# Dynamic Programming

1. Tabulation: Bottom Up
2. Memoization: Top Down

* Optimal Substructure: optimal solutions of subproblems, e.g. shortest path problem
* Overlapping subproblems: when solutions of same subproblems are needed again and again.
* Ways, arrangements, maximize, minimize.
* How to solve:

1) Identify if it is a DP problem

2) Decide a state expression with least parameters

3) Formulate state relationship

4) Do tabulation (or add memoization)

## **Longest Palindrome Subsequence**

Input : agbdba

Output : abdba

1. Initialize the 2d array lps[i][i]=1, substrings of length 1 have lps 1.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Lps | A | G | B | D | B | A |
| A | 1 |  |  |  |  |  |
| G |  | 1 |  |  |  |  |
| B |  |  | 1 |  |  |  |
| D |  |  |  | 1 |  |  |
| B |  |  |  |  | 1 |  |
| A |  |  |  |  |  | 1 |

1. Find lps for substrings of length 2 to n, hence difference in the positions of 1st and last characters of substring will be d:= 1 to n-1

i=0, j=i+d=0+1=1, lps(A0,G1)=1, lps(G1,B2)=1, and so on.

D=2, i=0, j=i+d=2, lps(A0,B2)=1, lps(G1,D3)=1, and so on.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | 0 | 1 | 2 | 3 | 4 | 5 |  |  |  | 0 | 1 | 2 | 3 | 4 | 5 |
|  | Lps | A | G | B | D | B | A |  | Lps | A | G | B | D | B | A |
| 0 | A | 1 | 1 |  |  |  |  | 0 | A | 1 | 1 | 1 |  |  |  |
| 1 | G |  | 1 | 1 |  |  |  | 1 | G |  | 1 | 1 | 1 |  |  |
| 2 | B |  |  | 1 | 1 |  |  | 2 | B |  |  | 1 | 1 | 3 |  |
| 3 | D |  |  |  | 1 | 1 |  | 3 | D |  |  |  | 1 | 1 | 1 |
| 4 | B |  |  |  |  | 1 | 1 | 4 | B |  |  |  |  | 1 | 1 |
| 5 | A |  |  |  |  |  | 1 | 5 | A |  |  |  |  |  | 1 |

1. This will fill upper half of the array and lps(0,5)=5 is the answer.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | 0 | 1 | 2 | 3 | 4 | 5 |
|  | Lps | A | G | B | D | B | A |
| 0 | A | 1 | 1 | 1 | 1 | 3 | 5 |
| 1 | G |  | 1 | 1 | 1 | 3 | 3 |
| 2 | B |  |  | 1 | 1 | 3 | 3 |
| 3 | D |  |  |  | 1 | 1 | 1 |
| 4 | B |  |  |  |  | 1 | 1 |
| 5 | A |  |  |  |  |  | 1 |

#include <iostream>

#include <stdlib.h>

using namespace std;

int LPS(string str){

int n=str.length();

int lps[n][n];

for(int i=0; i<n; i++){ //initialize array

lps[i][i]=1;

for(int j=0; j<n; j++){

if(i != j)

lps[i][j]=0;

}

}

for(int d=1; d<n; d++){//d is difference in the positions of 1st and last characters in the //substring

for(int i=0; i<n-d; i++){

int j=i+d;

if(str[i]==str[j]){

lps[i][j]=lps[j][i]=lps[i+1][j-1]+2;//lps increases by 2 with previous values

}

else{

lps[i][j]=lps[j][i]=max(lps[i+1][j],lps[i][j-1]); //max value of row-down and col-left

}

}

}

return lps[0][n-1];

}

int main() {

//code

int t;

cin>>t;

string str;

while(t--){

cin>>str;

cout<<LPS(str)<<endl;

}

return 0;

}

## **Maximum Common Subsequence**

input sequences = X[0..m-1] and Y[0..n-1], of lengths m and n resp.

LCS of 2 subsequences = L(X[0..m-1], Y[0..n-1])

If X[m-1] == Y[n-1], L(X[0..m-1], Y[0..n-1]) = 1 + L(X[0..m-2], Y[0..n-2])

Else X[m-1] != Y[n-1],

L(X[0..m-1], Y[0..n-1]) = MAX ( L(X[0..m-2], Y[0..n-1]), L(X[0..m-1], Y[0..n-2]) )

## **Longest Increasing-Decreasing Subsequence**

Given an array of integers, find the length of longest subsequence which is first increasing then decreasing.

\*\*Example: \*\*

For the given array [1 11 2 10 4 5 2 1]

Longest subsequence is [1 2 10 4 2 1]

Return value 6

Input array = arr[0..n-1]

LIS[0] = {arr[O]}

LIS[i] = {Max(LIS[j])} + arr[i] where j < i and arr[j] < arr[i]

= arr[i], if there is no such j

LDS[n] = {arr[n]}

LDS[i] = arr[i] + {Max(LDS[j])} where j > i and arr[j] < arr[i]

= arr[i], if there is no such j

For example, for array [1 11 2 10 4 5 2 1],

LIS[0]: 1

LIS[1]: 1 11

LIS[2]: 1 2

LIS[3]: 1 2 10

LIS[4]: 1 2 4

LIS[5]: 1 2 4 5

LIS[6]: 1 2

LIS[7]: 1

LDS[0]: 1

LDS[1]: 11 10 5 2 1

LDS[2]: 2 1

LDS[3]: 10 5 2 1

LDS[4]: 4 2 1

LDS[5]: 5 2 1

LDS[6]: 2 1

LDS[7]: 1

**Code:**

Sol1: Easy

int Solution::longestSubsequenceLength(const vector<int> &A) {

int n=A.size();

if(n==0 || n==1) return n;

int lis[n], lds[n];

lis[0]=1;

lds[n-1]=1;

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

lis[i]=1;

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

if(A[i]>A[j] && lis[i]<lis[j]+1)

lis[i]=lis[j]+1;

}

}

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

lds[i]=1;

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

if(A[i]>A[j] && lds[i]<lds[j]+1)

lds[i]=lds[j]+1;

}

}

int MID=1;

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

MID=max(MID, lis[i]+lds[i]-1);

}

return MID;

}

Sol2: complex

int Solution::longestSubsequenceLength(const vector<int> &A) {

int n=A.size();

if(n==0)

return 0;

vector<vector<int>> LIS(n), LDS(n);

LIS[0].push\_back(A[0]);

LDS[n-1].push\_back(A[n-1]);

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

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

if((A[i]>A[j]) && (LIS[j].size()>LIS[i].size()))

LIS[i]=LIS[j];

}

LIS[i].push\_back(A[i]);

}

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

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

if((A[i]>A[j]) && (LDS[j].size()>LDS[i].size()))

LDS[i]=LDS[j];

}

LDS[i].push\_back(A[i]);

}

int max=0;

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

if(max < LIS[i].size()+LDS[i].size())

max=LIS[i].size()+LDS[i].size()-1;

}

return max;

}

## Distinct Subsequences

* Asked in:
* [Google](https://www.interviewbit.com/search/?q=Google)
* Given two sequences S, T, count number of unique ways in sequence S, to form a subsequence that is identical to the sequence T.
* ***Subsequence :****A subsequence of a string is a new string which is formed from the original string by deleting some (can be none ) of the characters without disturbing the relative positions of the remaining characters. (ie, "ACE" is a subsequence of "ABCDE" while "AEC" is not).*
* **Example :**
* S = "rabbbit"
* T = "rabbit"
* Return 3. And the formations as follows:
* S1= "ra\_bbit"
* S2= "rab\_bit"
* S3="rabb\_it"
* "\_" marks the removed character.

Sol:

As a typical way to implement a dynamic programming algorithm, we construct a matrix dp, where each cell dp[i][j] represents the number of solutions of aligning substring T[0..i] with S[0..j];

Rule 1). dp[0][j] = 1, since aligning T = “” with any substring of S would have only ONE solution which is to delete all characters in S.

Rule 2). when i > 0, dp[i][j] can be derived by two cases:

case 1). if T[i] != S[j], then the solution would be to ignore the character S[j] and align substring T[0..i] with S[0..(j-1)]. Therefore, dp[i][j] = dp[i][j-1].

case 2). if T[i] == S[j], then first we could adopt the solution in case 1), but also we could match the characters T[i] and S[j] and align the rest of them (i.e. T[0..(i-1)] and S[0..(j-1)]. As a result, dp[i][j] = dp[i][j-1] + d[i-1][j-1]

e.g. T = B, S = ABC

dp[1][2]=1: Align T’=B and S’=AB, only one solution, which is to remove character A in S’.

int Solution::numDistinct(string S, string T) {

// Do not write main() function.

// Do not read input, instead use the arguments to the function.

// Do not print the output, instead return values as specified

// Still have a doubt. Checkout www.interviewbit.com/pages/sample\_codes/ for more details

int rows = T.size(), cols = S.size();

if(rows > cols){

return 0;

}

vector<vector<int> > temp(rows+1, vector<int>(cols+1, 0));

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

temp[0][i] = 1;

}

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

for(int j = 1; j <= cols; j++){

if(S[j-1] == T[i-1]){

temp[i][j] = temp[i-1][j-1] + temp[i][j-1];

}

else{

temp[i][j] = temp[i][j-1];

}

}

}

return temp[rows][cols];

}

## **Ways to Decode**

Facebook, Amazon

A message containing letters from A-Z is being encoded to numbers using the following mapping:

'A' -> 1

'B' -> 2

...

'Z' -> 26

Given an encoded message containing digits, determine the total number of ways to decode it.

**Example :**

Given encoded message "12", it could be decoded as "AB" (1 2) or "L" (12).

Sol:

So, when looking at the start of the string, we can either form a one digit code, and then look at the ways of forming the rest of the string of length L - 1, or we can form 2 digit code if its valid and add up the ways of decoding rest of the string of length L - 2.  
This obviously is exponential.

The code would go something like the following :

int ways(string &s, int startIndex) {

// BASE CASES

int answer = 0;

if (isValid(s[startIndex])) answer += ways(s, startIndex + 1);

if (isValid(s[startIndex] + s[startIndex + 1])) answer += ways(s, startIndex + 2);

return answer;

}

The number of ways decoding "12" is 2.

int Solution::numDecodings(string A) {

int n=A.size(), ways[n+1];

if(A[0]=='0'|| n==0) // if 1st char is 0 or string is empty, encoding is wrong

return 0;

ways[0]=1;//no chars in string

ways[1]=1; //1 char in string

for(int i=2; i<=n; i++){

ways[i]=0;

if(A[i-1] > '0') //consider ways till i-1th char

ways[i]=ways[i-1];

if(A[i-2] == '1' || (A[i-2] == '2' && A[i-1] < '7'))//combination possible

ways[i] += ways[i-2];

}

return ways[n];

}

## Stairs

* Asked in:
* [Morgan Stanley](https://www.interviewbit.com/search/?q=Morgan)
* [Amazon](https://www.interviewbit.com/search/?q=Amazon)
* [Intel](https://www.interviewbit.com/search/?q=Intel)

You are climbing a stair case. It takes n steps to reach to the top.

Each time you can either climb 1 or 2 steps. In how many distinct ways can you climb to the top?

**Example :**

Input : 3

Return : 3

Steps : [1 1 1], [1 2], [2 1]

int Solution::climbStairs(int A) {

int dp[A+1];

dp[0]=1;//0 steps can be overed in 1 unique way

dp[1]=1;

dp[2]=2;

for(int i=3; i<=A; i++){

dp[i]=dp[i-1]+dp[i-2];

}

return dp[A];

}

## **Largest area of rectangle with permutations**

Directi

Given a binary grid i.e. a 2D grid only consisting of 0’s and 1’s, find the area of the largest rectangle inside the grid such that all the cells inside the chosen rectangle should have 1 in them. You are allowed to permutate the columns matrix i.e. you can arrange each of the column in any order in the final grid. Please follow the below example for more clarity.

Lets say we are given a binary grid of 3 \* 3 size.

**1 0 1**

**0 1 0**

**1 0 0**

At present we can see that max rectangle satisfying the criteria mentioned in the problem is of **1 \* 1 = 1** area i.e either of the 4 cells which contain 1 in it. Now since we are allowed to permutate the columns of the given matrix, we can take column 1 and column 3 and make them neighbors. One of the possible configurations of the grid can be:

**1 1 0**

**0 0 1**

**1 0 0**

Now In this grid, first column is column 1, second column is column 3 and third column is column 2 from the original given grid. Now, we can see that if we calculate the max area rectangle, we get max area as **1 \* 2 = 2** which is bigger than the earlier case. Hence 2 will be the answer in this case.

Sol:

Create array dp[no of rows][no of cols]. dp[i][j] will denote the number of consecutive 1’s starting from the cell (i, j) and continuing upwards.

int Solution::solve(vector<vector<int> > &A) {

int n=A.size(); //no. of rows

int m=A[0].size(); //no. of cols

int dp[n+1][m+1];

memset(dp, 0, sizeof(dp));

for(int i=1; i<=m; i++){//go columnwise

for(int j=1; j<=n; j++){//go rowwise

if(A[j-1][i-1] == 0)

dp[j][i]=0;

else

dp[j][i]=dp[j-1][i]+1; // find dp value of upper row and same col add 1 to it

}

}

//creating histogram for each row, i.e. freqs of cols with height 0 to n of 1s

int res=0;

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

int hist[n+1], count=0;//hist:maximum possible height of consecutive 1s in a row

//count: no. of columns counted for a height in hist

memset(hist, 0, sizeof(hist));

for(int j=1; j<=m; j++){//go columnwise

hist[dp[i][j]]++; //dp[i][j] gives no. of consecutives 1’s at (i,j) n hist[dp[i][j]] gives //count of such cols

}

for(int j=i; j>=0; j--){//some cells might have 0 1's giving dp value 0

count +=hist[j];//count is no of cols with height of j consecutive 1's

res = max(res, count\*j);

}

}

return res;

}

## **Nearest Smaller Element**

* Asked in:
* [Amazon](https://www.interviewbit.com/search/?q=Amazon)
* [Microsoft](https://www.interviewbit.com/search/?q=Microsoft)

Given an array, find the **nearest** smaller element G[i] for every element A[i] in the array such that the element has an **index smaller than i**.

More formally,

G[i] for an element A[i] = an element A[j] such that

j is maximum possible AND

j < i AND

A[j] < A[i]

Elements for which no smaller element exist, consider next smaller element as -1.

**Example:**

Input : A : [4, 5, 2, 10, 8]  
Return : [-1, 4, -1, 2, 2]

vector<int> Solution::prevSmaller(vector<int> &A) {

stack<int> st;

vector<int> ans;

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

while(!st.empty() && st.top() >= A[i]) st.pop();

if(st.empty())

ans.push\_back(-1);

else

ans.push\_back(st.top());

st.push(A[i]);

}

return ans;

}

## **Intersecting Chords in a Circle**

Given a number N, return number of ways you can draw N chords in a circle with 2\*N points such that no 2 chords intersect.  
Two ways are different if there exists a chord which is present in one way and not in other.

For example,

N=2

If points are numbered 1 to 4 in clockwise direction, then different ways to draw chords are:

{(1-2), (3-4)} and {(1-4), (2-3)}

So, we return 2.

Notes:

* 1 ≤ N ≤ 1000
* Return answer modulo 109+7.

Sol:

If we draw a chord between any two points, can you observe current set of points getting broken into two smaller sets S\_1 and S\_2?

If we draw a chord from a point in S\_1 to a point in S\_2, it will surely intersect the chord we’ve just drawn.

So, we can arrive at a recurrence that Ways(n) = sum[i = 0 to n-1] { Ways(i)\*Ways(n-i-1) }. i.e. ways(i) is S\_1 and ways(n-i-1) is S\_2  
Here we iterate over i, assuming that size of one of the sets is i and size of other set automatically is (n-i-1) since we’ve already used a pair of points and i pair of points in one set.

int Solution::chordCnt(int A) {

long ways[A+1];

memset(ways,0, sizeof(ways));

ways[0]=1;

ways[1]=1;

for(int i=2; i<=A; i++){

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

ways[i]=(ways[i]+ways[j]\*ways[i-1-j])%1000000007;

}

}

return ways[A];

}

## Tushar's Birthday Bombs

It’s Tushar’s birthday today and he has N friends. Friends are numbered [0, 1, 2, …., N-1] and i-th friend have a positive strength S(i). Today being his birthday, his friends have planned to give him birthday bombs (kicks :P). Tushar’s friends know Tushar’s pain bearing limit and would hit accordingly.  
If Tushar’s resistance is denoted by R (>=0) then find the lexicographically smallest order of friends to kick Tushar so that the cumulative kick strength (sum of the strengths of friends who kicks) doesn’t exceed his resistance capacity and total no. of kicks hit are maximum. Also note that each friend can kick unlimited number of times (If a friend hits x times, his strength will be counted x times)

**Note:**

1. Answer format = Vector of indexes of friends in the order in which they will hit.
2. Answer should have the maximum no. of kicks that can be possibly hit. If two answer have the same no. of kicks, return one with the lexicographically smaller.
3. [a1, a2, …., am] is lexicographically smaller than [b1, b2, .., bm] if a1 < b1 or (a1 = b1 and a2 < b2) … .
4. Input cases are such that the length of the answer does not exceed 100000.

**Example:**  
R = 11, S = [6,8,5,4,7]

ans = [0,2]  
Here, [2,3], [2,2] or [3,3] also give the maximum no. kicks.

Sol:

Index of friend with minimum strength be mn then max\_kicks=R/S[mn]

[S[mn], S[mn], … max\_kicks times] will be the straight answer. But we need to consider lexicographical index, hence start comparing the strength of friend from index 0 till mn. We don’t need to consider indices after mn as that solution will be lexicographically higher than S[mn] max\_kicks times.

vector<int> Solution:: solve(int R, vector<int>S)

{

int mn=0, minS=S[0];

for(int i=1; i<S.size(); i++){

if(minS > S[i]){

minS=S[i];

mn=i;

}

}

//min strength and its index is found

vector<int> ans;

int max\_kicks=R/minS, j=0;

while(max\_kicks){

if(R-S[j] >= (max\_kicks-1)\*minS)

ans.push\_back(j);

else j++;

}

return ans;

}

## **Jump Game Array**

Asked in:

* [Amazon](https://www.interviewbit.com/search/?q=Amazon)
* [Ebay](https://www.interviewbit.com/search/?q=Ebay)

Given an array of non-negative integers, you are initially positioned at the first index of the array.

Each element in the array represents your maximum jump length at that position.

Determine if you are able to reach the last index.

For example:  
A = [2,3,1,1,4], return 1 ( true ).

A = [3,2,1,0,4], return 0 ( false ).

Sol1 with DP: (see sol2)

//dp in 1D array starting from last element of array

int Solution::canJump(vector<int> &A) {

int n=A.size();

if(n==1)

return 1;

bool jump[n];//array stating reachable 1 or not 0

memset(jump, 0, sizeof(jump));

jump[n-1]=1; //last element is the destination

for(int i=n-2; i>=0; i--){//start from 2nd last element

if(A[i]==0)//last index not reachable from ith index, do not check other conditions hence //else keyword is impotant

jump[i]=0;

else if(A[i]+i >= n-1) //if last index is reachable from ith index

jump[i]=1;

else{//find if all intermediate indices that are reaching last index or not

for(int j=i+1; j<n && j<=A[i]+i; j++){//2nd condition in for loop is to check if j is //reachable from i or not

if(jump[j]==1)

jump[i]=1;

}

}

}

return jump[0];

}

Sol2: No need of DP, Greedy will work

int Solution::canJump(vector<int> &A) {

int n=A.size();

int minPossibleIndex=n-1;

int isJumpPossible=1;

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

isJumpPossible=0;

if(A[i]+i >= minPossibleIndex){

isJumpPossible=1;

minPossibleIndex=i;

}

}

return isJumpPossible;

}

## **Min Jumps Array**

* Asked in:
* [Amazon](https://www.interviewbit.com/search/?q=Amazon)
* [Ebay](https://www.interviewbit.com/search/?q=Ebay)
* [Google](https://www.interviewbit.com/search/?q=Google)

Given an array of non-negative integers, you are initially positioned at the first index of the array.

Each element in the array represents your maximum jump length at that position.

Your goal is to reach the last index in the minimum number of jumps.

**Example :**  
Given array A = [2,3,1,1,4]

The minimum number of jumps to reach the last index is 2. (Jump 1 step from index 0 to 1, then 3 steps to the last index.)

If it is not possible to reach the end index, return -1.

Sol:

No need of DP, Greedy will work.

int Solution::jump(vector<int> &A) {

int n=A.size();

if(n<=1) return 0;

int curMaxReachPos=A[0];//where you can reach currently

int nextMaxReachPos=A[0];//where you can reach with each step

int jumps=1;

for(int i=1; i<=curMaxReachPos; i++){//take 1 step(i.e. i) at a time and see where you can reach next

if(i==n-1) return jumps; //if last pos is within curMaxReach, return jumps

nextMaxReachPos=max(nextMaxReachPos, A[i]+i);//update next possible //max reach position

if(i==curMaxReachPos){

jumps++; //increment jumps for reaching nextMaxReachPos

curMaxReachPos=nextMaxReachPos;//update the same

}

}

return -1;

}

## **Max Sum Without Adjacent Elements**

* Asked in:
* [Epic systems](https://www.interviewbit.com/search/?q=Epic)

Given a 2 \* N Grid of numbers, choose numbers such that the sum of the numbers  
is maximum and no two chosen numbers are adjacent horizontally, vertically or diagonally, and return it.

**Example:**

Grid:

1 2 3 4

2 3 4 5

so we will choose

3 and 5 so sum will be 3 + 5 = 8

Sol1: No DP, with Greedy

int Solution::adjacent(vector<vector<int> > &A) {

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

if(n==0) return 0;

int inclC, inclP, exclC, exclP=0;

inclP=max(A[0][0], A[1][0]);

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

inclC=exclP+max(A[0][i], A[1][i]);

exclC=max(inclP,exclP);

inclP=inclC;

exclP=exclC;

}

return max(inclP, exclP);

Sol2: With DP

int Solution::adjacent(vector<vector<int> > &A) {

int n=A[1].size();

if(n==0) return 0;

int dp[n+1];

dp[0]=0;

dp[1]=max(A[0][0], A[1][0]);

dp[2]=max(A[0][1],A[1][1]);

for(int i=3; i<=n; i++)

dp[i]=max(dp[i-2],dp[i-3])+max(A[0][i-1],A[1][i-1]);

return max(dp[n], dp[n-1]);

}

## **Edit Distance**

* Asked in:
* [Google](https://www.interviewbit.com/search/?q=Google)
* [LinkedIn](https://www.interviewbit.com/search/?q=LinkedIn)
* [Microsoft](https://www.interviewbit.com/search/?q=Microsoft)
* [Amazon](https://www.interviewbit.com/search/?q=Amazon)

Given two words A and B, find the minimum number of steps required to convert A to B. (each operation is counted as 1 step.)

You have the following 3 operations permitted on a word:

* Insert a character
* Delete a character
* Replace a character

**Example :**   
edit distance between  
"Anshuman" and "Antihuman" is 2.

* Operation 1: Replace s with t.
* Operation 2: Insert i.

int min(int a, int b, int c){

return min(min(a,b),c);

}

int Solution::minDistance(string A, string B) {

int a=A.size();

int b=B.size();

if(a==0)

return b;

if(b==0)

return a;

int dp[a+1][b+1];

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

dp[i][0]=i;

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

dp[0][j]=j;

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

for(int j=1; j<=b; j++){

if(A[i-1]==B[j-1]){

dp[i][j]=dp[i-1][j-1];

}

else

dp[i][j]=min(dp[i-1][j],dp[i][j-1],dp[i-1][j-1])+1;//insert, delete or replace

}

}

return dp[a][b];

}

## **Longest Increasing Subsequence**

* Asked in:
* [Facebook](https://www.interviewbit.com/search/?q=Facebook)
* [Yahoo](https://www.interviewbit.com/search/?q=Yahoo)
* [Epic systems](https://www.interviewbit.com/search/?q=Epic)
* [Amazon](https://www.interviewbit.com/search/?q=Amazon)
* [Microsoft](https://www.interviewbit.com/search/?q=Microsoft)

Find the **longest increasing subsequence** of a given sequence / array.

In other words, find a subsequence of array in which the subsequence’s elements are in strictly increasing order, and in which the subsequence is as long as possible.   
This subsequence is not necessarily contiguous, or unique.  
In this case, we only care about the **length** of the longest increasing subsequence.

**Example :**

Input : [0, 8, 4, 12, 2, 10, 6, 14, 1, 9, 5, 13, 3, 11, 7, 15]

Output : 6

The sequence : [0, 2, 6, 9, 13, 15] or [0, 4, 6, 9, 11, 15] or [0, 4, 6, 9, 13, 15]

## **Repeating Sub-Sequence**

Asked in:

* [Google](https://www.interviewbit.com/search/?q=Google)

Given a string, find if there is any sub-sequence that repeats itself.  
A sub-sequence of a string is defined as a sequence of characters generated by deleting some characters in the string without changing the order of the remaining characters.

**Input:**  
string

**Output:**

0/1

0 -> No

1 -> Yes

**Example:**

abab ------> yes, ab is repeated. So, return 1.

abba ------> No, a and b follow different order. So, return 0.

Sol1: Non DP solution

bool isPalindrome(string s){

int h=s.size()-1;

int l=0;

while(h>l){

if(s[h--]!=s[l++])

return false;

}

return true;

}

int Solution::anytwo(string A) {

int n=A.size();

int freq[256]={0};

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

freq[A[i]]++;

if(freq[A[i]] > 2) //there is repeating subseq

return 1;

}

string temp;

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

if(freq[A[i]] > 1)

temp.push\_back(A[i]);//forming string only of repeating chars

}

if(temp.size()==0)

return 0;

if(isPalindrome(temp)){//the palimdromic string with pattern “AAA” is already covered when frequency of certain character goes above 2.

return 0;

}

return 1;

}

Sol 2: DP solution

Find lcs of the string where i!=j

**int** **lcs**(string A, string B)

{

**int** n**=**A.length();

**int** dp[n**+**1][n**+**1];

**int** i,j;

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

**for**(j**=**0;j**<=**n;j**++**)

dp[i][j]**=**0;

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

**for**(j**=**1;j**<=**n;j**++**)

{

**if**(A[i**-**1]**==**B[j**-**1] **&&** i**!=**j)

{

dp[i][j]**=**1**+**dp[i**-**1][j**-**1];

}

**else**

dp[i][j]**=**max(dp[i][j**-**1],dp[i**-**1][j]);

*//cout<<i<<" "<<j<<" "<<dp[i][j]<<endl;*

}

**return** dp[n][n];

}

**int** Solution**::**anytwo(string A) {

**int** l**=**lcs(A,A);

**if**(l**>=**2)

**return** 1;

**else**

**return** 0;

*// Do not write main() function.*

*// Do not read input, instead use the arguments to the function.*

*// Do not print the output, instead return values as specified*

*// Still have a doubt. Checkout www.interviewbit.com/pages/sample\_codes/ for more details*

}

## **Longest Arithmetic Progression**

* Asked in:
* [Microsoft](https://www.interviewbit.com/search/?q=Microsoft)

Problem Setter: [dhruvi](https://www.interviewbit.com/profile/dhruvi) Problem Tester: sneh\_gupta

Find longest Arithmetic Progression in an integer array and return its length. More formally, find longest sequence of indeces, 0 < i1 < i2 < … < ik < ArraySize(0-indexed) such that sequence A[i1], A[i2], …, A[ik] is an Arithmetic Progression. Arithmetic Progression is a sequence in which all the differences between consecutive pairs are the same, i.e sequence B[0], B[1], B[2], …, B[m - 1] of length m is an Arithmetic Progression if and only if B[1] - B[0] == B[2] - B[1] == B[3] - B[2] == … == B[m - 1] - B[m - 2].  
**Examples**  
1) 1, 2, 3(All differences are equal to 1)  
2) 7, 7, 7(All differences are equal to 0)  
3) 8, 5, 2(Yes, difference can be negative too)

**Samples**  
1) Input: 3, 6, 9, 12  
Output: 4

Let dp[i][j] be the length of Longest Arithmetic progression that **ends** in positions i and j, i.e. last element is A[j] and element before last is A[i]. How can we calculate a value for fixed i and j? We know two last elements. So we know which number should be before position i. It’s number X such that A[i] - X == A[j] - A[i] -> X == 2 \* A[i] - A[j]. I.e we can iterate all over 0 <= k < i and if A[k] == X then update dp[i][j] by the value of dp[k][i] + 1(it’s easy to understand we only need to find rightmost such position).

int Solution::solve(const vector<int> &A) {

int n=A.size();

if(n<3)//AP can have size 2 or 1

return n;

int dp[n][n];//initialize 2D array

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

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

dp[i][j]=-1; //initialize dp to -1 for all values

map<int,int> pos;

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

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

dp[i][j]=2;//min value of AP is always 2 for any pair

int Ak=2\*A[i]-A[j];//find previous element in AP

if(pos.count(Ak)==0)//if it doesn’t exist

continue;

//AP having I, j includes k too

else dp[i][j]=max(dp[i][j], dp[pos[Ak]][i]+1);

}

pos[A[i]]=i;//as i moves ahead, previous elements of A with position <i become //available in the map

}

int res=2;

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

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

res=max(res, dp[i][j]);

}

return res;

}

## Interleaving Strings

Asked in:

* [Google](https://www.interviewbit.com/search/?q=Google)
* [Yahoo](https://www.interviewbit.com/search/?q=Yahoo)

Given s1, s2, s3, find whether s3 is formed by the interleaving of s1 and s2.

**Example,**  
Given:

s1 = "aabcc",

s2 = "dbbca",

When s3 = "aadbbcbcac", return true.  
When s3 = "aadbbbaccc", return false.

Return 0 / 1 ( 0 for false, 1 for true ) for this problem

Sol:

int isIL(string A, int a, string B, int b, string C, int c, vector<vector<int>> &dp){

if(a==A.size())

return (B.substr(b)==C.substr(c));

if(b==B.size())

return (A.substr(a)==C.substr(c));

int ans=0;

if(C[c]==A[a]&&isIL(A,a+1,B,b,C,c+1,dp) || C[c]==B[b]&&isIL(A,a,B,b+1,C,c+1,dp))

ans=1;

dp[a][b]=ans?1:0;

return ans;

}

int Solution::isInterleave(string A, string B, string C) {

if(A.size()+B.size()>C.size()) return 0;

vector<vector<int>> dp(A.size(),vector<int> (B.size(),0));

return isIL(A,0,B,0,C,0,dp);

}

## **Regular Expression Match**

* Asked in:
* [Facebook](https://www.interviewbit.com/search/?q=Facebook)
* [Microsoft](https://www.interviewbit.com/search/?q=Microsoft)

Implement wildcard pattern matching with support for '?' and '\*'.

* '?' : Matches any single character.
* '\*' : Matches any sequence of characters (including the empty sequence).

The matching should cover the entire input string (not partial).

The function prototype should be:

int isMatch(const char \*s, const char \*p)

**Examples :**

isMatch("aa","a") → 0

isMatch("aa","aa") → 1

isMatch("aaa","aa") → 0

isMatch("aa", "\*") → 1

isMatch("aa", "a\*") → 1

isMatch("ab", "?\*") → 1

isMatch("aab", "c\*a\*b") → 0

Return 1/0 for this problem.

Sol: No need of DP, DP used in string interleaving will take here exponential time

int Solution::isMatch(const string S, const string P) {

int n=S.size(), m=P.size();

int star=-1, prev\_start=0;

int i=0, j=0;

for(;i<n;){

if(S[i]==P[j]||P[j]=='?'){

i++;//skipping i

j++;//skipping j

}

else if(P[j]=='\*'){

star=j++;//skipping j

prev\_start=i; //not skipping i

}

else if(star!=-1){

j=star;

i=++prev\_start;//previous i is now skipped

}

else return 0;

}

while(j<m && P[j]=='\*')

j++;

return (j==m); //if rest pattern is \*

}

## **N digit numbers with digit sum S**

Find out the number of N digit numbers, whose digits on being added equals to a given number S. Note that a valid number starts from digits 1-9 except the number 0 itself. i.e. leading zeroes are not allowed.

Since the answer can be large, output answer modulo 1000000007

N = 2, S = 4   
Valid numbers are {22, 31, 13, 40}   
Hence output 4.

int\*\* dp;//declare pointer globally to access in all functions

long long int countRec(int n, int sum){

if(n==0) // when n=0 and sum=0 return 1;

return sum==0;

long long int ans=0;

if(dp[n][sum] > -1)//if dp is calculated, return the same

return dp[n][sum];

for(int i=0; i<=9; i++){//run loop for non MSB digit

if(sum-i >= 0)

ans +=countRec(n-1, sum-i);

ans %=1000000007;//put modulo operator late to avoid time exceed error

}

return dp[n][sum]=ans;

}

int Solution::solve(int A, int B) {

int sum=B, n=A;

int ans=0;

dp= new int\*[A+1]; // initalize 2d array dp to -1

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

dp[i]=new int[sum+1];

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

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

dp[i][j]=-1;

}

//MSB cannot be 0, can take digit between 1 to 9, run loop for MSB

for(int i=1; i<=9; i++){

if(sum-i >= 0){

ans += countRec(n-1, sum-i); // call recursive function

ans %=1000000007;

}

}

return ans;

}

## **Ways to color a 3xN Board**

* Asked in:
* [Codenation](https://www.interviewbit.com/search/?q=Codenation)

Given a 3Xn board, find the number of ways to color it using at most 4 colors such that no two adjacent boxes have same color. Diagonal neighbors are not treated as adjacent boxes.   
Output the ways%1000000007 as the answer grows quickly.

