1.Maximum Subarray Sum – Kadane‟s Algorithm

Kadane's Algorithm

Difficulty: **Medium** Accuracy: **36.28%** Submissions: **1M** Points: **4**

Given an integer array **arr[].** You need to find and return the **maximum** sum possible from all the subarrays.

Examples:

Input: arr[] = [2, 3, -8, 7, -1, 2, 3]
Output: 11
Explanation: The subarray {7, -1, 2, 3} has the largest sum 11.

Input: arr[] = [-2, -4]
Output: -2
Explanation: The subarray {-2} has the largest sum -2.

Input: arr[] = [5, 4, 1, 7, 8]
Output: 25
Explanation: The subarray {5, 4, 1, 7, 8} has the largest sum 25.

Constraints:

$1 \leq \text{arr.size()} \leq 10^5$
$-10^9 \leq \text{arr[i]} \leq 10^4$

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

int main(){
    int tc;
    cin>>tc;
    while(tc--){
        int n;

        cin>>n;
        vector<int>arr(n);
        for(int i=0;i<n;i++){
            cin>>arr[i];
        }
        int sum =0,maxi=arr[0];
        for(int i=0;i<n;i++){
            sum+=arr[i];
            maxi = max(sum,maxi);
            if(sum<0){
                sum=0;
            }
        }
        cout<<maxi<<endl;
    }

}
```

```
~/Desktop/practice (3.475s)
g++ kadenas.cpp

~/Desktop/practice (0.045s)
./a.out <testcase1.txt
11
-2
25
```

Time complexity: O(n)

Space Complexity: O(1)

2. Maximum Product Subarray

Maximum Product Subarray

Difficulty: **Medium** Accuracy: **18.09%** Submissions: **383K+** Points: **4**

Given an array **arr[]** that contains positive and negative integers (may contain 0 as well). Find the **maximum** product that we can get in a subarray of **arr**.

Note: It is guaranteed that the output fits in a 32-bit integer.

Examples

Input: arr[] = [-2, 6, -3, -10, 0, 2]
Output: 180
Explanation: The subarray with maximum product is {6, -3, -10} with product = 6 * (-3) * (-10) = 180.

Input: arr[] = [-1, -3, -10, 0, 60]
Output: 60
Explanation: The subarray with maximum product is {60}.

Input: arr[] = [2, 3, 4]
Output: 24
Explanation: For an array with all positive elements, the result is product of all elements.

as

```cpp
#include <bits/stdc++.h>
using namespace std;
int main(){
    int tc;
    cin>>tc;
    while(tc--){
        int n;
        cin>>n;
        vector<int>arr(n);

        for(int i=0;i<n;i++){
            cin>>arr[i];
        }
        int prod =1,maxi=INT_MIN;

        int cycle1=1,cycle2=1;

        for(int i=0;i<n;i++){
            if(cycle1 == 0) cycle1=1;
            if (cycle2 == 0) cycle2 =1;
            cycle1*=arr[i];
            cycle2*=arr[n-i-1];

            maxi =max(maxi,max(cycle1,cycle2));
        }
        cout<<maxi<<endl;
    }
}
```

```
~/Desktop/practice (3.472s)
g++ MaxSubarrayProd.cpp

~/Desktop/practice (0.044s)
./a.out <testcase2.txt
180
60
```

Time complexity: O(N)

Space Complexity:O(1)

3. Search in a sorted and rotated Array

Search in Rotated Sorted Array

Difficulty: **Medium** Accuracy: **37.64%** Submissions: **231K+** Points: **4**

Given a sorted (in ascending order) and rotated array **arr** of distinct elements which may be rotated at some point and given an element **key,** the task is to find the index of the given element **key** in the array **arr**. The whole array **arr** is given as the range to search.

> Rotation shifts elements of the array by a certain number of positions, with elements that fall off one end reappearing at the other end.

Note:- 0-based indexing is followed & returns **-1** if the key is not present.

as

```cpp
#include <bits/stdc++.h>
using namespace std;
int main()
{
    int tc;cin>>tc;
    while(tc--){
        int n;
        cin>>n;
        vector<int>arr(n);
        for(int i=0;i<n;i++){cin>>arr[i];}
        int key;
        cin>>key;
        int low =0,high = n-1;
        int ans=0;
        while(low<=high){
            int mid = (low+high)/2;
            if(arr[mid] == key){
                ans = mid;
                break;
            }
            if(arr[low]<=arr[mid]){
                if(arr[low]<=key and key<=arr[mid]){ high = mid-1;}
                else{low = mid+1;}
            }else if(arr[mid]<=arr[high]){
                if(arr[mid]<=key and key<=arr[high]){
                    low=mid+1;

