        while (left < right) {
            int mid = (left + right) / 2, j= 0, juices = 0;
            for (int j = 0; j < n; ++j) {
                if (arr[j] > mid) {
                   j= 0;
                } else if (++j >= k) {
                    juices++;
                   j= 0;
                }
            }
            if (juices < x) {
                left = mid + 1;
            } else {
                right = mid;
            }
        }
        return left;
    }

int main()
{
  int n,x,k;
  cin>>n>>x>>k;
  vector<int> arr(n);
  for(int i = 0;i<n;i++)
    cin>>arr[i];
  cout<<minDays(arr,x,k)<<endl;
  return 0;}
```

38. Max Difference

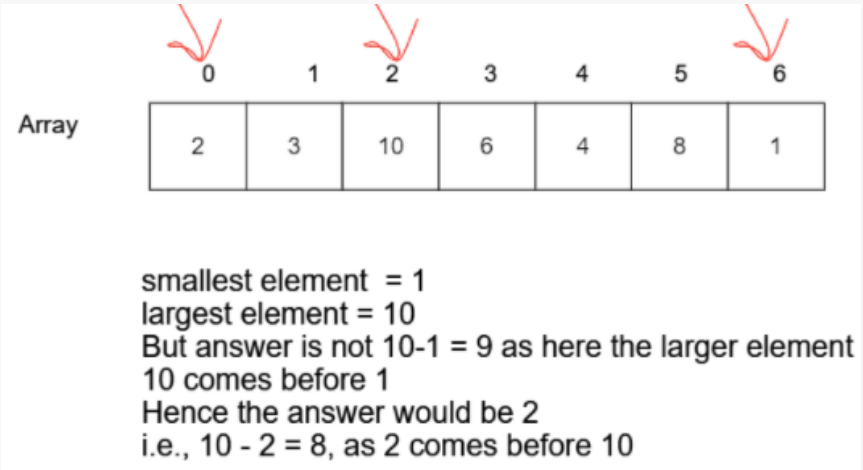
Problem Statement

Amar was given an array A with N distinct elements. Find out the maximum difference between any two elements such that if the index of the max element is i and the index of the minimum element is j then i>j.

In this problem, we are given an array and are asked to find the maximum difference between any two elements of the array. It's a pretty straightforward question, where we need to find the maximum element and then the minimum element but with a condition – the larger element must occur after the smaller element.

Example

N=[2 3 10 6 4 8 1] with size = 7



Logic

In this approach we are just keeping track of the smallest element that occurred till now and then taking the difference with our current element, if we get a current element that is smaller than the minimum element, we update our smaller element to the current element. This will require just one pass of the array and hence the time complexity reduces to linear terms.

Code

```
#include <bits/stdc++.h>
using namespace std;
int efficientmaxdiff(int *arr,int n){
    int ans = INT_MIN;
    int min_element = arr[0];
    for(int i = 1;i<n;i++){
        ans = max(ans,arr[i] - min_element);
        if(min_element > arr[i])
            min_element = arr[i];
    }
    //cout<<min_element<<endl;
    return ans;
}
int main(){
    int n;
    cin>>n;
    int *arr = new int[n];
    for(int i= 0;i<n;i++)
        cin>>arr[i];
    cout<<efficientmaxdiff(arr,n)<<endl;
    return 0;}
```

39. CHOCOLATE DILEMMA

Problem Statement

Arnab can take one candy from each row of candies but with the restrictions imposed:

1.) There are 3 sections in each row;

2.) If he takes candy from jth section in the ith row,then in (i+1)th row ,he can't choose candy from the same section i.e. jth section.

Example

6 6
1 3 5 2 4 6
6 4 5 1 3 2
1 3 5 2 4 6
6 4 5 1 3 2
6 4 5 1 3 2
1 3 5 2 4 6

Arnab selects sweets
row 1: 6
row 2: 6
row 3: 6
row 4: 6
row 5: 5
row 6: 6
sum =35

Logic

This question has a simple and elegant dynamic programming approach in data structures and algorithms.

Let's define a funtion sweetness_r(i,s) , where i denotes the row which we are currently on and s denotes the section. It is clearly mentioned in the question that we have only 3 sections.

Therefore,

sweetness_r(i,0)=a[i][0]+max(sweetness_r(i,1),sweetness_r(i,2));

sweetness_r(i,1)=a[i][1]+max(sweetness_r(i,0),sweetness_r(i,2));

sweetness_r(i,2)=a[i][0]+max(sweetness_r(i,1),sweetness_r(i,0));

Then print the maximum of sweetness_r(n,0),sweetness_r(n,1) and sweetness_r(n,2) ,where n is the number of rows.

Code

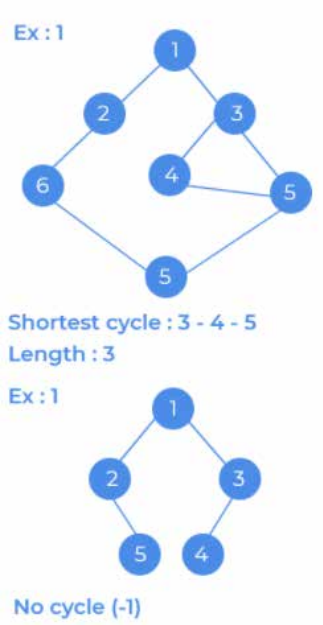
```
C++
#include <bits/stdc++.h>
     using namespace std;