1<= n < 100000

**Example:**  
**Input:** n = 1  
**Output:** 36

**Examples :**

Input : 1

Output : 36

We can use either a combination of 3 colors

or 2 colors. Now, choosing 3 colors out of

4 is {4}\choose{3} and arranging them

in 3! ways, similarly choosing 2 colors out

of 4 is {4}\choose{2} and while arranging

we can only choose which of them could be at

center, that would be 2 ways.

Answer = 4C3\*3! + 4C2\*2! = 36

c3(n+1) = 10\*c2(n)+11\*c3(n);

c2(n+1) = 7\*c2(n)+5\*c3(n);

int Solution::solve(int A) {

long long int c3, c2, temp;//if temp not declared as long long int

//values get truncated and give incorrect value

c3=24;

c2=12;

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

temp=c3;

c3=(11\*c3+10\*c2)%1000000007;

//c3=c3%1000000007;

c2=(5\*temp+7\*c2)%1000000007;

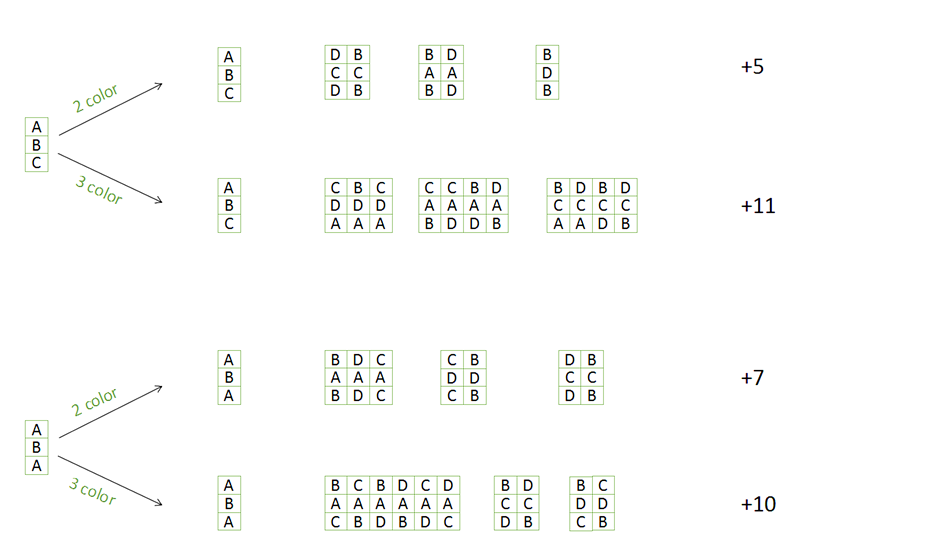
//c2=c2%1000000007;

}

long num=(c3+c2)%1000000007;

return num;

}



B

## Tushar's Birthday Party

* Asked in:
* [Snapdeal](https://www.interviewbit.com/search/?q=Snapdeal)

As it is Tushar’s Birthday on March 1st, he decided to throw a party to all his friends at TGI Fridays in Pune.  
Given are the eating capacity of each friend, filling capacity of each dish and cost of each dish. A friend is satisfied if the sum of the filling capacity of dishes he ate is equal to his capacity. Find the minimum cost such that all of Tushar’s friends are satisfied (reached their eating capacity).

**NOTE:**

1. Each dish is supposed to be eaten by only one person. Sharing is not allowed.
2. Each friend can take any dish unlimited number of times.
3. There always exists a dish with filling capacity 1 so that a solution always exists.

**Input Format**

Friends : List of integers denoting eating capacity of friends separated by space.

Capacity: List of integers denoting filling capacity of each type of dish.

Cost : List of integers denoting cost of each type of dish.

**Constraints:**  
1 <= Capacity of friend <= 1000  
1 <= No. of friends <= 1000  
1 <= No. of dishes <= 1000

**Example:**

Input:

2 4 6

2 1 3

2 5 3

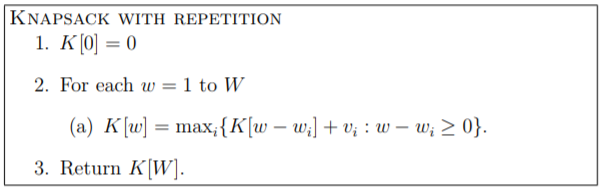
Output:

14

Explanation:

First friend will take 1st and 2nd dish, second friend will take 2nd dish twice. Thus, total cost = (5+3)+(3\*2)= 14

Sol:



#define INF 10000

int Solution::solve(const vector<int> &friends, const vector<int> &dishes, const vector<int> &cost) {

//this is knapsack with repetition problem

//declare array with max capacity

int capacity = \*max\_element(friends.begin(), friends.end());

int minCost[capacity+1];

memset(minCost, INF, sizeof(minCost));

minCost[0]=0;

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

for(int j=dishes[i]; j<=capacity; j++){

minCost[j]=min(minCost[j], minCost[j-dishes[i]]+cost[i]);

}

}

int ans=0;

for(auto afriend:friends){

ans+=minCost[afriend];

}

return ans;

}

Knapsack without repetition has below algo:

DP[i][j] ( j <= 0) = 0

DP[i][j] (i = 0, j > 0) = infinity

DP[i][j] (i > 0, j > 0) = min{ DP[i-1][j], DP[i-1][j - w[i-1]] + v[i-1] },

## Flip Array

Given an array of positive elements, you have to flip the sign of some of its elements such that the resultant sum of the elements of array should be minimum non-negative(as close to zero as possible). Return the minimum no. of elements whose sign needs to be flipped such that the resultant sum is minimum non-negative.

**Constraints:**

1 <= n <= 100

Sum of all the elements will not exceed 10,000.

**Example:**

A = [15, 10, 6]

ans = 1 (Here, we will flip the sign of 15 and the resultant sum will be 1 )

A = [14, 10, 4]

ans = 1 (Here, we will flip the sign of 14 and the resultant sum will be 0)

*Note that flipping the sign of 10 and 4 also gives the resultant sum 0 but flippings there are not minimum*

int Solution::solve(const vector<int> &A) {

int n=A.size();

int sum2=accumulate(A.begin(), A.end(), 0)/2;

//using array and memset to intialize to INT\_MAX fails, gives garbage values

vector<vector<int>> dp(n+1, vector<int>(sum2+1, INT\_MAX));

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

for(int j=0; j<=sum2; j++){

if(j==0) dp[i][j]=0; //when half sum is 0

else if(i==0) dp[i][j]=INT\_MAX;//0 elements in array

else{

int include=INT\_MAX, exclude=INT\_MAX;

if(j >= A[i-1] && dp[i-1][j-A[i-1]]!=INT\_MAX)

include=dp[i-1][j-A[i-1]]+1;//here value to add is 1 gives no. of elements

exclude=dp[i-1][j];

dp[i][j]=min(include, exclude);

}

}

}

//dp[n][sum2] doesn't give solution

for(int i=sum2; i>=0; i--){

if(dp[n][i]!=INT\_MAX)

return dp[n][i];

}

return -1;

}

## Equal Average Partition

* Asked in:
* [Amazon](https://www.interviewbit.com/search/?q=Amazon)

Given an array with non negative numbers, divide the array into two parts such that the average of both the parts is equal.   
Return both parts (If exist).  
If there is no solution. return an empty list.

**Example:**

Input:

[1 7 15 29 11 9]

Output:

[9 15] [1 7 11 29]

The average of part is (15+9)/2 = 12,

average of second part elements is (1 + 7 + 11 + 29) / 4 = 12

***NOTE 1:****If a solution exists, you should return a list of exactly 2 lists of integers A and B which follow the following condition :*

* numElements in A <= numElements in B
* If numElements in A = numElements in B, then A is lexicographically smaller than B ( https://en.wikipedia.org/wiki/Lexicographical\_order )

***NOTE 2:****If multiple solutions exist, return the solution where length(A) is minimum. If there is still a tie, return the one where A is lexicographically smallest.*

***NOTE 3:****Array will contain only non negative numbers.*

# STACKS AND QUEUES

## Generate all Parentheses

* Asked in:
* [Amazon](https://www.interviewbit.com/search/?q=Amazon)

Given a string containing just the characters '(', ')', '{', '}', '[' and ']', determine if the input string is valid.

The brackets must close in the correct order, "()" and "()[]{}" are all valid but "(]" and "([)]" are not.

Return 0 / 1 ( 0 for false, 1 for true ) for this problem

PROBLEM APPROACH :

Complete solution in hints.

int Solution::isValid(string A) {

stack<char> s;

map<char,char> matching;

matching['{']='}';

matching['[']=']';

matching['(']=')';

int n=A.size();

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

if(A[i]=='{' || A[i]=='[' || A[i]=='(')

s.push(A[i]);

else if(s.empty() || matching[s.top()]!=A[i])

return 0;

else s.pop();

}

return s.empty();

}

## **Balance Parenthesis**

int Solution::isValid(string A) {

stack<char> s;

map<char,char> matching;

matching['{']='}';

matching['[']=']';

matching['(']=')';

int n=A.size();

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

if(A[i]=='{' || A[i]=='[' || A[i]=='(')

s.push(A[i]);

else if(s.empty() || matching[s.top()]!=A[i])

return 0;

else s.pop();

}

return s.empty();

}

## Simplify Directory Path

* Asked in:
* [Microsoft](https://www.interviewbit.com/search/?q=Microsoft)

Given an absolute path for a file (Unix-style), simplify it.

**Examples:**

path = "/home/", => "/home"

path = "/a/./b/../../c/", => "/c"

Note that absolute path always begin with ‘/’ ( root directory )  
Path will not have whitespace characters.

Approach: find substrings that are lying within “/” chars, push them to stack, if “..” found, pop string from stack

Start of substring 🡪 size\_t find\_first\_not\_of(“args”, size\_t pos=0)🡪 find first occurrence which doesn’t match any of chars from args

End of substring 🡪find(‘/’,start) find position of ‘/’, get length of string as end-start

string Solution::simplifyPath(string A) {

stack<string> st;

size\_t start;

size\_t end=0;

string temp;

while((start=A.find\_first\_not\_of('/', end))!=std::string::npos){

end=A.find('/', start);

temp=A.substr(start, end-start);

if(temp == ".."){

if(!st.empty())

st.pop();

}

else if(temp != ".")

st.push(temp);

}

temp="";

while(!st.empty()){

temp="/"+st.top()+temp;

st.pop();

}

if(temp == "")

temp="/"+temp;

return temp;

}

## Nearest Smaller Element

* Asked in:
* [Amazon](https://www.interviewbit.com/search/?q=Amazon)
* [Microsoft](https://www.interviewbit.com/search/?q=Microsoft)

Given an array, find the **nearest** smaller element G[i] for every element A[i] in the array such that the element has an **index smaller than i**.

More formally,

G[i] for an element A[i] = an element A[j] such that

j is maximum possible AND

j < i AND

A[j] < A[i]

Elements for which no smaller element exist, consider next smaller element as -1.

**Example:**

Input : A : [4, 5, 2, 10, 8]  
Return : [-1, 4, -1, 2, 2]

**Example 2:**

Input : A : [3, 2, 1]  
Return : [-1, -1, -1]

Approach: Once we find a smaller number we don’t need to maintain numbers greater than that and appearing on its left.

Run a for loop, find its next smaller in stack. If stack is empty, put -1 in result vector, push current element to stack.

vector<int> Solution::prevSmaller(vector<int> &A) {

stack<int> st;

vector<int> ans;

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

while(!st.empty() && st.top() >= A[i]) st.pop();

if(st.empty())

ans.push\_back(-1);

else

ans.push\_back(st.top());

st.push(A[i]);

}

return ans;

}

## Largest Rectangle in Histogram

* Asked in:
* [Google](https://www.interviewbit.com/search/?q=Google)
* [Facebook](https://www.interviewbit.com/search/?q=Facebook)
* Given n non-negative integers representing the histogram’s bar height where the width of each bar is 1, find the area of largest rectangle in the histogram.

Above is a histogram where width of each bar is 1, given height = [2,1,5,6,2,3].

he largest rectangle is shown in the shaded area, which has area = 10 unit.

For example,  
Given height = [2,1,5,6,2,3],  
return 10.

int Solution::largestRectangleArea(vector<int> &A) {

int n=A.size();

stack<int> st;

if(n==0)

return 0;

int area=0, maxArea=0, top, i;

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

while(!st.empty() && A[st.top()]>A[i]){

top=st.top();

st.pop();

if(st.empty())

area=A[top]\*i;

else

area=A[top]\*(i-st.top()-1);

maxArea = max(area, maxArea);

}

st.push(i);

}

while(!st.empty()){

top=st.top();

st.pop();

if(st.empty()){

area=A[top]\*i;

}

else{

area=A[top]\*(i-st.top()-1);

}

maxArea=max(area, maxArea);

}

return maxArea;

}

## Evaluate Expression

* Asked in:
* [Yahoo](https://www.interviewbit.com/search/?q=Yahoo)
* [Google](https://www.interviewbit.com/search/?q=Google)
* [Facebook](https://www.interviewbit.com/search/?q=Facebook)
* Evaluate the value of an arithmetic expression in Reverse Polish Notation.
* Valid operators are +, -, \*, /. Each operand may be an integer or another expression.
* **Examples:**
* ["2", "1", "+", "3", "\*"] -> ((2 + 1) \* 3) -> 9
* ["4", "13", "5", "/", "+"] -> (4 + (13 / 5)) -> 6

bool isOperator(string s){

if(s=="+" || s=="-" || s=="\*" || s=="/") return 1;

return 0;

}

int cal(int n1, int n2, string op){

char c=op[0];

int res;

switch(c){

case '+':

res=n1+n2;

break;

case '-':

res=n1-n2;

break;

case '\*':

res=n1\*n2;

break;

case '/':

res=n1/n2;

}

return res;

}

int Solution::evalRPN(vector<string> &A) {

int n=A.size();

stack<int>st;

int res;

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

if(isOperator(A[i])){

int n2=st.top(); st.pop();

int n1=st.top(); st.pop();

int n=cal(n1, n2, A[i]);

st.push(n);

}

else{

stringstream ss(A[i]);

int n;

ss>>n;

st.push(n);

}

}

return st.top();

}

## **Sliding Window Maximum**

* Asked in:
* [Google](https://www.interviewbit.com/search/?q=Google)
* [Chronus](https://www.interviewbit.com/search/?q=Chronus)
* [Walmart labs](https://www.interviewbit.com/search/?q=Walmart)
* [Amazon](https://www.interviewbit.com/search/?q=Amazon)

BookmarkSuggest Edit

A long array A[] is given to you. There is a sliding window of size w which is moving from the very left of the array to the very right. You can only see the w numbers in the window. Each time the sliding window moves rightwards by one position. You have to find the maximum for each window. The following example will give you more clarity.

**Example :**

The array is [1 3 -1 -3 5 3 6 7], and w is 3.

| **Window position** | **Max** |
| --- | --- |
|  |  |
| [1 3 -1] -3 5 3 6 7 | 3 |
| 1 [3 -1 -3] 5 3 6 7 | 3 |
| 1 3 [-1 -3 5] 3 6 7 | 5 |
| 1 3 -1 [-3 5 3] 6 7 | 5 |
| 1 3 -1 -3 [5 3 6] 7 | 6 |
| 1 3 -1 -3 5 [3 6 7] | 7 |

**Input:** A long array A[], and a window

Sol1:

class Solution {

public:

vector<int> slidingMaximum(vector<int> &A, int w) {

int n = A.size();

vector<int> B;

if (n < w) return B;

B.resize(n - w + 1);

deque<int> Q;

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

while (!Q.empty() && A[i] >= A[Q.back()])

Q.pop\_back();

Q.push\_back(i);

}

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

B[i - w] = A[Q.front()];

while (!Q.empty() && A[i] >= A[Q.back()])

Q.pop\_back();

while (!Q.empty() && Q.front() <= i - w)

Q.pop\_front();

Q.push\_back(i);

}

B[n - w] = A[Q.front()];

return B;

}

};

Sol2:

vector<int> Solution::slidingMaximum(const vector<int> &A, int B) {

int n=A.size();

int k=B;

vector<int> res;

if(n<k) return res;

if(n==1) return A;

deque<int> Q;

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

while (!Q.empty() && A[Q.back()]<=A[i]) {

Q.pop\_back();

}

Q.push\_back(i);

}

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

res.push\_back(A[Q.front()]);

while (!Q.empty() && A[Q.back()]<=A[i]) {

Q.pop\_back();

}

while (!Q.empty() && Q.front()<i-k+1) {

Q.pop\_front();

}

Q.push\_back(i);

}

//each iteration adds result of its previous iteration

res.push\_back(A[Q.front()]);

return res;

}

## Rain Water Trapped

* Asked in:
* [Qualcomm](https://www.interviewbit.com/search/?q=Qualcomm)
* [Amazon](https://www.interviewbit.com/search/?q=Amazon)

Given n non-negative integers representing an elevation map where the width of each bar is 1, compute how much water it is able to trap after raining.

**Example :**

Given [0,1,0,2,1,0,1,3,2,1,2,1], return 6.



The above elevation map is represented by array [0,1,0,2,1,0,1,3,2,1,2,1].  
In this case, 6 units of rain water (blue section) are being trapped.

Sol: For each bar find its left highest bar and right highest bar. Take min of both and remove height of that bar. That will give volume for rain possible at current bar. Accumulate volumes for all bars and return the same.

int Solution::trap(const vector<int> &A) {

int n=A.size();

if(n==0 || n==1)

return 0;

int left[n], right[n];

left[0]=A[0];

right[n-1]=A[n-1];

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

left[i]=max(left[i-1],A[i]);

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

right[i]=max(right[i+1],A[i]);

int vol=0;

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

vol+=min(left[i],right[i])-A[i];

}

return vol;

}

## **Min Stack**

* Asked in:
* [Yahoo](https://www.interviewbit.com/search/?q=Yahoo)
* [Amazon](https://www.interviewbit.com/search/?q=Amazon)
* [Adobe](https://www.interviewbit.com/search/?q=Adobe)

Design a stack that supports push, pop, top, and retrieving the minimum element in constant time.

* **push(x)** – Push element x onto stack.
* **pop()** – Removes the element on top of the stack.
* **top()** – Get the top element.
* **getMin()** – Retrieve the minimum element in the stack.

Approach: if new element we are pushing is smaller than current min element in stack, then push y=2\*x-minElement to stack and set minElement=x. Here, y<x as minElement>x; Whenever x is to be popped, get previous min as 2\*minElement-y=2\*x-(2\*x-minElement(previous)

stack<int> s;

int minE;

MinStack::MinStack() {

while(!s.empty())

s.pop();

minE=-1;

}

void MinStack::push(int x) {

if(s.empty()){

s.push(x);

minE=x;

}

else{

if(x<minE){

s.push(2\*x-minE);

minE=x;

}

else

s.push(x);

}

}

void MinStack::pop() {

if(!s.empty()){

if(s.top() < minE){

minE=2\*minE-s.top();

s.pop();

}

else

s.pop();

}

}

int MinStack::top() {

if(s.empty())

return -1;

if(s.top() < minE)

return minE;

else

return s.top();

}

int MinStack::getMin() {

if(s.empty()) return -1;

else return minE;

}

vector**<int>** st;

vector**<int>** mins;

MinStack**::**MinStack() {

st.clear();

mins.clear();

}

**void** MinStack**::**push(**int** x) {

**int** sm **=** mins.size();

**if**(**!**sm **||** mins[sm**-**1] **>** x)

mins.push\_back(x);

st.push\_back(x);

}

**void** MinStack**::**pop() {

**if**(**!**st.size()) **return**;

**int** x **=** MinStack**::**top();

st.pop\_back();

**if**(x **==** mins[mins.size()**-**1]) mins.pop\_back();

}

**int** MinStack**::**top() {

**int** sz **=** st.size();

**if**(**!**sz) **return** **-**1;

**return** st[sz**-**1];

}

**int** MinStack**::**getMin() {

**if**(**!**mins.size()) **return** **-**1;

**return** mins[mins.size()**-**1];

}

# HEAPS AND MAPS

## N max pair combinations

* Asked in:
* [Liv.ai](https://www.interviewbit.com/search/?q=Liv.ai)

Given two arrays A & B of size N each.  
Find the maximum N elements from the sum combinations (Ai + Bj) formed from elements in array A and B.

For example if A = [1,2], B = [3,4], then possible pair sums can be 1+3 = 4 , 1+4=5 , 2+3=5 , 2+4=6  
and maximum 2 elements are 6, 5

**Example:**

N = 4  
a[]={1,4,2,3}  
b[]={2,5,1,6}

Maximum 4 elements of combinations sum are

10 (4+6),

9 (3+6),

9 (4+5),

8 (2+6)

Approach:

Sort both the arrays in ascending order.  
Let us take priority queue (heap).  
First max element is going to be the sum of the last two elements of array A and B i.e. (A[n-1] + B[n-1]).  
Insert that in heap with indices of both array i.e (n-1, n-1).  
Start popping from heap (n-iterations).  
And insert the sum (A[L-1]+A[R]) and (A[L]+B[R-1]).  
Take care that repeating indices should not be there in the heap (use map for that).

vector<int> Solution::solve(vector<int> &A, vector<int> &B) {

vector<int> res;

int n=A.size();

if(n==0 || n!=B.size())

return res;

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

set<pair<int, int>> S;

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

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

pq.push(make\_pair(A[n-1]+B[n-1], make\_pair(n-1, n-1)));

int k=n;

while(k--){

pair<int, pair<int, int>> p=pq.top();

pq.pop();

res.push\_back(p.first);

int L=p.second.first;

int R=p.second.second;

S.insert(make\_pair(L,R));

if(L>0 && R>=0 && S.find(make\_pair(L-1,R))==S.end()){

pq.push(make\_pair(A[L-1]+B[R], make\_pair(L-1,R)));

S.insert(make\_pair(L-1,R));

}

if(L>=0 && R>0 && S.find(make\_pair(L,R-1))==S.end()){

pq.push(make\_pair(A[L]+B[R-1], make\_pair(L,R-1)));

S.insert(make\_pair(L,R-1));

}

}

return res;

}

## Magician and Chocolates

Given **N** bags, each bag contains **Ai** chocolates. There is a kid and a magician. In one unit of time, kid chooses a random bag **i**, eats **Ai** chocolates, then the magician fills the ith bag with [*floor*](http://mathworld.wolfram.com/FloorFunction.html)(**Ai/2**) chocolates.

Given **Ai** for 1 <= *i* <= N, find the maximum number of chocolates kid can eat in **K** units of time.

For example,

K = 3

N = 2

A = 6 5

Return: 14

At t = 1 kid eats 6 chocolates from bag 0, and the bag gets filled by 3 chocolates  
At t = 2 kid eats 5 chocolates from bag 1, and the bag gets filled by 2 chocolates  
At t = 3 kid eats 3 chocolates from bag 0, and the bag gets filled by 1 chocolate  
so, total number of chocolates eaten: 6 + 5 + 3 = 14

**Note**: Return your answer modulo 10^9+7

Sol:

#define MOD 1000000007

typedef long long int LL;

int Solution::nchoc(int K, vector<int> &A){

int n=A.size();

LL sum=0;

priority\_queue<int> pq;

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

pq.push(A[i]);

while(K--){

int chocs=pq.top();

pq.pop();

sum=(sum+chocs)%MOD;

pq.push(floor(double(chocs)/2));

}

return sum;

}

## **Distinct Numbers in Window**

Asked in:

* [Amazon](https://www.interviewbit.com/search/?q=Amazon)

You are given an array of N integers, A1, A2 ,…, AN and an integer K. Return the of count of distinct numbers in all windows of size K.

Formally, return an array of size N-K+1 where i’th element in this array contains number of distinct elements in sequence Ai, Ai+1 ,…, Ai+k-1.

**Note**:

* If K > N, return empty array.

For example,

A=[1, 2, 1, 3, 4, 3] and K = 3

All windows of size K are

[1, 2, 1]

[2, 1, 3]

[1, 3, 4]

[3, 4, 3]

So, we return an array [2, 3, 3, 2].

vector<int> Solution::dNums(vector<int> &A, int B) {

int n=A.size();//size of array A

vector<int> res;

if(B > n)

return res;

map<int, int> m;//map to store nums in array

int dist\_count=0;//distinct numbers count

//find distinct count for first B numbers

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

if(m[A[i]]==0)

dist\_count++;

m[A[i]]+=1; //add the appearance count in map

}

res.push\_back(dist\_count); //add distint count of 1st window to res

//work on rest elements in array

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

//remove count of earliest number from map, to slide the window

if(m[A[i-B]]==1)

dist\_count--;//when earliest number appeared only once

m[A[i-B]]-=1;//reduce the appearance count in map

//add next number to window

if(m[A[i]]==0)

dist\_count++;

m[A[i]]+=1;

//push dist\_count to res

res.push\_back(dist\_count);

}

return res;

}

## **Merge K Sorted Lists**

Asked in:

* [Flipkart](https://www.interviewbit.com/search/?q=Flipkart)
* [Amazon](https://www.interviewbit.com/search/?q=Amazon)
* [Google](https://www.interviewbit.com/search/?q=Google)
* Merge k sorted linked lists and return it as one sorted list.

**Example :**

1 -> 10 -> 20

4 -> 11 -> 13

3 -> 8 -> 9

will result in

1 -> 3 -> 4 -> 8 -> 9 -> 10 -> 11 -> 13 -> 20

/\*\*

\* Definition for singly-linked list.

\* struct ListNode {

\* int val;

\* ListNode \*next;

\* ListNode(int x) : val(x), next(NULL) {}

\* };

\*/

void printQ(ListNode \*n){

ListNode \*t=n;

while(t){

cout<<t->val<<" ";

t=t->next;

}

cout<<endl;

}

//compare struct for priority queue

struct compare{

bool operator()(ListNode\* a, ListNode\* b){

return (a->val > b->val);

}

};

ListNode\* Solution::mergeKLists(vector<ListNode\*> &A) {

// Do not write main() function.

// Do not read input, instead use the arguments to the function.

// Do not print the output, instead return values as specified

// Still have a doubt. Checkout www.interviewbit.com/pages/sample\_codes/ for more details

//get size of vector

int k=A.size();

//declare priority queue

priority\_queue<ListNode\*, vector<ListNode\*>, compare> pq;

//declare nodes for final result list

ListNode \*head=NULL, \*tail;

//push first k heads of the lists

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

pq.push(A[i]);

}

while(!pq.empty()){

//get minimum number from heap

ListNode \*top=pq.top();

pq.pop();

//add next number of min to queue

if(top->next)

pq.push(top->next);

//insert number to result

if(head==NULL){

head=top;

tail=top;

}

else{

tail->next=top;

tail=top;

}

}

//printQ(head);

return head;

}

## **LRU Cache**

* Asked in:  [Adobe](https://www.interviewbit.com/search/?q=Adobe) [Citigroup](https://www.interviewbit.com/search/?q=Citigroup) [Amazon](https://www.interviewbit.com/search/?q=Amazon) [Microsoft](https://www.interviewbit.com/search/?q=Microsoft)

Design and implement a data structure for LRU (Least Recently Used) cache. It should support the following operations: get and set.

1. get(key) - Get the value (will always be positive) of the key if the key exists in the cache, otherwise return -1.
2. set(key, value) - Set or insert the value if the key is not already present. When the cache reaches its capacity, it should invalidate the least recently used item before inserting the new item.

The LRU Cache will be initialized with an integer corresponding to its capacity. Capacity indicates the maximum number of unique keys it can hold at a time.

**Definition of “least recently used”** : An access to an item is defined as a get or a set operation of the item. “Least recently used” item is the one with the oldest access time.

***NOTE:****If you are using any global variables, make sure to clear them in the constructor.*

#include <bits/stdc++.h>

typedef list<int>::iterator LIT;

typedef unordered\_map<int, LIT>::iterator MIT;

list<int> dq;

unordered\_map<int, LIT> ma;

int csize;

void printqueue(){

cout<<"\nqueue is ";

for(LIT lit=dq.begin(); lit!=dq.end(); ++lit)

cout<<\*lit<<" ";

cout<<"EOQ"<<endl;

}

LRUCache::LRUCache(int capacity) {

ma.clear();

dq.clear();

csize=capacity;

}

int LRUCache::get(int key) {

int res=-1;

MIT mit= ma.find(key);

if(mit!=ma.end()){//entry is in cache

res=\*(mit->second);

dq.erase(mit->second);

dq.push\_front(res);

ma[key]=dq.begin();

}

//printqueue();

return res;

}

void LRUCache::set(int key, int value) {

MIT mit= ma.find(key);

if(mit!=ma.end()){//entry already in cache

if(dq.begin() != mit->second){//entry is not latest

dq.erase(mit->second);

dq.push\_front(value);

mit->second=dq.begin();

}

else//entry is latest

\*(mit->second)=value;

}

else{//entry not in cache

if(dq.size()==csize){//cache full, remove least recent entry

//remove least recent cache entry from map

LIT it=dq.end();

--it;

MIT mit;

for(mit=ma.begin(); mit!=ma.end(); ++mit){//erasing entry from map and dq

if(mit->second == it){

ma.erase(mit->first);

dq.pop\_back();

break;

}

}

}

dq.push\_front(value);//make new entry in cache dq

ma[key]=dq.begin();

}

//printqueue();

}

Easy Implementation

**#include<list>**

**using** **namespace** std;

//map contains key and value and list contains only key, having most recent key accesses are at rear end

map**<int**,**int>** mp;

**int** sz;

list**<int>** lst;

LRUCache**::**LRUCache(**int** capacity) {

//clear map and list, since these are global variables

mp.clear();

sz**=**capacity;

lst.clear();

}

**int** LRUCache**::**get(**int** key) {

//search key in map, if not present return -1

**if**(mp.find(key)**==**mp.end()) **return** **-**1;

//if found, remove key from list,

//add to rear end of list,

//then return value from map

lst.remove(key);

lst.push\_back(key);

**return** mp[key];

}

**void** LRUCache**::**set(**int** key, **int** value) {

//if size of cache is zero, set not possible

**if**(sz**==**0) **return**;

//if key found in map

**if**(mp.find(key)**!=**mp.end())

{ //remove key from list

lst.remove(key);

//add to rear end of list

lst.push\_back(key);

//update value in map

mp[key]**=**value;

**return**;

}

//if key not found and size of cache is less than capacity

**if**(lst.size()**<**sz)

{ //push key to list

lst.push\_back(key);

//update value in map

mp[key]**=**value;

**return**;

}

//when cache is full, remove LR entry

//find LR entry in map

map**<int**,**int>::**iterator it**=**mp.find(lst.front());

//erase LR entry from map

mp.erase(it);

//remove key from list

lst.pop\_front();

//add key to list

lst.push\_back(key);

//update value in map

mp[key]**=**value;

}

## **Ways to form Max Heap**

* Asked in:
* [Directi](https://www.interviewbit.com/search/?q=Directi)

Problem Setter: aayushkapadia Problem Tester: sneh\_gupta

BookmarkSuggest Edit

**Max Heap** is a special kind of **complete binary tree** in which for every node the value present in that node is greater than the value present in it’s children nodes. If you want to know more about Heaps, please visit this [link](https://en.wikipedia.org/wiki/Heap_%28data_structure%29)

So now the problem statement for this question is:

**How many distinct Max Heap can be made from n distinct integers**

In short, you have to ensure the following properties for the max heap :

* Heap has to be a complete binary tree ( A complete binary tree is a binary tree in which every level, except possibly the last, is completely filled, and all nodes are as far left as possible. )
* Every node is greater than all its children

int height(int n){

int h=0;

while(n > 1){

h++;

n=n/2;

}

return h;

}

int nCr(int n, int r){

long long int C[r+1];

memset(C, 0, sizeof(C));

C[0]=1;

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

for(int j=min(i,r); j>0; j--){

C[j]=(C[j]+C[j-1]);

C[j]=C[j]%1000000007;

}

}

return C[r];

}

int Solution::solve(int A) {

if(A==0 || A==1)

return 1;

int h=height(A);

int m=pow(2,h-1);//half of max possible leaves

int p=A-pow(2,h)+1;//actual no. of nodes present at level h, i.e. leaf nodes

int l=m-1+min(m,p);//no. of nodes i lleft subtree, till level h-1,

//starting from root of left subtree (at height 0),

//hence level will become h-2 and no. of nodes are 2^0+2^1+...+2^(h-2)=2^(h-1)-1=m-1

//min(m,p) is min(max leaves in left subtree, actual leaves present)

int r=m-1+max(0,p-m);//p-m:= total leaves - half of leaves when p>= m

long long int C=nCr(A-1,l);

long long int x=solve(l);

long long int y=solve(r);

return (((C\*x)%1000000007)\*y)%1000000007;

}

Bit Manipulation

**Bit Manipulation**

## Tricks With Bits

* x & (x-1) will clear the lowest set bit of x
* x & ~(x-1) extracts the lowest set bit of x (all others are clear). Pretty patterns when applied to a linear sequence.
* x & (x + (1 << n)) = x, with the run of set bits (possibly length 0) starting at bit n cleared.
* x & ~(x + (1 << n)) = the run of set bits (possibly length 0) in x, starting at bit n.
* x | (x + 1) = x with the lowest cleared bit set.
* x | ~(x + 1) = extracts the lowest cleared bit of x (all others are set).
* x | (x - (1 << n)) = x, with the run of cleared bits (possibly length 0) starting at bit n set.
* x | ~(x - (1 << n)) = the lowest run of cleared bits (possibly length 0) in x, starting at bit n are the only clear bits.