                }
                else{
                    high=mid-1;
                }

            }
        }
        cout<<ans<<endl;
        }
}
```

```
~/Desktop/practice (3.461s)
g++ sortandrotated.cpp

~/Desktop/practice (0.045s)
./a.out <testcase3.txt
4
0
1
```

Time complexity: O(logN)

Space complexity: O(1)

4. Container with Most Water

Given non-negative integers arr1,arr2,....arrn where each represents a point at coordinate (i, arri). For each i vertical lines are drawn such that the two endpoints of line **i** is at (i, arri) and (i,0). Find two lines, which together with x-axis form a container, such that it contains the most water.

Note: In the case of a single verticle line it will not be able to hold water.

TEST CASES:

Input: arr = [1, 5, 4, 3]

Output: 6

Explanation: 5 and 3 are distance 2 apart. So the size of the base = 2. Height of container = min(5, 3) = 3. So total area = 3 * 2 = 6

Input: arr = [3, 1, 2, 4, 5]

Output: 12

Explanation: 5 and 3 are distance 4 apart. So the size of the base = 4. Height of container = min(5, 3) = 3. So total area = 4 * 3 = 12

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

int main()
{
    int tc;
    cin>>tc;
    while(tc--){
        int n;
        cin>>n;
        vector<int>height(n);
        for(int i=0;i<n;i++){
            cin>>height[i];
        }
        int left =0,right = n-1;
        int result =0;
        while(left<=right){
            int area = (right-left)* min(height[left],height[right]);
            result = max(area,result);
            if(height[left]>height[right]){
                right--;
            }
            else{
                left++;
            }
        }
        cout<<result<<endl;
  }


}
```

```
~/Desktop/practice (3.459s)
g++ container_with_most_water.cpp

~/Desktop/practice (0.045s)
./a.out <testcase4.txt
6
12
```

Time complexity:O(N)

Space Complexity:O(1)

5. Find the Factorial of a large number

```java
import java.util.*;
import java.math.BigInteger;
class Factorial {
    public static BigInteger fact(int n) {
        if (n == 0) return BigInteger.ONE;
        return BigInteger.valueOf(n).multiply(fact(n - 1));
    }
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int t = scanner.nextInt();
        for (int i = 0; i < t; i++) {
            int n = scanner.nextInt();
            System.out.println(fact(n));
        }
        scanner.close();
    }
}
```

OUTPUT:

```
~/Desktop/practice (1.157s)
javac Factorial.java

~/Desktop/practice (0.184s)
java Factorial <testcase5.txt
933262154439441526816992388562667004907159682643816214685929638952175999932299156089414639761565182862536979208272237582511852109168640000000000000000000000000
304140932017133780436126081660647688443776415689605120000000000
```

Time Complexity : O(n)

Space Complexity: O(1)

6. Trapping Rainwater Problem

Given an array with non-negative integers representing the height of blocks. If the width of each block is 1, compute how much water can be trapped between the blocks during the rainy season.

TEST CASES:

Input: arr[] = {3, 0, 1, 0, 4, 0, 2}

Output: 10

Explanation: The expected rainwater to be trapped is shown in the above image.

Input: arr[] = {3, 0, 2, 0, 4}

Output: 7

Explanation: We trap 0 + 3 + 1 + 3 + 0 = 7 units.

Input: arr[] = {1, 2, 3, 4}

Output: 0

Explanation : We cannot trap water as there is no height bound on both sides

```cpp
#include <bits/stdc++.h>
using namespace std;
int main()
{
    int tc;
    cin>>tc;
    while(tc--){
        int n;
        cin>>n;
        vector<int>height(n);
        for(int i=0;i<n;i++){
            cin>>height[i];
        }
        int result =0,cnt =0,left =0,right = n-1;

        int leftmax = height[left] , rightmax = height[right];
        while(left<right){
            if (leftmax<rightmax){
                left++;
                leftmax = max(leftmax,height[left]);
                result +=leftmax-height[left];
            }else{
                right--;
                rightmax = max(rightmax,height[right]);
                result +=rightmax-height[right];
            }
        }
        cout<<result<<endl;
  }
 }
```

```
~/Desktop/practice (3.46s)
g++ TrappingRainWater.cpp

~/Desktop/practice (0.044s)
./a.out <testcase6.txt
10
7
0
5
```