    int main()
    {
    //write your code here
    int n,m;
    cin>>n>>m;
    int a[n][3];
    for(int i=0;i<n;i++)
    {
    for(int j=0;j<3;j++)
    {int mx=0;
      for(int c=0;c<m/3;c++)
      {int x;
        cin>>x;
        mx=max(mx,x);
      }
      a[i][j]=mx;    }
    }
     int dp[n][3];
    dp[0][0]=a[0][0],dp[0][1]=a[0][1],dp[0][2]=a[0][2];
     for(int i=1;i<n;i++)
    {
    for(int j=0;j<3;j++)
    {
      if(j==0)
      dp[i][j]=a[i][j]+max(dp[i-1][1],dp[i-1][2]);
      if(j==1)
      dp[i][j]=a[i][j]+max(dp[i-1][2],dp[i-1][0]);
      if(j==2)
      dp[i][j]=a[i][j]+max(dp[i-1][0],dp[i-1][1]);

    }
    }
     cout<<max(dp[n-1][0],max(dp[n-1][1],dp[n-1][2]));
    return 0;
    }
```

40. Shortest cycle

Problem Statement

Given a graph we have to find the length of the shortest cycle in the given graph. If no cycle exists print −1.

Example



Logic

Create a queue and push the current vertex now perform following operations untill the queue is not empty:

Each time we pop a vertex v from queue, (v = queue.front()), we will mark the vertex v as visited (visited[v]= true).

Iterate for all the adjacent vertices of v and for every adjacent vertex a, do following :

update the parent and distance as, parent[a] = v and distance[a]= distance[v]+1.

push a into the queue.

if a is not visited,

and if parent[v] != a.

update the ans (ans = min(ans,current cycle length)).

Code

```
CPP
#include <bits/stdc++.h>
    using namespace std;
    #define N 10000

    vector<int> gr[N];

    // Function to add edge
    void Add_edge(int x, int y)
    {
        gr[x].push_back(y);
        gr[y].push_back(x);
    }

    // Function to find the length of
    // the shortest cycle in the graph
    int shortest_cycle(int n)
    {
        // To store length of the shortest cycle
        int ans = INT_MAX;

        // For all vertices
        for (int i = 0; i < n; i++) {

            // Make distance maximum
            vector<int> dist(n, INT_MAX);

            // Take a imaginary parent
            vector<int> par(n, -1);

            // Distance of source to source is 0
            dist[i] = 0;
            queue<int> q;

            // Push the source element
            q.push(i);

            // Continue until queue is not empty
            while (!q.empty()) {

                // Take the first element
                int x = q.front();
                q.pop();

                // Traverse for all it's childs
                for (int child : gr[x]) {
```

```
                // If it is not visited yet
                if (dist[child] == INT_MAX) {

                    // Increase distance by 1
                    dist[child] = 1 + dist[x];

                    // Change parent
                    par[child] = x;

                    // Push into the queue
                    q.push(child);
        }

                // If it is already visited
                else if (par[x] != child)
                    ans = min(ans, dist[x] + dist[child] + 1);
            }
        }
    }