## Min XOR value

* Asked in:
* [Booking.com](https://www.interviewbit.com/search/?q=Booking.com)
* Given an array of *N* integers, find the pair of integers in the array which have minimum XOR value. Report the minimum XOR value.
* **Examples :**   
  **Input**   
  0 2 5 7   
  **Output**   
  2 (0 XOR 2)   
  **Input**   
  0 4 7 9   
  **Output**   
  3 (4 XOR 7)
* **Constraints:**   
  2 <= N <= 100 000   
  0 <= A[i] <= 1 000 000 000

Approach: The first step is to sort the array. The answer will be the minimal value of X[i] XOR X[i+1] for every i.

int Solution::findMinXor(vector<int> &A) {

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

int minXor=A[0] xor A[1];

for(int i=1; i<A.size()-1; i++){

minXor=(A[i] xor A[i+1])<minXor?A[i] xor A[i+1]:minXor;

}

return minXor;

}

## Reverse Bits

* Asked in:
* [Nvidia](https://www.interviewbit.com/search/?q=Nvidia)
* [HCL](https://www.interviewbit.com/search/?q=HCL)
* [Amazon](https://www.interviewbit.com/search/?q=Amazon)

Reverse bits of an 32 bit unsigned integer

**Example 1:**

x = 0,

00000000000000000000000000000000

=> 00000000000000000000000000000000

return 0

**Example 2:**

x = 3,

00000000000000000000000000000011

=> 11000000000000000000000000000000

return 3221225472

**Since java does not have unsigned int, use long**

unsigned int Solution::reverse(unsigned int A) {

unsigned int res=0;

int i=0;

while(A){

if(A%2){

res=res+pow(2,(31-i));

}

A/=2;

i++;

}

return res;

}

## Number of 1 Bits

* Asked in:
* [Adobe](https://www.interviewbit.com/search/?q=Adobe)
* [Yahoo](https://www.interviewbit.com/search/?q=Yahoo)

Write a function that takes an unsigned integer and returns the number of 1 bits it has.

**Example:**

The 32-bit integer 11 has binary representation

00000000000000000000000000001011

so the function should return 3.

*Note that since Java does not have unsigned int, use long for Java*

Approach:

Notice what x - 1 does to bit representation of x.  
x - 1 would find the first set bit from the end, and then set it to 0, and set all the bits following it.

Which means if x = 10101001010100

^

|

|

|

First set bit from the end

Then x - 1 becomes 10101001010(011)

All other bits in x - 1 remain unaffected.  
This means that if we do (x & (x - 1)), it would just unset the last set bit in x *(which is why x&(x-1) is 0 for powers of 2)*.

Notice what x - 1 does to bit representation of x.  
x - 1 would find the first set bit from the end, and then set it to 0, and set all the bits following it.

Which means if x = 10101001010100

^

|

|

|

First set bit from the end

Then x - 1 becomes 10101001010(011)

All other bits in x - 1 remain unaffected.  
This means that if we do (x & (x - 1)), it would just unset the last set bit in x *(which is why x&(x-1) is 0 for powers of 2)*.

## Divide Integers

* Asked in:
* [Microsoft](https://www.interviewbit.com/search/?q=Microsoft)
* [Amazon](https://www.interviewbit.com/search/?q=Amazon)
* Divide two integers without using multiplication, division and mod operator.
* Return the floor of the result of the division.
* **Example:**
* 5 / 2 = 2
* **Also, consider if there can be overflow cases. For overflow case, return INT\_MAX.**

## Single Number II

* Asked in:
* [Google](https://www.interviewbit.com/search/?q=Google)
* [Amazon](https://www.interviewbit.com/search/?q=Amazon)
* Given an array of integers, every element appears thrice except for one which occurs once.
* Find that element which does not appear thrice.
* *Note: Your algorithm should have a linear runtime complexity.*
* Could you implement it without using extra memory?
* **Example :**
* Input : [1, 2, 4, 3, 3, 2, 2, 3, 1, 1]
* Output : 4

int Solution::singleNumber(const vector<int> &A) {

int ones=0, twos=0, threes=0;

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

twos |=ones & A[i];

ones= ones ^ A[i];

threes= ones & twos;

ones &=~threes;

twos &=~threes;

}

return ones;

}

## Single Number I

Given an array of integers, every element appears twice except for one. Find that single one.

*Note: Your algorithm should have a linear runtime complexity. Could you implement it without using extra memory?*

**Example :**

Input : [1 2 2 3 1]

Output : 3

int Solution::singleNumber(const vector<int> &A) {

int ones=0;

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

ones= ones ^ A[i];

}

return ones;

}

# Trees

## Next Greater Number BST / Inorder Successor

Given a BST node, return the node which has value just greater than the given node.

**1)** If right subtree of node is not NULL, then succ lies in right subtree. Do following.  
Go to right subtree and return the node with minimum key value in right subtree.  
**2)**If right sbtree of node is NULL, then start from root and use search like technique. Do following.  
Travel down the tree, if a node’s data is greater than root’s data then go right side, otherwise go to left side.

TreeNode**\*** Solution**::**getSuccessor(TreeNode**\***root, **int** B) {

TreeNode **\*** curr **=** root;

TreeNode **\***suc **=** NULL;

**while**(curr)

{

**if**(curr**->**val **>** B)

{

suc **=** curr;

curr **=** curr**->**left;

}**else**

{

curr **=** curr**->**right;

}

}

**return** suc;

}

Here condition 1) is also covered in condition 2).

## Valid BST

Efficient Method 1:

int isBSTUtil(node\* node, int min, int max);

/\* Returns true if the given

tree is a binary search tree

(efficient version). \*/

int isBST(node\* node)

{

    return(isBSTUtil(node, INT\_MIN, INT\_MAX));

}

/\* Returns true if the given

tree is a BST and its values

are >= min and <= max. \*/

int isBSTUtil(node\* node, int min, int max)

{

    /\* an empty tree is BST \*/

    if (node==NULL)

        return 1;

    /\* false if this node violates

    the min/max constraint \*/

    if (node->data < min || node->data > max)

        return 0;

    /\* otherwise check the subtrees recursively,

    tightening the min or max constraint \*/

    return

        isBSTUtil(node->left, min, node->data-1) && // Allow only distinct //values

        isBSTUtil(node->right, node->data+1, max); // Allow only distinct //values

}

Method 2:

int leftMax(TreeNode \*node){

if(!node) return INT\_MIN;

while(node->right)

node=node->right;

return node->val;

}

int rightMin(TreeNode \*node){

if(!node) return INT\_MAX;

while(node->left)

node=node->left;

return node->val;

}

/\*\*

\* Definition for binary tree

\* struct TreeNode {

\* int val;

\* TreeNode \*left;

\* TreeNode \*right;

\* TreeNode(int x) : val(x), left(NULL), right(NULL) {}

\* };

\*/

int Solution::isValidBST(TreeNode\* root) {

if(!root) return true; //Null tree is valid BST

if(root->left && root->left->val>=root->val) return false;

if(root->right && root->right->val<=root->val) return false;

if(!isValidBST(root->left) || !(isValidBST(root->right)))

return false;

int lmax=leftMax(root->left);

int rmin=rightMin(root->right);

if(lmax>=root->val || rmin<=root->val) return false;

return true;

}

## Inorder without recursion

Method 1:

/\* Iterative function for inorder tree

   traversal \*/

void inOrder(struct Node \*root)

{

    stack<Node \*> s;

    Node \*curr = root;

    while (curr != NULL || s.empty() == false)

    {

        /\* Reach the left most Node of the

           curr Node \*/

        while (curr !=  NULL)

        {

            /\* place pointer to a tree node on

               the stack before traversing

              the node's left subtree \*/

            s.push(curr);

            curr = curr->left;

        }

        /\* Current must be NULL at this point \*/

        curr = s.top();

        s.pop();

        cout << curr->data << " ";

        /\* we have visited the node and its

           left subtree.  Now, it's right

           subtree's turn \*/

        curr = curr->right;

    } /\* end of while \*/

}

/\* Function to traverse the binary tree without recursion and

   without stack \*/

void MorrisTraversal(struct tNode\* root)

{

    struct tNode \*current, \*pre;

    if (root == NULL)

        return;

    current = root;

    while (current != NULL) {

        if (current->left == NULL) {

            printf("%d ", current->data);

            current = current->right;

        }

        else {

            /\* Find the inorder predecessor of current \*/

            pre = current->left;

            while (pre->right != NULL && pre->right != current)

                pre = pre->right;

            /\* Make current as the right child of its inorder

               predecessor \*/

            if (pre->right == NULL) {

                pre->right = current;

                current = current->left;

            }

            /\* Revert the changes made in the 'if' part to restore

               the original tree i.e., fix the right child

               of predecessor \*/

            else {

                pre->right = NULL;

                printf("%d ", current->data);

                current = current->right;

            } /\* End of if condition pre->right == NULL \*/

        } /\* End of if condition current->left == NULL\*/

    } /\* End of while \*/

}

## Preorder Traversal

// An iterative process to print preorder traversal of Binary tree

void iterativePreorder(node \*root)

{

    // Base Case

    if (root == NULL)

       return;

    // Create an empty stack and push root to it

    stack<node \*> nodeStack;

    nodeStack.push(root);

    /\* Pop all items one by one. Do following for every popped item

       a) print it

       b) push its right child

       c) push its left child

    Note that right child is pushed first so that left is processed first \*/

    while (nodeStack.empty() == false)

    {

        // Pop the top item from stack and print it

        struct node \*node = nodeStack.top();

        printf ("%d ", node->data);

        nodeStack.pop();

        // Push right and left children of the popped node to stack

        if (node->right)

            nodeStack.push(node->right);

        if (node->left)

            nodeStack.push(node->left);

    }

}

## Postorder Traversal

Asked in:

* [Amazon](https://www.interviewbit.com/search/?q=Amazon)
* [Microsoft](https://www.interviewbit.com/search/?q=Microsoft)

Given a binary tree, return the postorder traversal of its nodes’ values.

**Example :**

Given binary tree

1

\

2

/

3

return [3,2,1].

**Using recursion is not allowed.**

void postOrderIterative(Node\* root)

{

    if (root == NULL)

        return;

    // Create two stacks

    stack<Node \*> s1, s2;

    // push root to first stack

    s1.push(root);

    Node\* node;

    // Run while first stack is not empty

    while (!s1.empty()) {

        // Pop an item from s1 and push it to s2

        node = s1.top();

        s1.pop();

        s2.push(node);

        // Push left and right children

        // of removed item to s1

        if (node->left)

            s1.push(node->left);

        if (node->right)

            s1.push(node->right);

    }

    // Print all elements of second stack

    while (!s2.empty()) {

        node = s2.top();

        s2.pop();

        cout << node->data << " ";

    }

}

Method 2:

Approach:

Recursive call would look something like this :

print(root->val);  
preorderprint(root->left);  
preorderprint(root->right);

If we use stack, then root->right must be pushed before root->left, so that left subtree will be processed first.

Create an empty stack, Push root node to the stack.  
Do following while stack is not empty.

2.1. pop an item from the stack and print it.

2.2. push the left child of popped item to stack.

2.3. push the right child of popped item to stack.

class Solution {

public:

vector<int> postorderTraversal(TreeNode \*root) {

stack<TreeNode\*> nodeStack;

vector<int> result;

//base case

if(root==NULL)

return result;

nodeStack.push(root);

while(!nodeStack.empty()) {

TreeNode\* node= nodeStack.top();

result.push\_back(node->val);

nodeStack.pop();

if(node->left)

nodeStack.push(node->left);

if(node->right)

nodeStack.push(node->right);

}

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

return result;

}

## Vertical Order traversal of Binary Tree

Asked in:

* [Amazon](https://www.interviewbit.com/search/?q=Amazon)

Problem Setter: yashpal1995 Problem Tester: [RAMBO\_tejasv](https://www.interviewbit.com/profile/RAMBO_tejasv)

BookmarkSuggest Edit

Given a binary tree, return a 2-D array with vertical order traversal of it.  
Go through the example and image for more details.

**Example :**  
Given binary tree:

6

/ \

3 7

/ \ \

2 5 9

returns

[

[2],

[3],

[6 5],

[7],

[9]

]

*/\**

*\* This problem can also be done in O(n) instead of O(n \* logn) by adding*

*\* one more traversal to compute minimum & maximum horizontal distance, start root's distance with (-minimumDistance) instead of '0'.*

*\*/*

/\*\*

\* Definition for binary tree

\* struct TreeNode {

\* int val;

\* TreeNode \*left;

\* TreeNode \*right;

\* TreeNode(int x) : val(x), left(NULL), right(NULL) {}

\* };

\*/

vector**<**vector**<int>** **>** Solution**::**verticalOrderTraversal(TreeNode**\*** A) {

vector**<**vector**<int>** **>** result;

*// Base case*

**if** (**!**A) **return** result;

*// Create a map of horizontalDistance, ListOfTreeNodes(at that horizontalDistance)*

map **<int**,vector**<int>** **>** m;

*// Create queue to do level order traversal.*

*// Every item of queue contains node and horizontal distance.*

queue**<**pair**<**TreeNode**\***, **int>** **>** q;

q.push(make\_pair(A, 0)); *// root's horizontalDistance is zero*

**while**(**!**q.empty()) {

*// get the queue front*

pair**<**TreeNode**\***,**int>** front **=** q.front();

q.pop();

**int** currentHorizontalDistance **=** front.second;

TreeNode**\*** currentTreeNode **=** front.first;

*// insert current node to hash map*

m[currentHorizontalDistance].push\_back(currentTreeNode **->** val);

**if** (currentTreeNode **->** left **!=** NULL) {

q.push(make\_pair(currentTreeNode **->** left, currentHorizontalDistance **-** 1));

}

**if** (currentTreeNode **->** right **!=** NULL) {

q.push(make\_pair(currentTreeNode **->** right, currentHorizontalDistance **+** 1));

}

}

*// Traverse the map and print nodes at every horigontal distance*

**for** (map**<int**,vector**<int>** **>** **::** iterator it **=** m.begin(); it **!=** m.end(); it**++**) {

result.push\_back(it **->** second);

}

**return** result;

}

## ZigZag Level Order Traversal BT

Asked in:

* [Amazon](https://www.interviewbit.com/search/?q=Amazon)
* [Microsoft](https://www.interviewbit.com/search/?q=Microsoft)

Given a binary tree, return the zigzag level order traversal of its nodes’ values. (ie, from left to right, then right to left for the next level and alternate between).

**Example :**   
Given binary tree

3

/ \

9 20

/ \

15 7

return

[

[3],

[20, 9],

[15, 7]

]

Sol1:

\* };

\*/

vector<vector<int> > Solution::zigzagLevelOrder(TreeNode\* root) {

vector<vector<int>> res;

if(!root) return res;

map<int, vector<int>> m;

queue<pair<TreeNode\*,int>> q;

q.push(make\_pair(root,0));

while(!q.empty()){

pair<TreeNode\*,int> front=q.front();

TreeNode \*node=front.first;

int level=front.second;

q.pop();

m[level].push\_back(node->val);

if(node->left)

q.push(make\_pair(node->left, level+1));

if(node->right)

q.push(make\_pair(node->right, level+1));

}

for(map<int, vector<int>>::iterator it=m.begin(); it!=m.end(); ++it){

if(it->first%2)

reverse(it->second.begin(), it->second.end());

res.push\_back(it->second);

}

return res;

}

Sol2: Simple

//Sol 2:

vector<vector<int>> res;

if(!root) return res;

stack<TreeNode\*> s1, s2;

int level=1;

s1.push(root);

vector<int> vec;

while(!s1.empty()){

TreeNode \*node=s1.top();

s1.pop();

vec.push\_back(node->val);

if(!level){

if(node->right) s2.push(node->right);

if(node->left) s2.push(node->left);

}

else{

if(node->left) s2.push(node->left);

if(node->right) s2.push(node->right);

}

if(s1.empty()){

s1.swap(s2);

level ^=1;

res.push\_back(vec);

vec.clear();

}

}

return res;

## Populate Next Right Pointers Tree / Connect nodes at same level

* Asked in:
* [Microsoft](https://www.interviewbit.com/search/?q=Microsoft)
* [Amazon](https://www.interviewbit.com/search/?q=Amazon)

BookmarkSuggest Edit

Given a binary tree

**struct** TreeLinkNode {

TreeLinkNode \*left;

TreeLinkNode \*right;

TreeLinkNode \*next;

}

Populate each next pointer to point to its next right node. If there is no next right node, the next pointer should be set to NULL.

Initially, all next pointers are set to NULL.

**Example :**

Given the following binary tree,

1

/ \

2 3

/ \ / \

4 5 6 7

After calling your function, the tree should look like:

1 -> NULL

/ \

2 -> 3 -> NULL

/ \ / \

4->5->6->7 -> NULL

*/\*\**

*\* Definition for binary tree with next pointer.*

*\* struct TreeLinkNode {*

*\* int val;*

*\* TreeLinkNode \*left, \*right, \*next;*

*\* TreeLinkNode(int x) : val(x), left(NULL), right(NULL), next(NULL) {}*

*\* };*

*\*/*

**void** Solution**::**connect(TreeLinkNode**\*** root) {

**while**(root)//loop run for number of levels times

{

TreeLinkNode**\*** s**=new** TreeLinkNode(0);

TreeLinkNode**\*** n**=**s;

**while**(root) //loop to traverse all nodes at a level

{

**if**(root**->**left)

{

n**->**next**=**root**->**left;

n**=**n**->**next;

}

**if**(root**->**right)

{

n**->**next**=**root**->**right;

n**=**n**->**next;

}

root**=**root**->**next;

}

root**=**s**->**next;

}

}

Sol2:

Using queue maintains order of children from left to right at level, stack doesn’t do that.

void Solution::connect(TreeLinkNode\* root) {

**if**(!root) return;

queue<TreeLinkNode\*>s1, s2;

s1.push(root);

**while**(!s1.empty()){

TreeLinkNode \*node=s1.front();

s1.pop();

**if**(node->left) s2.push(node->left);

**if**(node->right) s2.push(node->right);

**if**(s1.empty()){

node->next=NULL;

s1.swap(s2);

}

**else**{

node->next=s1.front();

}

}

}

## Balanced Binary Tree

* Asked in:
* [Amazon](https://www.interviewbit.com/search/?q=Amazon)

BookmarkSuggest Edit

Given a binary tree, determine if it is height-balanced.

***Height-balanced binary tree****: is defined as a binary tree in which the depth of the two subtrees of every node never differ by more than 1.*

Return 0 / 1 ( 0 for false, 1 for true ) for this problem

bool balance\_flag;

// depth function of binary tree is modified here

int depthCheck(TreeNode \*node){

if(!balance\_flag) return 0;

if(!node) return 0;

int leftdepth=depthCheck(node->left);

int rightdepth=depthCheck(node->right);

if(abs(leftdepth-rightdepth) >= 2){

balance\_flag=false;

return 0;

}

return 1+max(leftdepth,rightdepth);

}

/\*\*

\* Definition for binary tree

\* struct TreeNode {

\* int val;

\* TreeNode \*left;

\* TreeNode \*right;

\* TreeNode(int x) : val(x), left(NULL), right(NULL) {}

\* };

\*/

int Solution::isBalanced(TreeNode\* root) {

balance\_flag=true;

depthCheck(root);

return balance\_flag;

}

## Identical Binary Trees

* Asked in:
* [Amazon](https://www.interviewbit.com/search/?q=Amazon)

BookmarkSuggest Edit

Given two binary trees, write a function to check if they are equal or not.

Two binary trees are considered equal if they are structurally identical and the nodes have the same value.

Return 0 / 1 ( 0 for false, 1 for true ) for this problem

/\*\*

\* Definition for binary tree

\* struct TreeNode {

\* int val;

\* TreeNode \*left;

\* TreeNode \*right;

\* TreeNode(int x) : val(x), left(NULL), right(NULL) {}

\* };

\*/

int Solution::isSameTree(TreeNode\* A, TreeNode\* B) {

if(!A && !B) return 1;

if(!A || !B) return 0; //1&&1 is already covered above

return (A->val==B->val && isSameTree(A->left,B->left) && isSameTree(A->right, B->right));

}

## Symmetric Binary Tree Asked in:

* [Amazon](https://www.interviewbit.com/search/?q=Amazon)

BookmarkSuggest Edit

Given a binary tree, check whether it is a mirror of itself (ie, symmetric around its center).

**Example :**

1

/ \

2 2

/ \ / \

3 4 4 3

The above binary tree is symmetric.   
But the following is not:

1

/ \

2 2

\ \

3 3

//Solution

int isMirror(TreeNode \*A, TreeNode \*B){

if(!A && !B) return 1;

if(!A || !B) return 0;

return ((A->val == B->val) && isMirror(A->left, B->right) && isMirror(A->right, B->left));

}

/\*\*

\* Definition for binary tree

\* struct TreeNode {

\* int val;

\* TreeNode \*left;

\* TreeNode \*right;

\* TreeNode(int x) : val(x), left(NULL), right(NULL) {}

\* };

\*/

int Solution::isSymmetric(TreeNode\* A) {

if(!A) return 1;

return isMirror(A->left, A->right);

}

## Sorted Array To Balanced BST

* Asked in:
* [VMWare](https://www.interviewbit.com/search/?q=VMWare)
* [Amazon](https://www.interviewbit.com/search/?q=Amazon)
* Given an array where elements are sorted in ascending order, convert it to a height balanced BST.
* ***Balanced tree :****a height-balanced binary tree is defined as a binary tree in which the depth of the two subtrees of every node never differ by more than 1.*
* **Example :**
* Given A : [1, 2, 3]
* A height balanced BST :
* 2
* / \
* 1 3

TreeNode\* createBST(const vector<int> &A, int l, int r){

/\* Base Case \*/

    if (start > end)

    return NULL;

    /\* Get the middle element and make it root \*/

    int mid = (start + end)/2;

    TNode \*root = newNode(arr[mid]);

    /\* Recursively construct the left subtree

    and make it left child of root \*/

    root->left = sortedArrayToBST(arr, start,

                                    mid - 1);

    /\* Recursively construct the right subtree

    and make it right child of root \*/

    root->right = sortedArrayToBST(arr, mid + 1, end);

    return root;

}

/\*\*

\* Definition for binary tree

\* struct TreeNode {

\* int val;

\* TreeNode \*left;

\* TreeNode \*right;

\* TreeNode(int x) : val(x), left(NULL), right(NULL) {}

\* };

\*/

TreeNode\* Solution::sortedArrayToBST(const vector<int> &A) {

// Do not write main() function.

// Do not read input, instead use the arguments to the function.

// Do not print the output, instead return values as specified

// Still have a doubt. Checkout www.interviewbit.com/pages/sample\_codes/ for more details

int n=A.size();

int l=0, r=n-1;

return createBST(A, l, r);

}

## Inorder Traversal of Cartesian Tree

Asked in:

* [Amazon](https://www.interviewbit.com/search/?q=Amazon)

Given an inorder traversal of a cartesian tree, construct the tree.

***Cartesian tree****: is a heap ordered binary tree, where the root is greater than all the elements in the subtree.*

***Note:****You may assume that duplicates do not exist in the tree.*

**Example :**

Input : [1 2 3]

Return :

3

/

2

/

1

void printTree(TreeNode \* root){

if (!root) return;

printTree(root->left);

cout<<root->val<<" ";

printTree(root->right);

}

int findMax(vector<int> &A, int l, int r){

int max\_idx=l;

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

max\_idx=(A[max\_idx]>A[i])?max\_idx:i;

return max\_idx;

}

TreeNode \* formTree(vector<int> &A, int l, int r){

TreeNode \*root=NULL;

if(l <= r){

int root\_idx = findMax(A, l, r);

root=new TreeNode(A[root\_idx]);

root->left=formTree(A,l, root\_idx-1);

root->right=formTree(A, root\_idx+1, r);

}

return root;

}

/\*\*

\* Definition for binary tree

\* struct TreeNode {

\* int val;

\* TreeNode \*left;

\* TreeNode \*right;

\* TreeNode(int x) : val(x), left(NULL), right(NULL) {}

\* };

\*/

TreeNode\* Solution::buildTree(vector<int> &A) {

TreeNode\* root=formTree(A, 0, A.size()-1);

return root;

}

## Binary Tree From Inorder And Postorder

* Asked in:
* [Amazon](https://www.interviewbit.com/search/?q=Amazon)
* [Microsoft](https://www.interviewbit.com/search/?q=Microsoft)

Given inorder and postorder traversal of a tree, construct the binary tree.

***Note:****You may assume that duplicates do not exist in the tree.*

**Example :**

Input :

Inorder : [2, 1, 3]

Postorder : [2, 3, 1]

Return :

1

/ \

2 3

TreeNode\* buildTreeUtil(vector<int> &in, int l, int r, vector<int> &po, int &pIndex, map<int,int> &m){

if(l>r) return NULL;

int root\_val=po[pIndex];//root of subtree found

TreeNode \*root= new TreeNode(root\_val); //create root node

(pIndex)--; //do not consider this index in further BT creation

if(l==r) return root;

int iIndex=m[root\_val];

root->right=buildTreeUtil(in, iIndex+1, r, po, pIndex, m);

root->left=buildTreeUtil(in, l, iIndex-1, po, pIndex, m);

return root;

}

/\*\*

\* Definition for binary tree

\* struct TreeNode {

\* int val;

\* TreeNode \*left;

\* TreeNode \*right;

\* TreeNode(int x) : val(x), left(NULL), right(NULL) {}

\* };

\*/

TreeNode\* Solution::buildTree(vector<int> &inOrder, vector<int> &postOrder) {

map<int,int> m; //mapping inorder elements to their indices

//for easy search

int n=inOrder.size();

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

m[inOrder[i]]=i;

int pIndex=n-1; //post order index where root lies

return buildTreeUtil(inOrder, 0, n-1, postOrder, pIndex, m);

}

## Construct Binary Tree From Inorder And Preorder

* Asked in:
* [Amazon](https://www.interviewbit.com/search/?q=Amazon)
* [Microsoft](https://www.interviewbit.com/search/?q=Microsoft)

BookmarkSuggest Edit

Given preorder and inorder traversal of a tree, construct the binary tree.

***Note:****You may assume that duplicates do not exist in the tree.*

**Example :**

Input :

Preorder : [1, 2, 3]

Inorder : [2, 1, 3]

Return :

1

/ \

2 3

TreeNode \*buildTreeUtil(vector<int> &in, int l, int r, vector<int> &pre, int &preIndex, map<int,int> &m){

if(l > r) return NULL;

int root\_val=pre[preIndex];

TreeNode \* root=new TreeNode(root\_val);

preIndex++;

if(l==r) return root;

int iIndex=m[root\_val];

root->left=buildTreeUtil(in, l, iIndex-1, pre, preIndex, m);

//cout<<root->left->val<<endl;

root->right=buildTreeUtil(in, iIndex+1, r, pre, preIndex, m);

return root;

}

/\*\*

\* Definition for binary tree

\* struct TreeNode {

\* int val;

\* TreeNode \*left;

\* TreeNode \*right;

\* TreeNode(int x) : val(x), left(NULL), right(NULL) {}

\* };

\*/

TreeNode\* Solution::buildTree(vector<int> &pre, vector<int> &in) {

int preIndex=0;

int n=pre.size();

map<int,int> m;

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

m[in[i]]=i;

}

return buildTreeUtil(in, 0, n-1, pre, preIndex, m);

}

## Path Sum

* Asked in:
* [Microsoft](https://www.interviewbit.com/search/?q=Microsoft)
* [Yahoo](https://www.interviewbit.com/search/?q=Yahoo)
* [Amazon](https://www.interviewbit.com/search/?q=Amazon)

Given a binary tree and a sum, determine if the tree has a root-to-leaf path such that adding up all the values along the path equals the given sum.

**Example :**

Given the below binary tree and sum = 22,

5

/ \

4 8

/ / \

11 13 4

/ \ \

7 2 1

return true, as there exist a root-to-leaf path 5->4->11->2 which sum is 22.

**Return 0 / 1 ( 0 for false, 1 for true ) for this problem**

/\*\*

\* Definition for binary tree

\* struct TreeNode {

\* int val;

\* TreeNode \*left;

\* TreeNode \*right;

\* TreeNode(int x) : val(x), left(NULL), right(NULL) {}

\* };

\*/

Sol1:

Since sum should be on path to leaf, summing given value at intermediate node should return false.

int Solution::hasPathSum(TreeNode\* root, int sum) {

if(!root) return 0; //if you return sum==0 here, the test will fail when //node has 1 child and other null child pointer will return 1;

if((sum==root->val) && !root->left && !root->right) return 1;

return hasPathSum(root->left,sum-root->val)||hasPathSum(root->right,sum-root->val);

}

Sol2:

int Solution::hasPathSum(TreeNode\* root, int sum) {

if(!root) return (sum==0);

if((sum==root->val) && !root->left && !root->right) return 1;

int ans=0;

if(root->left)

ans = ans || hasPathSum(root->left, sum-root->val);

if(root->right)

ans = ans || hasPathSum(root->right, sum-root->val);

return ans;

}

## Root to Leaf Paths With Sum

* Asked in:
* [Microsoft](https://www.interviewbit.com/search/?q=Microsoft)
* [Yahoo](https://www.interviewbit.com/search/?q=Yahoo)
* [Amazon](https://www.interviewbit.com/search/?q=Amazon)

Given a binary tree and a sum, find all root-to-leaf paths where each path’s sum equals the given sum.

For example:  
Given the below binary tree and sum = 22,

5

/ \

4 8

/ / \

11 13 4

/ \ / \

7 2 5 1

return

[

[5,4,11,2],

[5,8,4,5]

]

vector<int> path;

void pathSumUtil(TreeNode \*node, int sum, vector<vector<int>> &res, int sum\_so\_far)

{

if(!node) return;

path.push\_back(node->val);

sum\_so\_far +=node->val;

if(!node->left && !node->right && sum\_so\_far==sum)

res.push\_back(path); //do not return after this as path vector should //get clear by end path.pop\_back()

if(node->left)

pathSumUtil(node->left, sum, res, sum\_so\_far);

if(node->right)

pathSumUtil(node->right, sum, res, sum\_so\_far);

path.pop\_back();

}

/\*\*

\* Definition for binary tree

\* struct TreeNode {

\* int val;

\* TreeNode \*left;

\* TreeNode \*right;

\* TreeNode(int x) : val(x), left(NULL), right(NULL) {}

\* };

\*/

vector<vector<int> > Solution::pathSum(TreeNode\* root, int sum) {

vector<vector<int>> res;

int sum\_so\_far=0;

if(!root) return res;

pathSumUtil(root, sum, res, sum\_so\_far);

return res;

}

## Min Depth of Binary Tree

* Asked in:
* [Facebook](https://www.interviewbit.com/search/?q=Facebook)
* [Amazon](https://www.interviewbit.com/search/?q=Amazon)

Given a binary tree, find its minimum depth.

The minimum depth is the number of nodes along the shortest path from the root node down to the nearest leaf node.

***NOTE :****The path has to end on a leaf node.*

**Example :**

1

/

2

min depth = 2.

/\*\*

\* Definition for binary tree

\* struct TreeNode {

\* int val;

\* TreeNode \*left;

\* TreeNode \*right;

\* TreeNode(int x) : val(x), left(NULL), right(NULL) {}

\* };

\*/

int Solution::minDepth(TreeNode\* root) {

if(!root) return 0;

if(!root->left && !root->right) return 1;

int ld=INT\_MAX, rd=INT\_MAX;

if(root->left) ld=minDepth(root->left);

if(root->right) rd=minDepth(root->right);

//cout<<(1+min(ld,rd))<<endl;

return 1+min(ld,rd);

}

## Sum Root to Leaf Numbers

* Asked in:
* [Google](https://www.interviewbit.com/search/?q=Google)
* [Microsoft](https://www.interviewbit.com/search/?q=Microsoft)
* Given a binary tree containing digits from 0-9 only, each root-to-leaf path could represent a number.
* An example is the root-to-leaf path 1->2->3 which represents the number 123.
* Find the total sum of all root-to-leaf numbers % 1003.
* **Example :**
* 1
* / \
* 2 3
* The root-to-leaf path 1->2 represents the number 12.  
  The root-to-leaf path 1->3 represents the number 13.
* Return the sum = (12 + 13) % 1003 = 25 % 1003 = 25.