TIME COMPLEXITY: O(N)

SPACE COMPLEXITY: O(1)

7. Chocolate Distribution Problem

```java
import java.util.*;
class ChocolateDistribution {

    public static int findMinDiff(int[] arr, int m) {
        int n = arr.length;
        if (n < m) {
            return -1; // Not enough chocolates to distribute
        }
        Arrays.sort(arr);
        int minDiff = Integer.MAX_VALUE;

        // Iterate to find the minimum difference
        for (int i = 0; i + m - 1 < n; i++) {
            int diff = arr[i + m - 1] - arr[i];
            minDiff = Math.min(minDiff, diff);
        }

        return minDiff;
    }
```

TEST CASES:

Input: arr[] = {7, 3, 2, 4, 9, 12, 56}, m = 3

Output: 2

Explanation: If we distribute chocolate packets {3, 2, 4}, we will get the minimum difference, that is 2.

Input: arr[] = {7, 3, 2, 4, 9, 12, 56}, m = 5

Output: 7

Explanation: If we distribute chocolate packets {3, 2, 4, 9, 7}, we will get the minimum difference, that is 9 – 2 = 7.

```
~/Desktop/practice (1.24s)
javac ChocoloteDistribution.java

~/Desktop/practice (0.226s)
java ChocolateDistribution < testcase7.txt
The minimum difference is: 2
The minimum difference is: 7
```

TIME COMPLEXITY:O(N)

SPACE COMPLEXITY:O(1)

8. Merge Overlapping Intervals

Overlapping Intervals

Difficulty: **Medium** Accuracy: **57.41%** Submissions: **67K+** Points: **4**

Given an array of Intervals **arr[][]**, where **arr[i] = [start$_i$, end$_i$].** The task is to merge all of the **overlapping Intervals**.

Examples:

Input: arr[][] = [[1,3],[2,4],[6,8],[9,10]]
Output: [[1,4], [6,8], [9,10]]
Explanation: In the given intervals we have only two overlapping intervals here, [1,3] and [2,4] which on merging will become [1,4]. Therefore we will return [[1,4], [6,8], [9,10]].

Input: arr[][] = [[6,8],[1,9],[2,4],[4,7]]
Output: [[1,9]]
Explanation: In the given intervals all the intervals overlap with the interval [1,9]. Therefore we will return [1,9].

Constraints:
$1 \leq arr.size() \leq 10^5$
$0 \leq start_i \leq end_i \leq 10^5$

as

```cpp
#include <bits/stdc++.h>
using namespace std;
int main()
{
    int tc;
    cin>>tc;
    while(tc--){
        int n;
        cin>>n;
        vector<vector<int>>intervals(n,vector<int>(2));
                for(int i=0;i<n;i++){
            cin>>intervals[i][0];
            cin>>intervals[i][1];
        }
        sort(intervals.begin(),intervals.end());
        vector<vector<int>>ans;
        for(int i=0;i<n;i++){
            if(ans.empty() || intervals[i][0] > ans.back()[1]){
                ans.push_back(intervals[i]);
            }else{
                ans.back()[1] = max(intervals[i][1] ,ans.back()[1]);
            }
        }
        for(int i=0;i<ans.size();i++){
            cout<<ans[i][0]<<" "<<ans[i][1]<<endl;
        }
    }
    return 0;

}
```

```
~/Desktop/practice (3.749s)
g++ mergeIntervals.cpp

~/Desktop/practice (0.045s)
./a.out <testcase8.txt
1 4
6 8
9 10
1 6
7 8
```

Time Complexity : O(nlogn)
Space Complexity : O(n)

9. A Boolean Matrix Question

Given a boolean matrix mat[M][N] of size M X N, modify it such that if a matrix cell mat[i][j] is 1 (or true) then make all the cells of ith row and jth column as 1.

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

void setOnes(vector<vector<int>>& matrix) {
    unordered_map<int, int> row;
    unordered_map<int, int> col;

    // Mark rows and columns to be set to 1
    for (int i = 0; i < matrix.size(); i++) {
        for (int j = 0; j < matrix[0].size(); j++) {
            if (matrix[i][j] == 1) {
                row[i] = 1;
                col[j] = 1;
            }
        }
    }
    // Set marked rows and columns to 1
    for (int i = 0; i < matrix.size(); i++) {
        for (int j = 0; j < matrix[0].size(); j++) {
            if (row[i] || col[j]) {
                matrix[i][j] = 1;
            }
        }
    }
}
```

Testcases:

Input: {{1, 0}, {0, 0}}

Output: {{1, 1} {1, 0}}

Input: {{0, 0, 0}, {0, 0, 1}}

Output: {{0, 0, 1}, {1, 1, 1}}

Input: {{1, 0, 0, 1}, {0, 0, 1, 0}, {0, 0, 0, 0}}

Output: {{1, 1, 1, 1}, {1, 1, 1, 1}, {1, 0, 1, 1}}