    // If graph contains no cycle
    if (ans == INT_MAX)
        return -1;

    // If graph contains cycle
    else
        return ans;
}

int main()
{
int t;
cin>>t;
while(t--)
{

    int n ,e;
    cin>>n>>e;
    for(int i=0;i<e;i++)
    gr[i].clear();

    while(e--)
    {

int u,v;
cin>>u>>v;
    Add_edge(u,v);
    }
    cout <<shortest_cycle(n)<<endl;

}
    return 0;
}
```

41. Clan War

Problem Statement

There are two clans numbered sequentially from 1 to N and given two integers, u and v denoting the clan numbers that are fighting each other. We assume that the clans of one nation do not fight each other, they only fight with the clans of the enemy nation. Check whether this assumption is correct or not.

Example

Logic

Given a graph represented as an adjacency-list (i.e. a list of edges), you can determine if it's bipartite as follows:

i.) Initialize a disjoint-set data structure SETS, with a singleton set for each vertex.
(If there is an even-length path between two vertices, then we will ultimately unify those two vertices into the same set, unless we return FALSE first.)

ii.) Initialize a parent[ ] for each vertex to NIL. (As we examine edges, we will update parent[ ] for each vertex to one of its neighbors.)

iii.) For each edge {u, v}: >* If u and v belong to the same parent in SETS, then return FALSE.

iv.) If parent[u] = NIL, set parent[u] := v.
Otherwise, update SETS to unify v with parent[u].

v.) If parent[v] = NIL, set parent[v] := u.
Otherwise, update SETS to unify u with parent[v].

vi.) Return TRUE.

Code

C++

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

    int BFS(vector<int> graph[],int n)
    {

        bool visited[n+1];
        int colour[n+1];
        int node,flag=0;

        memset(visited,0,sizeof(visited));
        memset(colour,-1,sizeof(colour));

        for(int k=1; k<=n ;++k)
        {
        if(!visited[k])
        {
            queue<int> q;
            q.push(k);
            colour[k] = 1;
            while(!q.empty())
            {
            node =     q.front();
            q.pop();
            visited[node] = true;
            for(int i=0; i<graph[node].size(); ++i)
            {
                if(colour[graph[node][i]] == -1)
                colour[graph[node][i]] = !colour[node];

                else if    (colour[graph[node][i]] == colour[node])
                {
                flag = 1;
                break;
                }

                if(!visited[graph[node][i]])
                q.push(graph[node][i]);
            }

            if(flag)
                break;
```

```
            }
            }
            if(flag)
            break;
        }
        return flag;
    }
    int main() {

        ios::sync_with_stdio(false);
        int t,n,m,u,v;

        cin>>t;
        while(t--)
        {

            cin>>n>>m;
            vector<int> graph[n+1];
            for(int i=0; i<m; ++i)
            {
                cin>>u>>v;
                graph[u].push_back(v);
                graph[v].push_back(u);
            }

            if(BFS(graph,n))
                cout<<"No"<<endl;
            else
                cout<<"Yes"<<endl;
        }
        return 0;
    }
```

42. Missing in AP

Problem Statement

Given an array A, such that it is arranged in an Arithmetic Progression, but one element from this A.P. is missing, find it.

Example

Input : A = [3, 6, 12, 15, 18]

Output : 9

Explanation : In the given A.P. series starting with initial element 3 and Common Difference 3, 9 is the element that is missing between 6 and 12.

Logic

check whether the missing element lies near one of the corner points by taking out difference of first and second elements and difference of last and second last elements and comparing these differences with each other. If they are not equal, this implies that the missing number lies near corner points and we can easily compute the missing number.
Else we will follow below approach –
1.) The idea is to use Binary Search.
2.) We initialize low and high as 0 and n-1, and calculate mid as mid = low + (high - low)/2
3.) If the element present at mid matches the calculated value at mid, this implies that elements at the lower half are all positioned correctly, so we search in the right half.
4. else we store the calculated value and search in the lower half till low 6. In this way, we will have the last value that was not at its right position stored with us which will be our result.