/\*\*

\* Definition for binary tree

\* struct TreeNode {

\* int val;

\* TreeNode \*left;

\* TreeNode \*right;

\* TreeNode(int x) : val(x), left(NULL), right(NULL) {}

\* };

\*/

int sum(TreeNode\* root, int val){

if(root == NULL){

return 0;

}

val = (((val)%1003\*10)%1003 + (root->val)%1003)%1003;

if(root->left == NULL && root->right == NULL){

return val;

}

return (sum(root->left, val)%1003 + sum(root->right, val)%1003)%1003;

}

int Solution::sumNumbers(TreeNode\* A) {

// Do not write main() function.

// Do not read input, instead use the arguments to the function.

// Do not print the output, instead return values as specified

// Still have a doubt. Checkout www.interviewbit.com/pages/sample\_codes/ for more details

return sum(A, 0);

}

## Shortest Unique Prefix

Asked in:

* [Google](https://www.interviewbit.com/search/?q=Google)

Find shortest unique prefix to represent each word in the list.

**Example:**

Input: [zebra, dog, duck, dove]

Output: {z, dog, du, dov}

where we can see that

zebra = z

dog = dog

duck = du

dove = dov

#define ALPHABET\_SIZE 26

struct Trie{

int repeat;

Trie \*child[ALPHABET\_SIZE];

Trie(){

repeat=0;

for(int i=0; i<ALPHABET\_SIZE; i++)

child[i]=NULL;

}

};

void insert(Trie \*root, string s){

int n=s.size();

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

if(!root->child[s[i]-'a']) root->child[s[i]-'a']=new Trie();

root=root->child[s[i]-'a'];

root->repeat++;

}

}

int prefixFind(Trie \*node, string word){

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

node=node->child[word[i]-'a'];

if(node->repeat==1)

return i;

}

}

vector<string> Solution::prefix(vector<string> &input) {

vector<string> output;

Trie \*trie=new Trie();

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

insert(trie, input[i]);

}

//printTrie(trie);

for(vector<string>::iterator it=input.begin(); it!=input.end(); ++it){

string word=\*it;

int count=prefixFind(trie, word);

output.push\_back(word.substr(0,count+1)) ;

}

//print(output);

return output;

}

## Kth Smallest Element In Tree

Asked in:

* [Amazon](https://www.interviewbit.com/search/?q=Amazon)

Given a binary search tree, write a function to find the kth smallest element in the tree.

**Example :**

Input :

2

/ \

1 3

and k = 2

Return : 2

As 2 is the second smallest element in the tree.

int countSubTree(TreeNode \*node){

if(!node) return 0;

return countSubTree(node->left)+countSubTree(node->right)+1;

}

int kSmallestUtil(TreeNode \*node, int &k){

if(!node) return 0;

int res=kSmallestUtil(node->left,k);

if(k==0) return res;

k--;

if(k==0) return node->val;

res=kSmallestUtil(node->right,k);

if(k==0) return res;

return 0;

}

/\*\*

\* Definition for binary tree

\* struct TreeNode {

\* int val;

\* TreeNode \*left;

\* TreeNode \*right;

\* TreeNode(int x) : val(x), left(NULL), right(NULL) {}

\* };

\*/

int Solution::kthsmallest(TreeNode\* root, int k) {

if(!root) return 0;

return kSmallestUtil(root,k);

}

**Approach 2: Morris Traversal**

|  |
| --- |
| // C++ program to find k'th largest element in BST  #include<iostream>  #include<climits>  using namespace std;    // A BST node  struct Node  {      int key;      Node \*left, \*right;  };    // A function to find  int KSmallestUsingMorris(Node \*root, int k)  {      // Count to iterate over elements till we      // get the kth smallest number      int count = 0;        int ksmall = INT\_MIN; // store the Kth smallest      Node \*curr = root; // to store the current node        while (curr != NULL)      {          // Like Morris traversal if current does          // not have left child rather than printing          // as we did in inorder, we will just          // increment the count as the number will          // be in an increasing order          if (curr->left == NULL)          {              count++;                // if count is equal to K then we found the              // kth smallest, so store it in ksmall              if (count==k)                  ksmall = curr->key;                // go to current's right child              curr = curr->right;          }          else          {              // we create links to Inorder Successor and              // count using these links              Node \*pre = curr->left;              while (pre->right != NULL && pre->right != curr)                  pre = pre->right;                // building links              if (pre->right==NULL)              {                  //link made to Inorder Successor                  pre->right = curr;                  curr = curr->left;              }                // While breaking the links in so made temporary              // threaded tree we will check for the K smallest              // condition              else              {                  // Revert the changes made in if part (break link                  // from the Inorder Successor)                  pre->right = NULL;                    count++;                    // If count is equal to K then we found                  // the kth smallest and so store it in ksmall                  if (count==k)                      ksmall = curr->key;                    curr = curr->right;              }          }      }      return ksmall; //return the found value  } |

This isn’t feasible…

int countSubTree(TreeNode \*node){

if(!node) return 0;

return countSubTree(node->left)+countSubTree(node->right)+1;

}

/\*\*

\* Definition for binary tree

\* struct TreeNode {

\* int val;

\* TreeNode \*left;

\* TreeNode \*right;

\* TreeNode(int x) : val(x), left(NULL), right(NULL) {}

\* };

\*/

int Solution::kthsmallest(TreeNode\* root, int k) {

if(!root) return -1;

int lcount = countSubTree(root->left);

if(lcount+1 == k) return root->val;

if(k <= lcount)

return kthsmallest(root->left, k);

else

return kthsmallest(root->right, k-lcount-1);

}

Method 2:

**class** **Solution** {

**public**:

**int** find(TreeNode\* root, **int** &k) {

**if** (!root) **return** -1;

*// We do an inorder traversal here.*

**int** k1 = find(root->left, k);

**if** (k == 0) **return** k1; *// left subtree has k or more elements.*

k--;

**if** (k == 0) **return** root->val; *// root is the kth element.*

**return** find(root->right, k); *// answer lies in the right node.*

}

**int** kthsmallest(TreeNode\* root, **int** k) {

**return** find(root, k); *// Call another function to pass k by reference.*

}

};

## 2-Sum Binary Tree

* Asked in:
* [Amazon](https://www.interviewbit.com/search/?q=Amazon)

Given a binary search tree T, where each node contains a positive integer, and an integer K, you have to find whether or not there exist two different nodes A and B such that A.value + B.value = K.

Return 1 to denote that two such nodes exist. Return 0, otherwise.

**Notes**

* Your solution should run in linear time and not take memory more than O(height of T).
* Assume all values in BST are distinct.

**Example :**

Input 1:

T : 10

/ \

9 20

K = 19

Return: 1

Input 2:

T: 10

/ \

9 20

K = 40

Return: 0

**Approach:**

Required TC=O(n) and SC<=height of tree

Storing inorder completely will not help. Sum can be checked by summing smallest value and largest value. When calculated sum becomes smaller than required , replace smallest value by its successor and when calculated sum becomes greater than required one, replace large values by its predecessor.

**For Half memory storage**

int Solution::t2Sum(TreeNode\* root, int sum) {

if(!root) return 0;

TreeNode \*lptr, \*rptr;

stack<TreeNode\*> s1, s2;

lptr=rptr=root;

while(lptr){

s1.push(lptr);

lptr=lptr->left;

}

lptr=s1.top();

while(rptr){

s2.push(rptr);

rptr=rptr->right;

}

rptr=s2.top();

while(lptr && rptr && lptr->val < rptr->val){

if(lptr->val+rptr->val==sum)

return 1;

if(lptr->val+rptr->val < sum){

s1.pop();

lptr=lptr->right;

while(lptr){

s1.push(lptr);

lptr=lptr->left;

}

lptr=s1.top();

}

else{

s2.pop();

rptr=rptr->left;

while(rptr){

s2.push(rptr);

rptr=rptr->right;

}

rptr=s2.top();

}

}

return 0;

}

**Other solution:**

int find(TreeNode \*node, int sum, set<int> &s){

if(!node) return 0;

if(s.find(sum-node->val)!=s.end())

return 1;

s.insert(node->val);

return find(node->left, sum, s) || find(node->right, sum, s);

}

/\*\*

\* Definition for binary tree

\* struct TreeNode {

\* int val;

\* TreeNode \*left;

\* TreeNode \*right;

\* TreeNode(int x) : val(x), left(NULL), right(NULL) {}

\* };

\*/

int Solution::t2Sum(TreeNode\* root, int sum) {

if(!root) return 0;

set<int> nums;

return find(root, sum, nums);

}

## BST Iterator

* Asked in:
* [Apple](https://www.interviewbit.com/search/?q=Apple)
* [Amazon](https://www.interviewbit.com/search/?q=Amazon)
* [Facebook](https://www.interviewbit.com/search/?q=Facebook)

Implement an iterator over a binary search tree (BST). Your iterator will be initialized with the root node of a BST.

The first call to next() will return the smallest number in BST. Calling next() again will return the next smallest number in the BST, and so on.

***Note:****next() and hasNext() should run in average O(1) time and uses O(h) memory, where h is the height of the tree.  
Try to optimize the additional space complexity apart from the amortized time complexity.*

Push leftmost subtree to stack. If a node has right child, when popping the same, push its right leftmost subtree to stack.

stack<TreeNode\*> st;

void leftmostInorder(TreeNode \*node){

while(node){

st.push(node);

node=node->left;

}

}

/\*\*

\* Definition for binary tree

\* struct TreeNode {

\* int val;

\* TreeNode \*left;

\* TreeNode \*right;

\* TreeNode(int x) : val(x), left(NULL), right(NULL) {}

\* };

\*/

BSTIterator::BSTIterator(TreeNode \*root) {

while(!st.empty())

st.pop();

leftmostInorder(root);

}

/\*\* @return whether we have a next smallest number \*/

bool BSTIterator::hasNext() {

if(!st.empty()) return true;

return false;

}

/\*\* @return the next smallest number \*/

int BSTIterator::next() {

if(st.empty()) return -1;

TreeNode \*top=st.top();

st.pop();

leftmostInorder(top->right);

return top->val;

}

/\*\*

\* Your BSTIterator will be called like this:

\* BSTIterator i = BSTIterator(root);

\* while (i.hasNext()) cout << i.next();

\*/

## Recover Binary Search Tree

* Asked in:
* [Microsoft](https://www.interviewbit.com/search/?q=Microsoft)
* [Amazon](https://www.interviewbit.com/search/?q=Amazon)

Two elements of a binary search tree (BST) are swapped by mistake.  
Tell us the 2 values swapping which the tree will be restored.

***Note:*** *A solution using O(n) space is pretty straight forward. Could you devise a constant space solution?*

**Example :**

Input :

1

/ \

2 3

Output :

[1, 2]

Explanation : Swapping 1 and 2 will change the BST to be

2

/ \

1 3

which is a valid BST

// This function does inorder traversal to find out the two swapped nodes.

// It sets three pointers, first, middle and last.  If the swapped nodes are

// adjacent to each other, then first and middle contain the resultant nodes

// Else, first and last contain the resultant nodes

void correctBSTUtil( struct node\* root, struct node\*\* first,

                     struct node\*\* middle, struct node\*\* last,

                     struct node\*\* prev )

{

    if( root )

    {

        // Recur for the left subtree

        correctBSTUtil( root->left, first, middle, last, prev );

        // If this node is smaller than the previous node, it's violating

        // the BST rule.

        if (\*prev && root->data < (\*prev)->data)

        {

            // If this is first violation, mark these two nodes as

            // 'first' and 'middle'

            if ( !\*first )

            {

                \*first = \*prev;

                \*middle = root;

            }

            // If this is second violation, mark this node as last

            else

                \*last = root;

        }

        // Mark this node as previous

        \*prev = root;

        // Recur for the right subtree

        correctBSTUtil( root->right, first, middle, last, prev );

    }

}

// A function to fix a given BST where two nodes are swapped.  This

// function uses correctBSTUtil() to find out two nodes and swaps the

// nodes to fix the BST

void correctBST( struct node\* root )

{

    // Initialize pointers needed for correctBSTUtil()

    struct node \*first, \*middle, \*last, \*prev;

    first = middle = last = prev = NULL;

    // Set the pointers to find out two nodes

    correctBSTUtil( root, &first, &middle, &last, &prev );

    // Fix (or correct) the tree

    if( first && last )

        swap( &(first->data), &(last->data) );

    else if( first && middle ) // Adjacent nodes swapped

        swap( &(first->data), &(middle->data) );

    // else nodes have not been swapped, passed tree is really BST.

}

## Least Common Ancestor

* Asked in:
* [Facebook](https://www.interviewbit.com/search/?q=Facebook)
* [Adobe](https://www.interviewbit.com/search/?q=Adobe)
* [Microsoft](https://www.interviewbit.com/search/?q=Microsoft)
* [Amazon](https://www.interviewbit.com/search/?q=Amazon)
* [Google](https://www.interviewbit.com/search/?q=Google)

Find the lowest common ancestor in an unordered binary tree given two values in the tree.

***Lowest common ancestor :****the lowest common ancestor (LCA) of two nodes v and w in a tree or directed acyclic graph (DAG) is the lowest (i.e. deepest) node that has both v and w as descendants.*

**Example :**

\_\_\_\_\_\_\_3\_\_\_\_\_\_

/ \

\_\_\_5\_\_ \_\_\_1\_\_

/ \ / \

6 \_2\_ 0 8

/ \

7 4

For the above tree, the LCA of nodes 5 and 1 is 3.

***LCA****= Lowest common ancestor*

Please note that LCA for nodes 5 and 4 is 5.

Sol1:

void printvec(vector<int> v){

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

cout<<v[i]<<" ";

cout<<endl;

}

bool findPath(TreeNode \*node, int n, vector<int> &res){

if(!node) return false;

res.push\_back(node->val);

if(node->val == n) return true;

if(findPath(node->left, n, res) || findPath(node->right, n, res))

return true;

res.pop\_back();

return false;

}

/\*\*

\* Definition for binary tree

\* struct TreeNode {

\* int val;

\* TreeNode \*left;

\* TreeNode \*right;

\* TreeNode(int x) : val(x), left(NULL), right(NULL) {}

\* };

\*/

int Solution::lca(TreeNode\* root, int A, int B) {

vector<int> a, b;

findPath(root, A, a);

findPath(root, B, b);

//printvec(a);

//printvec(b);

if(a.size()>0 && b.size()>0){

for(int i=a.size()-1; i>=0; i--){

if(find(b.rbegin(), b.rend(),a[i])!=b.rend())

return a[i];

}

}

return -1;

}

Sol2:

class Solution {

public:

TreeNode \*LCA(TreeNode \*root, int val1, int val2) {

if (!root) return NULL;

if (root->val == val1 || root->val == val2) return root;

TreeNode \*L = LCA(root->left, val1, val2);

TreeNode \*R = LCA(root->right, val1, val2);

if (L && R) return root; // If val1, val2 are on both sides

return L ? L : R; // either one of val1, val2 is on one side OR val1, val2 is not in L&R subtrees

}

bool find(TreeNode \*root, int val1) {

if (!root) return false;

if (root->val == val1) return true;

return (find(root->left, val1) || find(root->right, val1));

}

int lca(TreeNode \*root, int val1, int val2) {

//finding node is imp, if this is omitted then code will fail

//when one of the val1 and val2 is not present in tree

//it will return node which is present as LCA

if (!find(root, val1) || !find(root, val2)) return -1;

TreeNode \*ans = LCA(root, val1, val2);

if (!ans) return -1;

return ans->val;

}

};

## Flatten Binary Tree to Linked List

* Asked in:
* [Adobe](https://www.interviewbit.com/search/?q=Adobe)
* [Amazon](https://www.interviewbit.com/search/?q=Amazon)
* [Microsoft](https://www.interviewbit.com/search/?q=Microsoft)

Given a binary tree, flatten it to a linked list in-place.

**Example :**  
Given

1

/ \

2 5

/ \ \

3 4 6

The flattened tree should look like:

1

\

2

\

3

\

4

\

5

\

6

Note that the left child of all nodes should be NULL.

/\*\*

\* Definition for binary tree

\* struct TreeNode {

\* int val;

\* TreeNode \*left;

\* TreeNode \*right;

\* TreeNode(int x) : val(x), left(NULL), right(NULL) {}

\* };

\*/

TreeNode\* Solution::flatten(TreeNode\* root) {

// Do not write main() function.

// Do not read input, instead use the arguments to the function.

// Do not print the output, instead return values as specified

// Still have a doubt. Checkout www.interviewbit.com/pages/sample\_codes/ for more details

if(!root) return NULL;

root->left=flatten(root->left);//flatten left subtree

root->right=flatten(root->right);//flatten right subtree

//attach left subtree to right child of root and

//attach right subtree to right child of left subtree

if(root->left){

TreeNode \*newpos=root->right;

root->right=root->left;

TreeNode \*tail=root->left;

root->left=NULL;

while(tail->right)

tail=tail->right;

tail->right=newpos;

}

return root;

}

Sol2:

lass Solution {

public:

void flatten(TreeNode \*root) {

if (!root) return;

TreeNode\* node = root;

while (node) {

// Attatches the right sub-tree to the rightmost leaf of the left sub-tree:

if (node->left) {

TreeNode \*rightMost = node->left;

while (rightMost->right) {

rightMost = rightMost->right;

}

rightMost->right = node->right;

// Makes the left sub-tree to the right sub-tree:

node->right = node->left;

node->left = NULL;

}

// Flatten the rest of the tree:

node = node->right;

}

}

};

## Order of People Heights

* Asked in:
* [Google](https://www.interviewbit.com/search/?q=Google)

You are given the following :

* A positive number N
* Heights : A list of heights of N persons standing in a queue
* Infronts : A list of numbers corresponding to each person (P) that gives the **number of persons**who are **taller** than P and standing in front of P

You need to return list of actual order of persons’s height

**Consider that heights will be unique**

**Example**

Input :

Heights: 5 3 2 6 1 4

InFronts: 0 1 2 0 3 2

Output :

actual order is: 5 3 2 1 6 4

So, you can see that for the person with height 5, there is no one taller than him who is in front of him, and hence Infronts has 0 for him.

For person with height 3, there is 1 person ( Height : 5 ) in front of him who is taller than him.

You can do similar inference for other people in the list.

If you know how many people elements there in first half of the array are, and the second half of the array.

Please read the previous hint if you haven’t done so already.

Here, we will explore how to efficiently answer the query of finding the ith empty space.

The query can be solved using **segment / interval tree**.  
The root contains the number of elements in [0, N].  
Left node contains the number of elements in [0, N/2]  
Right node contains the number of elements in [N/2 + 1, N]

Lets say we need to find the ith empty position.  
We look at the number of elements X in [0, N/2].

If

N / 2 - X >= i, the position lies in the left part of array and we move down to the left node.

N / 2 - X < i, we now look for i - (N / 2 - X) th position in the right part of the array and move to the right node in the tree.

This is a fairly standard use of the segment tree.

vector<int> Solution::order(vector<int> &height, vector<int> &infront) {

vector<int> res(height.size(),-1);

map<int, int> mp;

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

mp[height[i]]=infront[i]; //pairs of height and infront with //increasing order of height

for(auto it: mp){

int empty\_pos=0, pos=0; //empty\_pos tells how many empty positions //are

//pos tells at which position the person with given height to place

while(empty\_pos<=it.second){ //increase pos unless empty positions //aren't reached what infront has

if(res[pos]==-1)

empty\_pos++;

pos++;

}

res[pos-1]=it.first; //subtract 1 from pos as position starts from 0

}

return res;

}

## Invert the Binary Tree

* Asked in:
* [Google](https://www.interviewbit.com/search/?q=Google)

Given a binary tree, invert the binary tree and return it.   
Look at the example for more details.

**Example :**   
Given binary tree

1

/ \

2 3

/ \ / \

4 5 6 7

invert and return

1

/ \

3 2

/ \ / \

7 6 5 4

/\*\*

\* Definition for binary tree

\* struct TreeNode {

\* int val;

\* TreeNode \*left;

\* TreeNode \*right;

\* TreeNode(int x) : val(x), left(NULL), right(NULL) {}

\* };

\*/

TreeNode\* Solution::invertTree(TreeNode\* root) {

if(!root) return NULL;

root->left=invertTree(root->left);

root->right=invertTree(root->right);

TreeNode \*temp=root->left;

root->left=root->right;

root->right=temp;

return root;

}

## Hotel Reviews

* Asked in:
* [Booking.com](https://www.interviewbit.com/search/?q=Booking.com)
* Given a set of reviews provided by the customers for different hotels and a string containing “Good Words”, you need to sort the reviews in descending order according to their “Goodness Value” (Higher goodness value first). We define the “Goodness Value” of a string as the number of “Good Words” in that string.
* Note: Sorting should be stable. If review i and review j have the same “Goodness Value” then their original order would be preserved.
* You are expected to use Trie in an Interview for such problems
* **Constraints:**
* 1. 1 <= No.of reviews <= 200
* 2. 1 <= No. of words in a review <= 1000
* 3. 1 <= Length of an individual review <= 10,000
* 4. 1 <= Number of Good Words <= 10,000
* 5. 1 <= Length of an individual Good Word <= 4
* 6. All the alphabets are lower case (a - z)
* **Input:**
* S : A string S containing "Good Words" separated by "\_" character. (See example below)
* R : A vector of strings containing Hotel Reviews. Review strings are also separated by "\_" character.
* **Output:**
* A vector V of integer which contain the original indexes of the reviews in the sorted order of reviews.
* V[i] = k means the review R[k] comes at i-th position in the sorted order. (See example below)
* In simple words, V[i]=Original index of the review which comes at i-th position in the sorted order. (Indexing is 0 based)
* **Example:**
* Input:
* S = "cool\_ice\_wifi"
* R = ["water\_is\_cool", "cold\_ice\_drink", "cool\_wifi\_speed"]
* Output:
* ans = [2, 0, 1]
* Here, sorted reviews are ["cool\_wifi\_speed", "water\_is\_cool", "cold\_ice\_drink"]

void print(vector<string> s){

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

cout<<s[i]<<" ";

cout<<endl;

}

struct mycomp{

bool operator()(const pair<int,int> &a, const pair<int,int> &b){

if(a.second==b.second) return a.first<b.first;

return a.second>b.second;}

}myobj;

struct Trie{

bool isEnd;//finding end of word

Trie\* child[26]; //alphabet size

Trie(bool ie=false){ //by default end of word is false

isEnd=ie;

for(int i=0; i<26; i++)

child[i]=NULL;//initialize child ptrs to null

}

};

void insert(Trie \*trie, string word){

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

if(!trie->child[word[i]-'a'])//if letter not found

trie->child[word[i]-'a']=new Trie();//create new node

trie=trie->child[word[i]-'a'];//move to this node

}

trie->isEnd=true;//set end of word to true for last letter of the word

}

bool search(Trie \*trie, string word){

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

if(trie->child[word[i]-'a'])//search for each letter of word

trie=trie->child[word[i]-'a'];//if node found move to it

else return false;//return false if not found any letter of the word

}

return trie->isEnd;//return end of word value

}

vector<string> split(string line, char delim){

vector<string> res;

stringstream ss(line);

string item;

while(getline(ss, item, delim))

res.push\_back(item);

return res;

}

vector<int> Solution::solve(string input, vector<string> &review) {

int n=review.size();//no. of total reviews

vector<string> gwords;//get good words at gwords

vector<pair<int,int>> review\_rate(n);//pair of review index and its rating

vector<int> res; //result will have indices with decreasing order of rating

Trie \*trie=new Trie();

gwords=split(input, '\_');

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

insert(trie, gwords[i]);//insert each good word into trie

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

vector<string> r;//get words of each review in r

r=split(review[i], '\_');

int k=0; //intial rating 0

for(int j=0; j<r.size(); j++){//for each word in review[i]

if(search(trie, r[j]))//if word is found in trie

k++;//increase rating

}

review\_rate[i].first=i; //store index

review\_rate[i].second=k;//store rating

}

sort(review\_rate.begin(), review\_rate.end(), myobj);//sort in decreasing order of rating

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

res.push\_back(review\_rate[i].first);//push indices to res

return res; //return res

}

# Binary Search

## Rotated Array

* Asked in:
* [Facebook](https://www.interviewbit.com/search/?q=Facebook)

Suppose a sorted array A is rotated at some pivot unknown to you beforehand.

*(i.e., 0 1 2 4 5 6 7 might become 4 5 6 7 0 1 2)*.

Find the minimum element.

The array will not contain duplicates.

* **NOTE 1:** Also think about the case when there are duplicates. Does your current solution work? How does the time complexity change?\*

**PROBLEM APPROACH:**

**Note:** If you know the number of times the array is rotated, then this problem becomes trivial. If the number of rotation is x, then minimum element is A[x].

Properties of minimum element in rotated array:

Minimum element < elements at left

Minimum element < elements at right

int Solution::findMin(const vector<int> &A) {

//checking 1 property of min element is enough

int l=0, r=A.size()-1;

while(l<r){

int m=(l+r)/2;

if(A[m]>A[r])

l=m+1;

else

r=m;

}

if(l==r) return A[l];

else return -1;

}

## Count Element Occurrence

Given a sorted array of integers, find the number of occurrences of a given target value.  
Your algorithm’s runtime complexity must be in the order of O(log n).  
If the target is not found in the array, **return 0**

\*\*Example : \*\*  
Given [5, 7, 7, 8, 8, 10] and target value 8,  
return 2.

**int** findCount(**const** vector**<int>** **&**A, **int** target) {

**int** n **=** A.size();

**int** i **=** 0, j **=** n **-** 1;

**int** start **=** **-**1, end **=** **-**1;

*// FIND FIRST*

**while** (i **<** j)

{

**int** mid **=** (i **+** j) **/**2;

**if** (A[mid] **<** target) i **=** mid **+** 1;

**else** j **=** mid;

}

**if** (A[i] **!=** target) **return** 0; *// the element does not exist in the array.*

start **=** i;

*// FINDLAST*

j **=** n **-** 1; *// We don't have to set i to 0 the second time.*

**while** (i **<** j)

{

**int** mid **=** (i **+** j) **/**2 **+** 1; *// Make mid biased to the right*

**if** (A[mid] **>** target) j **=** mid **-** 1;

**else** i **=** mid; *// So that this won't make the search range stuck.*

}

end **=** j;

**return** (end **-** start **+** 1);

}

};

**Approach Not feasible…**

int binSearch(vector<int>A, int l, int r, int num){

while(l<=r){

int m=(l+r)/2;

if(A[m]==num)

return m;

else if(num < A[m])

return binSearch(A, l, m-1, num);

else

return binSearch(A, m+1, r, num);

}

return -1;

}

int Solution::findCount(const vector<int> &A, int B) {

int n=A.size();

int pos=binSearch(A, 0, n-1, B);

int pos2=pos;

int count=0;

if(pos != -1){

count++;

pos--;

//searching on left side for repetition

while(pos>=0 && A[pos]==B){

count++;

pos--;

}

//searching on right side for repetition

pos2++;

while(pos2<n && A[pos2]==B)

{

count++;

pos2++;

}

}

return count;

}

## Matrix Median

Given a N cross M matrix in which each row is sorted, find the overall median of the matrix. Assume N\*M is odd.

For example,

Matrix=

[1, 3, 5]

[2, 6, 9]

[3, 6, 9]

A = [1, 2, 3, 3, 5, 6, 6, 9, 9]

Median is 5. So, we return 5.

**Note:** No extra memory is allowed.

int Solution::findMedian(vector<vector<int> > &A) {

int r=A.size();

int c=A[0].size();

//find min and max

int minv=A[0][0];

int maxv=A[0][c-1];

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

minv=min(minv, A[i][0]);

maxv=max(maxv, A[i][c-1]);

}

//median will have half of the array size items smaller than it

int half\_size=(r\*c+1)/2;

//binary search modified

//usual binary search works on indices here actual values are considered

while(minv<maxv){

int mid=(minv+maxv)/2;

int size=0;

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

size+=upper\_bound(A[i].begin(), A[i].end(), mid)-A[i].begin();

if(size < half\_size)

minv=mid+1;

else

maxv=mid;

}

return minv;

}

## Square Root of Integer

Asked in:

* [Facebook](https://www.interviewbit.com/search/?q=Facebook)
* [Amazon](https://www.interviewbit.com/search/?q=Amazon)
* [Microsoft](https://www.interviewbit.com/search/?q=Microsoft)

Implement int sqrt(int x).

Compute and return the **square root of x**.

If x is not a perfect square, return floor(sqrt(x))

**Example :**

Input : 11

Output : 3

**DO NOT USE SQRT FUNCTION FROM STANDARD LIBRARY**

int Solution::sqrt(int num) {

if(num==0 || num==1) return num;

//Every number > 4 has square root < number/2; hence starting r from num/2

int l=1, r=num/2, ans=1;

while(l<=r){

int m=l+(r-l)/2;

if(m<=num/m){

ans=m;//current possible square root

l=m+1;

}

else r=m-1;

}

return ans;

}

Code2:

int Solution::sqrt(int num) {

if(num==0 || num==1) return num;

int l=1, r=num/2+1, ans=1;

while(l<r){

int m=l+(r-l)/2;

//if(m==num/m) return m;

if(m<=num/m){

ans=m;

l=m+1;

}

else r=m;

}

return ans;

}

## Matrix Search

Write an efficient algorithm that searches for a value in an m x n matrix.

This matrix has the following properties:

1. Integers in each row are sorted from left to right.
2. The **first integer** of each row is greater than or equal to the **last integer** of the previous row.

**Example:**

Consider the following matrix:

[

[1, 3, 5, 7],

[10, 11, 16, 20],

[23, 30, 34, 50]

]

Given target = 3, return 1 ( 1 corresponds to true )

Return 0 / 1 ( 0 if the element is not present, 1 if the element is present ) for this problem

int Solution::searchMatrix(vector<vector<int> > &A, int x) {

int rows=A.size();

int cols=A[0].size();

//left and right boundaries of array when converted from matrix A

int l=0, r=rows\*cols-1;

while(l<=r){

int m=l+(r-l)/2;

int rm=m/cols;//row number of element m

int cm=m%cols;//col number of element m

if(A[rm][cm]==x)

return 1;

if(A[rm][cm]<x)

l=m+1;

else

r=m-1;

}

return 0;

}

## Sorted Insert Position

* Asked in:
* [Yahoo](https://www.interviewbit.com/search/?q=Yahoo)

Given a sorted array and a target value, return the index if the target is found. If not, return the index where it would be if it were inserted in order.

You may assume no duplicates in the array.

Here are few examples.

[1,3,5,6], 5 → 2

[1,3,5,6], 2 → 1

[1,3,5,6], 7 → 4

[1,3,5,6], 0 → 0

int Solution::searchInsert(vector<int> &A, int x) {

int l=0;

int r=A.size()-1;

int index=-1;//index is to find position of largest number in array which is just smaller than x

////when x is smallest in array, its insert position will be 0, hence index is intialized with -1

while(l<=r){

int m=l+(r-l)/2;

if(A[m]==x)

return m;

if(A[m]<x){

index=m;//index is updated with just smaller number than x

l=m+1;

}

else r=m-1;

}

return index+1;

}

## Search for a Range

* Asked in:
* [Google](https://www.interviewbit.com/search/?q=Google)
* [Microsoft](https://www.interviewbit.com/search/?q=Microsoft)
* Given a sorted array of integers, find the starting and ending position of a given target value.
* Your algorithm’s runtime complexity must be in the order of O(log n).
* If the target is not found in the array, return [-1, -1].
* **Example:**
* Given [5, 7, 7, 8, 8, 10]
* and target value 8,
* return [3, 4].

**Approach:** exactly same as counting element occurrences

**Sol1: Easy**

vector<int> Solution::searchRange(const vector<int> &A, int x) {

vector<int>res;

/\*int a=searchFirst(A,0,A.size()-1,x);//searching first occurence of x

int b=searchLast(A,0,A.size()-1,x);//searching last occurrence of x

res.push\_back(a);

res.push\_back(b);\*/

int l=0, r=A.size()-1;

int start=-1, end=-1;

//search first occurrence

while(l<r){

int m=(l+r)/2;

if(A[m]<x) l=m+1;

else r=m;

}

if(A[l]==x) start=l;

//search last occurrence

r=A.size()-1;

while(l<r){

int m=(l+r)/2+1;

if(A[m]>x) r=m-1;

else l=m;

}

if(A[r]==x) end=r;

res.push\_back(start);

res.push\_back(end);

return res;

}

**Sol2: Bit complex**

int searchFirst(const vector<int>&A, int l, int r, int x){

while(l<=r){

int m=l+(r-l)/2;

if((m==0 || A[m-1]<x) && A[m]==x)//if m is first occurence of x or m-1 has smaller number than x

return m;

//rest is same as binary search

else if(A[m]<x)

l=m+1;

//2nd else should be executed when A[m-1]=A[x]=x

else r=m-1;

}

return -1;

}

int searchLast(const vector<int>&A, int l, int r, int x){

while(l<=r){

int m=l+(r-l)/2;

if((m==(A.size()-1) || A[m+1]>x) && A[m]==x)//if m is last occurence or m+1 has higher number than x

return m;

//rest is same as binary search

else if(A[m]>x)

r=m-1;

//this else should be executed when A[m+1]=A[m]=x

else l=m+1;

}

return -1;

}

vector<int> Solution::searchRange(const vector<int> &A, int x) {

vector<int>res;

int a=searchFirst(A,0,A.size()-1,x);//searching first occurence of x

int b=searchLast(A,0,A.size()-1,x);//searching last occurrence of x

res.push\_back(a);

res.push\_back(b);

return res;

}

## Implement Power Function

Asked in:

* [Google](https://www.interviewbit.com/search/?q=Google)
* [LinkedIn](https://www.interviewbit.com/search/?q=LinkedIn)

Implement pow(x, n) % d.

In other words, given x, n and d,

**find (xn % d)**

Note that remainders on division cannot be negative.   
In other words, make sure the answer you return is non negative.

Input : x = 2, n = 3, d = 3

Output : 2

2^3 % 3 = 8 % 3 = 2.

//Given: n can never be negative, ans can never be negative

typedef long long int LL;

//Given: n can never be negative, ans can never be negative

int Solution::pow(int x, int n, int d) {

if(x==0) return x; //when base is 0

if(n==0) return 1; //when power is 0

//avoid overflow, use long long int

LL halfP=pow(x, n/2, d);

LL P=(halfP \* halfP)%d;

if(n%2==0)

return P%d;

else{

P=(P\*x)%d;

if(x>0) return P;

//when x is -ve, remainder also be -ve, add d to get then answer

else return (d+P)%d;

}

}

## Painter's Partition Problem

* Asked in:
* [Google](https://www.interviewbit.com/search/?q=Google)
* [Codenation](https://www.interviewbit.com/search/?q=Codenation)

BookmarkSuggest Edit

You have to paint N boards of length {A0, A1, A2, A3 … AN-1}. There are K painters available and you are also given how much time a painter takes to paint 1 unit of board. You have to get this job done as soon as possible under the constraints that **any painter will only paint contiguous sections of board.**

* 2 painters cannot share a board to paint. That is to say, a board  
  cannot be painted partially by one painter, and partially by another.
* A painter will only paint contiguous boards. Which means a  
  configuration where painter 1 paints board 1 and 3 but not 2 is  
  invalid.

Return the ans % 10000003

Input :

K : Number of painters

T : Time taken by painter to paint 1 unit of board

L : A List which will represent length of each board

Output:

return minimum time to paint all boards % 10000003

**Example**

Input :

K : 2

T : 5

L : [1, 10]

Output : 50

//Return the ans % 10000003

#define MOD 10000003

int getPainters(vector<int> &boards, int time){

int i=0, sum=0, p=1;

while(i<boards.size()){

if(sum+boards[i]<=time){

sum+=boards[i];

}

else{

p++;

sum=boards[i];

}

i++;

}

return p;

}

int Solution::paint(int painters, int time, vector<int> &boards) {

//considering 1 unit length of board needs 1 unit time instead of K or time

//maximum time taken to paint boards is when 1 painter paints all boards

long long int maxT=(accumulate(boards.begin(), boards.end(), 0))%MOD;

//minimum time taken maximum length of the board

int minT=(\*max\_element(boards.begin(), boards.end()))%MOD;

int ans;

//apply binary search between mintime and maxtime

while(minT<=maxT){

int midT=(minT+(maxT-minT)/2)%MOD;//addition can cause overflow, apply %MOD

int p=getPainters(boards, midT);//find no. of painter to paint all boards in midT time

if(p<=painters){//if p is less than or equal to given painters, save the current answer

ans=midT;

maxT=midT-1;

}

else minT=(midT+1)%MOD;//can cause overflow, apply %MOD

}

return ((long long)ans\*time)%MOD;//since ans and time are int, multiplication can cause overflow

}

## Allocate Books

Asked in:

* [Google](https://www.interviewbit.com/search/?q=Google)

N number of books are given.   
The ith book has Pi number of pages.   
You have to allocate books to M number of students so that maximum number of pages alloted to a student is minimum. A book will be allocated to exactly one student. Each student has to be allocated at least one book. Allotment should be in contiguous order, for example: A student cannot be allocated book 1 and book 3, skipping book 2.

**NOTE:** Return -1 if a valid assignment is not possible

**Input:**

List of Books

M number of students

Your function should return an integer corresponding to the minimum number.

**Example:**

P : [12, 34, 67, 90]

M : 2

Output : 113

There are 2 number of students. Books can be distributed in following fashion :

1) [12] and [34, 67, 90]

Max number of pages is allocated to student 2 with 34 + 67 + 90 = 191 pages

2) [12, 34] and [67, 90]

Max number of pages is allocated to student 2 with 67 + 90 = 157 pages

3) [12, 34, 67] and [90]

Max number of pages is allocated to student 1 with 12 + 34 + 67 = 113 pages

Of the 3 cases, Option 3 has the minimum pages = 113.

//given books and max pages can be assigned to each student

int getStuds(vector<int> &books, int pages){

int i=0, sum=0, s=1;

while(i<books.size()){

if(sum+books[i]<=pages)

sum+=books[i];

else{

s++;

sum=books[i];

}

i++;

}

return s;

}

int Solution::books(vector<int> &books, int students) {

//minimum pages are book with max pages

int minP=\*max\_element(books.begin(), books.end());

//maximum pages is all books pages

int maxP=accumulate(books.begin(), books.end(), 0);

int ans=-1;

//each student should get at least 1 book

if(books.size()<students) return ans;

while(minP <= maxP){

int midP=minP+(maxP-minP)/2;

int studs=getStuds(books, midP);//find studs required to allot midP pages

if(studs<=students){

ans=midP;//if books can be alloted to less number of students then can also be alloted to total no. students

maxP=midP-1;//search if next lower max pages possible to allot

}

else minP=midP+1;

}

return ans;

}

## Rotated Sorted Array Search

Asked in:

* [Facebook](https://www.interviewbit.com/search/?q=Facebook)
* [Google](https://www.interviewbit.com/search/?q=Google)
* [Microsoft](https://www.interviewbit.com/search/?q=Microsoft)
* [Amazon](https://www.interviewbit.com/search/?q=Amazon)

BookmarkSuggest Edit

Suppose a sorted array is rotated at some pivot unknown to you beforehand.

(i.e., *0 1 2 4 5 6 7* might become *4 5 6 7 0 1 2* ).

You are given a target value to search. If found in the array, return its index, otherwise return -1.

You may assume no duplicate exists in the array.

Input : [4 5 6 7 0 1 2] and target = 4

Output : 0

* **NOTE :** Think about the case when there are duplicates. Does your current solution work? How does the time complexity change?\*

**Solution doesn’t work when duplicates occur**

int findMin(const vector<int> &A){

int l=0, r=A.size()-1;

while(l<r){//same as binary search, no = sign as loop stops when l crosses r

//but we want l and r pointing same element which is the minimum one

int m=l+(r-l)/2;

if(A[m]<A[r])//smaller elements are at left of m, including m

r=m;//no m-1 as m can be minimum element

else l=m+1; // here A[m]>A[r], minimum is not at m and at right of m;

//A[m]==A[r] is not possible, except when l=r, but that won't happen inside loop

}

return r;

}

//regular binary search algo

int find(const vector<int> &A, int l, int r, int x){

while(l<=r){

int m=l+(r-l)/2;

if(A[m]==x)

return m;

if(A[m]<x)

l=m+1;

else r=m-1;

}

return -1;

}

//trick is to find minimum element position,

//divide array between 2 subarrays, i) 0 to minPos-1 , having all numbers in ascending order

//and ii) minPos, end pos, having all element in ascending order

int Solution::search(const vector<int> &A, int num) {

if(A.size()<1) return -1;

int minP=findMin(A);//find minimum elment

int pos=-1;

pos=find(A,0,minP-1, num);

if(pos == -1){

pos=find(A, minP, A.size()-1, num);

}

return pos;

}

## Median of Array

* Asked in:
* [Amazon](https://www.interviewbit.com/search/?q=Amazon)
* [VMWare](https://www.interviewbit.com/search/?q=VMWare)
* [Google](https://www.interviewbit.com/search/?q=Google)
* [Microsoft](https://www.interviewbit.com/search/?q=Microsoft)
* [Goldman Sachs](https://www.interviewbit.com/search/?q=Goldman)
* There are two sorted arrays A and B of size m and n respectively.
* Find the median of the two sorted arrays ( The median of the array formed by merging both the arrays ).
* The overall run time complexity should be O(log (m+n)).
* **Sample Input**
* A : [1 4 5]
* B : [2 3]
* **Sample Output**
* 3
* *NOTE: IF the number of elements in the merged array is even, then the median is the average of n / 2 th and n/2 + 1th element.   
  For example, if the array is [1 2 3 4], the median is (2 + 3) / 2.0 = 2.5*

Given a sorted array A of length m, we can split it into two parts:

|  |  |
| --- | --- |
| { A[0], A[1], … , A[i - 1] } | { A[i], A[i + 1], … , A[m - 1] } |

All the elements in right part are greater than the elements in left part.

The left part has “i” elements, and right part has “m - i” elements.

There are “m + 1” kinds of splits. (i = 0 ~ m)

When i = 0, the left part has “0” elements, right part has “m” elements.

When i = m, the left part has “m” elements, right part has “0” elements.

For array B, we can split it with the same way:

|  |  |
| --- | --- |
| { B[0], B[1], … , B[j - 1] } | { B[j], B[j + 1], … , B[n - 1] } |

The left part has “j” elements, and right part has “n - j” elements.

Put A’s left part and B’s left part into one set. (Let us name this set “LeftPart”)

Put A’s right part and B’s right part into one set. (Let us name this set”RightPart”)

LeftPart | RightPart

|  |  |
| --- | --- |
| { A[0], A[1], … , A[i - 1] } | { A[i], A[i + 1], … , A[m - 1] } |

|  |  |
| --- | --- |
| { B[0], B[1], … , B[j - 1] } | { B[j], B[j + 1], … , B[n - 1] } |

If we can ensure the following:

1) LeftPart’s length == RightPart’s length (or RightPart’s length + 1)

2) All elements in RightPart is greater than elements in LeftPart,

then we split all elements in {A, B} into two parts with eqaul length, and one part is

always greater than the other part.

Then the median can be easily found.

The expected time complexity gives away binary search in this case.  
We are going to do **binary search for the answer** in this case.

Given a sorted array A of length m, we can split it into two parts:

|  |  |
| --- | --- |
| { A[0], A[1], … , A[i - 1] } | { A[i], A[i + 1], … , A[m - 1] } |

All elements in right part are greater than elements in the left part.

The left part has i elements, and right part has m - i elements.  
There are m + 1 kinds of splits.

**(i = 0 ~ m)**

*When i = 0, the left part has “0” elements, the right part has “m” elements.*  
*When i = m, the left part has “m” elements, right part has “0” elements.*

For the array B, we can split it in the same way:

|  |  |
| --- | --- |
| { B[0], B[1], … , B[j - 1] } | { B[j], B[j + 1], … , B[n - 1] } |

The left part has “j” elements, and right part has “n - j” elements.

Put A’s left part and B’s left part into one set. *(Let’s name this set “LeftPart”)*

Put A’s right part and B’s right part into one set. *(Let’s name this set”RightPart”)*

LeftPart | RightPart

|  |  |
| --- | --- |
| { A[0], A[1], … , A[i - 1] } | { A[i], A[i + 1], … , A[m - 1] } |

|  |  |
| --- | --- |
| { B[0], B[1], … , B[j - 1] } | { B[j], B[j + 1], … , B[n - 1] } |

If we can ensure the following:

* LeftPart’s length == RightPart’s length (or RightPart’s length + 1)
* All elements in RightPart is greater than elements in LeftPart,

then we can split all elements in {A, B} into two parts with equal length, and one part is always greater than the other part.

Then the median can thus be easily found.

To ensure these two condition, we just need to ensure:

* **Condition 1** :
* i + j == (m - i) + (n - j)
* OR i + j == (m - i) + (n - j) + 1

Which means if n >= m,

i = 0 to m

j = (m + n + 1) / 2 - i

* **Condition 2**

B[j - 1] <= A[i] and A[i - 1] <= B[j]

Considering edge values, we need to ensure:

(j == 0 or i == m or B[j - 1] <= A[i]) and

(i == 0 or j == n or A[i - 1] <= B[j])

So, all we need to do is:

* Search i from 0 to m, to find an object i to meet condition (1) and (2) above.

**And we can do this search by binary search.**

**How?**

* If B[j0 - 1] > A[i0], than the object ix can’t be in [0, i0].

Why?

Because if

ix < i0,

=> jx = (m + n + 1) / 2 - ix > j0

=> B[jx - 1] >= B[j0 - 1] > A[i0] >= A[ix].

This **violates** the **condition (2)**. So ix can’t be less than i0.

* And if A[i0 - 1] > B[j0], than the object ix can’t be in [i0, m].

So we can do the binary search following the steps described below:

* set imin, imax = 0, m, then start searching in [imin, imax]

Search in [imin, imax]:

i = (imin + imax) / 2

j = ((m + n + 1) / 2) - i

if B[j - 1] > A[i]:

search in [i + 1, imax]

else if A[i - 1] > B[j]:

search in [imin, i - 1]

else:

if m + n is odd:

answer is max(A[i - 1], B[j - 1])

else:

answer is (max(A[i - 1], B[j - 1]) + min(A[i], B[j])) / 2

double Solution::findMedianSortedArrays(const vector<int> &A, const vector<int> &B) {

int m=A.size();

int n=B.size();

if(m>n) return findMedianSortedArrays(B,A);

int i,j;

int imin=0, imax=m;

int med1, med2;

while(imin<=imax){

i=(imin+imax)/2;

j=(m+n+1)/2-i;

if(j>0 && i<m && B[j-1]>A[i]) imin=i+1;

else if(i>0 && j<n && A[i-1]>B[j]) imax=i-1;

else{

if(i==0) med1=B[j-1];

else if(j==0) med1=A[i-1];

else med1=max(A[i-1],B[j-1]);

if((m+n)%2==1) return med1;

else{

if(i==m) med2=B[j];

else if(j==n) med2=A[i];

else med2=min(A[i],B[j]);

return (1.0\*(med1+med2)/2);

}

}

}

}

# String

## Longest Palindromic Substring

* Asked in:
* [Amazon](https://www.interviewbit.com/search/?q=Amazon)
* [Microsoft](https://www.interviewbit.com/search/?q=Microsoft)
* [Groupon](https://www.interviewbit.com/search/?q=Groupon)

Given a string S, find the longest palindromic substring in S.

**Substring of string S:**

S[i...j] where 0 <= i <= j < len(S)

**Palindrome string:**

A string which reads the same backwards. More formally, S is palindrome if reverse(S) = S.

**Incase of conflict**, return the substring which occurs first ( with the least starting index ).

**Example :**

Input : "aaaabaaa"

Output : "aaabaaa"

string Solution::longestPalindrome(string s) {

int len=s.size();

string res;

int dp[len][len];

memset(dp,0, sizeof(dp));

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

dp[i][i]=1;

int maxlen=1, start=0;

for(int l=2; l<=len; l++){

for(int i=0; i<=len-l; i++){

int j=i+l-1;

if(s[i]==s[j]){

if(l==2 || dp[i+1][j-1]){

dp[i][j]=1;

if(maxlen < l){

maxlen=l;

start=i;

}

}

}

// cout<<l<<" "<<j<<" "<<maxlen<<endl;

}

}

return s.substr(start,maxlen);

}

## Implement StrStr

* Asked in:
* [Facebook](https://www.interviewbit.com/search/?q=Facebook)
* [Amazon](https://www.interviewbit.com/search/?q=Amazon)
* [Qualcomm](https://www.interviewbit.com/search/?q=Qualcomm)
* [Wipro](https://www.interviewbit.com/search/?q=Wipro)
* [Microsoft](https://www.interviewbit.com/search/?q=Microsoft)

Please Note:

Another question which belongs to the category of questions which are intentionally stated vaguely.   
Expectation is that you will ask for correct clarification or you will state your assumptions before you start coding.

**Implement strStr().**

*strstr - locate a substring ( needle ) in a string ( haystack ).*

**Try not to use standard library string functions for this question.**

Returns the index of the first occurrence of needle in haystack, or -1 if needle is not part of haystack.

***NOTE:***

***Good clarification questions:***

1. *What should be the return value if the needle is empty?*
2. *What if both haystack and needle are empty?*

*For the purpose of this problem, assume that the return value should be -1 in both cases.*

int Solution::strStr(const string s2, const string s1) {

int m=s1.size();

int n=s2.size();

if(m>n || m==0 || n==0) return -1;

int i=0, j=0, start=0;

while(i<m && j<n && start<=n-m){

if(s1[i]==s2[j]){

if(i==m-1) return start;

i++;

j++;

}

else{

i=0;

j=++start;

}

}

return -1;

}

# Linked List

## Intersection of Linked Lists

* Asked in:
* [Amazon](https://www.interviewbit.com/search/?q=Amazon)
* [Microsoft](https://www.interviewbit.com/search/?q=Microsoft)
* [NetApp](https://www.interviewbit.com/search/?q=NetApp)
* [Apache Design](https://www.interviewbit.com/search/?q=Apache)

Write a program to find the node at which the intersection of two singly linked lists begins.

For example, the following two linked lists:

A: a1 → a2

↘

c1 → c2 → c3

↗

B: b1 → b2 → b3

begin to intersect at node c1.

***Notes:***

* If the two linked lists have no intersection at all, return null.
* The linked lists must retain their original structure after the function returns.
* You may assume there are no cycles anywhere in the entire linked structure.
* Your code should preferably run in O(n) time and use only O(1) memory.

int length(ListNode \*node){

int len=0;

while(node){

len++;

node=node->next;

}

return len;

}

/\*\*

\* Definition for singly-linked list.

\* struct ListNode {

\* int val;

\* ListNode \*next;

\* ListNode(int x) : val(x), next(NULL) {}

\* };

\*/

ListNode\* Solution::getIntersectionNode(ListNode\* A, ListNode\* B) {

int lenA=length(A);

int lenB=length(B);

if(lenA==0 || lenB==0) return NULL;

if(lenA < lenB) {

return getIntersectionNode(B,A);

}

int reach=lenA-lenB;

while(reach){ A=A->next; reach--;}

while(A && B){

if(A==B) return A;

A=A->next;

B=B->next;

}

return NULL;

}

## Reverse Linked List

Reverse a linked list. Do it in-place and in one-pass.

For example:  
Given 1->2->3->4->5->NULL,

return 5->4->3->2->1->NULL.

Sol1: Iterative

/\*\*

\* Definition for singly-linked list.

\* struct ListNode {

\* int val;

\* ListNode \*next;

\* ListNode(int x) : val(x), next(NULL) {}

\* };

\*/

ListNode\* Solution::reverseList(ListNode\* head) {

if(!head) return head;

ListNode \*prev=NULL, \*cur=head, \*next;

while(cur){

next=cur->next;

cur->next=prev;

prev=cur;

cur=next;

}

return prev;

}

Sol2: Recursive

ListNode \* Solution::reverseList(ListNode \*node){

if(!node || !node->next) return node;

ListNode \*revHead=reverseList(node->next);

node->next->next=node;

node->next=NULL;

return revHead;

}

## K reverse linked list

* Asked in:
* [Microsoft](https://www.interviewbit.com/search/?q=Microsoft)
* [Amazon](https://www.interviewbit.com/search/?q=Amazon)

Given a singly linked list and an integer K, reverses the nodes of the

list K at a time and returns modified linked list.

***NOTE :****The length of the list is divisible by K*

**Example :**

Given linked list 1 -> 2 -> 3 -> 4 -> 5 -> 6 and K=2,

You should return 2 -> 1 -> 4 -> 3 -> 6 -> 5

Try to solve the problem using constant extra space.

* Reverse the first sub-list of size k. While reversing keep track of the next node and previous node. Let the pointer to the next node be next and pointer to the previous node be prev. See [this post](https://www.geeksforgeeks.org/reverse-a-linked-list/) for reversing a linked list.
* head->next = reverse(next, k) ( Recursively call for rest of the list and link the two sub-lists )
* Return prev ( prev becomes the new head of the list (see the diagrams of iterative method of [this post)](https://www.geeksforgeeks.org/reverse-a-linked-list/))

/\*\*

\* Definition for singly-linked list.

\* struct ListNode {

\* int val;

\* ListNode \*next;

\* ListNode(int x) : val(x), next(NULL) {}

\* };

\*/

ListNode\* Solution::reverseList(ListNode\* head, int K) {

ListNode \*prev=NULL, \*cur=head, \*next=NULL;

int count=0;

while(cur && count<K){

next=cur->next;

cur->next=prev;

prev=cur;

cur=next;

count++;

}

if(next)

head->next=reverseList(next, K);

return prev;

}

## Reverse Link List II

* Asked in:
* [Facebook](https://www.interviewbit.com/search/?q=Facebook)
* [Microsoft](https://www.interviewbit.com/search/?q=Microsoft)
* [Amazon](https://www.interviewbit.com/search/?q=Amazon)
* Reverse a linked list from position m to n. Do it in-place and in one-pass.
* For example:  
  Given 1->2->3->4->5->NULL, m = 2 and n = 4,
* return 1->4->3->2->5->NULL.
* ***Note:*** *Given m, n satisfy the following condition:  
  1 ≤ m ≤ n ≤ length of list.*
* ***Note 2:*** *Usually the version often seen in the interviews is reversing the whole linked list which is obviously an easier version of this question.*

/\*\*

\* Definition for singly-linked list.

\* struct ListNode {

\* int val;

\* ListNode \*next;

\* ListNode(int x) : val(x), next(NULL) {}

\* };

\*/

ListNode \*reverseK(ListNode \*head, int k){

if(!head || !head->next) return head;

ListNode \*prev=NULL, \*cur=head, \*next=NULL;

while(cur && k--){

next=cur->next;

cur->next=prev;

prev=cur;

cur=next;

}

if(cur) head->next=cur;

return prev;

}

ListNode\* Solution::reverseBetween(ListNode\* head, int m, int n) {

if(m>n || !head || !head->next) return head;

int k=n-m+1;

if(m==1){

return reverseK(head,k);

}

ListNode \*pre=head;

int count=1;

while(pre && count++<m-1)

pre=pre->next;

//cout<<pre->val<<endl;

pre->next=reverseK(pre->next,k);

return head;

}

## Remove Duplicates from Sorted List

* Asked in:
* [Microsoft](https://www.interviewbit.com/search/?q=Microsoft)
* [VMWare](https://www.interviewbit.com/search/?q=VMWare)

Given a sorted linked list, delete all duplicates such that each element appear only once.

For example,  
Given 1->1->2, return 1->2.  
Given 1->1->2->3->3, return 1->2->3.

/\*\*

\* Definition for singly-linked list.

\* struct ListNode {

\* int val;

\* ListNode \*next;

\* ListNode(int x) : val(x), next(NULL) {}

\* };

\*/

ListNode\* Solution::deleteDuplicates(ListNode\* head) {

if(!head || !head->next) return head;

//maintain 2 ptrs, keep checking their values

//if values are same then remove the next ptr

ListNode \*prev=head, \*cur;

while(prev->next){

cur=prev->next;

if(prev->val==cur->val){

prev->next=cur->next;

cur->next=NULL;

delete cur;

}

else prev=cur;

}

return head;

}

**//Sol 2**

ListNode\* Solution::deleteDuplicates(ListNode\* head) {

if(!head || !head->next) return head;

ListNode \*prev, \*cur=head;

while(cur && cur->next){

prev=cur;

cur=cur->next;

if(prev->val==cur->val){

prev->next=cur->next;

cur->next=NULL;

delete cur;

cur=prev;

}

}

return head;

}

## Remove Duplicates from Sorted List II Asked in:

* [Microsoft](https://www.interviewbit.com/search/?q=Microsoft)
* [VMWare](https://www.interviewbit.com/search/?q=VMWare)

Given a sorted linked list, delete all nodes that have duplicate numbers, leaving only distinct numbers from the original list.

For example,  
Given 1->2->3->3->4->4->5, return 1->2->5.  
Given 1->1->1->2->3, return 2->3.

/\*\*

\* Definition for singly-linked list.

\* struct ListNode {

\* int val;

\* ListNode \*next;

\* ListNode(int x) : val(x), next(NULL) {}

\* };

\*/

ListNode\* Solution::deleteDuplicates(ListNode\* head) {

if(!head || !head->next) return head;

// create a dummy node that acts like a fake

// head of list pointing to the original head

ListNode \*dummy=new ListNode(-1);

dummy->next=head;

// Node pointing to last node which has no duplicate.

ListNode \*prev=dummy;

// Node to traverse the list

ListNode \*cur=head;

while(cur){

// Until the current and previous values are same,

// keep moving current ahead

while(cur->next && prev->next->val==cur->next->val)

cur=cur->next;

//when current is unique, it is not updated

//move prev to current

if(prev->next==cur)

prev=cur;

//when current is last duplicate,

//unlink duplicate nodes by updating prev->next

else

prev->next=cur->next;

//traverse to next node in list

cur=cur->next;

}

//update original head as first unique node

head=dummy->next;

return head;

}

## Merge Two Sorted Lists

* Asked in:
* [Microsoft](https://www.interviewbit.com/search/?q=Microsoft)
* [Yahoo](https://www.interviewbit.com/search/?q=Yahoo)
* [Amazon](https://www.interviewbit.com/search/?q=Amazon)

Merge two sorted linked lists and return it as a new list.   
The new list should be made by splicing together the nodes of the first two lists, and should also be sorted.

For example, given following linked lists :

5 -> 8 -> 20

4 -> 11 -> 15

The merged list should be :

4 -> 5 -> 8 -> 11 -> 15 -> 20

Sol1: modify A and insert nodes from B

/\*\*

\* Definition for singly-linked list.

\* struct ListNode {

\* int val;

\* ListNode \*next;

\* ListNode(int x) : val(x), next(NULL) {}

\* };

\*/

ListNode\* Solution::mergeTwoLists(ListNode\* A, ListNode\* B) {

if(!A) return B;

if(!B) return A;

if(A->val > B->val) return mergeTwoLists(B,A);

ListNode \*h1=A, \*h2=B, \*temp;

while(h1 && h2){

while(h1->next && h1->next->val<h2->val)

h1=h1->next;

temp=h2;

h2=h2->next;

temp->next=h1->next;

h1->next=temp;

h1=h1->next;

}

if(!h2) return A;

temp->next=h2;

return A;

}

**Sol2**: First smallest of A and B will become head then traversing the list in same way to update further.

/\*\*

\* Definition for singly-linked list.

\* struct ListNode {

\* int val;

\* ListNode \*next;

\* ListNode(int x) : val(x), next(NULL) {}

\* };

\*/

ListNode\* Solution::mergeTwoLists(ListNode\* A, ListNode\* B) {

if(!A) return B;

if(!B) return A;

ListNode \*head, \*p;

if(A->val < B->val){

head=A;

A=A->next;

}

else{

head=B;

B=B->next;

}

p=head;

while(A && B){

if(A->val < B->val){

p->next=A;

A=A->next;

}

else{

p->next=B;

B=B->next;

}

p=p->next;

}

if(A) p->next=A;

if(B) p->next=B;

return head;

}

## Remove Nth Node from List End

* Asked in:
* [HCL](https://www.interviewbit.com/search/?q=HCL)
* [Amazon](https://www.interviewbit.com/search/?q=Amazon)

Given a linked list, remove the nth node from the end of list and return its head.

For example,  
Given linked list: 1->2->3->4->5, and n = 2.  
After removing the second node from the end, the linked list becomes 1->2->3->5.

***Note:***

* If n is greater than the size of the list, remove the first node of the list.

## Try doing it using constant additional space.

Sol1:

1) Make the first pointer go n nodes. Then move the second and first pointer simultaneously. This way, the first pointer is always ahead of the second pointer by n nodes. So when first pointer reaches the end, you are on the node to be removed.

**Sol1:**

ListNode \*nptr=A;

ListNode \*ptr=A;

while(n-- && nptr)

nptr=nptr->next;

// if n>length of A, remove first node of list

if(!nptr){

return A->next;

}

//now ptr is n position away from nptr, save previous of ptr and update the next at end

while(nptr->next){

nptr=nptr->next;

ptr=ptr->next;

}

ptr->next=ptr->next->next;

return A;

**Sol2: inefficient**

ListNode\* Solution::removeNthFromEnd(ListNode\* A, int B) {

int lenA=length(A);

//case 1: lenA<=B

if(lenA <= B){

A=A->next;

return A;

}

//case 2: lenA > B

int pos=lenA-B-1;

ListNode \*temp=A;

while(pos){

temp=temp->next;

pos--;

}

if(temp->next) temp->next=temp->next->next;

return A;

}

## Rotate List

* Asked in:
* [Amazon](https://www.interviewbit.com/search/?q=Amazon)

Given a list, rotate the list to the right by k places, where k is non-negative.

For example:

Given 1->2->3->4->5->NULL and k = 2,  
return 4->5->1->2->3->NULL.

**Sol1: no need to calculate length, entire list will be traversed when k > size of list**

/\*\*

\* Definition for singly-linked list.

\* struct ListNode {

\* int val;

\* ListNode \*next;

\* ListNode(int x) : val(x), next(NULL) {}

\* };

\*/

ListNode\* Solution::rotateRight(ListNode\* head, int k) {

if(!head || !head->next) return head;

ListNode \*kptr=head, \*ptr=head;

while(k && kptr){

kptr=kptr->next;

k--;

}

//if k > size of list

if(k>0) return rotateRight(head, k);

if(k==0 && !kptr) return head;

while(kptr->next){

kptr=kptr->next;

ptr=ptr->next;

}

ListNode \*head1=ptr->next;

ptr->next=NULL;

kptr->next=head;

head=head1;

return head;

}

**Sol2: calculate length first**

int length(ListNode \*head){

int len=0;

while(head){

len++;

head=head->next;

}

return len;

}

/\*\*

\* Definition for singly-linked list.

\* struct ListNode {

\* int val;

\* ListNode \*next;

\* ListNode(int x) : val(x), next(NULL) {}

\* };

\*/

ListNode\* Solution::rotateRight(ListNode\* head, int k) {

if(!head || !head->next) return head;

int len=length(head);

//k can be greather than lenght of list

k=k%len;

//no rotate required

if(k==0)

return head;

ListNode \*cur=head;

//place cur to exactly k space away from head

while(cur && k--)

cur=cur->next;

ListNode \*knode, \*prev=head;

//traverse list till cur reaches 2nd last element

while(cur->next){

prev=prev->next;

cur=cur->next;

}

knode=prev->next;

prev->next=NULL;

cur->next=head;

head=knode;

return head;

}

## Reorder List

* Asked in:
* [Amazon](https://www.interviewbit.com/search/?q=Amazon)
* [Microsoft](https://www.interviewbit.com/search/?q=Microsoft)

Given a singly linked list

L: L0 → L1 → … → Ln-1 → Ln,

reorder it to:

L0 → Ln → L1 → Ln-1 → L2 → Ln-2 → …

You must do this in-place without altering the nodes’ values.

For example,  
Given {1,2,3,4}, reorder it to {1,4,2,3}.

* 1. Break the list from middle into 2 lists.  
     2) Reverse the latter half of the list.   
     3) Now merge the lists so that the nodes alternate.

ListNode \* reverse(ListNode\* head){

if(!head|| !head->next) return head;

ListNode \*revHead=reverse(head->next);

head->next->next=head;

head->next=NULL;

return revHead;

}

/\*\*

\* Definition for singly-linked list.

\* struct ListNode {

\* int val;

\* ListNode \*next;

\* ListNode(int x) : val(x), next(NULL) {}

\* };

\*/

ListNode\* Solution::reorderList(ListNode\* head) {

if(!head || !head->next) return head;

ListNode \*fast\_ptr=head, \*slow\_ptr=head;

while(fast\_ptr && fast\_ptr->next){

slow\_ptr=slow\_ptr->next;

fast\_ptr=fast\_ptr->next->next;

}

ListNode\* head2=slow\_ptr->next;

slow\_ptr->next=NULL;

head2=reverse(head2);

ListNode\* head1=head;

while(head1 && head2){

ListNode \*temp=head2;

head2=head2->next;

temp->next=head1->next;

head1->next=temp;

head1=temp->next;

}

return head;

}

## Swap List Nodes in pairs

* Asked in:
* [Microsoft](https://www.interviewbit.com/search/?q=Microsoft)
* [Amazon](https://www.interviewbit.com/search/?q=Amazon)

Given a linked list, swap every two adjacent nodes and return its head.

For example,  
Given 1->2->3->4, you should return the list as 2->1->4->3.

Your algorithm should use only constant space. You may not modify the values in the list, only nodes itself can be changed.

/\*\*

\* Definition for singly-linked list.

\* struct ListNode {

\* int val;

\* ListNode \*next;

\* ListNode(int x) : val(x), next(NULL) {}

\* };

\*/

ListNode\* Solution::swapPairs(ListNode\* head) {

ListNode \*prev=NULL, \*cur=head, \*next;

int count=2;

while(cur && count--){

next=cur->next;

cur->next=prev;

prev=cur;

cur=next;

}

if(next)

head->next=swapPairs(next);

return prev;

}

## Add Two Numbers as Lists

* Asked in:
* [Amazon](https://www.interviewbit.com/search/?q=Amazon)
* [Qualcomm](https://www.interviewbit.com/search/?q=Qualcomm)
* [Microsoft](https://www.interviewbit.com/search/?q=Microsoft)
* [Facebook](https://www.interviewbit.com/search/?q=Facebook)

You are given two linked lists representing two non-negative numbers. The digits are stored in **reverse order**and each of their nodes contain a single digit. Add the two numbers and return it as a linked list.

Input: (2 -> 4 -> 3) + (5 -> 6 -> 4)  
Output: 7 -> 0 -> 8

342 + 465 = 807

Make sure there are no trailing zeros in the output list  
So, 7 -> 0 -> 8 -> 0 is not a valid response even though the value is still 807.

Sol1:

/\*\*

\* Definition for singly-linked list.

\* struct ListNode {

\* int val;

\* ListNode \*next;

\* ListNode(int x) : val(x), next(NULL) {}

\* };

\*/

ListNode\* Solution::addTwoNumbers(ListNode\* A, ListNode\* B) {

if(!A) return B;

if(!B) return A;

ListNode \*head, \*tail;

int sum=A->val+B->val;

int carry=sum/10;

head=new ListNode(sum%10);

tail=head;

A=A->next;

B=B->next;

while(A || B || carry){

sum=(A?A->val:0)+(B?B->val:0)+carry;

carry=sum/10;

tail->next=new ListNode(sum%10);

tail=tail->next;

if(A) A=A->next;

if(B) B=B->next;

}

return head;

}

Sol2: modifying A and saving result in that

/\*\*

\* Definition for singly-linked list.

\* struct ListNode {

\* int val;

\* ListNode \*next;

\* ListNode(int x) : val(x), next(NULL) {}

\* };

\*/

ListNode\* Solution::addTwoNumbers(ListNode\* A, ListNode\* B) {

ListNode\* head=A;

ListNode \* prev=A;

ListNode\* car;

int sum=0, carry=0;

if(!A) return B;

if(!B) return A;

while(A && B){

sum=carry+A->val+B->val;

A->val=sum%10;

carry=sum/10;

prev=A;

A=A->next;

B=B->next;

}

if(!A && B){

//cout<<"Inside this"<<endl;

prev->next=B;

while(B){

sum=B->val+carry;

B->val=sum%10;

carry=sum/10;

prev=B;

B=B->next;

}

}

else if(A && !B){

while(A){

sum=carry+A->val;

A->val=sum%10;

carry=sum/10;

prev=A;

A=A->next;

}

}

if(carry){

car=new ListNode(carry);

prev->next=car;

}

return head;

}

## Partition List

* Asked in:
* [Microsoft](https://www.interviewbit.com/search/?q=Microsoft)

Given a linked list and a value x, partition it such that all nodes less than x come before nodes greater than or equal to x.

You should preserve the original relative order of the nodes in each of the two partitions.

For example,  
Given 1->4->3->2->5->2 and x = 3,  
return 1->2->2->4->3->5.

/\*\*

\* Definition for singly-linked list.

\* struct ListNode {

\* int val;

\* ListNode \*next;

\* ListNode(int x) : val(x), next(NULL) {}

\* };

\*/

ListNode\* Solution::partition(ListNode\* head, int x) {

ListNode \*prev, \*dummy;

//dummy node needed to point to head of the updated list

dummy=new ListNode(-1);

dummy->next=head;

prev=dummy;

//head1 and tail1 maintain list of less than x elements

ListNode \*cur=head, \*head1=NULL, \*tail1=NULL; // initializing head1=NULL helps in if(!head) check

//traverse given list, if node < x is found add to list of head1 and remove from given list

while(cur){

if(cur->val<x){

//remove cur from existing list

prev->next=cur->next;

cur->next=NULL;

//update list with head1 and tail1

if(!head1){

head1=cur;

tail1=cur;

}

else{

tail1->next=cur;

tail1=tail1->next;

}

//get next element in cur

cur=prev->next;

}

else{

prev=cur;

cur=cur->next;

}

}

if(head1){//useful when head1 is empty

tail1->next=dummy->next;

return head1;

}

else return head;

}

## List Cycle

* Asked in:
* [Amazon](https://www.interviewbit.com/search/?q=Amazon)
* [Microsoft](https://www.interviewbit.com/search/?q=Microsoft)
* [NetApp](https://www.interviewbit.com/search/?q=NetApp)

Given a linked list, return the node where the cycle begins. If there is no cycle, return null.

Try solving it using constant additional space.

**Example :**

Input :

\_\_\_\_\_\_

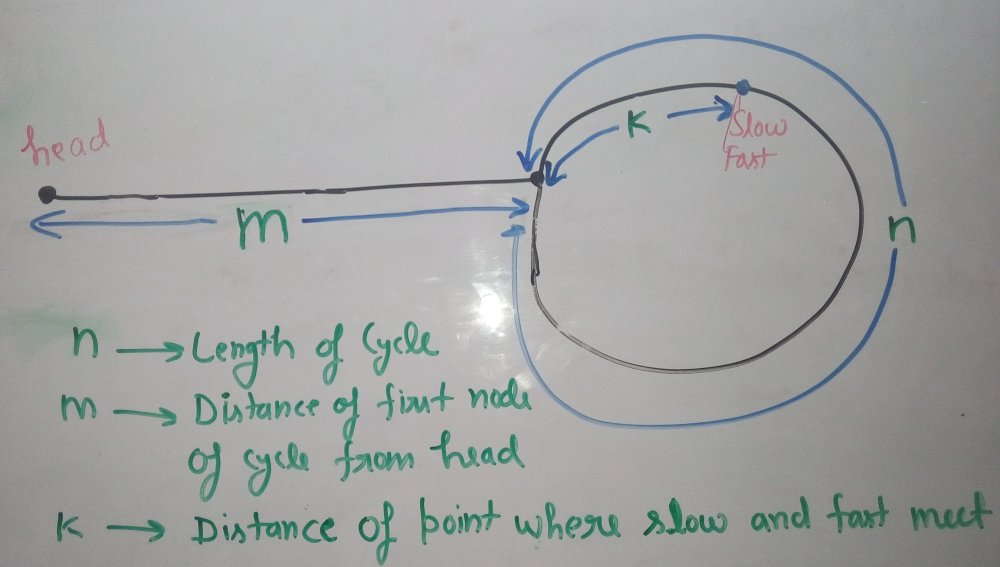
| |

\/ |

1 -> 2 -> 3 -> 4

Return the node corresponding to node 3.

**Approach / Theory**



Distance traveled by fast pointer = 2 \* (Distance traveled

by slow pointer)

(m + n\*x + k) = 2\*(m + n\*y + k)

Note that before meeting the point shown above, fast

was moving at twice speed.

x --> Number of complete cyclic rounds made by

fast pointer before they meet first time

y --> Number of complete cyclic rounds made by

slow pointer before they meet first time

From above equation, we can conclude below

m + k = (x-2y)\*n

Which means m+k is a multiple of n.

So if we start moving both pointers again at same speed such that one pointer (say slow) begins from head node of linked list and other pointer (say fast) begins from meeting point. When slow pointer reaches beginning of loop (has made m steps), fast pointer would have made also moved m steps as they are now moving same pace. Since m+k is a multiple of n and fast starts from k, they would meet at the beginning. Can they meet before also? No because slow pointer enters the cycle first time after m steps.

/\*\*

\* Definition for singly-linked list.

\* struct ListNode {

\* int val;

\* ListNode \*next;

\* ListNode(int x) : val(x), next(NULL) {}

\* };

\*/

ListNode\* Solution::detectCycle(ListNode\* head) {

ListNode \*slow, \*fast;

slow=fast=head;

while(fast && fast->next){

slow=slow->next;

fast=fast->next->next;

if(slow==fast)

break;

}

if(!slow || !fast || !fast->next)

return NULL;

slow=head;

while(slow!=fast){

slow=slow->next;

fast=fast->next;

}

return slow;

}

## Insertion Sort List

* Asked in:
* [Microsoft](https://www.interviewbit.com/search/?q=Microsoft)
* [Google](https://www.interviewbit.com/search/?q=Google)

Sort a linked list using insertion sort.

We have explained Insertion Sort at Slide 7 of [Arrays](http://www.interviewbit.com/courses/programming/topics/arrays/)

[Insertion Sort Wiki](http://en.wikipedia.org/wiki/Insertion_sort#Algorithm) has some details on Insertion Sort as well.

**Example :**

Input : 1 -> 3 -> 2

Return 1 -> 2 -> 3

**Approach: Insertion sort**

/\*\*

\* Definition for singly-linked list.

\* struct ListNode {

\* int val;

\* ListNode \*next;

\* ListNode(int x) : val(x), next(NULL) {}

\* };

\*/

/\*

void swap(ListNode \*A, ListNode \*B){

int tmp=A->val;

A->val=B->val;

B->val=tmp;

return;

}

ListNode\* Solution::insertionSortList(ListNode\* head) {

if(!head || !head->next) return head;

ListNode \*cur=head;

while(cur){

ListNode \*next=cur->next;

while(next){

if(cur->val > next->val)

swap(cur, next);

next=next->next;

}

cur=cur->next;

}

return head;

}

\*/

ListNode\* insert(ListNode \*head, ListNode \*node){

if(!head || head->val>=node->val){

node->next=head;

head=node;

}

else{

ListNode \*cur=head;

while(cur->next && cur->next->val<node->val)

cur=cur->next;

node->next=cur->next;

cur->next=node;

}

return head;

}

ListNode\* Solution::insertionSortList(ListNode\* head) {

if(!head || !head->next) return head;

ListNode \*sorted=NULL, \*unsorted=head;

ListNode \*next;

while(unsorted){

next=unsorted->next;

sorted=insert(sorted,unsorted);

unsorted=next;

}

return sorted;

}

**Approach: This isn’t insertion sort technique**

/\*\*

\* Definition for singly-linked list.

\* struct ListNode {

\* int val;

\* ListNode \*next;

\* ListNode(int x) : val(x), next(NULL) {}

\* };

\*/

void swap(ListNode \*A, ListNode \*B){

int tmp=A->val;

A->val=B->val;

B->val=tmp;

return;

}

ListNode\* Solution::insertionSortList(ListNode\* head) {

if(!head || !head->next) return head;

ListNode \*cur=head;

while(cur){

ListNode \*next=cur->next;

while(next){

if(cur->val > next->val)

swap(cur, next);

next=next->next;

}

cur=cur->next;

}

return head;

}

## Palindrome List

* Asked in:
* [Amazon](https://www.interviewbit.com/search/?q=Amazon)
* [Microsoft](https://www.interviewbit.com/search/?q=Microsoft)

Given a singly linked list, determine if its a palindrome. Return 1 or 0 denoting if its a palindrome or not, respectively.

**Notes**:

* Expected solution is linear in time and constant in space.

For example,

List 1-->2-->1 is a palindrome.

List 1-->2-->3 is not a palindrome.

Sol:

For finding mid point, first we can in O(N) traverse whole list and calculate total number of elements.  
Reversing is again O(N).

/\*\*

\* Definition for singly-linked list.

\* struct ListNode {

\* int val;

\* ListNode \*next;

\* ListNode(int x) : val(x), next(NULL) {}

\* };

\*/

ListNode \*reverse(ListNode \*head){

if(!head || !head->next) return head;

ListNode \*revHead=reverse(head->next);

head->next->next=head;

head->next=NULL;

return revHead;

}

int Solution::lPalin(ListNode\* head) {

if(!head || !head->next) return 1;

ListNode \*slow=head, \*fast=head->next;

while(fast && fast->next){

fast=fast->next->next;

slow=slow->next;

}

ListNode\* head1=slow->next;

slow->next=NULL;

ListNode\* head2=reverse(head1);

while(head && head2){

if(head->val != head2->val) return 0;

head=head->next;

head2=head2->next;

}

return 1;

}

## Sort List

Asked in:

* [Google](https://www.interviewbit.com/search/?q=Google)

Sort a linked list in O(n log n) time using constant space complexity.

**Example :**

Input : 1 -> 5 -> 4 -> 3

Returned list : 1 -> 3 -> 4 -> 5

Approach: mergesort gives O(n log n) time

/\*\*

\* Definition for singly-linked list.

\* struct ListNode {

\* int val;

\* ListNode \*next;

\* ListNode(int x) : val(x), next(NULL) {}

\* };

\*/

/\*

ListNode \*merge(ListNode \*A, ListNode \*B){

if(!A) return B;

if(!B) return A;

//cout<<"inside merge"<<endl;

ListNode \*head, \*tail;

if(A->val<=B->val){

head=A;

tail=A;

A=A->next;

}

else{

head=B;

tail=B;

B=B->next;

}

while(A && B){

if(A->val <= B->val){

tail->next=A;

A=A->next;

}

else{

tail->next=B;

B=B->next;

}

tail=tail->next;

}

if(A) tail->next=A;

if(B) tail->next=B;

return head;

}

ListNode\* Solution::sortList(ListNode \*head){

if(!head || !head->next) return head;

ListNode \*slow=head, \*fast=head->next;

//don't make fast=head, recursive call of sortList fails

//when list has 2 elements, and so list keeps dividing itself into size 2 and 0,

//this causes stack overflow

while(fast && fast->next){

slow=slow->next;

fast=fast->next->next;

}

ListNode \*head1=slow->next;

slow->next=NULL;

return merge(sortList(head), sortList(head1));

}

# Greedy Algorithm

## Highest Product

* Asked in:
* [Coursera](https://www.interviewbit.com/search/?q=Coursera)
* [Amazon](https://www.interviewbit.com/search/?q=Amazon)

Given an array of integers, return the highest product possible by multiplying 3 numbers from the array

**Input:**

array of integers e.g {1, 2, 3}

***NOTE:****Solution will fit in a 32-bit signed integer*

**Example:**

[0, -1, 3, 100, 70, 50]

=> 70\*50\*100 = 350000

Sol: When array is sorted in descending order, maximum triplet product can be

* + - * 1. The product of first 3 elements of the array OR
        2. The product of first element (i.e. highest positive number) and last two element (highest magnitude negative numbers)

int Solution::maxp3(vector<int> &A) {

sort(A.begin(), A.end(), greater<int>());

int n=A.size();

int product1=A[0]\*A[1]\*A[2];

int product2=A[0]\*A[n-1]\*A[n-2];

return max(product1, product2);

}

## Bulbs

**N** light bulbs are connected by a wire. Each bulb has a switch associated with it, however due to faulty wiring, a switch also changes the state of all the bulbs to the right of current bulb. Given an initial state of all bulbs, find the minimum number of switches you have to press to turn on all the bulbs. You can press the same switch multiple times.

**Note :** 0 represents the bulb is off and 1 represents the bulb is on.

**Example:**

Input : [0 1 0 1]

Return : 4

Explanation :

press switch 0 : [1 0 1 0]

press switch 1 : [1 1 0 1]

press switch 2 : [1 1 1 0]

press switch 3 : [1 1 1 1]

Sol: Assume count=number of state switches

When count is odd, it changes the binary array, and when count is even it brings the binary array back to its original values

int Solution::bulbs(vector<int> &bulb) {

int n=bulb.size();

int count=0;

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

if(bulb[i]==count%2)

count++;

}

return count;

}

## Assign Mice to Holes

* Asked in:
* [Amazon](https://www.interviewbit.com/search/?q=Amazon)
* There are N Mice and N holes are placed in a straight line.   
  Each hole can accomodate only 1 mouse.   
  A mouse can stay at his position, move one step right from x to x + 1, or move one step left from x to x − 1. Any of these moves consumes 1 minute.  
  Assign mice to holes so that the time when the last mouse gets inside a hole is minimized.
* **Example:**
* positions of mice are:
* 4 -4 2
* positions of holes are:
* 4 0 5
* Assign mouse at position x=4 to hole at position x=4 : Time taken is 0 minutes
* Assign mouse at position x=-4 to hole at position x=0 : Time taken is 4 minutes
* Assign mouse at position x=2 to hole at position x=5 : Time taken is 3 minutes
* After 4 minutes all of the mice are in the holes.
* Since, there is no combination possible where the last mouse's time is less than 4,
* answer = 4.
* **Input:**
* A : list of positions of mice
* B : list of positions of holes
* **Output:**
* single integer value
* ***NOTE:****The final answer will fit in a 32 bit signed integer.*

Sol: distance between position of mouse and position of hole will be minimum when both arrays are sorted.

int Solution::mice(vector<int> &mouse, vector<int> &hole) {

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

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

int maxdiff=INT\_MIN;

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

maxdiff=max(maxdiff, abs(mouse[i]-hole[i]));

}

return maxdiff;

}

## Majority Element

* Asked in:
* [Microsoft](https://www.interviewbit.com/search/?q=Microsoft)
* [Yahoo](https://www.interviewbit.com/search/?q=Yahoo)
* [Google](https://www.interviewbit.com/search/?q=Google)
* [Amazon](https://www.interviewbit.com/search/?q=Amazon)

Given an array of size n, find the majority element. The majority element is the element that appears more than floor(n/2) times.

You may assume that the array is non-empty and the majority element always exist in the array.

**Example :**

Input : [2, 1, 2]

Return : 2 which occurs 2 times which is greater than 3/2.

**Moore’s Voting Algorithm**

This Method only works when we are given that majority element do exist in the array.

The algorithm for first phase that works in O(n) is known as Moore’s Voting Algorithm. Basic idea of the algorithm is that if we cancel out each occurrence of an element *e* with all the other elements that are different from *e* then *e* will exist till end if it is a majority element.

findCandidate(a[], size)

1. Initialize index and count of majority element

maj\_index = 0, count = 1

2. Loop for i = 1 to size – 1

(a) If a[maj\_index] == a[i]

count++

(b) Else

count--;

(c) If count == 0

maj\_index = i;

count = 1

3. Return a[maj\_index]

int Solution::majorityElement(const vector<int> &A) {

int mindex=0, count=1;

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

if(A[i]==A[mindex]) count++;

else count--;

if(count==0){

mindex=i;

count=1;

}

}

return A[mindex];

}

## Distribute Candy

* Asked in:
* [Microsoft](https://www.interviewbit.com/search/?q=Microsoft)
* [Flipkart](https://www.interviewbit.com/search/?q=Flipkart)
* [Amazon](https://www.interviewbit.com/search/?q=Amazon)

There are N children standing in a line. Each child is assigned a rating value.

*You are giving candies to these children subjected to the following requirements:*

* Each child must have at least one candy.
* Children with a higher rating get more candies than their neighbors.

What is the minimum candies you must give?

**Sample Input :**

Ratings : [1 2]

**Sample Output :**

3

The candidate with 1 rating gets 1 candy and candidate with rating cannot get 1 candy as 1 is its neighbor. So rating 2 candidate gets 2 candies. In total, 2+1 = 3 candies need to be given out.

Sol: start assigning candies from 0th position as 1. When rating of ith child is grater than rating of i-1th child, assign candies[i-1]+1 candies to ith child, else assign 1.

Once candies assigned to child i, traverse back from i-1 to 0th child to check if constraints are violating.s

int Solution::candy(vector<int> &rating) {

int n=rating.size();

int sum=0;

vector<int> candies(n);

int i;

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

if(i>0 && rating[i]>rating[i-1])

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

else candies[i]=1;

int j=i;

while(j>0 && rating[j-1]>rating[j] && candies[j-1]<=candies[j]){

candies[j-1]++;

j--;

}

}

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

sum+=candies[i];

return sum;

}

## Gas Station

* Asked in:
* [Bloomberg](https://www.interviewbit.com/search/?q=Bloomberg)
* [Google](https://www.interviewbit.com/search/?q=Google)
* [DE Shaw](https://www.interviewbit.com/search/?q=DE)
* [Amazon](https://www.interviewbit.com/search/?q=Amazon)

There are N gas stations along a circular route, where the amount of gas at station i is gas[i].

You have a car with an unlimited gas tank and it costs cost[i] of gas to travel from station i to its next station (i+1). You begin the journey with an empty tank at one of the gas stations.

Return the minimum starting gas station’s index if you can travel around the circuit once, otherwise return -1.

You can only travel in one direction. i to i+1, i+2, ... n-1, 0, 1, 2..  
Completing the circuit means starting at i and ending up at i again.

**Example :**

Input :

Gas : [1, 2]

Cost : [2, 1]

Output : 1

If you start from index 0, you can fill in gas[0] = 1 amount of gas. Now your tank has 1 unit of gas. But you need cost[0] = 2 gas to travel to station 1.

If you start from index 1, you can fill in gas[1] = 2 amount of gas. Now your tank has 2 units of gas. You need cost[1] = 1 gas to get to station 0. So, you travel to station 0 and still have 1 unit of gas left over. You fill in gas[0] = 1 unit of additional gas, making your current gas = 2. It costs you cost[0] = 2 to get to station 1, which you do and complete the circuit.

int Solution::canCompleteCircuit(const vector<int> &gas, const vector<int> &cost) {

int tank=0, n=cost.size();

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

tank=gas[i];

if(tank<cost[i])

continue;

tank-=cost[i];

int j=(i+1)%n;

while(j!=i){

tank+=gas[j];

if(tank<cost[j])

break;

else{

tank-=cost[j];

j=(j+1)%n;

}

}

if(i==j)

return i;

}

return -1;

}

Sol2: Easy but doesn’t check circular path

int canCompleteCircuit(vector<int> &gas, vector<int> &cost) {

int sumGas = 0;

int sumCost = 0;

int start = 0;

int tank = 0;

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

sumGas += gas[i];

sumCost += cost[i];

tank += gas[i] - cost[i];

if (tank < 0) {

start = i + 1;

tank = 0;

}

}

if (sumGas < sumCost) {

return -1;

} else {

return start;

}

}

# Arrays

## Spiral Order Matrix I

* Asked in:
* [Microsoft](https://www.interviewbit.com/search/?q=Microsoft)
* [JP Morgan](https://www.interviewbit.com/search/?q=JP)
* [Amazon](https://www.interviewbit.com/search/?q=Amazon)
* [Flipkart](https://www.interviewbit.com/search/?q=Flipkart)
* Given a matrix of m \* n elements (m rows, n columns), return all elements of the matrix in spiral order.
* **Example:**
* Given the following matrix:
* [
* [ 1, 2, 3 ],
* [ 4, 5, 6 ],
* [ 7, 8, 9 ]
* ]
* You should return
* [1, 2, 3, 6, 9, 8, 7, 4, 5]

vector<int> Solution::spiralOrder(const vector<vector<int> > &A) {

vector<int> res;

int rstart=0, rend=A.size();

int cstart=0, cend=A[0].size();

while(rstart<rend && cstart<cend){

for(int j=cstart; j<cend; j++)

res.push\_back(A[rstart][j]);

rstart++;

for(int i=rstart; i<rend; i++)

res.push\_back(A[i][cend-1]);

cend--;

if(rstart<rend){//if this isn't checked, same row could be printed

for(int j=cend-1; j>=cstart; j--)

res.push\_back(A[rend-1][j]);

rend--;

}

if(cstart<cend){//if this isn't checked, same column could be printed

for(int i=rend-1; i>=rstart; i--)

res.push\_back(A[i][cstart]);

cstart++;

}

}

return res;

}

## Min Steps in Infinite Grid

* Asked in:
* [Directi](https://www.interviewbit.com/search/?q=Directi)
* You are in an infinite 2D grid where you can move in any of the 8 directions :
* (x,y) to
* (x+1, y),
* (x - 1, y),
* (x, y+1),
* (x, y-1),
* (x-1, y-1),
* (x+1,y+1),
* (x-1,y+1),
* (x+1,y-1)
* You are given a sequence of points and **the order in which you need to cover the points**. Give the minimum number of steps in which you can achieve it. You start from the first point.
* **Input :**
* Given two integer arrays A and B, where A[i] is x coordinate and B[i] is y coordinate of ith point respectively.
* **Output :**
* Return an Integer, i.e minimum number of steps.
* **Example :**
* Input : [(0, 0), (1, 1), (1, 2)]
* Output : 2
* It takes 1 step to move from (0, 0) to (1, 1). It takes one more step to move from (1, 1) to (1, 2).
* This question is intentionally left slightly vague. Clarify the question by trying out a few cases in the “See Expected Output” section.

Sol:

int Solution::coverPoints(vector<int> &X, vector<int> &Y) {

int n=X.size();

if(n==0 || n==1) return 0;

int minDistance=0;

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

int diffX=abs(X[i]-X[i+1]);

int diffY=abs(Y[i]-Y[i+1]);

minDistance+=max(diffX,diffY);

}

return minDistance;

}

## Add One To Number

* Asked in:
* [Google](https://www.interviewbit.com/search/?q=Google)
* [Microsoft](https://www.interviewbit.com/search/?q=Microsoft)

Given a non-negative number represented as an array of digits,

add 1 to the number ( increment the number represented by the digits ).

The digits are stored such that the most significant digit is at the head of the list.

**Example:**

If the vector has [1, 2, 3]

the returned vector should be [1, 2, 4]

as 123 + 1 = 124.

***NOTE:****Certain things are intentionally left unclear in this question which you should practice asking the interviewer.  
For example, for this problem, following are some good questions to ask :*

**Q :** Can the input have 0’s before the most significant digit. Or in other words, is 0 1 2 3 a valid input?

***A :****For the purpose of this question,****YES***

**Q :** Can the output have 0’s before the most significant digit? Or in other words, is 0 1 2 4 a valid output?

* **A :** For the purpose of this question, **NO**. Even if the input has zeroes before the most significant digit.

Sol:

vector<int> Solution::plusOne(vector<int> &A) {

vector<int> res;

int n=A.size();

if(n==0){

res.push\_back(1);

return res;

}

int carry=1;

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

int sum=A[i]+carry;

res.push\_back(sum%10);

carry=sum/10;

}

if(carry) res.push\_back(carry);

for(vector<int>::reverse\_iterator it=res.rbegin(); it!=res.rend(); ++it){

if(\*it==0) res.pop\_back();

else break;

}

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

return res;

}

## Max Sum Contiguous Subarray

* Asked in:
* [Facebook](https://www.interviewbit.com/search/?q=Facebook)
* [Paypal](https://www.interviewbit.com/search/?q=Paypal)
* [Yahoo](https://www.interviewbit.com/search/?q=Yahoo)
* [Microsoft](https://www.interviewbit.com/search/?q=Microsoft)
* [LinkedIn](https://www.interviewbit.com/search/?q=LinkedIn)
* [Amazon](https://www.interviewbit.com/search/?q=Amazon)
* [Goldman Sachs](https://www.interviewbit.com/search/?q=Goldman)

Find the contiguous subarray within an array (containing at least one number) which has the largest sum.

**For example:**

Given the array [-2,1,-3,4,-1,2,1,-5,4],

the contiguous subarray [4,-1,2,1] has the largest sum = 6.

For this problem, return the maximum sum.

int Solution::maxSubArray(const vector<int> &A) {

int n=A.size();

if(n==0) return 0;

int msum=A[0], sum=A[0];

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

if(sum+A[i]>A[i]) //adding cur element doesn’t increase sum more than cur //element then new subarray to be found from cur element

sum+=A[i];

else sum=A[i];

msum=max(msum,sum);

}

return msum;

}

## Maximum Absolute Difference

* Asked in:
* [Amazon](https://www.interviewbit.com/search/?q=Amazon)

You are given an array of N integers, A1, A2 ,…, AN. Return maximum value of f(i, j) for all 1 ≤ *i, j* ≤ N.  
f(i, j) is defined as |A[i] - A[j]| + |i - j|, where |x| denotes absolute value of x.

**For example**,

A=[1, 3, -1]

f(1, 1) = f(2, 2) = f(3, 3) = 0

f(1, 2) = f(2, 1) = |1 - 3| + |1 - 2| = 3

f(1, 3) = f(3, 1) = |1 - (-1)| + |1 - 3| = 4

f(2, 3) = f(3, 2) = |3 - (-1)| + |2 - 3| = 5

So, we return 5.

An **efficient** solution in O(n) time complexity can be worked out using the properties of absolute values.  
f(i, j) = |A[i] – A[j]| + |i – j| can be written in 4 ways (Since we are looking at max value, we don’t even care if the value becomes negative as long as we are also covering the max value in some way).

Case 1: A[i] > A[j] and i > j

|A[i] - A[j]| = A[i] - A[j]

|i -j| = i - j

hence, f(i, j) = (A[i] + i) - (A[j] + j)

Case 2: A[i] < A[j] and i < j

|A[i] - A[j]| = -(A[i]) + A[j]

|i -j| = -(i) + j

hence, f(i, j) = -(A[i] + i) + (A[j] + j)

Case 3: A[i] > A[j] and i < j

|A[i] - A[j]| = A[i] - A[j]

|i -j| = -(i) + j

hence, f(i, j) = (A[i] - i) - (A[j] - j)

Case 4: A[i] < A[j] and i > j

|A[i] - A[j]| = -(A[i]) + A[j]

|i -j| = i - j

hence, f(i, j) = -(A[i] - i) + (A[j] - j)

Note that case 1 and 2 are equivalent and so are case 3 and 4 and hence we can design our algorithm only for two cases as it will cover all the possible cases.

*1. Calculate the value of A[i] + i and A[i] – i for every element of the array while traversing through the array.*

*2. Then for the two equivalent cases, we find the maximum possible value. For that, we have to store minimum and maximum values of expressions A[i] + i and A[i] – i for all i.*

*3. Hence the required maximum absolute difference is maximum of two values i.e. max((A[i] + i) – (A[j] + j)) and max((A[i] – i) – (A[j] – j)). These values can be found easily in linear time.  
     a. For max((A[i] + i) – (A[j] + j)) Maintain two variables max1 and min1 which will store maximum and minimum values of A[i] + i respectively. max((A[i] + i) – (A[j] + j)) = max1 – min1  
     b. For max((A[i] – i) – (A[j] – j)). Maintain two variables max2 and min2 which will store maximum and minimum values of A[i] – i respectively. max((A[i] – i) – (A[j] – j)) = max2 – min2*

int Solution::maxArr(vector<int> &A) {

int n=A.size();

if(n==0) return 0;

int max1=A[0]+1;

int min1=max1;

int max2=A[0]-1;

int min2=max2;

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

int t1=A[i]+i+1;

int t2=A[i]-i-1;

max1=max(max1, t1);

min1=min(min1, t1);

max2=max(max2, t2);

min2=min(min2, t2);

}

return max((max1-min1),(max2-min2));

}

## Repeat and Missing Number Array

* Asked in:
* [Amazon](https://www.interviewbit.com/search/?q=Amazon)
* take care of **overflows** in your problem.
* order the operations cleverly so that the numbers do not overflow.

typedef long long LL;

vector<int> Solution::repeatedNumber(const vector<int> &A) {

vector<int> res;

int n=A.size();

if(n==0) return res;

LL sum=0, sum2=0;

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

LL temp=A[i];

sum+=temp;

sum-=(i+1);

sum2+=temp\*temp;

sum2-=LL(i+1)\*(i+1);

}

LL sum3=sum2/sum;

int A1=(sum3+sum)/2;

int B=A1-sum;

res.push\_back(A1);

res.push\_back(B);

return res;

}

## Flip

* Asked in:
* [VMWare](https://www.interviewbit.com/search/?q=VMWare)

You are given a binary string(*i.e.* with characters 0 and 1) S consisting of characters S1, S2, …, SN. In a single operation, you can choose two indices L and R such that 1 ≤ L ≤ R ≤ N and flip the characters SL, SL+1, …, SR. By flipping, we mean change character 0 to 1 and vice-versa.

Your aim is to perform ATMOST one operation such that in final string number of 1s is maximised. If you don’t want to perform the operation, return an empty array. Else, return an array consisting of two elements denoting L and R. If there are multiple solutions, return the lexicographically smallest pair of L and R.

**Notes**:

* Pair (a, b) is lexicographically smaller than pair (c, d) if a < c or, if a == c and b < d.

For example,

S = 010

Pair of [L, R] | Final string

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_\_\_

[1 1] | 110

[1 2] | 100

[1 3] | 101

[2 2] | 000

[2 3] | 001

We see that two pairs [1, 1] and [1, 3] give same number of 1s in final string. So, we return [1, 1].

Another example,

If S = 111

No operation can give us more than three 1s in final string. So, we return empty array [].

**Sol:**

**//Kadane’s algorithm modified**

vector<int> Solution::flip(string A) {

vector<int> res;

int cur0=0, max0=0;

int l=0;

int L=-1, R=-1;

int n=A.size();

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

cur0 += (A[i]=='0'?1:-1);

if(cur0 > max0){

max0=cur0;

L=l;

R=i;

}

if(cur0 < 0)//1s are more in string at position i

{

//skip current L and set it to next of current position i

l=i+1;

cur0=0;

}

}

if(L==-1 || R==-1)

return res;

L++; R++;

res.push\_back(L);

res.push\_back(R);

return res;

}

## Merge Intervals

* Asked in:
* [Google](https://www.interviewbit.com/search/?q=Google)

Given a set of non-overlapping intervals, insert a new interval into the intervals (merge if necessary).

You may assume that the intervals were initially sorted according to their start times.

**Example 1:**

Given intervals [1,3],[6,9] insert and merge [2,5] would result in [1,5],[6,9].

**Example 2:**

Given [1,2],[3,5],[6,7],[8,10],[12,16], insert and merge [4,9] would result in [1,2],[3,10],[12,16].

This is because the new interval [4,9] overlaps with [3,5],[6,7],[8,10].

Make sure the returned intervals are also sorted.

**Approach:**

1) Size of interval array as 0.

2) newInterval being an interval preceding all intervals in the array.

Given interval (3,6),(8,10), insert and merge (1,2)

3) newInterval being an interval succeeding all intervals in the array.

Given interval (1,2), (3,6), insert and merge (8,10)

4) newInterval not overlapping with any interval and falling in between 2 intervals in the array.

Given interval (1,2), (8,10) insert and merge (3,6)

5) newInterval covering all given intervals.

Given interval (3, 5), (7, 9) insert and merge (1, 10)

6) newInterval overlapping with other intervals

1. Current interval in vector is falling before newInterval
2. Current interval in vector is falling after newInterval
3. Case BAAB
4. Case BABA
5. Case ABBA
6. Case ABAB

Corner cases: when end of merge interval is end of interval B

When B overlaps all intervals

/\*\*

\* Definition for an interval.

\* struct Interval {

\* int start;

\* int end;

\* Interval() : start(0), end(0) {}

\* Interval(int s, int e) : start(s), end(e) {}

\* };

\*/

vector<Interval> Solution::insert(vector<Interval> &A, Interval B) {

// Do not write main() function.

// Do not read input, instead use the arguments to the function.

// Do not print the output, instead return values as specified

// Still have a doubt. Checkout www.interviewbit.com/pages/sample\_codes/ for more details

if(B.start > B.end)

swap(B.start, B.end);

vector<Interval> res;

int n=A.size();

int start=B.start, end=B.end;

//case 1:given vector is empty

if(n==0) {

res.push\_back(B);

return res;

}

//case 2: B at very first

if(end<A[0].start){

A.push\_back(B);

rotate(A.rbegin(), A.rbegin()+1, A.rend());

return A;

}

//case 3: B at very end

if(start>A[n-1].end){

res=A; res.push\_back(B);

return res;

}

//case 4: B overlaps all intervals

if(start<A[0].start && end>A[n-1].end){

res.push\_back(B);

return res;

}

//case 5: B is separate/overlapping and in between

int left=-1, right=-1; //currently no start and end of merged interval found

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

if(A[i].end<start){//when current interval is coming before B

res.push\_back(A[i]);

if(i+1<n && A[i+1].start>end) res.push\_back(B);//when B is not overlapped

}

else if(A[i].start>end){//when current interval is coming after B

if(left!=-1 && right!=-1){//insert previously found merged interval

res.push\_back(Interval(left, right));

left=-1;//reset left, right as interval is merged

right=-1;

}

res.push\_back(A[i]); //insert current interval

}

else if(start<A[i].start){//case BAAB and BABA

if(left==-1) left=start;

right=max(A[i].end, end);

}

else{//case ABBA and ABAB

if(left==-1) left=A[i].start;

right=max(A[i].end, end);

}

}

if(left!=-1 && right!=-1){//for loop doesn't add merged interval for below conditions

Interval m=Interval(left, right);//merged interval

if(res.empty()) res.push\_back(m);

else{

Interval i=res.back();

if(i.start!=m.start && i.end!=m.end)

res.push\_back(m);

}

}

return res;

}

## Merge Overlapping Intervals

* Asked in:
* [Google](https://www.interviewbit.com/search/?q=Google)
* [Amazon](https://www.interviewbit.com/search/?q=Amazon)

Given a collection of intervals, merge all overlapping intervals.

**For example:**

Given [1,3],[2,6],[8,10],[15,18],

return [1,6],[8,10],[15,18].

Make sure the returned intervals are sorted.

struct mystruct{

bool operator()(const Interval &a, const Interval &b){return a.start<b.start;}

}myobject;

/\*\*

\* Definition for an interval.

\* struct Interval {

\* int start;

\* int end;

\* Interval() : start(0), end(0) {}

\* Interval(int s, int e) : start(s), end(e) {}

\* };

\*/

vector<Interval> Solution::merge(vector<Interval> &A) {

int n=A.size();

vector<Interval> res;

if(n<=1) return A;

sort(A.begin(), A.end(), myobject);

int preStart=A[0].start;

int preEnd=A[0].end;

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

if(preEnd<A[i].start){

res.push\_back(Interval(preStart, preEnd));

preStart=A[i].start;

preEnd=A[i].end;

}

else{

preEnd=max(preEnd, A[i].end);

}

}

res.push\_back(Interval(preStart, preEnd));

return res;

}

## Max Non Negative SubArray

* Asked in:
* [Google](https://www.interviewbit.com/search/?q=Google)

Find out the maximum sub-array of non negative numbers from an array.  
The sub-array should be continuous. That is, a sub-array created by choosing the second and fourth element and skipping the third element is invalid.

Maximum sub-array is defined in terms of the sum of the elements in the sub-array. Sub-array A is greater than sub-array B if sum(A) > sum(B).

**Example:**

A : [1, 2, 5, -7, 2, 3]

The two sub-arrays are [1, 2, 5] [2, 3].

The answer is [1, 2, 5] as its sum is larger than [2, 3]

**NOTE:** If there is a tie, then compare with segment's length and return segment which has maximum length  
**NOTE 2:** If there is still a tie, then return the segment with minimum starting index

Approach:

Loop i = 1 to Array.length :

IF current element is positive :

update current sum

compare max sum with current sum

update max sum

update max ranges

ELSE :

current sum := 0

update current ranges.

EndLoop;

return elements of max ranges

typedef long long int LL;

vector<int> Solution::maxset(vector<int> &A) {

vector<int> res;

int n=A.size();

LL csum=0, msum=0;

int left=-1, right=-1, l=-1, r=-1;

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

if(A[i]>=0){

csum+=A[i];

if(l==-1) l=i;

r=i;

if(csum>msum || (csum==msum && (right-left)<(r-l))){

left=l;

right=r;

msum=csum;

}

}

else{

csum=0; l=-1; r=-1;

}

}

if(left==-1) return res;

for(int i=left; i<=right; i++)

res.push\_back(A[i]);

return res;

}

## Spiral Order Matrix II

* Asked in:
* [Microsoft](https://www.interviewbit.com/search/?q=Microsoft)
* [JP Morgan](https://www.interviewbit.com/search/?q=JP)
* [Amazon](https://www.interviewbit.com/search/?q=Amazon)
* Given an integer n, generate a square matrix filled with elements from 1 to n2 in spiral order.
* **Example:**
* Given n = 3,
* You should return the following matrix:
* [
* [ 1, 2, 3 ],
* [ 8, 9, 4 ],
* [ 7, 6, 5 ]
* ]

vector<vector<int> > Solution::generateMatrix(int n) {

vector<vector<int>> A(n, vector<int>(n));

int rstart=0, rend=n, cstart=0, cend=n;

int num=1;

while(rstart<rend && cstart<cend){

for(int c=cstart; c<cend; c++)

A[rstart][c]=num++;

rstart++;

for(int r=rstart; r<rend; r++)

A[r][cend-1]=num++;

cend--;

if(rstart<rend){

for(int c=cend-1; c>=cstart; c--)

A[rend-1][c]=num++;

rend--;

}

if(cstart<cend){

for(int r=rend-1; r>=rstart; r--)

A[r][cstart]=num++;

cstart++;

}

}

return A;

}

## Pascal Triangle

* Asked in:
* [Google](https://www.interviewbit.com/search/?q=Google)
* [Amazon](https://www.interviewbit.com/search/?q=Amazon)
* Given numRows, generate the first numRows of Pascal’s triangle.
* Pascal’s triangle : To generate A[C] in row R, sum up A’[C] and A’[C-1] from previous row R - 1.
* **Example:**
* Given numRows = 5,
* Return
* [
* [1],
* [1,1],
* [1,2,1],
* [1,3,3,1],
* [1,4,6,4,1]
* ]

void print(vector<int> &A){

int n=A.size();

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

cout<<A[i]<<" ";

cout<<endl;

}

vector<vector<int> > Solution::solve(int n) {

vector<vector<int>> res;

for(int r=0; r<n; r++){

int col=r+1;

vector<int> temp(col);//initializes vector with col elements and 0 value

if(r==0){

temp[r]=1;//don’t use push\_back; it adds to default values

res.push\_back(temp);

//print(temp);

temp.clear();

continue;

}

for(int c=0; c<col; c++){

if(c==0||c==col-1)

temp[c]=1;

else temp[c]=res[r-1][c-1]+res[r-1][c];

}

res.push\_back(temp);

temp.clear();

}

return res;

}

## Kth Row of Pascal's Triangle

* Asked in:
* [Google](https://www.interviewbit.com/search/?q=Google)

Given an index k, return the kth row of the Pascal’s triangle.

Pascal’s triangle : To generate A[C] in row R, sum up A’[C] and A’[C-1] from previous row R - 1.

**Example:**

Input : k = 3

Return : [1,3,3,1]

***NOTE****: k is 0 based. k = 0, corresponds to the row [1].*

 Note:Could you optimize your algorithm to use only O(k) extra space?

**Mathematics Formula:**

vector<int> Solution::getRow(int k) {

vector<int> pascal;

int n=1;

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

pascal.push\_back(n);

n=n\*(k-i)/(i+1);

}

return pascal;

}

**Brute force:**

vector<int> Solution::getRow(int k) {

vector<vector<int>> pascal;

for(int r=0; r<=k; r++){

int col=r+1;

vector<int>temp(col);

if(r==0){

temp[r]=1;

if(k==r) return temp;

pascal.push\_back(temp);

temp.clear();

continue;

}

for(int c=0; c<col; c++){

if(c==0||c==col-1) temp[c]=1;

else temp[c]=pascal[r-1][c-1]+pascal[r-1][c];

}

pascal.push\_back(temp);

if(r==k) return temp;

temp.clear();

}

}

## Anti Diagonals

* Asked in:
* [Microsoft](https://www.interviewbit.com/search/?q=Microsoft)
* [Adobe](https://www.interviewbit.com/search/?q=Adobe)
* Give a N\*N square matrix, return an array of its anti-diagonals. Look at the example for more details.

**Example:**

Input:

1 2 3

4 5 6

7 8 9

Return the following :

[

[1],

[2, 4],

[3, 5, 7],

[6, 8],

[9]

]

Input :

1 2

3 4

Return the following :

[

[1],

[2, 3],

[4]

]

vector<vector<int> > Solution::diagonal(vector<vector<int> > &A) {

int n=A.size();

vector<vector<int> > res(2\*n-1);

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

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

res[i+j].push\_back(A[i][j]);

}

return res;

}

## Noble Integer

Given an integer array, find if an integer p exists in the array such that the number of integers greater than p in the array equals to p  
If such an integer is found return 1 else return -1.

struct myclass{

bool operator()(int a, int b)

{

return (a<b);

}

}myobject;

int Solution::solve(vector<int> &A) {

sort(A.begin(), A.end(), myobject);

int n=A.size();

if(A[n-1]==0) return 1;

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

if(A[i]<A[i+1]){

if(A[i]==n-i-1) return 1;

}

}

return -1;

}

## Largest Number

* Asked in:
* [Amazon](https://www.interviewbit.com/search/?q=Amazon)
* [Goldman Sachs](https://www.interviewbit.com/search/?q=Goldman)
* [Microsoft](https://www.interviewbit.com/search/?q=Microsoft)

Given a list of non negative integers, arrange them such that they form the largest number.

**For example:**

Given [3, 30, 34, 5, 9], the largest formed number is 9534330.

*Note: The result may be very large, so you need to return a string instead of an integer.*

**Approach:** Given two numbers X and Y, how should myCompare() decide which number to put first – we compare two numbers XY (Y appended at the end of X) and YX (X appended at the end of Y).

If XY is larger, then, in the output, X should come before Y, else Y should come before X.

struct myclass{

bool operator()(string a, string b){

return (a+b>b+a);

}

}myobject;

string Solution::largestNumber(const vector<int> &A) {

string res;

vector<string> input;

int n=A.size();

int zeros=0;

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

zeros+=(A[i]==0?1:0);

input.push\_back(to\_string(A[i]));

}

if(zeros==n) return "0";

sort(input.begin(), input.end(), myobject);

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

res+=input[i];

return res;

}

## Wave Array

* Asked in:
* [Google](https://www.interviewbit.com/search/?q=Google)
* [Adobe](https://www.interviewbit.com/search/?q=Adobe)
* [Amazon](https://www.interviewbit.com/search/?q=Amazon)

Given an array of integers, sort the array into a wave like array and return it,   
In other words, arrange the elements into a sequence such that a1 >= a2 <= a3 >= a4 <= a5.....

**Example**

Given [1, 2, 3, 4]

One possible answer : [2, 1, 4, 3]

Another possible answer : [4, 1, 3, 2]

***NOTE :****If there are multiple answers possible, return the ne thats lexicographically smallest.   
So, in example case, you will return [2, 1, 4, 3]*

vector<int> Solution::wave(vector<int> &A) {

int n=A.size();

if(n==0 || n==1) return A;

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

for(int a=0; a<A.size()-1; a=a+2){

swap(A[a],A[a+1]);

return A;

}

## Hotel Bookings Possible

* Asked in:
* [Goldman Sachs](https://www.interviewbit.com/search/?q=Goldman)
* [Microsoft](https://www.interviewbit.com/search/?q=Microsoft)

BookmarkSuggest Edit

A hotel manager has to process N advance bookings of rooms for the next season. His hotel has Krooms. Bookings contain an arrival date and a departure date. He wants to find out whether there are enough rooms in the hotel to satisfy the demand. Write a program that solves this problem in time O(N log N) .

Input:

First list for arrival time of booking.

Second list for departure time of booking.

Third is K which denotes count of rooms.

Output:

A boolean which tells whether its possible to make a booking.

Return 0/1 for C programs.

O -> No there are not enough rooms for N booking.

1 -> Yes there are enough rooms for N booking.

**Example :**

Input :

Arrivals : [1 3 5]

Departures : [2 6 8]

K : 1

Return : False / 0

At day = 5, there are 2 guests in the hotel. But I have only one room.

bool Solution::hotel(vector<int> &arrive, vector<int> &depart, int K) {

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

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

int arr=0, dep=0;

while(arr<arrive.size()){

arr++; //increase guest count of arrival

K--; //decrease available rooms

if(K==0){

if(arrive[arr]>=depart[dep]){

K++;//increase available rooms

dep++;//increase guest count of departing

}

}

if(K<0) return 0;

}

return 1;

}

## Find Duplicate in Array

* Asked in:
* [Amazon](https://www.interviewbit.com/search/?q=Amazon)
* [VMWare](https://www.interviewbit.com/search/?q=VMWare)
* [Riverbed](https://www.interviewbit.com/search/?q=Riverbed)
* Given a read only array of n + 1 integers between 1 and n, find one number that repeats in linear time using less than O(n) space and traversing the stream sequentially O(1) times.
* **Sample Input:**
* [3 4 1 4 1]
* **Sample Output:**
* 1
* If there are multiple possible answers ( like in the sample case above ), output any one.
* If there is no duplicate, output -1

A **simple solution** is to create a count array and store counts of all elements. As soon as we encounter an element with count more than 1, we return it. This solution works in O(n) time and requires O(n) extra space.

A **space optimized solution** is to break the given range (from 1 to n) into blocks of size equal to sqrt(n). We maintain the count of elements belonging to each block for every block. Now as the size of array is (n+1) and blocks are of size sqrt(n), then there will be one such block whose size will be more than sqrt(n). For the block whose count is greater than sqrt(n), we can use hashing for the elements of this block to find which element appears more than once.

Sol:

int Solution::repeatedNumber(const vector<int> &A) {

int n=A.size()-1;//array has n+1 elements

int range=sqrt(n);

int block=n/range+1;//total blocks

int count[block];

memset(count,0, sizeof(count));//initial count for all blocks is 0

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

count[(A[i]-1)/range]++;//subtract 1 from A[i] to have elements upto range in current block

}

int selected\_block=block-1; //last block has <=range elements

//when all other blocks have exact range elements, duplicate number lies in last block

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

if(count[i]>range){

selected\_block=i;

break;

}

}

unordered\_map<int,int> m;//build map for block having >range elements

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

if((A[i]>selected\_block\*range)&&A[i]<=(selected\_block+1)\*range)

{ m[A[i]]++;

if(m[A[i]]>1) return A[i];

}

}

return -1;

}

## Maximum Consecutive Gap

* Asked in:
* [Hunan Asset](https://www.interviewbit.com/search/?q=Hunan)

Given an unsorted array, find the maximum difference between the successive elements in its sorted form.

Try to solve it in linear time/space.

**Example :**

Input : [1, 10, 5]

Output : 5

**Return 0 if the array contains less than 2 elements.**

* You may assume that all the elements in the array are non-negative integers and fit in the 32-bit signed integer range.
* You may also assume that the difference will not overflow.
* A **simple solution**is to first sort the array, then traverse it and keep track of maximum difference between adjacent elements. Time complexity of this
* An **efficient solution** is based on idea of [Pigeonhole sorting](https://www.geeksforgeeks.org/pigeonhole-sort/). We dont actually sort the array, we just have to fill the buckets and keep track of maximum and minimum value of each bucket. If we found an empty bucket, The maximum gap would be the difference of **maximum value in previous bucket – minimum value in next bucket**.

int Solution::maximumGap(const vector<int> &A) {

int n=A.size();

if(n<2) return 0;

int minV=\*min\_element(A.begin(), A.end());

int maxV=\*max\_element(A.begin(), A.end());

//n elements have n-1 buckets, each bucket holds min value at right and max value at left

int leftMax[n-1], rightMin[n-1];

fill\_n(leftMax, n-1, INT\_MIN);

fill\_n(rightMin, n-1, INT\_MAX);

float avgGap=float(maxV-minV)/(n-1);//avg gap between n-1 buckets

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

if(A[i]==minV || A[i]==maxV)

continue;

int index=(A[i]-minV)/avgGap;//find bucket index in which ith element will fall

//update leftMax and rightMin

if(leftMax[index]==INT\_MIN)

leftMax[index]=A[i];

else

leftMax[index]=max(leftMax[index], A[i]);

if(rightMin[index]==INT\_MAX)

rightMin[index]=A[i];

else

rightMin[index]=min(rightMin[index], A[i]);

}

//max\_gap = max of rightMin(current)-leftMax(previous)

int max\_gap=0, prevMax=minV;//smallest element in array

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

if(rightMin[i]==INT\_MAX)//ith bucket has no element

continue;

max\_gap=max(max\_gap, rightMin[i]-prevMax);

prevMax=leftMax[i];//update previous max value as max at current bucket

}

max\_gap=max(max\_gap, maxV-prevMax);

return max\_gap;

}

## Maximum Unsorted Subarray

* Asked in:
* [Amazon](https://www.interviewbit.com/search/?q=Amazon)
* You are given an array (zero indexed) of **N** non-negative integers, **A0**, **A1** ,…, **AN-1**.  
  Find the minimum sub array **Al**, **Al+1** ,…, **Ar** so if we sort(in ascending order) that sub array, then the whole array should get sorted.  
  If **A** is already sorted, output **-1**.
* **Example :**
* Input 1:
* A = [1, 3, 2, 4, 5]
* Return: [1, 2]
* Input 2:
* A = [1, 2, 3, 4, 5]
* Return: [-1]
* In the above example(Input 1), if we sort the subarray **A1**, **A2**, then whole array **A** should get sorted.

**Solution:**  
**1) Find the candidate unsorted subarray**  
a) Scan from left to right and find the first element which is greater than the next element. Let *s*be the index of such an element. In the above example 1, *s*is 3 (index of 30).  
b) Scan from right to left and find the first element (first in right to left order) which is smaller than the next element (next in right to left order). Let *e*be the index of such an element. In the above example 1, e is 7 (index of 31).

**2) Check whether sorting the candidate unsorted subarray makes the complete array sorted or not. If not, then include more elements in the subarray.**  
a) Find the minimum and maximum values in *arr[s..e]*. Let minimum and maximum values be *min*and *max*. *min*and *max*for [30, 25, 40, 32, 31] are 25 and 40 respectively.  
b) Find the first element (if there is any) in *arr[0..s-1]* which is greater than *min*, change *s*to index of this element. There is no such element in above example 1.  
c) Find the last element (if there is any) in *arr[e+1..n-1]*which is smaller than max, change *e*to index of this element. In the above example 1, e is changed to 8 (index of 35)

**3) Print *s*and *e*.**

vector<int> Solution::subUnsort(vector<int> &A) {

int s=-1, e=-1;

vector<int> res;

int n=A.size();

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

if((A[i]>A[i+1])&&(s==-1)){

s=i;

break;

}

}

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

if((A[j]<A[j-1])&&e==-1){

e=j;

break;

}

}

int max=\*max\_element(A.begin()+s, A.begin()+e+1);

int min=\*min\_element(A.begin()+s, A.begin()+e+1);

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

if(A[i]>min){

s=i; break;

}

}

for(int j=n-1; j>e; j--){

if(A[j]<max){

e=j; break;

}

}

res.push\_back(s);

res.push\_back(e);

if(s==-1 || e==-1) res.pop\_back();

return res;

}

## MAXSPPROD

You are given an array **A** containing **N** integers. The special product of each ith integer in this array is defined as the product of the following:<ul>

 LeftSpecialValue: For an index i, it is defined as the index j such that A[j]>A[i] (i>j). If multiple A[j]’s are present in multiple positions, the LeftSpecialValue is the maximum value of j.

 RightSpecialValue: For an index i, it is defined as the index j such that A[j]>A[i] (j>i). If multiple A[j]s are present in multiple positions, the RightSpecialValue is the minimum value of j.

Write a program to find the maximum special product of any integer in the array.

Input: You will receive array of integers as argument to function.

Return: Maximum special product of any integer in the array modulo 1000000007.

**Note**: If j does not exist, the LeftSpecialValue and RightSpecialValue are considered to be 0.

Constraints 1 <= N <= 10^5 1 <= A[i] <= 10^9

**Solution**:Given an array a[1..N]. For each element at position i (1 <= i <= N). Where

1. **L(i)** is defined as closest index j such that j < i and a[j] > a[i]. If no such j exists then **L(i) = 0**.
2. **R(i)** is defined as closest index k such that k > i and a[k] > a[i]. If no such k exists then **R(i) = 0**.

**LRProduct(i) = L(i)\*R(i)**.

typedef long long LL;

#define MOD 1000000007;

vector<int> rightNextGreater(vector<int> &A){

int n=A.size();

vector<int> res(n,0);

stack<int> st;

st.push(0);

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

while(!st.empty() && A[st.top()]<A[i]){

res[st.top()]=i;

st.pop();

}

st.push(i);

}

//res[n-1]=0;

return res;

}

vector<int> leftNextGreater(vector<int> &A){

int n=A.size();

vector<int> res(n,0);

stack<int> st;

st.push(n-1);

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

while(!st.empty() && A[st.top()]<A[i]){

res[st.top()]=i;

st.pop();

}

st.push(i);

}

//res[0]=0;

return res;

}

int Solution::maxSpecialProduct(vector<int> &A) {

int n=A.size();

if(n<=2) return 0;

vector<int> rightSE=rightNextGreater(A);

vector<int> leftSE=leftNextGreater(A);

LL max\_prod=0;

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

LL prod=((LL)rightSE[i]\*leftSE[i]);

max\_prod=max(max\_prod, prod);

}

return max\_prod%MOD;

}

## Rotate Matrix

* Asked in:
* [Google](https://www.interviewbit.com/search/?q=Google)
* [Facebook](https://www.interviewbit.com/search/?q=Facebook)
* [Amazon](https://www.interviewbit.com/search/?q=Amazon)

You are given an n x n 2D matrix representing an image.

Rotate the image by 90 degrees (clockwise).

You need to do this in place.

Note that if you end up using an additional array, you will only receive partial score.

**Example:**

If the array is

[

[1, 2],

[3, 4]

]

Then the rotated array becomes:

[

[3, 1],

[4, 2]

]

Now lets say we have to do things in place ( **no extra space allowed** ). This implies that we have to make things work with just moving elements around with constant extra memory.

Again, a group of 4. So, we can move these elements group by group without requiring creating a copy of the array.

void Solution::rotate(vector<vector<int> > &A) {

int len=A.size();

for(int i=0; i<len/2; i++){

for(int j=i; j<len-i-1; j++){

int tmp=A[i][j];

A[i][j]=A[len-j-1][i];

A[len-j-1][i]=A[len-i-1][len-j-1];

A[len-i-1][len-j-1]=A[j][len-i-1];

A[j][len-i-1]=tmp;

}

}

}

## Next Permutation

* Asked in:
* [Microsoft](https://www.interviewbit.com/search/?q=Microsoft)
* [Amazon](https://www.interviewbit.com/search/?q=Amazon)

Implement the next permutation, which rearranges numbers into the numerically next greater permutation of numbers.

If such arrangement is not possible, it must be rearranged as the lowest possible order ie, sorted in an ascending order.

The replacement must be in-place, do not allocate extra memory.

**Examples:**

1,2,3 → 1,3,2

3,2,1 → 1,2,3

1,1,5 → 1,5,1

20, 50, 113 → 20, 113, 50

Following is the algorithm for finding the next greater number.  
**I)**Traverse the given number from rightmost digit, keep traversing till you find a digit which is smaller than the previously traversed digit. For example, if the input number is “534976”, we stop at **4** because 4 is smaller than next digit 9. If we do not find such a digit, then output is “Not Possible”.

**II)** Now search the right side of above found digit ‘d’ for the smallest digit greater than ‘d’. For “53**4**976″, the right side of 4 contains “976”. The smallest digit greater than 4 is **6**.

**III)**Swap the above found two digits, we get 53**6**97**4** in above example.

**IV)** Now sort all digits from position next to ‘d’ to the end of number. The number that we get after sorting is the output. For above example, we sort digits in bold 536**974**. We get “536**479**” which is the next greater number for input 534976.

void Solution::nextPermutation(vector<int> &A) {

int n=A.size();

int i;

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

if(A[i]>A[i-1]) break;

if(i==0){

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

return;

}

int idx=i-1;

int cur=i;

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

if(A[j]>A[idx]&&A[j]<A[cur])

cur=j;

swap(A[idx], A[cur]);

sort(A.begin()+idx+1, A.end());

}

## Find Permutation

Given a positive integer n and a string s consisting only of letters *D* or *I*, you have to find any permutation of first n positive integer that satisfy the given input string.

*D* means the next number is smaller, while *I* means the next number is greater.

**Notes**

* Length of given string s will always equal to n - 1
* Your solution should run in linear time and space.

**Example :**

Input 1:

n = 3

s = ID

Return: [1, 3, 2]

When the input string contains only *D* or *I* we just need to return all positive number upto n either in descending or ascending orders respectively.  
So if *n* = 3, *s* = “II”, return [1, 2, 3]

Now, starting with each character of the input string, we need to substitute an appropriate number(from *1* to *n*) corresponding to each character(*I* or *D*).

So, Suppose we started with a set corresponding to all the elements from that we need to make permutation(i.e all integer from *1* to *n*).

As *I* denotes the next number should be larger, we need to substitute smallest remaining number from our set corresponding to subsequent *I* as it automatically makes the next element to be larger.

Similar things will happens with character *D*, we need to substitute the largest remaining number from our set.

As the input string size is n - 1, we to append the last integer to our answer

vector<int> Solution::findPerm(const string A, int N) {

int n=A.size()+1;

vector<int> res;

set<int> s;

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

s.insert(i);

int num;

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

if(A[i]=='I'){

num=\*s.begin();

res.push\_back(num);

}

else{

num=\*s.rbegin();

res.push\_back(num);

}

s.erase(num);

}

res.push\_back(\*s.begin());

return res;

}

## Set Matrix Zeros

* Asked in:
* [Oracle](https://www.interviewbit.com/search/?q=Oracle)
* [Amazon](https://www.interviewbit.com/search/?q=Amazon)

Given an **m x n matrix** of 0s and 1s, if an element is 0, set its entire row and column to 0.

Do it in place.

**Example**

Given array A as

1 0 1

1 1 1

1 1 1

On returning, the array A should be :

0 0 0

1 0 1

1 0 1

Use the first row and first column of the input matrix in as auxiliary arrays row[] and col[] to store the info of submatrix([1,1]…[m-1][n-1]). i.e. if A[i][j]==0 then set A[0][j]=A[i][0]=0;

This will modify the first row and first column.

To decide whether entire first row or column needs modification, maintain variables rowflag and colflag. If 0th row or 0th col has 0, then set rowflag and colflag accordingly.

Update submatrix ((1,1),…, (m-1),(n-1)) first and later update 0th row and 0th column.

void Solution::setZeroes(vector<vector<int> > &A) {

int m=A.size();

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

bool rowflag=false, colflag=false;

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

for(int j=0; j<n; j++){

if(i==0 && A[i][j]==0)

rowflag=true;

if(j==0 && A[i][j]==0)

colflag=true;

if(A[i][j]==0)

A[0][j]=A[i][0]=0;

}

}

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

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

if(A[i][0]==0 || A[0][j]==0)

A[i][j]=0;

}

if(rowflag){

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

A[0][j]=0;

}

if(colflag){

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

A[i][0]=0;

}

}

## First Missing Integer

* Asked in:
* [Model N](https://www.interviewbit.com/search/?q=Model)
* [InMobi](https://www.interviewbit.com/search/?q=InMobi)
* [Amazon](https://www.interviewbit.com/search/?q=Amazon)

Given an unsorted integer array, find the first missing positive integer.

**Example:**

Given [1,2,0] return 3,

[3,4,-1,1] return 2,

[-8, -7, -6] returns 1

Your algorithm should run in O(n) time and use constant space.

Approach: numbers A[i]<=0 and A[i]>N ( N being the size of the array ) is not important to us since **the missing positive integer will be in the range [1, N+1]**.

The answer will be N+1 only if all of the elements of the array are exact one occurrence of [1, N].

Creating buckets would have been an easy solution if using extra space was allowed.

We traverse the array and if A[i] is in [1,N] range, we try to put in the index of same value in the array.

int Solution::firstMissingPositive(vector<int> &A) {

int n=A.size();

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

if(A[i]>0 && A[i]<=n){

int pos=A[i]-1;

if(A[pos]!=A[i]){

swap(A[pos],A[i]);

i--;

}

}

}

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

if(A[i]!=i+1) return i+1;

}

return n+1;

}

## N/3 Repeat Number

Asked in:

* [Google](https://www.interviewbit.com/search/?q=Google)
* You’re given a read only array of n integers. Find out if any integer occurs more than n/3 times in the array in linear time and constant additional space.
* If so, return the integer. If not, return -1.
* If there are multiple solutions, return any one.
* **Example :**
* Input : [1 2 3 1 1]
* Output : 1
* 1 occurs 3 times which is more than 5/3 times.

int Solution::repeatedNumber(const vector<int> &A) {

int n=A.size();

int count1=0, count2=0;

int maj1=INT\_MAX;

int maj2=INT\_MAX;

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

if(maj1==A[i])

count1++;

else if(maj2==A[i])

count2++;

else if(count1==0){

maj1=A[i]; count1=1;

}

else if(count2==0){

maj2=A[i]; count2=1;

}

else{

count1--;

count2--;

}

}

count1=count2=0;

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

if(maj1==A[i]) count1++;

if(maj2==A[i]) count2++;

}

if(count1>n/3) return maj1;

if(count2>n/3) return maj2;

return -1;

}

# Graph Data Structure & Algorithms

## Smallest sequence with given Primes

* Asked in:
* [Booking.com](https://www.interviewbit.com/search/?q=Booking.com)
* [Microsoft](https://www.interviewbit.com/search/?q=Microsoft)
* Given three prime number(p1, p2, p3) and an integer k. Find the first(smallest) k integers which have only p1, p2, p3 or a combination of them as their prime factors.
* **Example:**
* *Input :*   
  Prime numbers : [2,3,5]   
  k : 5
* If primes are given as p1=2, p2=3 and p3=5 and k is given as 5, then the sequence of first 5 integers will be:
* *Output:*   
  {2,3,4,5,6}
* *Explanation :*   
  4 = p1 \* p1 ( 2 \* 2 )  
  6 = p1 \* p2 ( 2 \* 3 )
* **Note:** The sequence should be sorted in ascending order

**Approach:** We use the fact that there are only 3 possibilities of getting to a new number, Multiply by p1 or p2 or p3.

int min(int a, int b, int c){

return min(a, min(b,c));

}

vector<int> Solution::solve(int A, int B, int C, int k) {

vector<int>res;

res.push\_back(1);

int nextA, nextB, nextC;

int iA=0, iB=0, iC=0;

int nextN;

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

nextA=res[iA]\*A;

nextB=res[iB]\*B;

nextC=res[iC]\*C;

nextN=min(nextA, nextB, nextC);

res.push\_back(nextN);

if(nextN==nextA) iA++;

if(nextN==nextB) iB++;

if(nextN==nextC) iC++;

}

res.erase(res.begin());

return res;

}

## Valid Path

* Asked in:
* [Morgan Stanley](https://www.interviewbit.com/search/?q=Morgan)
* [Amazon](https://www.interviewbit.com/search/?q=Amazon)
* [Codenation](https://www.interviewbit.com/search/?q=Codenation)

There is a rectangle with left bottom as  (0, 0) and right up as (x, y). There are N circles such that their centers are inside the rectangle.  
Radius of each circle is R. Now we need to find out if it is possible that we can move from (0, 0) to (x, y) without touching any circle.

**Note :**We can move from any cell to any of its 8 adjecent neighbours and we cannot move outside the boundary of the rectangle at any point of time.  
  
  
**Input Format**

1st argument given is an Integer x.

2nd argument given is an Integer y.

3rd argument given is an Integer N, number of circles.

4th argument given is an Integer R, radius of each circle.

5th argument given is an Array A of size N, where A[i] = x cordinate of ith circle

6th argument given is an Array B of size N, where B[i] = y cordinate of ith circle

**Output Format**

Return YES or NO depending on weather it is possible to reach cell (x,y) or not starting from (0,0).

**Constraints**

0 <= x, y, R <= 100

1 <= N <= 1000

Center of each circle would lie within the grid

**For Example**

Input:

x = 2

y = 3

N = 1

R = 1

A = [2]

B = [3]

Output:

NO

Explanation:

There is NO valid path in this case

See Expected Output

Seen this question in a real interview beforeYesNo

Check if (i,j) is a valid point for all 0<=i<=x, 0<=j<=y. By valid point we mean that none of the circle should contain it.

Now you know all the valid point in rectangle. You need to figure out if you can go from (0,0) to (x,y) through valid points. This can be done with any graph traversal algorithms like BFS/DFS.

bool inCircle(int x, int y, vector<int> &X, vector<int> &Y, int R){

int rSq=R\*R;

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

int dx=abs(x-X[i]);

int dy=abs(y-Y[i]);

if(dx\*dx+dy\*dy <= rSq)

return true;

}

return false;

}

bool inGrid(int x, int y, int m,int n){

if(x>=0 && x<=m && y>=0 && y<=n)

return true;

return false;

}

string Solution::solve(int m, int n, int N, int R, vector<int> &X, vector<int> &Y) {

bool visited[m+1][n+1];

memset(visited, false, sizeof(visited));

visited[0][0]=true;

queue<pair<int,int>> q;

q.push(make\_pair(0,0));

while(!q.empty()){

pair<int,int> point=q.front();

q.pop();

int x=point.first;

int y=point.second;

if(x==m && y==n) return "YES";

for(int i=-1; i<2; i++){

for(int j=-1; j<2; j++){

if(!inCircle(x+i,y+j, X, Y, R) && inGrid(x+i, y+j, m, n) && !visited[x+i][y+j])

{

visited[x+i][y+j]=true;

q.push(make\_pair(x+i, y+j));

}

}

}

}

return "NO";

}

## Level Order

* Asked in:
* [Facebook](https://www.interviewbit.com/search/?q=Facebook)
* [Groupon](https://www.interviewbit.com/search/?q=Groupon)
* [Goldman Sachs](https://www.interviewbit.com/search/?q=Goldman)

Given a binary tree, return the level order traversal of its nodes’ values. (ie, from left to right, level by level).

**Example :**  
Given binary tree

3

/ \

9 20

/ \

15 7

return its level order traversal as:

[

[3],

[9,20],

[15,7]

]

Also think about a version of the question where you are asked to do a level order traversal of the tree when depth of the tree is much greater than number of nodes on a level.

Sol1: Maintain a vector of size ‘depth’ of the tree. Do any kind of tree traversal keeping track of the current depth. Append the current element to vector[currentDepth]. Since we need stuff left to right, make sure left subtree is visited before the right subtree ( Any of traditional pre/post/inorder traversal should suffice ).

/\*\*

\* Definition for binary tree

\* struct TreeNode {

\* int val;

\* TreeNode \*left;

\* TreeNode \*right;

\* TreeNode(int x) : val(x), left(NULL), right(NULL) {}

\* };

\*/

void buildVector(TreeNode \*node, vector<vector<int>>&res, int depth){

if(!node) return;

if(res.size()==depth)

res.push\_back(vector<int>(0));

res[depth].push\_back(node->val);

buildVector(node->left,res,depth+1);

buildVector(node->right,res,depth+1);

}

vector<vector<int> > Solution::levelOrder(TreeNode\* root) {

vector<vector<int>> res;

buildVector(root,res,0);

return res;

}

Sol2: Typical level order

/\*\*

\* Definition for binary tree

\* struct TreeNode {

\* int val;

\* TreeNode \*left;

\* TreeNode \*right;

\* TreeNode(int x) : val(x), left(NULL), right(NULL) {}

\* };

\*/

vector<vector<int> > Solution::levelOrder(TreeNode\* root) {

vector<vector<int>> res;

vector<int> level;

if(!root) return res;

queue<TreeNode\*> q1, q2;

q1.push(root);

while(!q1.empty()){

TreeNode \*node=q1.front();

q1.pop();

level.push\_back(node->val);

if(node->left)

q2.push(node->left);

if(node->right)

q2.push(node->right);

if(q1.empty()){

res.push\_back(level);

level.clear();

q1.swap(q2);

}

}

return res;

}

## Smallest Multiple With 0 and 1

* Asked in:
* [Amazon](https://www.interviewbit.com/search/?q=Amazon)

You are given an integer N. You have to find smallest multiple of N which consists of digits 0 and 1 only. Since this multiple could be large, return it in form of a string.

**Note**:

* Returned string should not contain leading zeroes.

For example,

For N = 55, 110 is smallest multiple consisting of digits 0 and 1.

For N = 2, 10 is the answer.

**Approach:** Maintain graph for remainders 0 to n-1.

Each node has i) digit 0 or 1 which is appended to previous binary number string,

ii) boolean visited saying remainder is already visited or not

(When visited[rem]=1, smallest bin number with reminder=rem is already processed

iii)parent maintains parent node having previous remainder value before adding 0 or 1 to number string

create 3 vectors, visited, parent and digit. Create queue for storing and processing remainders, start with digit 1 as root of the graph, and add 1%n to queue

Sol:

string Solution::multiple(int n) {

//create

vector<bool> visited(n,0);

vector<int> parent(n);

vector<bool> digit(n);

queue<int> q;//for bfs

int rem=1%n;//first binary number considered is 1

//and its remainder is 1%n, which is the root of graph

digit[rem]=1;

parent[rem]=-1;//root has no parent

visited[rem]=1;

q.push(rem);

while(!q.empty()){

int curRem=q.front();

q.pop();

if(curRem==0) //remainder is 0, hence smallest bin multiple is found

break;

int mod0=(curRem\*10)%n; //append 0 to curRem

int mod1=(curRem\*10+1)%n; //append 1 to curRem

if(!visited[mod0]){

visited[mod0]=1;

digit[mod0]=0;

parent[mod0]=curRem;

q.push(mod0);

}

if(!visited[mod1]){

visited[mod1]=1;

digit[mod1]=1;

parent[mod1]=curRem;

q.push(mod1);

}

}

rem=0;

string res="";

while(rem!=-1){

res+=(digit[rem]+'0');//adding '0' for char conversion

rem=parent[rem];

}

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

return res;

}

## Commutable Islands

Asked in:

* [Amazon](https://www.interviewbit.com/search/?q=Amazon)

There are n islands and there are many bridges connecting them. Each bridge has some cost attached to it.

We need to find bridges with minimal cost such that all islands are connected.

It is guaranteed that input data will contain at least one possible scenario in which all islands are connected with each other.

**Example :**  
**Input**

Number of islands ( n ) = 4

1 2 1

2 3 4

1 4 3

4 3 2

1 3 10

In this example, we have number of islands(n) = 4 . Each row then represents a bridge configuration. In each row first two numbers represent the islands number which are connected by this bridge and the third integer is the cost associated with this bridge.

In the above example, we can select bridges 1(connecting islands 1 and 2 with cost 1), 3(connecting islands 1 and 4 with cost 3), 4(connecting islands 4 and 3 with cost 2). Thus we will have all islands connected with the minimum possible cost(1+3+2 = 6).   
In any other case, cost incurred will be more.

Below are the steps for finding MST using Kruskal’s algorithm

***1.****Sort all the edges in non-decreasing order of their weight.****2.****Pick the smallest edge. Check if it forms a cycle with the spanning tree formed so far. If cycle is not formed, include this edge. Else, discard it.****3.****Repeat step#2 until there are (V-1) edges in the spanning tree.*

/\*

Below are the steps for finding MST using Kruskal’s algorithm

1. Sort all the edges in non-decreasing order of their weight.

2. Pick the smallest edge. Check if it forms a cycle with the spanning tree formed so far.

If cycle is not formed, include this edge. Else, discard it.

3. Repeat step#2 until there are (V-1) edges in the spanning tree.

The step#2 uses Union-Find (Union By Rank and Path Compression) algorithm to detect cycle.

Start with total disjoint sets equal to no. of vertices

for each min weight edge, check if its endpoints are connected, if not merge their disjoint sets

Maintain RootParent node that represents disjoint set.

Whenever disjoint sets are merged, update RootParent of added nodes

\*/

//compare function to sort vector based on weights of edges.

//here 3rd element is weight of edge

struct myclass{

bool operator()(const vector<int> &a, const vector<int> &b){

return a[2]<b[2];

}

}myobj;

//class for disjoint\_set having fun create\_set, find\_set and merge\_set

class DisjointSet{

private:

int NoC;//no of disjoint components

vector<int> parent; //root parent of each component

vector<int> rank; //rank of each component's root parent

public:

//constructor, parent and rank has 1 element extra as node numbering start from 1

DisjointSet(int numOfComp):NoC(numOfComp), parent(numOfComp+1,-1), rank(numOfComp+1,1)

{

resetParents();

}

//initially every node has separate disjoint set, hence RootParent of each node is same node itself

void resetParents()

{

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

parent[i]=i;

}

//find RootParent which has its own value as parent

int getRootParent(int a)

{

int p=parent[a];

while(p!=parent[p])

p=parent[p];

return p;

}

//get RootParents to decide nodes connected or not

bool connected(int a, int b)

{

int rootA=getRootParent(a);

int rootB=getRootParent(b);

return (rootA==rootB);

}

//get no of disjoint sets

int getNoC()

{

return NoC;

}

//merge disjoint set

void merge(int a, int b)

{

int rootA=getRootParent(a);

int rootB=getRootParent(b);

int tmp;

if(rootA==rootB) return;

if(rank[rootA]>rank[rootB]){

parent[rootB]=rootA;

rank[rootA]+=rank[rootB];

tmp=b;//update root parent of b

}

else{

parent[rootA]=rootB;

rank[rootB]+=rank[rootA];

tmp=a;//updateroot parent of a

}

NoC--; //2 sets merged, decrease NoC by 1

//not needed below line

//parent[tmp]=parent[parent[tmp]];//update RootParent of newly added node

}

};

int Solution::solve(int N, vector<vector<int> > &V) {

//make\_set

DisjointSet DS(N);

sort(V.begin(), V.end(), myobj);

int sum=0;

int i=0;

while(DS.getNoC()!=1){

vector<int> e=V[i];

if(!DS.connected(e[0],e[1])){ //find\_set

DS.merge(e[0], e[1]);//merge\_set

sum+=e[2];

}

i++;

}

return sum;

}

## Possibility of finishing all courses given pre-requisites

Asked in:

[Amazon](https://www.interviewbit.com/search/?q=Amazon)

*There are a total of****N****courses you have to take, labeled from****1****to****N****. Some courses may have prerequisites, for example to take course****2****you have to first take course****1****, which is expressed as a pair:****[1,2]****.*

Given the total number of courses and a list of prerequisite pairs, is it possible for you to finish all courses. return **1/0** if it is **possible/not possible**.  
The list of prerequisite pair are given in two integer arrays **B** and **C** where **B[i]** is a prerequisite for **C[i]**.

***Example:***

*If****N = 3****and the prerequisite pairs are****[1,2]****and****[2,3]****, then you can finish courses in the following order:****1****,****2****and****3****.*

*But if****N = 2****and the prerequisite pairs are****[1,2]****and****[2,1]****, then it is not possible for you to finish all the courses.*

This is detecting cycle in DAG

#include<list>

class Graph{

private:

int n;

vector<list<int>> edges;

public:

Graph(int n):n(n),edges(n+1){

}

void addEdge(int u, int v){

edges[u].push\_back(v);

}

bool isCyclic(){

vector<bool> visited(n+1,0), recStack(n+1,0);

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

if(isCyclicUtil(i, visited, recStack)) return 1;

return false;

}

bool isCyclicUtil(int v, vector<bool> &visited, vector<bool> &recStack){

if(!visited[v]){

visited[v]=1;

recStack[v]=1;

for(list<int>::iterator i=edges[v].begin(); i!=edges[v].end(); ++i){

if(!visited[\*i]&&isCyclicUtil(\*i,visited,recStack))

return true;

else if (recStack[\*i]) return true;

}

}

recStack[v]=0;

return false;

}

};

int Solution::solve(int N, vector<int> &u, vector<int> &v) {

Graph g(N);

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

g.addEdge(u[i],v[i]);

}

return !g.isCyclic();

}

## Black Shapes

* Asked in:
* [Amazon](https://www.interviewbit.com/search/?q=Amazon)

Given N \* M field of O's and X's, where O=white, X=black  
Return the number of black shapes. A black shape consists of one or more adjacent X's (diagonals not included)

**Example:**

OOOXOOO

OOXXOXO

OXOOOXO

answer is 3 shapes are :

(i) X

X X

(ii)

X

(iii)

X

X

Note that we are looking for connected shapes here.

For example,

XXX

XXX

XXX

is just one single connected black shape.

Simple graph traversal approach:

Answer := 0

Loop i = 1 to N :

Loop j = 1 to M:

IF MATRIX at i, j equal to 'X' and not visited:

BFS/DFS to mark the connected area as visited

update Answer

EndLoop

EndLoop

return Answer

vector<int>row={-1,1,0,0};

vector<int>col={0,0,-1,1};

bool safe(int x, int y, int m, int n){

if(x>=0 && x<m && y>=0 && y<n) return true;

return false;

}

void DFSUtil(int i, int j, int m, int n, vector<string> &s){

s[i][j]='O';

for(int k=0; k<4; k++){

int x=i+row[k];

int y=j+col[k];

if(safe(x,y,m,n)&&s[x][y]=='X')

DFSUtil(x,y,m,n,s);

}

}

int Solution::black(vector<string> &s) {

int m=s.size();

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

int count=0;

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

for(int j=0; j<n; j++){

if(s[i][j]=='X'){

count++;

DFSUtil(i,j,m,n,s);

}

}

}

return count;

}

## Capture Regions on Board

* Asked in:
* [Google](https://www.interviewbit.com/search/?q=Google)

Given a 2D board containing 'X' and 'O', capture all regions surrounded by 'X'.

A region is captured by flipping all 'O's into 'X's in that surrounded region.

For example,

X X X X

X O O X

X X O X

X O X X

After running your function, the board should be:

X X X X

X X X X

X X X X

X O X X

**Approach:**

**1)** Traverse the given matrix and replace all ‘O’ with a special character ‘-‘.

**2)** Traverse four edges of given matrix and call [floodFill(‘-‘, ‘O’)](https://www.geeksforgeeks.org/flood-fill-algorithm-implement-fill-paint/) for every ‘-‘ on edges. The remaining ‘-‘ are the characters that indicate ‘O’s (in the original matrix) to be replaced by ‘X’.

**3)**Traverse the matrix and replace all ‘-‘s with ‘X’s.

Floodfill

// A recursive function to replace previous color 'prevC' at '(x, y)'

// and all surrounding pixels of (x, y) with new color 'newC' and

**floodFil(screen[M][N], x, y, prevC, newC)**

1) If x or y is outside the screen, then return.

2) If color of screen[x][y] is not same as prevC, then return

3) Recur for north, south, east and west.

floodFillUtil(screen, x+1, y, prevC, newC);

floodFillUtil(screen, x-1, y, prevC, newC);

floodFillUtil(screen, x, y+1, prevC, newC);

floodFillUtil(screen, x, y-1, prevC, newC);

vector<int>row={0,0,-1,1};

vector<int>col={-1,1,0,0};

bool safe(int i, int j, int m, int n){

if(i>=0 && i<m && j>=0 && j<n) return true;

return false;

}

void floodfill(int i, int j, int m, int n, vector<vector<char>>&A, char oldChar, char newChar)

{

if(!safe(i,j,m,n)) return;

if(A[i][j]!=oldChar) return;

A[i][j]=newChar;

for(int k=0; k<4; k++){

int x=i+row[k];

int y=j+col[k];

floodfill(x,y,m,n,A,'-','O');

//printVector(A);

}

}

void Solution::solve(vector<vector<char> > &A) {

int m=A.size();

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

if(m<3 || n<3) return;

//replace all '0's with '-'

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

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

if(A[i][j]=='O')

A[i][j]='-';

}

//printVector(A);

int row, col;

row=0;

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

if(A[row][j]=='-')

floodfill(row,j,m,n,A,'-','O');

//printVector(A);

col=n-1;

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

if(A[i][col]=='-')

floodfill(i,col,m,n,A,'-','O');

row=m-1;

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

if(A[row][j]=='-')

floodfill(row,j,m,n,A,'-','O');

col=0;

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

if(A[i][col]=='-')

floodfill(i,col,m,n,A,'-','O');

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

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

if(A[i][j]=='-')

A[i][j]='X';

}

## Largest Distance between nodes of a Tree

* Asked in:
* [Facebook](https://www.interviewbit.com/search/?q=Facebook)
* [Google](https://www.interviewbit.com/search/?q=Google)

**Find largest distance**  
Given an arbitrary unweighted rooted tree which consists of N (2 <= N <= 40000) nodes. The goal of the problem is to find largest distance between two nodes in a tree. Distance between two nodes is a number of edges on a path between the nodes (there will be a unique path between any pair of nodes since it is a tree). The nodes will be numbered 0 through N - 1.

The tree is given as an array P, there is an edge between nodes P[i] and i (0 <= i < N). Exactly one of the i’s will have P[i] equal to -1, it will be root node.

***Example:*** *If given P is [-1, 0, 0, 0, 3], then node 0 is the root and the whole tree looks like this:*

0

/ | \

1 2 3

\

4

*One of the longest path is 1 -> 0 -> 3 -> 4 and its length is 3, thus the answer is 3. Note that there are other paths with maximal distance.*

We can find longest path using two BFSs. First BFS to find an end point of the longest path and second BFS from this end point to find the actual longest path.

#include<list>

class Graph{

private:

int V;

vector<list<int>>edges;

public:

Graph(int n):V(n),edges(n){}

void addEdge(int u, int v){

edges[u].push\_back(v);

}

pair<int,int> dfs(int v){

vector<int>dist(V,-1);

dist[v]=0;

queue<int>q;

q.push(v);

list<int>::iterator it;

while(!q.empty()){

int u=q.front();

q.pop();

for(it=edges[u].begin();it!=edges[u].end();++it){

if(dist[\*it]==-1){

dist[\*it]=dist[u]+1;

q.push(\*it);

}

}

}

int mIndex=v, mDist=0;

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

if(mDist<dist[i]){

mIndex=i;

mDist=dist[i];

}

}

return make\_pair(mIndex,mDist);

}

};

int Solution::solve(vector<int> &A) {

int n=A.size();

Graph g(n);

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

if(A[i]!=-1){

g.addEdge(A[i],i);

g.addEdge(i, A[i]);

}

}

pair<int,int> end1, end2;

end1=g.dfs(0);

end2=g.dfs(end1.first);

return end2.second;

}

## Word Search Board

* Asked in:
* [Epic systems](https://www.interviewbit.com/search/?q=Epic)
* [Amazon](https://www.interviewbit.com/search/?q=Amazon)

Given a 2D board and a word, find if the word exists in the grid.

The word can be constructed from letters of sequentially adjacent cell, where "adjacent" cells are those horizontally or vertically neighboring. The cell itself does not count as an adjacent cell.   
The same letter cell may be used more than once.

**Example :**

Given board =

[

["ABCE"],

["SFCS"],

["ADEE"]

]

word = "ABCCED", -> returns 1,

word = "SEE", -> returns 1,

* word = "ABCB", -> returns 1,

word = "ABFSAB" -> returns 1

word = "ABCD" -> returns 0

Note that 1 corresponds to true, and 0 corresponds to false.

vector<int> row={0,0,-1,1};

vector<int> col={1,-1,0,0};

vector<string> Board;

string word;

bool found;

bool safe(int i, int j){

if(i>=0 && i<Board.size() && j>=0 && j<Board[0].size())

return true;

return false;

}

void solve(int pos, int i, int j){

if(found) return;

if(pos==word.size()){

found=true;

return;

}

for(int k=0; k<4; k++){

int i1=row[k]+i;

int j1=col[k]+j;

if(!safe(i1,j1)) continue;

if(word[pos]==Board[i1][j1])

solve(pos+1,i1,j1);

}

}

int Solution::exist(vector<string> &A, string B) {

Board=A;

word=B;

int m=A.size();

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

found=false;

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

for(int j=0; j<n; j++){

if(A[i][j]==B[0]) solve(1,i,j);

}

}

return found;

}

## Convert Sorted List to Binary Search Tree

* Asked in:
* [Google](https://www.interviewbit.com/search/?q=Google)
* Given a singly linked list where elements are sorted in ascending order, convert it to a height balanced BST.
* ***A height balanced BST :****a height-balanced binary tree is defined as a binary tree in which the depth of the two subtrees of every node never differ by more than 1.*
* **Example :**
* Given A : 1 -> 2 -> 3
* A height balanced BST :
* 2
* / \
* 1 3

n this method, we construct from leaves to root. The idea is to insert nodes in BST in the same order as they appear in Linked List so that the tree can be constructed in O(n) time complexity. We first count the number of nodes in the given Linked List. Let the count be n. After counting nodes, we take left n/2 nodes and recursively construct the left subtree. After left subtree is constructed, we allocate memory for root and link the left subtree with root. Finally, we recursively construct the right subtree and link it with root.  
While constructing the BST, we also keep moving the list head pointer to next so that we have the appropriate pointer in each recursive call.

/\*\*

\* Definition for binary tree

\* struct TreeNode {

\* int val;

\* TreeNode \*left;

\* TreeNode \*right;

\* TreeNode(int x) : val(x), left(NULL), right(NULL) {}

\* };

\*/

/\*\*

\* Definition for singly-linked list.

\* struct ListNode {

\* int val;

\* ListNode \*next;

\* ListNode(int x) : val(x), next(NULL) {}

\* };

\*/

int count(ListNode \*A){

int c=0;

while(A){

c++;

A=A->next;

}

return c;

}

//it is necessary to provide address of pointer, since we want the main pointer gets //updated in each recursive call

TreeNode \*sortedListToBSTUtil(ListNode \* &A, int n){

if(n<=0) return NULL;

TreeNode \*left=sortedListToBSTUtil(A,n/2);

TreeNode \*root=new TreeNode(A->val);

root->left=left;

A=A->next;

root->right=sortedListToBSTUtil(A,n-n/2-1);

return root;

}

TreeNode\* Solution::sortedListToBST(ListNode\* A) {

int n=count(A);

return sortedListToBSTUtil(A,n);

}

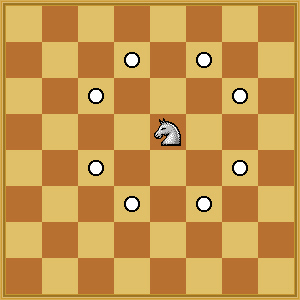
## Knight On Chess Board

Asked in:

* [Goldman Sachs](https://www.interviewbit.com/search/?q=Goldman)
* [Amazon](https://www.interviewbit.com/search/?q=Amazon)

**Knight movement on a chess board**

Given any source point and destination point on a chess board, we need to find whether Knight can move to the destination or not.



The above figure details the movements for a knight ( 8 possibilities ). Note that a knight cannot go out of the board.

If yes, then what would be the minimum number of steps for the knight to move to the said point.  
If knight can not move from the source point to the destination point, then return -1

**Input:**

N, M, x1, y1, x2, y2

where N and M are size of chess board

x1, y1 coordinates of source point

x2, y2 coordinates of destination point

**Output:**

return Minimum moves or -1

**Example**

Input : 8 8 1 1 8 8

Output : 6

**Approach:**

This problem can be seen as shortest path in unweighted graph. Therefore we use [BFS](https://www.geeksforgeeks.org/breadth-first-traversal-for-a-graph/) to solve this problem. We try all 8 possible positions where a Knight can reach from its position. If reachable position is not already visited and is inside the board, we push this state into queue with distance 1 more than its parent state. Finally we return distance of target position, when it gets pop out from queue.

struct cell{

int x,y,dist;

cell(){}

cell(int x, int y, int dist):x(x),y(y),dist(dist){}

};

vector<int> dx={-2, -1, 1, 2, -2, -1, 1, 2};

vector<int> dy={-1, -2, -2, -1, 1, 2, 2, 1};

bool safe(int x, int y, int m, int n){

if(x>0 && x<=m && y>0 && y<=n) return true;

return false;

}

int Solution::knight(int m, int n, int x1, int y1, int x2, int y2) {

vector<vector<bool>> visited(m+1,vector<bool>(n+1,false));

cell c(x1,y1,0);

visited[x1][y1]=true;

queue<cell>q;

q.push(c);

while(!q.empty()){

cell c1=q.front();

q.pop();

if(c1.x==x2 && c1.y==y2){

return c1.dist;

}

for(int k=0; k<8; k++){

int xi=dx[k]+c1.x;

int yi=dy[k]+c1.y;

if(safe(xi,yi,m,n) && !visited[xi][yi]){

visited[xi][yi]=true;

q.push(cell(xi,yi,c1.dist+1));

}

}

}

return -1;

}

## Clone Graph

* Asked in:
* [Google](https://www.interviewbit.com/search/?q=Google)
* [Facebook](https://www.interviewbit.com/search/?q=Facebook)
* [Amazon](https://www.interviewbit.com/search/?q=Amazon)

Clone an undirected graph. Each node in the graph contains a label and a list of its neighbors.

/\*\*

\* Definition for undirected graph.

\* struct UndirectedGraphNode {

\* int label;

\* vector<UndirectedGraphNode \*> neighbors;

\* UndirectedGraphNode(int x) : label(x) {};

\* };

\*/

UndirectedGraphNode \*Solution::cloneGraph(UndirectedGraphNode \*node) {

if(!node) return NULL;

UndirectedGraphNode\* clone=new UndirectedGraphNode(node->label);

map<UndirectedGraphNode\*,UndirectedGraphNode\*> m;

queue<UndirectedGraphNode\*> q;

m[node]=clone;

q.push(node);

while(!q.empty()){

UndirectedGraphNode\* u=q.front();

q.pop();

vector<UndirectedGraphNode\*> v=u->neighbors;

int n=v.size();

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

if(m[v[i]]==NULL){

UndirectedGraphNode\* clone1=new UndirectedGraphNode(v[i]->label);

m[v[i]]=clone1;

q.push(v[i]);

}

m[u]->neighbors.push\_back(m[v[i]]);

}

}

return clone;

}

## Word Ladder I

* Asked in:
* [Google](https://www.interviewbit.com/search/?q=Google)
* [Microsoft](https://www.interviewbit.com/search/?q=Microsoft)
* [Ebay](https://www.interviewbit.com/search/?q=Ebay)

Given two words (start and end), and a dictionary, find the length of shortest transformation sequence from start to end, such that:

* You must change exactly one character in every transformation
* Each intermediate word must exist in the dictionary

**Example :**

Given:

start = "hit"

end = "cog"

dict = ["hot","dot","dog","lot","log"]

As one shortest transformation is "hit" -> "hot" -> "dot" -> "dog" -> "cog",  
return its length 5.

Note that we account for the length of the transformation path instead of the number of transformation itself.

***Note:***

* Return 0 if there is no such transformation sequence.
* All words have the same length.
* All words contain only lowercase alphabetic characters.

The idea is to use [BFS](https://www.geeksforgeeks.org/breadth-first-traversal-for-a-graph/). We start from the given start word, traverse all words that adjacent (differ by one character) to it and keep doing so until we find the target word or we have traversed all words.

struct Word{

string text;

int len;

Word(string text,int len):text(text),len(len){}

};

bool neighbors(string s1, string s2){

int n=s1.size();

if(n != s2.size()) return 0;

int diff=0;

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

if(s1[i]!=s2[i]) diff++;

}

if(diff==1) return 1;

return 0;

}

int Solution::ladderLength(string start, string end, vector<string> &dictV) {

Word w(start,1);

queue<Word>q;

q.push(w);

vector<string>::iterator it, it1;

while(!q.empty()){

Word front=q.front();

q.pop();

if(front.text==end) return 1;

for(it=dictV.begin(); it!=dictV.end(); ++it){

if(neighbors(front.text, \*it)){

if(\*it==end) return front.len+1;

q.push(Word(\*it,front.len+1));

it1=it;

dictV.erase(it);

it=it1;

}

}

}

return 0;

}

# Backtracking

## Pruned builder

### Palindrome Partitioning

### Generate all Parentheses II

### Sudoku

* Asked in:
* [Microsoft](https://www.interviewbit.com/search/?q=Microsoft)
* [Qualcomm](https://www.interviewbit.com/search/?q=Qualcomm)

Write a program to solve a Sudoku puzzle by filling the empty cells.  
Empty cells are indicated by the character '.'  
You may assume that there will be only one unique solution.



A sudoku puzzle,



and its solution numbers marked in red.

**Example :**

For the above given diagrams, the corresponding input to your program will be

[[53..7....], [6..195...], [.98....6.], [8...6...3], [4..8.3..1], [7...2...6], [.6....28.], [...419..5], [....8..79]]

and we would expect your program to modify the above array of array of characters to

[[534678912], [672195348], [198342567], [859761423], [426853791], [713924856], [961537284], [287419635], [345286179]]

#define BLANK '.'

bool findUnassigned(vector<vector<char>> &A, int &row, int &col){

for(int i=0; i<9; i++){

for(int j=0; j<9; j++){

if(A[i][j]==BLANK){

row=i;

col=j;

return true;

}

}

}

return false;

}

bool inRow(vector<vector<char>> &A, int row, int num){

for(int i=0; i<9; i++)

if(A[row][i]==(num+'0'))

return true;

return false;

}

bool inCol(vector<vector<char>> &A, int col, int num){

for(int i=0; i<9; i++)

if(A[i][col]==(num+'0'))

return true;

return false;

}

bool inGrid(vector<vector<char>> &A, int rowStart, int colStart, int num){

for(int i=0; i<3; i++){

for(int j=0; j<3; j++){

if(A[rowStart+i][colStart+j]==(num+'0'))

return true;

}

}

return false;

}

bool safe(vector<vector<char>> &A, int row, int col, int num){

if(!inRow(A,row,num) && !inCol(A,col,num) && !inGrid(A,row-row%3, col-col%3,num) && A[row][col]==BLANK)

return true;

return false;

}

void print(vector<vector<char>> &A){

for(int i=0; i<9; i++){

for(int j=0; j<9; j++)

cout<<A[i][j];

cout<<endl;

}

}

bool solveSudokuUtil(vector<vector<char>>&A){

int row, col;

if(!findUnassigned(A, row, col))

return true;

for(int num=1; num<=9; num++){

if(safe(A, row, col, num)){

A[row][col]=num+'0';

if(solveSudokuUtil(A)) return true;

A[row][col]=BLANK;

}

}

return false;

}

void Solution::solveSudoku(vector<vector<char> > &A) {

solveSudokuUtil(A);

}

### NQueens

Asked in:

* [Qualcomm](https://www.interviewbit.com/search/?q=Qualcomm)
* [Google](https://www.interviewbit.com/search/?q=Google)
* [Amazon](https://www.interviewbit.com/search/?q=Amazon)

The n-queens puzzle is the problem of placing n queens on an n×n chessboard such that no two queens attack each other.



Given an integer n, return all distinct solutions to the n-queens puzzle.

Each solution contains a distinct board configuration of the n-queens’ placement, where 'Q' and '.' both indicate a queen and an empty space respectively.

For example,  
There exist two distinct solutions to the 4-queens puzzle:

[

[".Q..", // Solution 1

"...Q",

"Q...",

"..Q."],

["..Q.", // Solution 2

"Q...",

"...Q",

".Q.."]

]

#### Approach: Start placing each queen column wise and try attempting in every row.

1. tor<vector<string>> res;
3. bool safe(vector<string> &tmp, int row, int col, int n){
4. int i,j;
5. for(i=0; i<col; i++) // check prev cols on same row having queens or not
6. if(tmp[row][i]=='Q')
7. return false;
8. for(i=row, j=col; i>=0 && j>=0; i--, j--) //check left upper diagonal
9. if(tmp[i][j]=='Q')
10. return false;
11. for(i=row, j=col; i<n && j>=0; i++, j--) //check left lower diagonal
12. if(tmp[i][j]=='Q')
13. return false;
14. return true;
15. }
17. void solveNQueensUtil(vector<string> &tmp, int col, int n){
18. if(col >= n){
19. res.push\_back(tmp);
20. return;
21. }
22. for(int i=0; i<n; i++){
23. if(safe(tmp, i, col, n)){
24. tmp[i][col]='Q';
25. solveNQueensUtil(tmp,col+1,n);
26. tmp[i][col]='.';
27. }
28. }
29. }
30. vector<vector<string>> Solution::solveNQueens(int n) {
31. res.clear();
32. string str;
33. for(int i=0; i<n; i++)
34. str.push\_back('.');
35. vector<string> tmp(n,str);
36. solveNQueensUtil(tmp, 0, n);
37. return res;
38. }