```
~/Desktop/practice (3.733s)
g++ BooleanMatrix.cpp

~/Desktop/practice (0.044s)
./a.out <testcase9.txt
1 1
1 0
0 0 1
1 1 1
1 1 1 1
1 1 1 1
1 0 1 1
```

Time Complexity: O(m×n)

Space Complexity: O(m+n)

10. Print a given matrix in spiral form

Given an m x n matrix, the task is to print all elements of the matrix in spiral form.

```cpp
vector<int> spiralOrder(vector<vector<int>>& mat) {
    int n = mat.size();
    if (n == 0) return {};
    int m = mat[0].size();
    vector<int> arr;
    int left = 0, right = m - 1;
    int top = 0, bottom = n - 1;
    while (left <= right && top <= bottom) {
        // Traverse from left to right
        for (int i = left; i <= right; i++) {
            arr.push_back(mat[top][i]);
        }
        top++;
        // Traverse from top to bottom
        for (int i = top; i <= bottom; i++) {
            arr.push_back(mat[i][right]);
        }
        right--;
        // Check if there is a single row left
        if (top <= bottom) {
            // Traverse from right to left
            for (int i = right; i >= left; i--) {
                arr.push_back(mat[bottom][i]);
            }
            bottom--;
        }
        // Check if there is a single column left
        if (left <= right) {
            // Traverse from bottom to top
            for (int i = bottom; i >= top; i--) {
                arr.push_back(mat[i][left]);
            }
            left++;
        }
    }
    return arr;
}
```

TEST CASES:

Input: matrix = {{1, 2, 3, 4}, {5, 6, 7, 8}, {9, 10, 11, 12}, {13, 14, 15, 16 }}

Output: 1 2 3 4 8 12 16 15 14 13 9 5 6 7 11 10

Input: matrix = { {1, 2, 3, 4, 5, 6}, {7, 8, 9, 10, 11, 12}, {13, 14, 15, 16, 17, 18}}

Output: 1 2 3 4 5 6 12 18 17 16 15 14 13 7 8 9 10 11

Explanation: The output is matrix in spiral format.

```
~/Desktop/practice (0.044s)
./a.out <testcase9.txt
1 1
1 0
0 0 1
1 1 1
1 1 1 1
1 1 1 1
1 0 1 1

~/Desktop/practice (1.196s)
g++ SpiralMatrix.cpp

~/Desktop/practice (0.043s)
./a.out <testcase10.txt
1 2 3 4 8 12 16 15 14 13 9 5 6 7 11 10
1 2 3 4 5 6 12 18 17 16 15 14 13 7 8 9 10 11
```

Time Complexity: O(m×n)

Space Complexity: O(m×n)

13. Check if given Parentheses expression is balanced or not

Given a string str of length N, consisting of „(„ and „)„ only, the task is to check whether it is balanced or not.

```cpp
#include <bits/stdc++.h>
using namespace std;
int main()
{
    int tc;
    cin>>tc;
    while(tc--){
        string str;
        cin>>str;
        stack<char> st;
        int n = str.size();
        for(int i=0;i<n;i++){
            if(!st.empty()){
                if(st.top() == '(' and str[i] == ')'){
                    st.pop();
                }else{
                    st.push(str[i]);
                }
            }else{
                st.push(str[i]);
            }
        }
        if(st.empty()){
            cout<<"string is balanced"<<endl;
        }
        else{
            cout<<"String is unbalanced"<<endl;
        }
    }

}
```

TEST CASES:

Input: str = “(((()))()()”

Output: Balanced

Input: str = “())(())”

Output: Not Balanced

```
~/Desktop/practice (3.565s)
g++ balanced_paranthesis.cpp

~/Desktop/practice (0.045s)
./a.out <testcase13.txt
string is balanced
String is unbalanced
```

TIME COMPLEXITY:O(N)

SPACE COMPLEXITY:O(1)

14. Check if two Strings are Anagrams of each other

Given two strings s1 and s2 consisting of lowercase characters, the task is to check whether the two given strings are anagrams of each other or not. An anagram of a string is another string that contains the same characters, only the order of characters can be different.

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

bool isAnagram(const string& s, const string& t) {
    if (s.size() != t.size()) return false;
    unordered_map<char, int> charCount;
    for (char c : s) charCount[c]++;
    for (char c : t) {
        if (--charCount[c] < 0) return false;
    }
    return true;
}
int main() {
    int testCases;
    cin >> testCases;
    for (int i = 1; i <= testCases; i++) {
        string s, t;
        cin >> s;
        cin >> t;
        bool result = isAnagram(s, t);
        cout  << (result ? "True" : "False") << endl;
    }

    return 0;
}
```

TEST CASES:

Input: s1 = “geeks” s2 = “kseeg”

Output: true

Explanation: Both the string have same characters with same frequency. So, they are anagrams.

Input: s1 = “allergy” s2 = “allergic”

Output: false

Explanation: Characters in both the strings are not same. s1 has extra character „y‟ and s2 has extra characters „i‟ and „c‟, so they are not anagrams.

Input: s1 = “g”, s2 = “g”

Output: true

Explanation: Characters in both the strings are same, so they are anagrams.

```
~/Desktop/practice (3.653s)
g++ ValidAnagrams.cpp

~/Desktop/practice (0.041s)
./a.out <testcase14.txt
True
False
True
```

TIME COMPLEXITY:O(N)

SPACE COMPLEXITY:O(N)

15. Longest Palindromic Substring

Given a string str, the task is to find the longest substring which is a palindrome. If there are multiple answers, then return the first appearing substring.

```cpp
#include <iostream>
#include <string>
using namespace std;

string longestPalindrome(string s) {
    if (s.empty()) return "";

    string res;
    int resLen = 0;

    for (int i = 0; i < s.size(); i++) {
        // Odd-length palindromes
        int l = i, r = i;
        while (l >= 0 && r < s.size() && s[l] == s[r]) {
            if ((r - l + 1) > resLen) {
                res = s.substr(l, r - l + 1);
                resLen = r - l + 1;
            }
            l--;
            r++;
        }

        // Even-length palindromes
        l = i;
        r = i + 1;
        while (l >= 0 && r < s.size() && s[l] == s[r]) {
            if ((r - l + 1) > resLen) {
                res = s.substr(l, r - l + 1);
                resLen = r - l + 1;
            }
            l--;
            r++;
        }
    }
    return res;
}
```

TEST CASES:
Input: str = “forgeeksskeegfor”

Output: “geeksskeeg”

Explanation: There are several possible palindromic substrings like “kssk”, “ss”, “eeksskee” etc. But the substring “geeksskeeg” is the longest among all.

Input: str = “Geeks” Output: “ee”

Input: str = “abc” Output: “a”

Input: str = “” Output: “”

```
~/Desktop/practice (0.967s)
g++ LongestPalindromicSubstring.cpp

~/Desktop/practice (0.044s)
./a.out <testcase15.txt
geeksskeeg
ee
a
```

Time Complexity: O(t \times k^2)

Space Complexity : O(k)

16. Longest Common Prefix using Sorting

Given an array of strings arr[]. The task is to return the longest common prefix among each and every strings present in the array. If there‟s no prefix common in all the strings, return “-1”.

Input: arr[] = [“geeksforgeeks”, “geeks”, “geek”, “geezer”]

Output: gee

Explanation: “gee” is the longest common prefix in all the given strings. Input: arr[] = [“hello”, “world”] Output: -1 Explanation: There‟s no common prefix in the given strings.

```cpp
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

class Solution {
public:
    string longestCommonPrefix(vector<string>& arr) {
        string ans = "";
        sort(arr.begin(), arr.end());
        int n = arr.size();
        if (n == 0) return "-1";
        if (n == 1) return arr[0];
        string first = arr[0], last = arr[n - 1];
        for (int i = 0; i < min(first.length(), last.length()); i++) {
            if (first[i] == last[i]) ans += first[i];
            else break;
        }
        return ans.empty() ? "" : ans;
    }
};
```

```
~/Desktop/practice (1.016s)
g++ LongestCommonPrefix.cpp

~/Desktop/practice (0.037s)
./a.out < longcompre.txt
Longest common prefix: gee
No common prefix
```

Time Complexity: O(nlogn+m)

Space Complexity: O(1)

17. Delete middle element of a stack

Given a stack with push(), pop(), and empty() operations, The task is to delete the middle element of it without using any additional data structure.

```cpp
void deleteMiddle(stack<int>& st, int n, int current = 0) {
    if (st.empty() || current == n / 2) {
        st.pop();
        return;
    }
    int top = st.top();
    st.pop();
    deleteMiddle(st, n, current + 1);
    st.push(top);
}
int main() {
    int t;
    cin >> t;
    while (t--) {
        int n, element;
        cin >> n;
        stack<int> st;
        for (int i = 0; i < n; i++) {
            cin >> element;
            st.push(element);
        }
        deleteMiddle(st, n);
        stack<int> resultStack;
        while (!st.empty()) {
            resultStack.push(st.top());
            st.pop();
        }
        if (resultStack.empty()) {
            cout << "-1" << endl;
        } else {
            while (!resultStack.empty()) {
                cout << resultStack.top() << " ";
                resultStack.pop();
            }
            cout << endl;
        }
    }
    return 0;
}
```

Input : Stack[] = [1, 2, 3, 4, 5]

Output : Stack[] = [1, 2, 4, 5]

Input : Stack[] = [1, 2, 3, 4, 5, 6]
Output : Stack[] = [1, 2, 4, 5, 6]

```
~/Desktop/practice (0.782s)
g++ deleteMid.cpp

~/Desktop/practice (0.034s)
./a.out < deletemid.txt
1 2 4 5
1 2 4 5 6
```

Time Complexity: O(n),

Space Complexity: O(n)

18. Next Greater Element

Next Greater Element

Difficulty: **Medium** Accuracy: **32.95%** Submissions: **410K+** Points: **4**

Given an array **arr[ ]** of integers, the task is to find the next greater element for each element of the array in order of their appearance in the array. Next greater element of an element in the array is the nearest element on the right which is greater than the current element.
If there does not exist next greater of current element, then next greater element for current element is -1. For example, next greater of the last element is always -1.

Examples

Input: arr[] = [1, 3, 2, 4]
Output: [3, 4, 4, -1]
Explanation: The next larger element to 1 is 3, 3 is 4, 2 is 4 and for 4, since it doesn't exist, it is -1.

Input: arr[] = [6, 8, 0, 1, 3]
Output: [8, -1, 1, 3, -1]
Explanation: The next larger element to 6 is 8, for 8 there is no larger elements hence it is -1, for 0 it is 1 , for 1 it is 3 and then for 3 there is no larger element on right and hence -1.

Input: arr[] = [10, 20, 30, 50]
Output: [20, 30, 50, -1]
Explanation: For a sorted array, the next element is next greater element also exxept for the last element.

