S.Tejeswar AI&DS – 22AD104

Date: 9th Nov, 2024 (practice Problems)

1. Maximum Subarray Sum – Kadane"s Algorithm:

Given an array arr[], the task is to find the subarray that has the maximum sum and return its sum.

C++ Program:

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

int maximumSubArraySum(vector<int> array){
    if(array.size() < 1){
        return 0;
    }
    int maximumSubArraySum = array[0];
    int currentSum = 0;
    for(int num: array){
        currentSum += num;
        maximumSubArraySum = max(maximumSubArraySum, currentSum);
        if(currentSum < 0){
            currentSum = 0;
        }
    }
    return maximumSubArraySum;
}

int main()
{
    int n;
    int num;
    string inputLine;
    vector<int> inputArray;
    cout << "Enter size of array: ";
    cin >> n;

    cout << "Enter the space seperated elements: ";
    for(int i = 0; i < n; i++){
        cin >> num;
        inputArray.push_back(num);
    }
    cout << maximumSubArraySum(inputArray);
    return 0;
}
```

Time Complexity: O(N)
Space Complexity: O(1)

Input 1:
n = 7
array = {2, 3, -8, 7, -1, 2, 3}

Output1:

```
Enter size of array: 7
Enter the space seperated elements: 2 3 -8 7 -1 2 3
11
```

Input 2:
n = 2
array = [-2, -4]

Output 2:

```
Enter size of array: 2
Enter the space seperated elements: -2 -4
-2
```

2. Maximum Product Subarray
Given an integer array, the task is to find the maximum product of any subarray.

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

int maximumSubArrayProduct(vector<int> array){
    if(array.size() < 1){
        return 0;
    }
    int arraySize = array.size();
    int currMin = array[0];
    int currMax = array[0];
    int maximumProduct = array[0];

    for(int i = 1; i < arraySize; i++){
        int temp = max(array[i], max(array[i] * currMax, array[i] * currMin ));
        currMin = min(array[i], min(currMax * array[i], currMin * array[i]));
        currMax = temp;
        maximumProduct = max(maximumProduct, currMax);

    }
    return maximumProduct;
}

int main()
{
```

```
    int n;
    int num;
    string inputLine;
    vector<int> inputArray;
    cout << "Enter size of array: ";
    cin >> n;
    cout << "Enter the space seperated elements: ";
    for(int i = 0; i < n; i++){
        cin >> num;
        inputArray.push_back(num);
    }
    cout << maximumSubArrayProduct(inputArray);
    return 0;
}
```

Time Complexity: O(N)
Space Complexity: O(1)

Input 1:
n = 6
arr = [-2, 6, -3, -10, 0, 2]

Output 1:

```
Enter size of array: 6
Enter the space seperated elements: -2 6 -3 -10 0 2
180
```

Input 2:
N = 5
arr = [-1, -3, -10, 0, 60]

```
Enter size of array: 5
Enter the space seperated elements: -1 -3 -10 0 60
60
```

3. Search in a sorted and rotated Array
Given a sorted and rotated array arr[] of n distinct elements, the task is to find the index of given key in the array. If the key is not present in the array, return -1

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

int findElement(vector<int>& arr, int target){
    int low = 0;
```

```
    int high = arr.size() - 1;
    int mid = (low + high) / 2;
    while(low <= high){
        mid = (low + high) / 2;
        if(arr[mid] == target){
            return mid;
        }
        if(arr[low] <= arr[mid]){
            if(arr[low] <= target && target <= arr[mid]){
                high = mid - 1;
            }
            else{
                low = mid + 1;
            }
        }
        else {
            if(arr[mid] < target && target <= arr[high]){
                low = mid + 1;
            }
            else{
                high = mid - 1;
            }
        }
    }
    return -1;
}



int main()
{
    vector<int> arr = {4, 5, 6, 7, 0, 1, 2};
    int target = 6;
    int result = findElement(arr, target);

    cout << result;
    return 0;
}
```

Time Complexity: O(logn)
Space Complexity: O(1)

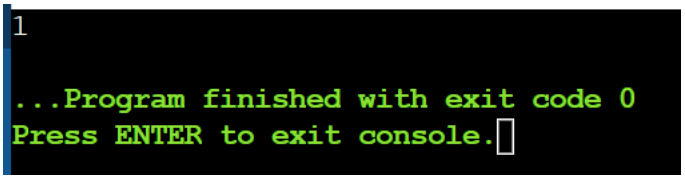
Input1:
Arr = { 4, 5, 6, 7, 0, 1, 2 }, key = 3
Output1:

```
2

...Program finished with exit code 0
Press ENTER to exit console.
```

Input2:
 arr= {50, 10, 20, 30, 40}, key = 10

OutPut2:



4. Container with Most Water

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


int maxArea(vector<int>& array) {
        int area ;
        int maxArea = 0;
        int low = 0;
        int high = array.size() - 1;
        while(low <= high){
            area = min(array[low], array[high]) * (high - low);
            maxArea = max(maxArea, area);
            if(array[low] <= array[high]){
                low ++;
            }
            else{
                high--;
            }
        }
        return maxArea;
    }



int main()
{
    vector<int> arr = {1, 5, 4, 3};
    int result = maxArea(arr);
    cout << result;
    return 0;
}
```

Time Complexity: O(N)
Space Complexity: O(1)

Input1:
Array = {1, 5, 4, 3}

Output1:

```
6

...Program finished with exit code 0
Press ENTER to exit console.
```

Input2:
Array ={3, 1, 2, 4, 5}
Output2:

```
12

...Program finished with exit code 0
Press ENTER to exit console.
```

5. Find the Factorial of a large number

C++ Program

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

using namespace std;

int factorial(int n){
    vector<int> result = {1};
    for(int num = 2; num <= n; num++){
        int carry = 0;
        for(int i = 0; i < result.size(); i++){
            int ans = result[i] * num + carry;
            result[i] = ans % 10;
            carry = ans / 10;
        }
        while(carry){
            result.push_back(carry % 10);
            carry = carry / 10;
        }

    }
    for (int i = result.size() - 1; i >= 0; i--) {
            cout << result[i];
        }
    cout << '\n';
    return 0;
}

int main()
{
    int n1 = 100;
    int n2 = 50;
```

```
    cout << "Factorial of " << n1 << ": \n";
    factorial(n1);
    cout <<"\nFactorial of " << n2 << ": \n";
    factorial(n2);
    return 0;
}
```

Time Complexity: O(n^2 logN)
Space Complexity: O(maximum digits in n!)

Input:
N1 = 100
N2 = 50

Output:

```
Factorial of 100:
93326215443944152681699238856266700490715968264381621468592963895217599993229915608941463976156518286253697920827223
758251185210916864000000000000000000000000

Factorial of 50:
30414093201713378043612608166064768844377641568960512000000000000
```

6.
Trapping Rainwater Problem states that given an array of n non-negative integers arr[] representing an elevation map where the width of each bar is 1, compute how much water it can trap after rain.

C++ Program:

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

using namespace std;

int findRainwaterTrapped(vector<int>& nums){
    int arrayLength = nums.size();
    int trappedWater = 0;

    vector<int> maxPrefix(arrayLength, nums[0]);
    vector<int> maxSuffix(arrayLength, nums[arrayLength - 1]);

    for(int i = 1; i < arrayLength; i++){
        maxPrefix[i] = max(maxPrefix[i - 1], nums[i]);
    }
    for(int i = arrayLength - 2; i >= 0; i--){
        maxSuffix[i] = max(maxSuffix[i + 1], nums[i]);
    }

    for(int i = 0; i < arrayLength - 1; i++){
        int minElevation = min(maxPrefix[i], maxSuffix[i]);
        if(minElevation > nums[i]){
            trappedWater += minElevation - nums[i];
```

```
        }
    }
    return trappedWater;
}

int main()
{
    vector<int> nums ={3, 0, 2, 0, 4};
    cout << "Trapped Rainwater is: " << findRainwaterTrapped(nums);
    return 0;
}
```

Time Complexity: O(N)
Space Complexity: O(N)