Code

C++

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

int MissingAP(int *arr, int low, int high, int diff, int missing_ele){
  if(low <= high){
    int mid = low + (high - low)/2;

    /* if curr element is at its right place in AP
    we will search in the lower half then */
    if(arr[mid] == arr[0] + mid*diff)
      return MissingAP(arr, mid+1, high, diff, missing_ele);

    /* else save the value that should be here and further
    search in the lower half */
    else{
      missing_ele = arr[0] + mid*diff;
      return MissingAP(arr, low, mid - 1, diff, missing_ele);
    }
  }
  //finally return the ele that was not found on its place
  return missing_ele;
}

int main()
{
```

```
  int t; cin>>t;
  while(t--){
    int n; cin>>n;
    int arr[n];
    for(int i=0; i<n; i++)
      cin>>arr[i];

    //check if missing element lies at one of the ends or close to one of the ends
    int left_diff = arr[1] - arr[0];
    int right_diff = arr[n-1] - arr[n-2];
    int diff, missing_ele;

    //if left == right, missing element lies inside of the array
    if(left_diff == right_diff){
      diff = left_diff;
      //we will go on checking inside the array
      cout<<MissingAP(arr, 0, n-1, diff, -1)<<"\n";
      continue;
    }

    //else missing ele is at one of the corners or near corner
    if(left_diff < right_diff){
      missing_ele = arr[n-1] - left_diff;
      cout<<missing_ele<<"\n";
    }
    else{
      missing_ele = arr[0] + right_diff;
      cout<<missing_ele<<"\n";
    }
  }
  return 0;
}
```

43. Find Mountain Top

Problem Statement

Given an array A of N integers, such that till a point these integers are strictly increasing and after that strictly decreasing, find the element present at that point.

Example

Input : A = [10, 12, 14, 15, 8, 7, 6]
Output : 15
Explanation : 15 is the top point. As before 15 all elements are strictly increasing and after 15 all elements are strictly decreasing.

Logic

1.) The idea is to use Binary Search.
2.) Initialize start and end as starting and ending indexes of the array.
3.) Calculate, mid = start + (end - start) / 2.
4.) For every such mid calculated check if the element at mid is greater than both of its left and right element. If Yes return this element.
5.) Else if element at mid is only greater than left element, then go on searching in the right half.
6.) Else, go on searching in the left half till the element is found.

Code

C++

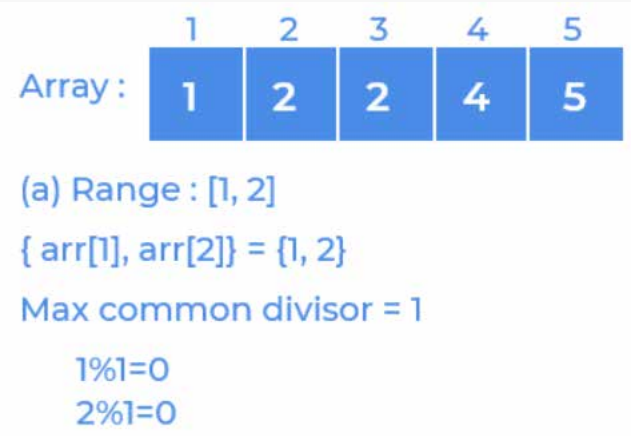
```
#include <bits/stdc++.h>
using namespace std;
int MountainTop(int *arr, int start, int end){
  while(start <= end){
    int mid = start + (end - start) / 2;

    /*if mid index element is greater than both of
    its adjacent elements, this is our top element*/
    if(arr[mid] > arr[mid - 1] && arr[mid] > arr[mid + 1])
      return arr[mid];

    /* if mid indexed element is only greater than left element
      this implies top is in the right half */
    else if(arr[mid] > arr[mid - 1])
      start = mid + 1;

    /* if mid indexed element is only greater than right element
      this implies top is in the left half */
    else
      end = mid - 1;
  }
}


int main()
{
  int t; cin>>t;
  while(t--){
    int n; cin>>n;
    int arr[n];
    for(int i=0; i<n; i++)
      cin>>arr[i];
    cout<<MountainTop(arr, 0, n-1)<<"\n";
  }
  return 0;}
```

44. Maximum Divisor

Problem Statement

Given an array of N elements and Q queries. In each query, two values is given l and r. We have to find a maximum positive integer x where arr[l]%x = arr[l+1]%x =...=arr[r]%x = 0

Example



Logic

As the number of queries and array size is too large for linear search in every query, we will use segment tree to solve this problem.
A Segment Tree is a data structure which allows answering range queries very effectively over a large input. Each query takes logarithmic time. Range queries includes sum over a range, or finding a minimum value over a given range etc. Query be of any type we can use segment trees and modify it accordingly.
Leaf nodes of the tree stores the actual array values and intermediate nodes stores the information of subarrays with is require to solve the problem. Lets say if we have to find a sum between different ranges, so now the intermediate nodes will store the sum of the current subarray. We fill the nodes by recursively calling left and right subtree (dividing into segements), untill there is a single element left, which can be directly assigned the value of the array.

Array representation of the tree is used to represent segment tree, where (i*2)+1 represents the left node and (i*2)+2 represents right node, parent will be represented by (i-1)/2 for every index i.

We will construct our tree by starting at the original array and dividing it into two halves (left and right), untill there is a single element left (leaf) which can directly be filled with a[i] for any index i. Now for every range say l to r, we will store the gcd of the current range in the node.
Now that our tree is constructed, we will answer queries (gcd of the given range). The queries can be of 3 types:

1.) The range of the tree exactly matches with the query, in this case we will return the value stored in this node.
2.) The range either belongs to the left or right node, in this case we will make two recursive calls for left and right subtrees respectively.
3.) The range overlaps with two of more ranges, in this case we are forced to go to the lower levels of both the subtrees and find the gcd of the range which fits the current range and finally return the gcd of the values returned by both sub trees.

Code

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


int *st;

int gcd(int a, int b)
{
    if (b == 0)
        return a;
    return gcd(b, a % b);
}


int findGcd(int ss, int se, int qs, int qe, int si)
{
    if (ss>qe || se < qs)
        return 0;
     if (qs<=ss && qe>=se)
        return st[si];
    int mid = ss+(se-ss)/2;
    return gcd(findGcd(ss, mid, qs, qe, si*2+1),findGcd(mid+1, se, qs, qe, si*2+2));
}
int findRangeGcd(int ss, int se, int arr[],int n)
{
    if (ss<0 || se > n-1 || ss>se)
    {
```

```
        return -1;
    }
    return findGcd(0, n-1, ss, se, 0);
}


int constructST(int arr[], int ss, int se, int si)
{
    if (ss==se)
    {
        st[si] = arr[ss];
        return st[si];
    }
    int mid = ss+(se-ss)/2;
    st[si] = gcd(constructST(arr, ss, mid, si*2+1),constructST(arr, mid+1, se,
si*2+2));
    return st[si];
}


int *constructSegmentTree(int arr[], int n)
{
int height = (int)(ceil(log2(n)));
int size = 2*(int)pow(2, height)-1;
st = new int[size];
constructST(arr, 0, n-1, 0);
return st;
}

int main()
{
  int t;
  cin>>t;
  while(t--)
  {
      int n;
      cin>>n;
      int arr[n];
      for(int i=0;i<n;i++)
       cin>>arr[i];

      constructSegmentTree(arr, n);

      int q;
      cin>>q;
      while(q--)
      {
```

```
        int l,r;
        cin>>l>>r;
         l-=1;
         r -=1;
        cout << findRangeGcd(l, r, arr, n) << "\n";
      }
  }

    return 0;
}
```

45. Magical Ropes

Problem Statement

Given an array H of lengths of Magical Ropes and array R of the rate by which rope increases daily, print the minimum number of days required to collect ropes that sum up to length X

Restrictions:

1.) You cannot cut the rope, you have to take complete rope for collecting it.
2.) You cannot take a rope with length less than K.

Example

Input : N = 4, X = 100, K = 45
Heights[] = [2, 5, 2, 6]
Rate[] = [10, 13, 15, 12]
Output : 4

Explanation :

Day 0 = [2, 5, 2, 6]

Day 1 = [2 + 10, 5 + 13, 2 + 15, 6 + 12] = [12, 18, 17, 18]

Day 2 = [12 + 10, 18 + 13, 17 + 15, 18 + 12] = [22, 31, 32, 30]

Day 3 = [22 + 10, 31 + 13, 32 + 15, 30 + 12] = [32, 44, 47, 42]

Day 4 = [32 + 10, 44 + 13, 47 + 15, 42 + 12] = [42, 57, 62, 54]

Ans : Day 4 as (57, 62, 54 are all greater than 45 and sum up to value that is greater than equal to 100)

Logic

Lenght at ith day = ith day * Rate of Growth + Initial length

- The idea is to go through the best online programming courses and perform a Binary Search such that whenever we find a day in which ropes length sum up to X, we will save it and search for a even lower value (as Minimum days are needed).
- initialize low as 0 and high as 10^{18}, by this we understand that minimum days can be as low as value of low and as high as value of high.
- Calculate mid = (low + high)/2, say mid is our desired day.
- Now, Linearly traverse and check for all ropes if their length at mid day is greater than or equal to K, if Yes add its value to a variable Sum.
- After traversal if Sum becomes greater or equal to X, simply save it, and check for a even lower value in the left half i.e. (low - (mid - 1)).
- Else, search in the right half i.e. ((mid + 1) - high).

Code

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

int main()
{
  ll N, X, K; cin>>N>>X>>K;
  ll height[N], rate[N];

  //input heights of ropes
  for(ll i=0; i<N; i++)
    cin>>height[i];

  //input rate of growth of ropes
  for(ll i=0; i<N; i++)
    cin>>rate[i];

  //initialize minimum and maximum days that can be possible as low and high
  ll low = 0;
  ll high = 1000000000000000000;
  ll min_days = 0;

  //Check for a mid value if it is satisfying the conditions
  while(low <= high){
    ll mid = (low + high)/2;
    ll sum = 0, flag = 0;
    for(ll i=0 ;i<N; i++){
      if(K <= (mid*rate[i] + height[i])){
        sum += (mid*rate[i] + height[i]);
        if(sum >= X){
          flag = 1;
          break;
        }
      }
    }

    /* if mid is one of the possible day, save it and
    keep on checking for more lower values in the left half */
    if(flag == 1){
      min_days = mid;
      high = mid - 1;
    }
    /* if mid is not a possible day skip it and
      go on searching in the right half */
    else
      low = mid + 1;
  }
  cout<<min_days;
  return 0;
}
```

46. Large Small Difference

Problem Statement

In this problem, we are given an array of N integers and we have exactly 3 moves that we have to perform. In one move we can choose an element from the array and change its value to anything we want and then we have to return the minimum difference between the smallest and largest element in the array.

Example

arr[]=[1 5 6 13 14 15 16 17], then for this our answer is 4 which comes from changing smallest 3 elements to 13, 17-13 = 4.

Logic

The idea is to minimize the difference between the maximum and minimum elements. This can be done after sorting; we can change 3 elements such that our minimum and maximum elements come closer. So, let suppose we have 4 elements then, obviously we can pick all the 3 elements and change them to the maximum or minimum, and hence their difference will always be 0.
If we can do 0 move, return (max(A) - min(A))
If we can do 1 move, return
min(the second max(A) - min(A), the max(A) - second min(A)).

Algorithm

So basically, we will have 3 options for n>4

- Change 3 biggest elements
- Change 2 biggest + 1 smallest element
- Change 1 biggest + 2 smallest element
- Change 3 smallest elements

Take minimum of these 4 cases.

Code

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

#define ll long long
const ll MOD = 1e9+7;

int minDifference(vector<int>& A, int n) {
    if (n < 5){
        return 0;
    }
    partial_sort(A.begin(), A.begin() + 4, A.end());
    nth_element(A.begin() + 4, A.end() - 4, A.end());
    sort(A.end() - 4, A.end());
    return min({A[n - 1] - A[3], A[n - 2] - A[2], A[n - 3] - A[1], A[n - 4] -
A[0]});
}


int main(){
    int n;
    cin>>n;
    vector<int> A(n);
    for(int i = 0; i<n;i++){
        cin>>A[i];
    }
    int ans = minDifference(A, n);
    cout<<ans<<endl;
    return 0;
}
```

47. Largest Integer with digit cost K

Problem Statement

In this problem, we are given the cost of each integer from 1 to 9. We are also provided with an integer k and are asked to form the largest possible number that can be formed for which the cost is equal to k. This means, for each digit that we place in our array/string will incur the cost provided corresponding to the cost array.

Let's make this clearer, suppose we have cost=[9 1 4 6 7 8 2 4 1] and k=8, the maximum possible number with cost equal to k is "99999999" as the cost of 9 is 1 so we can place eight 9s in our answer which will also be the maximum.

Logic

We will be using the concept of knapsack here with some modifications. Well, the best idea is to start with 1 and check for all possible values trying to maximize the answer. We also keep a visited 2D array depending on the state where the state has index and target k as the variables. For every state, we have two options, either we can include that digit or we can skip it. If we include that digit, we might include it multiple times, hence we start the next recursion in this case from the same digit only whereas for the case of skipping, we start from the next digit. We keep our answer in a 2D dp of strings. At the last, we compare the two strings obtained from the two options and return whichever is greater.

Algorithm

1.) Start the recursion function from digit 1 and recur down till 9.
2.) For each digit, we have two options either we keep on including that digit using recursion until the target becomes zero or we skip this digit.
   a.) solve(i+1, target), skipping the current i
   b.) solve(i, target-w[ind]) including the current i.
3.) If both these answers are zero then we return “0” as no such string/number can be formed.
4.) If both are not “0” then we have to take the maximum of these two.

Code

```
#include <bits/stdc++.h>
using namespace std;
#define ll long long
const ll MOD = 1e9+7;

string dp[12][5005];
int vis[12][5005];
int w[12];
string max_fun(string s1, string s2){
    if(s1.length()!=s2.length()){
        return s1.length()>s2.length()?s1:s2;
    }
```

```
    if(s1>s2){
        return s1;
    }
    else{
        return s2;
    }
}
string solve(int i, int target){
    if(target == 0){
        return "";
    }
    if(target<0 or i == 10){
        return "0";
    }
    if(vis[i][target] == 1){
        return dp[i][target];
    }
    vis[i][target] = 1;
    string ans1 = solve(i+1, target);
    string ans2 = solve(i, target-w[ind]);
    if(ans1 == "0" and ans2 == "0"){
        return dp[i][target] = "0";
    }
    if(ans1 == "0"){
        return dp[i][target] = (ans2 + to_string(i));
    }
    if(ans2 == "0"){
        return dp[i][target] = ans1;
    }
    dp[i][target] = max_fun(ans1, ans2 + to_string(i));
    return dp[i][target];
}
int main(){
    int K;
    cin>>K;
    for(int i=1;i<=9;i++){
        cin>>w[i];
    }
    memset(vis, 0, sizeof(vis));
    string ans = solve(1, K);
    cout<<ans<<endl;
    return 0;
}
```

48. Subarray with the target sum

Problem Statement

In this problem, we are given an array of integers and a value k. We are asked to find the minimum lengths of non-overlapping subarrays.

Example

Suppose we have arr=[3 2 5 4 1 5] and k = 5 then we can say our subarrays are [5] and [5] and their sum of lengths is 1+1 = 2.

Logic

We have to find a subarray and their sum should be k. We can use a prefix sum to make it more efficient. Now if we want to see if we have a subarray with sum K then for a particular prefixi-k is present in the prefix array or not. If it does then we have our subarray with such sum. From this, we can also find their respective index which will be stored in map. We need the map to store the prefix sum and corresponding index. Then we have to iterate over the array to find the minimum length subarray.

- Build the prefix sum array
- Create a HashMap to store the position of the prefix sums.
- Now we iterate through the array and find currsum-target in the map, if we get it this means we have a subarray before the current index whose sum is k and we can take this as one minimum.

Code

C++

```
#include <bits/stdc++.h>
using namespace std;
int minSumOfLengths(vector<int>& arr, int target) {
       int n = arr.size();
        unordered_map<int, int>hm;
        hm[0] = -1;
        int sum = 0;

        for (int i = 0; i < n; i++) {
            sum += arr[i];
            hm[sum] = i;
        }
        sum = 0;
        int msize = INT_MAX, res = INT_MAX;
        for (int i = 0; i < n; i++) {
            sum += arr[i];

            if (hm.find(sum-target) != hm.end())
                msize = min(msize, i-hm[sum-target]);

            if (hm.find(sum+target) != hm.end() && msize != INT_MAX)
                res = min(res, msize + hm[sum+target]-i);
        }
        return res == INT_MAX ? -1 : res;
    }

int main()
{
  int n,k;
  cin>>n>>k;
  vector<int> arr(n);
  for(int i = 0;i<n;i++)
    cin>>arr[i];
  cout<<minSumOfLengths(arr,k);
  return 0;
}
```

49. Fibonacci sum equal to S

Problem Statement

We all have come across the Fibonacci sequence, right? Do not worry if you haven't. It's just a series of numbers following the property – Fib1=1, Fib2=1, and Fibn=Fibn-1+Fibn-2 for all n>2. We can jot down this sequence as – 1, 1, 2, 3, 5, 8, 13... and so on. In this problem, we are given an Integer sum S and are asked to find out the minimum number of Fibonacci numbers when summed up to get a target S. We can use the same Fibonacci number multiple times.

Example

If we have
S=22
And all the Fibonacci numbers below this – 1, 1,2, 3,5, 8, 13, 21
Then S=1+21 or 8+13 hence minimum we require two elements – ans = 2

Logic

The idea is to find all the Fibonacci numbers below S and start iterating from the largest Fibonacci number just less than or equal to S. For each of the Fibonacci numbers, we keep on dividing S by the Fibonacci number until our S becomes zero. Each of these divisions is one step.

- Find all the Fibonacci numbers below S.
- Start Iterating from largest Fibonacci number and divide S by that number (greedily choosing the largest element)
- count += (S / fibnum[j]);
- S %= fibnumj;
- Return count

Code

```
C++
#include <bits/stdc++.h>
using namespace std;
long long nearestfibpos = 0;
vector<long long> fib(long long S)
{
  vector<long long> fibnum;
    int i = 3, next;
    fibnum.push_back(0);
    fibnum.push_back(1);
    fibnum.push_back(1);
    while (1) {
        next = fibnum[i - 1] + fibnum[i - 2];
        if (next > S)
            break;
        fibnum.push_back(next);
        i++;
    }
    return fibnum;
}
long long minSteps(long long S){
  long long count = 0;
  vector<long long> fibnum = fib(S);
   long long j = fibnum.size() - 1;
    while (S > 0) {
        count += (S / fibnum[j]);
        S %= (fibnum[j]);
        j--;
    }
    return count;
}
int main()
{
  //write your code here
  long long S;
  cin>>S;
  cout<<minSteps(S)<<endl;
  return 0;
}
```

50. Kth Element in Two Sorted Arrays

Problem Statement

Given two sorted arrays A and B, we need to find the kth element in the resultant array which will be formed when these two arrays will be merged

Example

A = [1,4,7]
B = [5,6,8,9]
K = 3
Res = [1,4,5,6,7,8,9]
3rd element is 5.

Logic

First of all, to observe here is the arrays are sorted, can we take advantage of this property. I think we can use a two-pointer technique here as after each iteration we have a definitive way to change our pointers.
We can have a variable say x to count the number of elements we have placed in the sorted array just we do not need to place them; we just need to move the pointers.

- When Ai<Bj, this will mean we have to place A[i] in our resultant array and increment the value of x by 1 and check if x==k, this means our current A[i] is out kth element. Also, increment i by 1.

- When Ai>Bj, this will mean we have to place B[j] in our resultant array and incre ment the value of x by 1 and check if x==k, this means our current B[j] is out kth element. Also, increment j by 1.

Code

```
C++
#include <bits/stdc++.h>
using namespace std;
int Kthelement(vector<int> &A,vector<int> &B,int k){
  int n = A.size();
  int m = B.size();
  int i = 0,j= 0,x = 0;
  while(i < n && j<m){
    if(A[i] < B[j]){
      x++;
      if(x == k)
        return A[i];
      i++;
    }else{
      x++;
      if(x == k)
        return B[j];
      j++;
    }
  }
  while(i<n){
      x++;
      if(x == k)
        return A[i];
      i++;
  }
  while(j<m){
    x++;
      if(x == k)
        return B[j];
      j++;
    }  return -1;  }

int main()
{
  //write your code here
  int t;
  cin>>t;
  while(t--){
    int n,m,k;
    cin>>n>>m>>k;
    vector<int> A(n);
    vector<int> B(m);
    for(int i = 0;i<n;i++)
      cin>>A[i];
    for(int i = 0;i<m;i++)
      cin>>B[i];
    cout<<Kthelement(A,B,k)<<endl;
  }

  return 0;
}
```