**Unique Questions According to Me**

1. Smallest Positive Integer that can not be represented as Sum

ANS. 1st approach (Find all the possible subsets and compare with the numbers from 1):

2nd approach ():

long long smallestpositive(vector<long long> array, int n)

{

sort(array.begin(),array.end());

long long macx=1;

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

{

if (array[i]<=macx)

macx+=array[i];

else

break;

}

return macx;

}

2. Print matrix in Anti-Spiral form

ANS. 1st approach (find the spiral order and then reverse the vector):

#include <bits/stdc++.h>

using namespace std;

vector<int> spiralOrder(vector<vector<int>>& matrix) {

int m=matrix.size();

int n=matrix[0].size();

int t=0,b=m-1,l=0,r=n-1;

vector<int> res;

while (l<=r && t<=b)

{

for (int i=l;i<=r;i++)

res.push\_back(matrix[t][i]);

t++;

if (t>b)

break;

for (int i=t;i<=b;i++)

res.push\_back(matrix[i][r]);

r--;

if (r<l)

break;

for (int i=r;i>=l;i--)

res.push\_back(matrix[b][i]);

b--;

if (b<t)

break;

for (int i=b;i>=t;i--)

res.push\_back(matrix[i][l]);

l++;

}

return res;

}

int main()

{

vector<vector<int>> mat

{

{1, 2, 3, 4, 5},

{6, 7, 8, 9, 10},

{11, 12, 13, 14, 15},

{16, 17, 18, 19, 20}

};

vector<int> res= spiralOrder(mat);

reverse(res.begin(),res.end());

for (int i:res)

cout<<i<<" ";

return 0;

}

3. My Calender-I

ANS. 1st approach (Brute force):

vector<pair<int,int>> v;

MyCalendar() {

}

bool book(int start, int end) {

if (v.size()==0)

v.push\_back({start,end});

else

{

for (auto it:v)

{

if (start<it.second && end>it.first)

return false;

}

v.push\_back({start,end});

}

return true;

}

2nd approach ():

multiset<pair<int,int>> m;

MyCalendar() {

}

bool book(int start, int end) {

m.insert({start,1});

m.insert({end,-1});

int s=0;

for (auto it:m)

{

s+=it.second;

if (s>1)

{

m.erase(m.find({start,1}));

m.erase(m.find({end,-1}));

return false;

}

}

return true;

}

4. Product of Array Except Self

ANS. 1st approach (multiply all numbers and divide the multiplication by each number):

2nd approach (calculate left and right products in two vectors TC-O(N),SC-O(N)):

vector<int> productExceptSelf(vector<int>& nums) {

int n=nums.size();

vector<int> left(n,1);

vector<int> right(n,1);

for (int i=1;i<n;i++)

left[i]=left[i-1]\*nums[i-1];

for (int i=n-2;i>=0;i--)

right[i]=right[i+1]\*nums[i+1];

vector<int> ans(n,0);

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

ans[i]=left[i]\*right[i];

return ans;

}

3rd approach (Use ans vector for storing the left multiplication TC-O(N),SC-O(1)):

vector<int> productExceptSelf(vector<int>& nums) {

int n=nums.size();

vector<int> ans(n,1);

int prod=nums[n-1];

for (int i=1;i<n;i++)

ans[i]=ans[i-1]\*nums[i-1];

for (int i=n-2;i>=0;i--)

{

ans[i]=ans[i]\*prod;

prod\*=nums[i];

}

return ans;

}

5. Sort with Difficulty

ANS. 1st approach ():

#include <bits/stdc++.h>

using namespace std;

int main() {

int p, s; cin>>p>>s;

vector<pair<int, int>> ans;

for(int i =0; i<p; i++){

vector<pair<int, int>> problem(s);

for(int i = 0;i<s;i++){

cin>>problem[i].first;

}

for(int i = 0;i<s;i++){

cin>>problem[i].second;

}

sort(problem.begin(), problem.end());

int c = 0;

for(int i = 0; i<s-1; i++){

if(problem[i].second>problem[i+1].second){

c++;

}

}

ans.push\_back({c,i+1});

}

sort(ans.begin(),ans.end());

for (int i=0;i<ans.size();i++)

cout<<ans[i].second<<endl;

return 0;

}

6. Greedy Candidates

Ans. 1st approach ():

#include <bits/stdc++.h>

using namespace std;

int main() {

int t;

cin>>t;

while (t--)