Input 1:
Height = {3, 0, 1, 0, 4, 0, 2}
Output1:

```
Trapped Rainwater is: 10

...Program finished with exit code 0
Press ENTER to exit console.
```

Input2:
Height = {3, 0, 2, 0, 4}
Output2:

```
Trapped Rainwater is: 7

...Program finished with exit code 0
Press ENTER to exit console.
```

Input3:
Height = {1, 2, 3, 4}
Output3:

```
Trapped Rainwater is: 0

...Program finished with exit code 0
Press ENTER to exit console.
```

Input4:
Height = {10, 9, 0, 5}

```
Trapped Rainwater is: 5

...Program finished with exit code 0
Press ENTER to exit console.
```

7. Chocolate Distribution Problem

Given an array arr[] of n integers where arr[i] represents the number of chocolates in ith packet. Each packet can have a variable number of chocolates. There are m students, the task is to distribute chocolate packets such that:

Each student gets exactly one packet.

The difference between the maximum and minimum number of chocolates in the packets given to the students is minimized.

C++ Program

```
#include <iostream>
#include <vector>
#include <bits/stdc++.h>
#include <climits>


using namespace std;

int minimumDifference(vector<int>& nums, int m){
    int arrayLength = nums.size();
    int minimumDifference = INT_MAX;
    sort(nums.begin(), nums.end());
    for(int i = 0; i < arrayLength - m + 1; i++){
        minimumDifference = min(minimumDifference, nums[i + m - 1] - nums[i]);
    }
    return minimumDifference;
}

int main()
{
    vector<int> nums = {7, 3, 2, 4, 9, 12, 56};
    int m = 5;
    cout << "Minimum Difference is: " << minimumDifference(nums, m);
    return 0;
}
```

Time Complexity: O(NlogN)

Space Complexity: O(1)

Input1:

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

Output1:

```
Minimum Difference is: 7

...Program finished with exit code 0
Press ENTER to exit console.
```

Input2:

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

Output2:

```
Minimum Difference is: 7

...Program finished with exit code 0
Press ENTER to exit console.
```

8. Merge Overlapping Intervals
Given an array of time intervals where arr[i] = [starti, endi], the task is to merge all the overlapping intervals into one and output the result which should have only mutually exclusive intervals.

C++ Program:

```
#include <iostream>
#include <vector>
#include <bits/stdc++.h>
#include <climits>


using namespace std;

vector<vector<int>> mergeIntervals(vector<vector<int>>& intervals){
    sort(intervals.begin(), intervals.end());
    int intervalsCount = intervals.size();
    vector<vector<int>> mergedIntervals;
    vector<int> currInterval = intervals[0];
    for(int i = 1; i < intervalsCount; i++){
        if(currInterval[1] >= intervals[i][0]){
            currInterval[1] = max(intervals[i][1], currInterval[1]);
        }
        else{
            mergedIntervals.push_back(currInterval);
            currInterval = intervals[i];
        }
    }
    mergedIntervals.push_back(currInterval);
    return mergedIntervals;
}

int main()
{
    vector<vector<int>> nums = {{1, 3}, {2, 4}, {6, 8}, {9, 10}};
    cout << "Merged Intervals is: \n";
    vector<vector<int>>mergedIntervals = mergeIntervals(nums);
    for(int i = 0; i < mergedIntervals.size(); i++){
        cout<<"[" << mergedIntervals[i][0] << ',' << mergedIntervals[i][1] <<']'
<< '\n';
    }
    return 0;
}
```

Time Complexity: O(N)
Space Complexity: O(N)

Input1:
arr[] = [[1, 3], [2, 4], [6, 8], [9, 10]]
Output1:

```
Merged Intervals is:
[1,4]
[6,8]
[9,10]
```

Input2:
[[7, 8], [1, 5], [2, 4], [4, 6]]
Output2:

```
Merged Intervals is:
[1,6]
[7,8]
```

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.