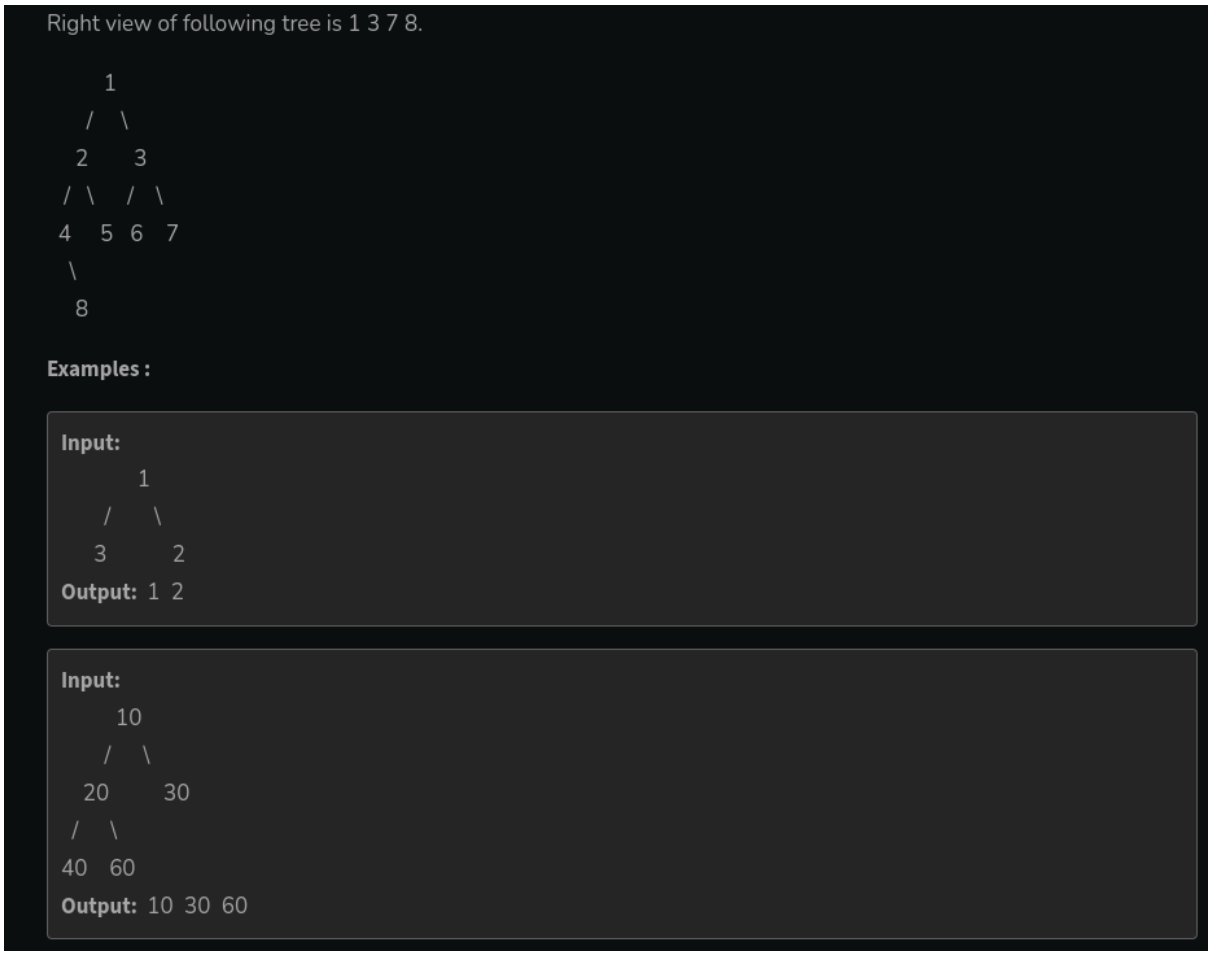
```cpp
#include <bits/stdc++.h>
using namespace std;

void NGE(int arr[], int n) {
    stack<int> s;
    s.push(arr[0]);

    for (int i = 1; i < n; i++) {
        while (!s.empty() && s.top() < arr[i]) {
            cout << s.top() << " --> " << arr[i] << endl;
            s.pop();
        }
        s.push(arr[i]);
    }

    while (!s.empty()) {
        cout << s.top() << " --> " << -1 << endl;
        s.pop();
    }
}

int main() {

    int n;
    cin >> n;
    int arr[n];
    for (int i = 0; i < n; i++) {
        cin >> arr[i];
    }
    NGE(arr, n);
    return 0;
}
```

Tme Complexity:O(n)

Space Complexity: O(n)

19.Print Right View of a Binary Tree

Given a Binary Tree, find **Right view** of it. Right view of a Binary Tree is set of nodes visible when tree is viewed from **right** side. Return the right view as a list.



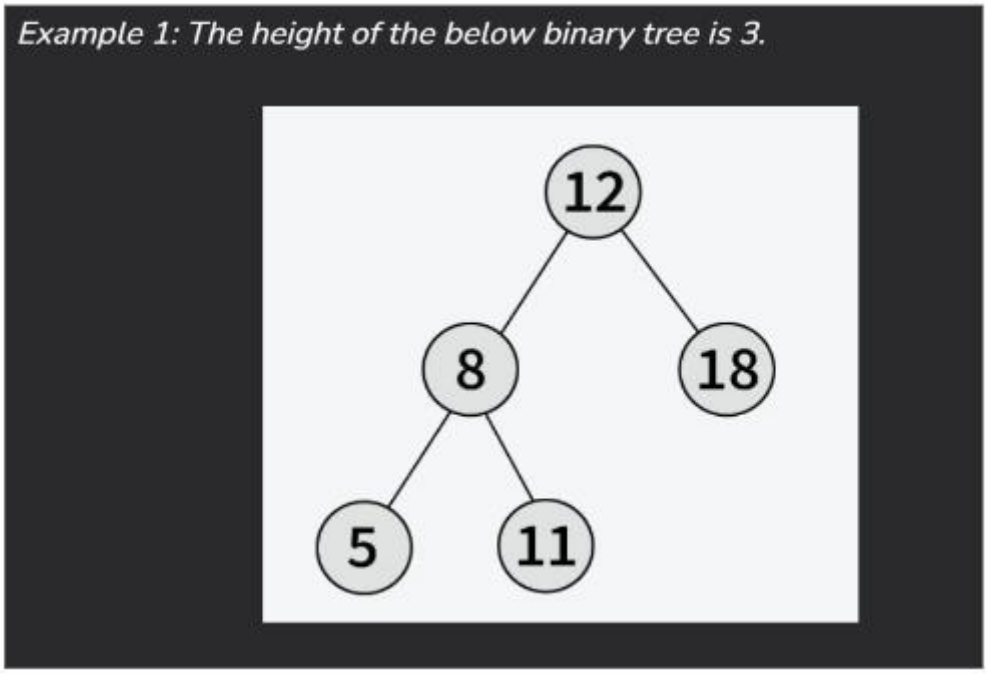
```cpp
#include <iostream>
#include <vector>
using namespace std;

struct TreeNode {
    int val;
    TreeNode* left;
    TreeNode* right;
    TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
};

void rightView(TreeNode* curr, vector<int>& ans, int currDepth) {
    if (!curr) return;

    if (currDepth == ans.size()) ans.push_back(curr->val);

    rightView(curr->right, ans, currDepth + 1);
    rightView(curr->left, ans, currDepth + 1);
}

vector<int> rightSideView(TreeNode* root) {
    vector<int> ans;
    rightView(root, ans, 0);
    return ans;
}
```

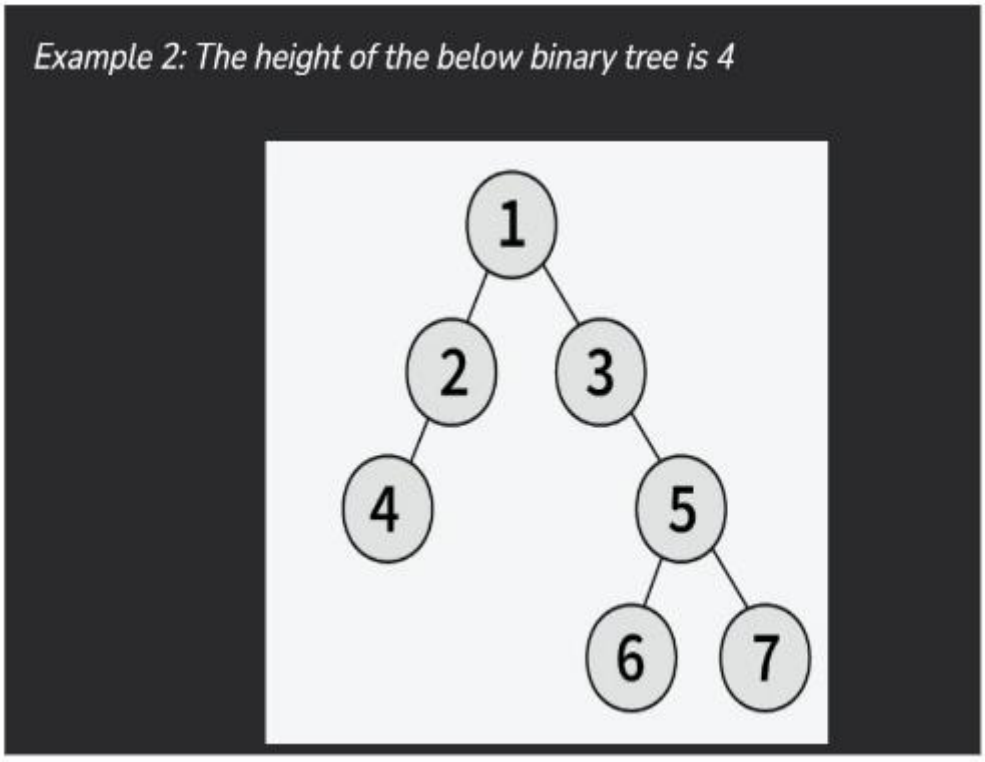
Time Complexity : O(n)

Space Complexity : O(h) where h is height of the tree O(n) to store the right view nodes.

12. Maximum Depth or Height of Binary Tree

Given a binary tree, the task is to find the maximum depth or height of the tree. The height of the tree is the number of vertices in the tree from the root to the deepest node.





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

struct node {
  int data;
  struct node *left;
  struct node *right;
};

struct node *newNode(int data) {
  struct node *node = (struct node *)malloc(sizeof(struct node));

  node->data = data;

  node->left = NULL;
  node->right = NULL;
  return (node);
}

int helper(node* root){
  if(!root) return 0;
  int left=1+helper(root->left);
  int right=1+helper(root->right);
  return max(left,right);
}
```

Time Complexity: O(n)

Space Complexity: O(1)