{

int N,M;

cin>>N>>M;

vector<int> minSalary(N,0);

for (int i=0;i<N;i++)

cin>>minSalary[i];

vector<int> offeredSalary(M,0);

vector<int> maxJobOffers(M,0);

for (int i=0;i<M;i++)

{

cin>>offeredSalary[i];

cin>>maxJobOffers[i];

}

vector<vector<int>> arr(N,vector<int> (M,0));

for (int i=0;i<N;i++)

{

string s;

cin>>s;

for(int j = 0; j < M; j++)

arr[i][j] = s[j] - '0';

}

int jobs=0;

long long salary=0;

set<int> s;

for (int i=0;i<N;i++)

{

int selected=-1;

for (int j=0;j<M;j++)

{

if (arr[i][j] && maxJobOffers[j]>0 && offeredSalary[j]>minSalary[i])

{

if (selected==-1 || offeredSalary[selected]<offeredSalary[j])

{

selected=j;

}

}

}

if (selected>=0)

{

jobs++;

salary+=(long long)offeredSalary[selected];

maxJobOffers[selected]--;

s.insert(selected);

}

}

cout<<jobs<<" "<<salary<<" "<<M-s.size()<<endl;

}

return 0;

}

7. Redcue To Zero

ANS. 1st approach ():

#include <bits/stdc++.h>

using namespace std;

int main() {

int t;

cin>>t;

while (t--)

{

long long x,y;

cin>>x>>y;

if (x>y) swap(x,y);

if (x==y) cout<<x<<endl;

else if (x==0) cout<<-1<<endl;

else

{

long long ans=0;

while (x<y)

{

x\*=2;

ans++;

}

cout<<ans+y<<endl;

}

}

return 0;

}

8. Count Pairs in array whose sum is divisible by k

ANS. 1st approach (O(n2)):

2nd approach (O(n)):

int countKdivPairs(int A[], int n, int K)

{

int freq[k]={0};

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

++freq[A[i] % K];

int sum = freq[0] \* (freq[0] - 1) / 2;

for (int i = 1; i <= K / 2 && i != (K - i); i++)

sum += freq[i] \* freq[K – i];

if (K % 2 == 0)

sum += (freq[K / 2] \* (freq[K / 2] - 1) / 2);

return sum;

}

9. Swapping pairs make sum equal

Ans. 1st Approach (Use Hashing to store the second array):

int findSwapValues(int A[], int n, int B[], int m)

{

unordered\_map<int,int> mp;

int suma=accumulate(A,A+n,0);

int sumb=accumulate(B,B+m,0);

if ((suma-sumb)%2==1) return -1;

int target=(suma-sumb)/2;

for (int i=0;i<m;i++)

mp[B[i]]=i;

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

{

if (mp.find(A[i]-target)!=mp.end())

return 1;

}

return -1;

}

2nd Approach (Sort the array and check for Target):

int findSwapValues(int A[], int n, int B[], int m)

{

long long suma=accumulate(A,A+n,0);

long long sumb=accumulate(B,B+m,0);

if (suma==sumb) return 1;

if ((suma-sumb)%2==1) return -1;

long long target=(suma-sumb)/2;

sort(A,A+n);

sort(B,B+m);

int i=0,j=0;

while (i<n && j<m)

{

long long currdiff=A[i]-B[j];

if (currdiff==target)

return 1;

else if (currdiff<target)

i++;

else

j++;

}

return -1;

}

10. Phone Number(Leetcode 17)

ANS. 1st approach (BackTracking Solution):

void solve(int i,string digits,vector<string>&ans,string temp,map<char,string> &mp)

{

if (i==digits.size())

{

ans.push\_back(temp);

return;

}

string comb=mp[digits[i]];

for (int j=0;j<comb.size();j++)

solve(i+1,digits,ans,temp+comb[j],mp);

}

vector<string> letterCombinations(string digits) {

map<char,string> mp;

mp['2']="abc";

mp['3']="def";

mp['4']="ghi";

mp['5']="jkl";

mp['6']="mno";

mp['7']="pqrs";

mp['8']="tuv";

mp['9']="wxyz";

vector<string> ans;

if (digits=="") return ans;

solve(0,digits,ans,"",mp);

return ans;

}

10.1 Find which words can be made using the Digits and which are also present in the given set.

Ans.