C++ Program:

```
#include <iostream>
#include <vector>
#include <bits/stdc++.h>
#include <climits>


using namespace std;

vector<vector<int>> modifyMatrix(vector<vector<int>> matrix){
    vector<int>rows(matrix.size(), 0);
    vector<int>columns(matrix[0].size(), 0);
    for(int i = 0; i < matrix.size(); i++){
        for(int j = 0; j < matrix[0].size(); j++){
            if(matrix[i][j] == 1){
                rows[i] = 1;
                columns[j] = 1;
            }
        }
    }
    for(int i = 0; i < matrix.size(); i++){
        for(int j = 0; j < matrix[0].size(); j++){
            if(rows[i] == 1 || columns[j] == 1){
                matrix[i][j] = 1;
            }
        }
    }
    return matrix;
```

```
}

int main()
{
    vector<vector<int>> nums = {{1, 0},
{0, 0}};
    cout << "Modified Matrix is: \n";
    vector<vector<int>>matrix1 = modifyMatrix(nums);
    for(int i = 0; i < matrix1.size(); i++){
        for(int j = 0; j < matrix1[0].size(); j++){
            cout << matrix1[i][j] << ' ';
        }
        cout << '\n';
    }
    return 0;
}
```

Time Complexity: O(m * n)
Space Complexity: O(m) + O(n)

Input1:
{{1, 0},
{0, 0}}

Output1:

```
Modified Matrix is:
1 1
1 0
```

Input2:
{{0, 0, 0},
{0, 0, 1}}
Output2:

```
Modified Matrix is:
0 0 1
1 1 1
```

Input3:
{{1, 0, 0, 1},
{0, 0, 1, 0},
{0, 0, 0, 0}}

```
Modified Matrix is:
1 1 1 1
1 1 1 1
1 0 1 1
```

10.
Print a given matrix in spiral form

C++ Program:

```
#include <iostream>
```

```
#include <vector>
using namespace std;
void printspiral(vector<vector<int>> &mat, int n, int m) {
    int top = 0;
    int bottom = n - 1;
    int left = 0;
    int right = m - 1;
    while (top <= bottom && left <= right) {
        for (int i = left; i <= right; i++) {
            cout << mat[top][i] << " ";
        }
        top++;
        for (int i = top; i <= bottom; i++) {
            cout << mat[i][right] << " ";
        }
        right--;
        if (top <= bottom) {
            for (int i = right; i >= left; i--) {
                cout << mat[bottom][i] << " ";
            }
            bottom--;
        }
        if (left <= right) {
            for (int i = bottom; i >= top; i--) {
                cout << mat[i][left] << " ";
            }
            left++;
        }
    }
}
int main() {
    vector<vector<int>> mat = {
    {1, 2, 3, 4}, {5, 6, 7, 8}, {9, 10, 11, 12}, {13, 14, 15, 16}};
    vector<vector<int>> mat1 = {
    {1, 2, 3, 4, 5, 6}, {7, 8, 9, 10, 11, 12}, {13, 14, 15, 16, 17,
    18}};
    printspiral(mat, mat.size(), mat[0].size());
    cout << endl;
    printspiral(mat1, mat1.size(), mat1[0].size());
    return 0;
}
```

Time Complexity: O(m*n)
Space Complexity: O(1)

Input1:
{{1, 2, 3, 4},
{5, 6, 7, 8},
{9, 10, 11, 12},
{13, 14, 15, 16 }}

Output1:

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

Input2:

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

Output2:

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

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.

C++ Program:

```
#include <iostream>
#include <stack>
#include <string>
#include <vector>
using namespace std;

bool isbalanced(string s) {
    stack<char> st;
    st.push(s[0]);
    int i = 1;
    while (i < s.size()) {
        if (s[i] == '(') {
            st.push(s[i]);
        }
        else {
            if (st.empty() || st.top() != '(') {
            return false;
            }
            st.pop();
        }
        i++;
    }
    return true;
}
int main() {
    vector<string> tc = {"((()))()()", "())((())"};
    for (string t : tc) {
        if (isbalanced(t)) {
            cout << "Balanced" << endl;
        }
        else {
            cout << "Not Balanced" << endl;
        }
    }
```

```
}
```

Output:

```
Balanced
Not Balanced
```

Time Complexity: O(N)
Space Complexity: O(N)

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.

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.

C++Program:

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

bool isAnagram(string s1, string s2) {
    int l1 = s1.size();
    int l2 = s2.size();
    if(l1 != l2){
        return false;
    }
    sort(s1.begin(), s1.end());
    sort(s2.begin(), s2.end());
    for (int i = 0; i < l1; i++) {
        if (s1[i] != s2[i]) {
        return false;
        }
    }
    return true;
}
int main() {
```

```
    vector<vector<string>> input = {{"geeks", "kseeg"}, {"allergy", "allergic"},
{"g", "g"}};
    for(auto test : input){
        if(isAnagram(test[0], test[1])){
            cout<< "True" << '\n';
        }
        else{
            cout << "False" << '\n';
        }
    }
}
```

Time Complexity: O(NlogN)
Space Complexity: O(1)

Output:

```
True
False
True
```

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.

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.

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


using namespace std;

string getpal(const string &s, int left, int right) {
    while (left >= 0 && right < s.size() && s[left] == s[right]) {
        left--;
        right++;
    }
    return s.substr(left + 1, right - left - 1);
}

int main() {
    vector<string> st = {"forgeeksskeegfor", "Geeks", "abc", ""};
    for (string s : st) {
        string large;
        for (int i = 0; i < s.size(); i++) {
```

```
            string s1 = getpal(s, i, i);
            string s2 = getpal(s, i, i + 1);
            if (s1.size() > large.size()) {
                large = s1;
            }
            if (s2.size() > large.size()) {
                large = s2;
            }
        }
        cout << large << endl;
    }
}
```

Time Complexity: O(NlogN)
Space Complexity: O(1)

OUTPUT:

```
geeksskeeg
ee
a
```

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.

Program:

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

string findcommonPrefix(vector<string> arr) {
    sort(arr.begin(), arr.end());
    string first = arr[0];
    string last = arr[arr.size() - 1];
    string cp = "";
    int n = min(first.size(), last.size());
    for (int i = 0; i < n; i++) {
        if (first[i] == last[i]) {
        cp += first[i];
        } else {
        break;
        }
    }
```

```
    if (cp.empty()) {
        return "-1";
    } else {
        return cp;
    }
}
int main() {
    vector<vector<string>> arrs = {{"geeksforgeeks", "geeks", "geek","geezer"},
    {"hello", "world"}};
    string res;
    for (auto arr : arrs) {
        res = findcommonPrefix(arr);
        cout << res << endl;
    }
}
```

Output:

```
gee
-1
```

Time Complexity: O(N logN)
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.

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]

Program:

```
#include <iostream>
#include <stack>
using namespace std;

void deletemid(stack<int> &st, int size, int curr = 0) {
    if (curr == size / 2) {
        st.pop();
        return;
    }
    int temp = st.top();
    st.pop();
    deletemid(st, size, curr + 1);
    st.push(temp);
    }


int main() {
```

```
    stack<int> st;
    stack<int> st1;
    st.push(1);
    st.push(2);
    st.push(3);
    st.push(4);
    st.push(5);
    st1.push(1);
    st1.push(2);
    st1.push(3);
    st1.push(4);
    st1.push(5);
    st1.push(6);
    deletemid(st, st.size());
    deletemid(st1, st1.size());
    while (!st.empty()) {
        cout << st.top() << " ";
        st.pop();
    }
    cout << endl;
    while (!st1.empty()) {
        cout << st1.top() << " ";
        st1.pop();
    }
    cout << endl;
    return 0;
}
```

OUTPUT:

```
5 4 2 1
6 5 4 2 1
```

Time Complexity: O(N)

18. Next Greater Element (NGE) for every element in given Array

Given an array, print the Next Greater Element (NGE) for every element.

Note: The Next greater Element for an element x is the first greater element on the right side of x in the array. Elements for which no greater element exist, consider the next greater element as -1.

Input: arr[] = [4 , 5 , 2 , 25]

Output: 4 -> 5

5 -> 25

2 -> 25

25 -> -1

Explanation: Except 25 every element has an element greater than them present on the right side

Input: arr[] = [13 , 7, 6 , 12]

Output: 13 -> -1

7 -> 12

6 -> 12

12 -> -1
Explanation: 13 and 12 don‟t have any element greater than them present on the right side

Program:

```
#include <iostream>
#include <vector>
using namespace std;
vector<int> findnextlarge(vector<int> arr) {
    int n = arr.size();
    vector<int> res(n);
    for (int i = 0; i < n; i++) {
        int curmax = arr[i];
    for (int j = i + 1; j < n; j++) {
        curmax = max(curmax, arr[j]);
        if (curmax != arr[i]) {
            break;
        }
    }
        if (curmax != arr[i]) {
            res[i] = curmax;
        }
        else {
            res[i] = -1;
        }
    }
    return res;
}
int main() {

vector<vector<int>> arrs = {{4, 5, 2, 25}, {13, 7, 6, 12}};
vector<int> res;
for (auto arr : arrs) {
    res = findnextlarge(arr);
    for (int i = 0; i < arr.size(); i++) {
        cout << arr[i] << "  ---next--- " << res[i] << endl;
    }
    cout << endl;
    }
}
```

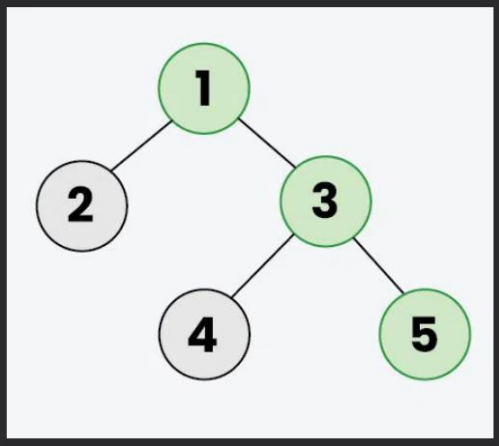
OUTPUT:

```
13 ---next--- -1
7 ---next--- 12
6 ---next--- 12
12 ---next--- -1
```

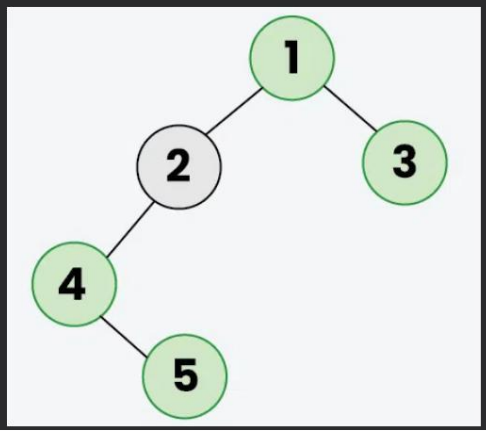
19. Print Right View of a Binary Tree

Given a Binary Tree, the task is to print the Right view of it. The right view of a Binary Tree is a set of rightmost nodes for every level.

*Example 1: The **Green** colored nodes (1, 3, 5) represents the Right view in the below Binary tree.*



*Example 2: The **Green** colored nodes (1, 3, 4, 5) represents the Right view in the below Binary tree.*



PROGRAM:

```
#include <iostream>
#include <queue>
using namespace std;

struct Node {
    int data;
    Node *left;
    Node *right;
    Node(int val) : data(val), left(nullptr), right(nullptr) {}
};


class BinaryTree {
    public:
    Node *root;
    void printright() {
        queue<Node *> q;
        q.push(root);
        while (!q.empty()) {
            int n = q.size();
            for (int i = 0; i < n; i++) {
```

```
                Node *cur = q.front();
                q.pop();
                if (i == n - 1) {
                    cout << cur->data << " ";
                }
                if (cur->left != nullptr) {
                q.push(cur->left);
                }
                if (cur->right != nullptr) {
                    q.push(cur->right);
                }
            }
        }
    }
};