11. Find Maximum of Two Numbers without using any conditional Operators

Ans. 1st approach ():

12. Find Number of Trailing Zeroes in Factorial of Number

Ans. 1st approach (we have to count the number of 5 and 2 because these two will comtribute towards making zeroes at the end of Number i.e 10 . we will get 2’s easily by any even number. So we will count only 5’s):

int trailingZeroes(int n) {

int count=0;

for (long long i=5;n/i>0;i\*=5)

count+=n/i;

return count;

}

13. Find two Numbers each from one array such that their abs Diff is Minimum

Ans. 1st approach ():

void findPair(int A[], int B[])

{

int m = A.length - 1;

int n = B.length - 1;

int i = 0;

int j = 0;

int min = Integer.MAX\_VALUE;

int a, b;

while (i < m && j < n) {

if (Math.abs(A[i]-B[j]) < min) {

min = Math.abs(A[i]-B[j]);

a = A[i];

b = B[j];

if (min == 0)

break; /\* absolute difference cannot be less than 0 \*/

}

if (A[i] < B[j])

i++;

else

j++;

}

System.out.println("The pair with minimum absolute difference is "+a+","+b);

}

14. Given two Line segments. find the intersection point of two line segment.

Ans. 1st approach (if the slope of lines segment are equal, then intersection can’t be possible. So return -1.else find the intersection point by solving both equation and find whether intersection points lie between given line segments or not.)

15. Plank of 2 size s and l.you have to make diving board using k planks. return all the length of diving board which can be achieved.

Ans. 1st approach (using backtracking and recursion, fill all the places with all possible choices) :

Void findlength(unordered\_set<int>&st,int &s, int &l, int k,int total)

{

If (k==0)

{

St.insert(total);

Return;

}

Findlength(st,s,l,k-1,total-s);

Findlength(st,s,l,k-1,total-l);

}

Calling 🡺 findlength(st,s,l,k,0);

2nd approach (given only 2 sizes we can fill easily the k places which does not generate duplicates reseults):

Unordered\_set<int> st;

For (int i=0;i<=n;i++)

St.insert(s\*i+l\*(n-i));

16. Tic Tac Toe Design Methods

Ans.

17. Print all the arrays that generate unique BST.

Ans.

18. Check Balanced BT.

Ans.

19. Validate BST

Ans.

20. Inorder Successor of a Node

Ans.

21. Preorder, postorder and Inorder using morris traversal

Ans.

22. Given two large tree t1 and t2 where t1>>>t2.check whether t2 is subtree in t1 or not.

Ans.

23. Predict the Winner

Ans.

24. Max Consecutive ones II (at most 1 flips)

Ans.

It is best to use a general solution to deal with this kind of problem. A window [left,right] can be maintained to hold at least k zeros. When it encounters 0, it accumulates the number of zero, and then judges if the number of 0 is greater than k at this time, then shifts the left boundary left to the right, and if the removed nums[left] is 0, then zero decrements by 1. If it is not greater than k, use the number of digits in the window to update res.

int findMaxConsecutiveOnes(vector<int>& A) {

int zero = 0, i = 0, j = 0, N = A.size(), ans = 0;

while (j < N) {

zero += A[j++] == 0;

while (zero > 1) zero -= A[i++] == 0;

ans = max(ans, j - i);

}

return ans;

}

25. Max Consecutive ones III (for k most flips)

Ans. int findMaxConsecutiveOnes(vector<int>& A) {

int zero = 0, i = 0, j = 0, N = A.size(), ans = 0;

while (j < N) {

zero += A[j++] == 0;

while (zero > k) zero -= A[i++] == 0;

ans = max(ans, j - i);

}

return ans;

}

26. Construct Binary Tree from Preorder and Inorder Traversal

Ans.

TreeNode \*Build(vector<int>& preorder,int preStart,int preEnd ,vector<int>& inorder,int inStart,int inEnd,map<int,int> &inMap)

{

if (preStart>preEnd || inStart>inEnd)

return nullptr;

TreeNode \*root=new TreeNode(preorder[preStart]);

int inroot=inMap[root->val];

int numsleft\_in\_leftside=inroot-inStart; //conatins

root->left=Build(preorder,preStart+1,preStart+numsleft\_in\_leftside,inorder,inStart,inroot-1,inMap);

root->right=Build(preorder,preStart+numsleft\_in\_leftside+1,preEnd,inorder,inroot+1,inEnd,inMap);

return root;

}

TreeNode\* buildTree(vector<int>& preorder, vector<int>& inorder) {

map<int,int> inMap;

for (int i=0;i<inorder.size();i++)

inMap[inorder[i]]=i;

TreeNode \*root=Build(preorder,0,preorder.size()-1,inorder,0,inorder.size()-1,inMap);

return root;

}

27. Construct Binary Tree from Postorder and Inorder Traversal

Ans.

TreeNode \*Build(vector<int>& postorder,int postStart,int postEnd ,vector<int>& inorder,int inStart,int inEnd,map<int,int> &inMap)

{

if (postStart>postEnd || inStart>inEnd)

return nullptr;

TreeNode \*root=new TreeNode(postorder[postEnd]);

int inroot=inMap[root->val];

int numsLeft=inroot-inStart;

root->left=Build(postorder,postStart,postStart+numsLeft-1,inorder,inStart,inroot-1,inMap);

root->right=Build(postorder,postStart+numsLeft,postEnd-1,inorder,inroot+1,inEnd,inMap);

return root;

}

TreeNode\* buildTree(vector<int>& inorder, vector<int>& postorder) {

map<int,int> inMap;

for (int i=0;i<inorder.size();i++)

inMap[inorder[i]]=i;

TreeNode \*root=Build(postorder,0,postorder.size()-1,inorder,0,inorder.size()-1,inMap);

return root;

}

28. Find number of subsequences where all elements are equal

Ans. 1st approach (count the frequency of each element and calculate subsequences on that basis) :

#include<bits/stdc++.h>

long mod=1e9+7;

long power(long x, int n) {

long power=1;

long nn=n;

if (n<0) nn\*=-1;

while (nn)

{

if (nn%2==0)

{

x=(x\*x)%mod;

nn/=2;

}

else

{

power=(power\*x)%mod;

nn=nn-1;

}

}

if (n<0)

power=1/power;

return power;

}

long countSubsequences(int\* arr, int n) {

long ans=0;

map<int,int> mp;

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

mp[arr[i]]++;

for (auto it:mp)

{

ans=(ans+power(2,it.second)-1)%mod;

}

return ans%mod;

}

29. Prime Number using Sieve of Eratosthenes

Ans.

bool prime[MAX + 1];

void SieveOfEratosthenes()

{

memset(prime, true, sizeof(prime));

prime[1] = false;

for (int p = 2; p \* p <= MAX; p++) {

if (prime[p] == true) {

for (int i = p \* P; i <= MAX; i += p)

prime[i] = false;

}

}

}

30. Sum of all Substring of N

Ans.

Intuition :

sumofdigit[0] = 1 = 1

sumofdigit[1] = 2 + 12 = 14

sumofdigit[2] = 3 + 23 + 123 = 149

sumofdigit[3] = 4 + 34 + 234 + 1234 = 1506

sumofdigit[3] = 4 + 34 + 234 + 1234

= 4 + 30 + 4 + 230 + 4 + 1230 + 4

= 4\*4 + 10\*(3 + 23 +123)

= 4\*4 + 10\*(sumofdigit[2])

In general, sumofdigit[i] = (i+1)\*num[i] + 10\*sumofdigit[i-1]

int sumOfSubstrings(string num)

{

int n = num.length();

int sumofdigit[n];

sumofdigit[0] = toDigit(num[0]);

int res = sumofdigit[0];

for (int i = 1; i < n; i++) {

int numi = toDigit(num[i]);

sumofdigit[i]= (i + 1) \* numi + 10 \* sumofdigit[i - 1];

res += sumofdigit[i];

}

return res;

}

int sumOfSubstrings(string num)

{

int n = num.length();

int prev = toDigit(num[0]);

int res = prev;

int current = 0;

for (int i = 1; i < n; i++) {

int numi = toDigit(num[i]);

current = (i + 1) \* numi + 10 \* prev;

res += current;

prev = current;

}

return res;

}

31. Prime Matrix

Ans

#include<bits/stdc++.h>

using namespace std;

int N = 1000000;

vector<bool> isPrime(N + 1, true);

void setPrime()

{

isPrime[1] = false;

for (int i = 2; i <= sqrt(N); i++)

{

if (isPrime[i])

{

for (int j = i \* i; j <= N; j += i)

isPrime[j] = false;

}

}

}

int main()

{

setPrime();

int n, m;

cin >> n >> m;

vector<vector<int>> matrix;

matrix.resize(n, vector<int>(m));

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

for (int j = 0; j < m; j++)

cin >> matrix[i][j];

int ans = INT\_MAX;

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

{

int sum = 0;

for (int j = 0; j < m; j++)

{

if (!isPrime[matrix[i][j]])

{

int diff = 1;

while (!isPrime[matrix[i][j] + diff])

diff++;

sum += diff;

}

}

ans = min(ans, sum);

}

for (int j = 0; j < m; j++)

{

int sum = 0;

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

{

if (!isPrime[matrix[i][j]])

{

int diff = 1;

while (!isPrime[matrix[i][j] + diff])

diff++;

sum += diff;

}

}

ans = min(ans, sum);

}

cout << ans;

return 0;

}

32. Maximum Balls

Ans. Calculate GCD for all elements except the current element using Prefix and Suffix GCD

int Maximum\_balls(int n,vector<int> &marks)

{

if (n==1) return marks[0];

vector<int> prefix\_gcd(n);

vector<int> suffix\_gcd(n);

prefix\_gcd[0]=marks[0];

for (int i=1;i<n;i++)

prefix\_gcd[i]=\_\_gcd(prefix\_gcd[i],marks[i]);

suffix\_gcd[n-1]=marks[n-1];

for (int i=n-2;i>=0;i--)

suffix\_gcd[i]=\_\_gcd(suffix\_gcd[i+1],marks[i]);

int ans=0;

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

{

if (i==0)

ans=max(ans,suffix\_gcd[i+1]);

else if (i==n-1)

ans=max(ans,prefix\_gcd[i-1]);

else

ans=max(ans,\_\_gcd(prefix\_gcd[i-1],suffix\_gcd[i+1]));

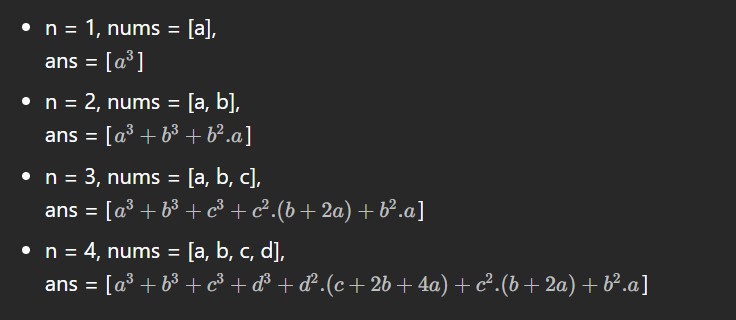
}

return max(ans,prefix\_gcd[n-1]);

}

33. power of heores(2681)

Ans.



int sumOfPower(vector<int>& nums) {

long long mod = 1e9 + 7, pre = 0, res = 0;

sort(nums.begin(), nums.end());

for (long long x: nums) {

res = (res + (x \* x % mod) \* x % mod + (x \* x % mod) \* pre % mod) % mod;

pre = (pre \* 2 + x) % mod;

}

return res;

}

34. 2663. Lexicographically Smallest Beautiful String

Ans.

/If the string does not contain any palindromic substrings of lengths 2 and 3, then the string does not contain any palindromic substrings at all. This is the main condition

string smallestBeautifulString(string s, int k) {

int n = s.size();

// Starting from the end of the string

for (int i = n-1; i >= 0; --i) {

for (char ch = s[i]+1; ch < 'a' + k; ++ch)

{

if ((i == 0 || s[i-1] != ch) && (i <= 1 || s[i-2] != ch))

{

s[i] = ch;

for (int j = i+1; j < n; ++j)

{

for (char cand = 'a'; cand < 'a' + k; ++cand)

{

if (cand != s[j-1] && (j == 1 || cand != s[j-2]))

{

s[j] = cand;

break;

}

}

}

return s;

}

}

}

// If no such string is possible, return an empty string

return "";

}

35. 2662. Minimum Cost of a Path With Special Roads

Ans. Dijkastra for finding shortest path in directed graph

int minimumCost(vector<int>& start, vector<int>& target, vector<vector<int>>& specialRoads) {

int startX = start[0] , startY = start[1];

int targetX = target[0] , targetY = target[1];

if(startX == targetX && startY == targetY) return 0;

map<pair<int,int>,int> cost; /\* {{X:CoOrdinate,Y:CoOrdinate},minCostToXY} \*/

cost[{startX,startY}] = 0; /\* Starting Co-Ordinstes \*/

cost[{targetX,targetY}] = 1e9 + 8621; /\* Target Co-ordinates : Initially Unreachable \*/

/\* priority Queue (minHeap) -> {CostToReachXY,{X,Y}} \*/

priority\_queue<pair<int,pair<int,int>>,vector<pair<int,pair<int,int>>>,greater<pair<int,pair<int,int>>>> pq;

pair<int,pair<int,int>> tuple = {0,{startX,startY}};

pq.push(tuple);

while(!pq.empty()){

/\* Always Get The Minimum Cost Path \*/

tuple = pq.top(); pq.pop();

int currX = tuple.second.first;

int currY = tuple.second.second;

int currCost = tuple.first;

/\* If Reached Target \*/

if(currX == targetX && currY == targetY) return currCost;

/\* Directly to target \*/

if(cost[{targetX,targetY}] > currCost + abs(currX-targetX)+abs(currY-targetY)){

/\* Total Cost = Current Cost + |X1-X2| + |Y1-Y2| \*/

cost[{targetX,targetY}] = currCost + abs(currX-targetX) + abs(currY-targetY);

pq.push({cost[{targetX,targetY}],{targetX,targetY}});

}

/\* Via Special Road \*/

for(vector<int> &road : specialRoads){

/\* Total Cost = Current Cost + Cost To Reach Special Road + Cost To Cross The Special Road \*/

int toReachCost = currCost + abs(currX - road[0]) + abs(currY - road[1]) + road[4];

if(cost.find({road[2],road[3]}) == cost.end() || cost[{road[2],road[3]}] > toReachCost){

cost[{road[2],road[3]}] = toReachCost;

pq.push({toReachCost,{road[2],road[3]}});

}

}

}

return -1;

}

36. Count the Number of Subsequences in which all the elements are equal

Ans.

void CountSubSequence(int A[], int N)

{

// Stores the count

// of subsequences

int result = 0;

// Stores the frequency

// of array elements

map<int, int> mp;

for (int i = 0; i < N; i++) {

// Update frequency of A[i]

mp[A[i]]++;

}

for (auto it : mp) {

// Calculate number of subsequences

result

= result + pow(2, it.second) - 1;

}

// Print the result

cout << result << endl;

}

37. Number of substrings with count of each character as k

Ans.

bool have\_same\_frequency(map<char, int>& freq, int k)

{

for (auto& pair : freq) {

if (pair.second != k && pair.second != 0) {

return false;

}

}

return true;

}

int count\_substrings(string s, int k)

{

int count = 0;

int distinct = (set<char>(s.begin(), s.end())).size();

for (int length = 1; length <= distinct; length++) {

int window\_length = length \* k;

map<char, int> freq;

int window\_start = 0;

int window\_end = window\_start + window\_length - 1;

for (int i = window\_start;

i <= min(window\_end, s.length() - 1); i++) {

freq[s[i]]++;

}

while (window\_end < s.length()) {

if (have\_same\_frequency(freq, k)) {

count++;

}

freq[s[window\_start]]--;

window\_start++;

window\_end++;

if (window\_length < s.length()) {

freq[s[window\_end]]++;

}

}

}

return count;

}

38. You have two numbers number1 and number2, your job is to check the number of borrow operations needed for subtraction of number1 from number2. If the subtraction is not possible

then return the string not possible.

Ans.

#include <bits/stdc++.h>

using namespace std;

int main()

{

string s1,s2;

int c=0,f=0;

cin>>s1>>s2;

if(stoi(s1)< stoi(s2)) {cout<<"Impossible";}

reverse(s1.begin(),s1.end());

reverse(s2.begin(),s2.end());

for(int i=0;i< s1.length();i++)

if(s1[i]< s2[i]) {f=1;c++;}

else if(s1[i]==s2[i])

{

if(f==1) {c++;}

}

else

f=0;

cout<< c;

}

39. Given an array of n integers, your task is to find the maximum sum of values in a contiguous subarray with length between a and b.

Ans.

We will compute the array of prefix sums prefix\_sums first. Then for each index s, we will compute the maximum elements of prefix\_sums[s+a..s+b]. The difference between prefix\_sum[s] and that maximum element is the maximum sum of a subarray that starts at index s + 1 and of length between a and b.

#include <bits/stdc++.h>

using namespace std;

typedef long long ll;

const int maxN = 2e5+1;

int N, A, B;

ll pre[maxN];

multiset<ll> S;

int main(){

scanf("%d %d %d", &N, &A, &B);

for(int i = 1; i <= N; i++){

scanf("%lld", &pre[i]);

pre[i] += pre[i-1];

}

for(int i = A; i <= B; i++)

S.insert(pre[i]);

ll best = \*S.rbegin();

for(int i = 1; i+A <= N; i++){

S.erase(pre[i+A-1]);

if(i+B <= N)

S.insert(pre[i+B]);

best = max(best, \*S.rbegin()-pre[i]);

}

printf("%lld\n", best);

}