int main() {

    BinaryTree t;
    t.root = new Node(1);
    t.root->left = new Node(2);
    t.root->right = new Node(3);
    t.root->left->left = new Node(4);
    t.root->left->left->right = new Node(5);
    t.printright();
    cout << endl;
    BinaryTree t1;
    t1.root = new Node(1);
    t1.root->left = new Node(2);
    t1.root->right = new Node(3);
    t1.root->right->left = new Node(4);
    t1.root->right->right = new Node(5);
    t1.printright();
    return 0;
}
```

OUTPUT:

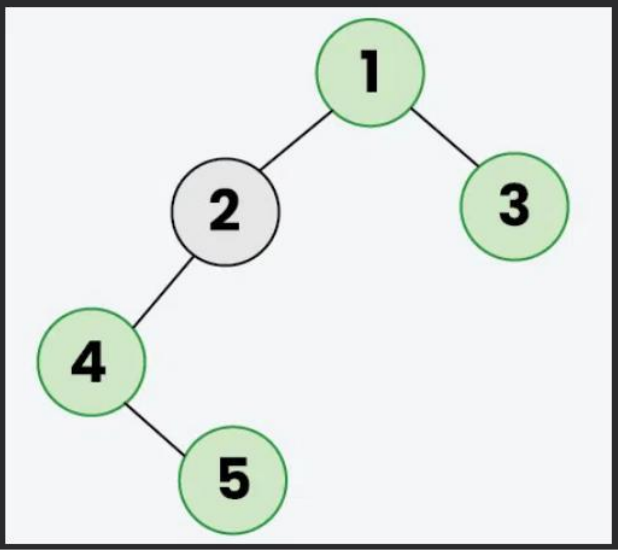
```
1 3 4 5
1 3 5
```

Time Complexity: O(N)

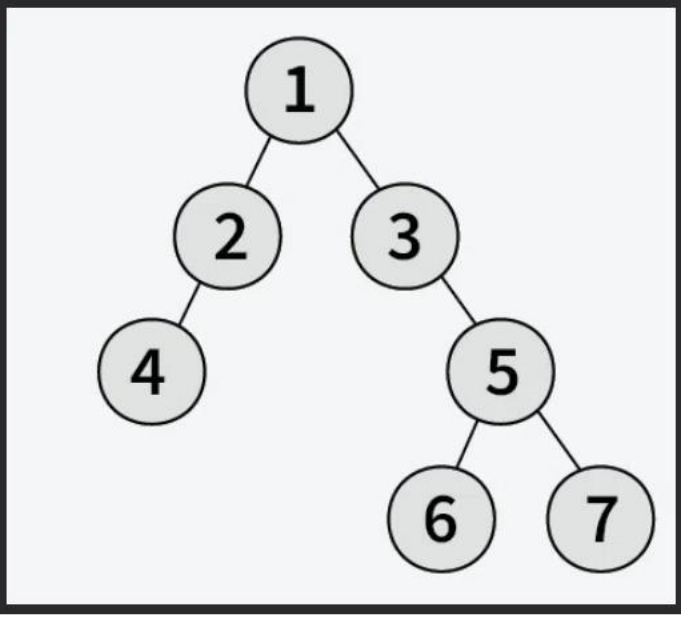
20. Maximum Depth or Height of Binary Tree

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.

*Example 2: The **Green** colored nodes (1, 3, 4, 5) represents the Right view in the below Binary tree.*



Example 2: The height of the below binary tree is 4



PROGRAM:

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

struct Node {
    int data;
    Node *left;
    Node *right;
    Node(int val) {
```

```
        data = val;
        left = nullptr;
        right = nullptr;
    }
};


int maxDepth(Node *node) {
      if (node == nullptr ) return 0;
           int lDepth = maxDepth(node->left);
           int rDepth = maxDepth(node->right);
           return max(lDepth, rDepth) + 1;
}

int main() {

Node *root = new Node(12);
root->left = new Node(8);
root->right = new Node(18);
root->left->left = new Node(5);
root->left->right = new Node(11);
cout << maxDepth(root) << endl;

Node *root1 = new Node(1);
root1->left = new Node(2);
root1->right = new Node(3);
root1->left->left = new Node(4);
root1->right->right = new Node(5);
root1->right->right->left = new Node(6);
root1->right->right->right = new Node(7);
cout << maxDepth(root1) << endl;
return 0;
}
```

Time Complexity: O(N)

OUTPUT: