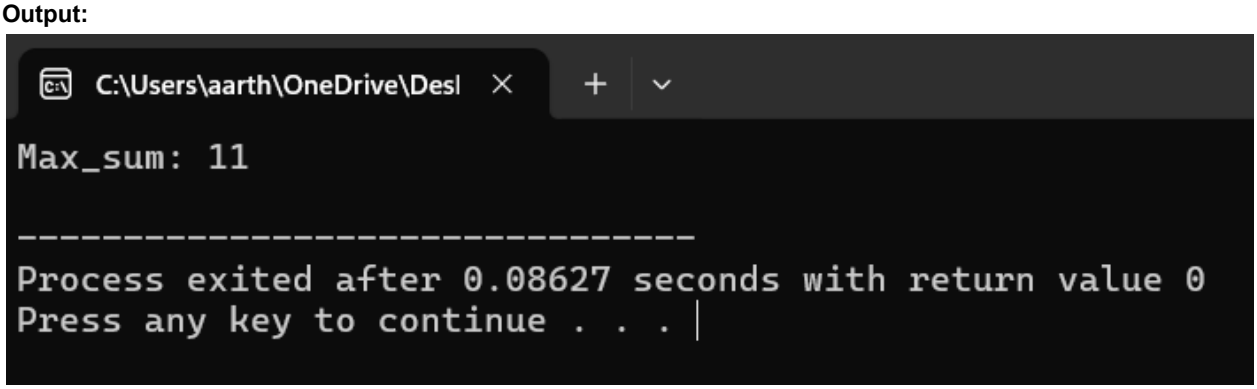
1. **Kadane's Algo -> Maximum SubArray:**

```
#include <bits/stdc++.h>
using namespace std;
int FindmaxSubarray(int arr[], int n)
{
	int max_sum = arr[0];
	int current_sum = 0;
	for(int i=0; i<n; i++)
	{
		current_sum += arr[i];
		max_sum = max(max_sum , current_sum);
		if(current_sum < 0)
		{
			current_sum = 0;
		}
	}
	return max_sum;
}
int main()
{
	int arr[] = {2, 3, -8, 7, -1, 2, 3} ;
	int n = sizeof(arr)/sizeof(arr[0]);
	int result = FindmaxSubarray(arr, n);
	cout << "Max_sum: " <<result << endl;
	return 0;
}
```

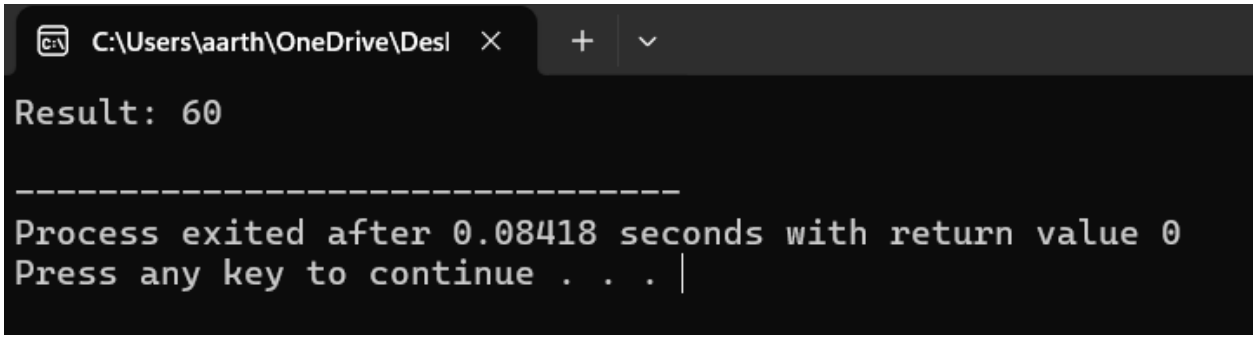
Time Complexity : O(n)
Output:



2. **Maximum Product Subarray**

```
#include <bits/stdc++.h>
using namespace std;
int findMaxProd(int nums[], int n)
{
	int max_prod = nums[0];
	int min_prod = nums[0];
	int result = nums[0];
	for(int i=1; i<=n; i++)
	{
		if(nums[i] < 0)
		{
			swap(max_prod, min_prod);
		}
		max_prod = max(nums[i], max_prod * nums[i]);
		min_prod = min(nums[i], min_prod * nums[i]);
		result = max(result, max_prod);
	}
	return result;
}
int main()
{
	int nums[] = {-1, -3, -10, 0, 60} ;

	int n = sizeof(nums)/sizeof(nums[0]);
	int result = findMaxProd(nums, n);
	cout << "Result: " << result<<endl;
	return 0;
}
```

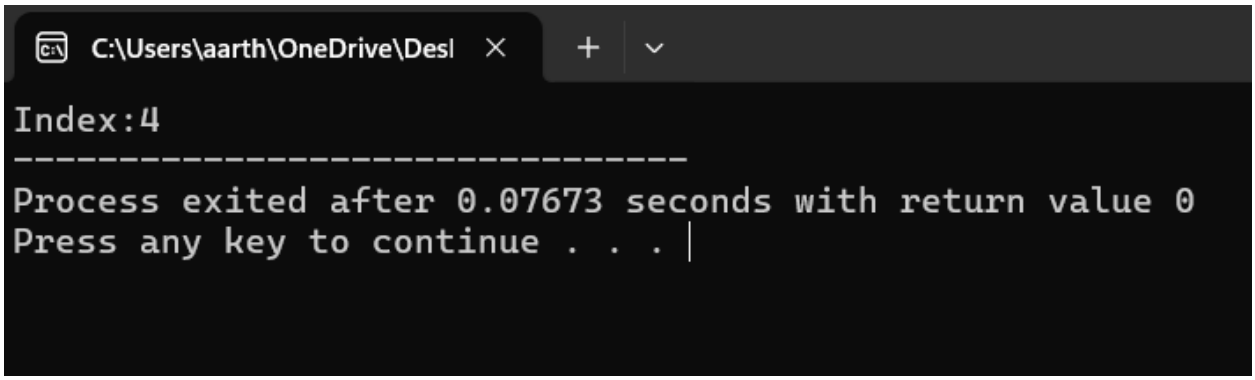
Time Complexity : O(n)
Output:



3. **Search in a sorted and rotated Array**

```
#include <bits/stdc++.h>
using namespace std;
int main()
{
        int nums[] = { 4, 5, 6, 7, 0, 1, 2 };
        int target = 0;
        int n = sizeof(nums)/sizeof(nums[0]);
        bool found = false;
  for(int i=0; i<n; i++)
  {
        if(target == nums[i])
        {
                cout << "Index:" << i;
                found = true;
                break;
                }
        }
        if(!found){
                cout << "Index not found";
        }
        return 0;

}
```

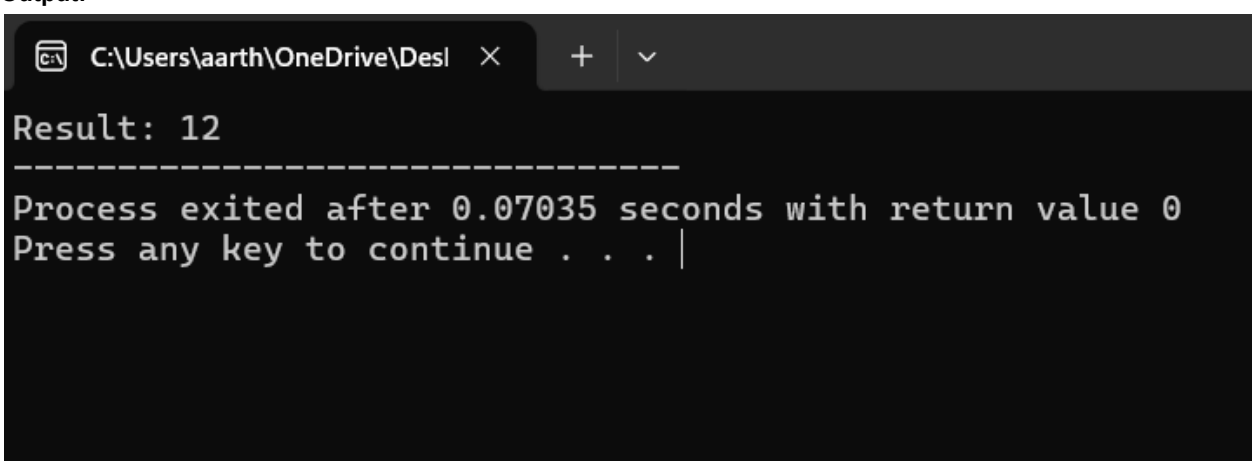
Time Complexity : O(n)
Output:



4. **Container with Most Water**

```
#include <bits/stdc++.h>
using namespace std;
int main()
{
	int nums[] = {3, 1, 2, 4, 5};
	int n = sizeof(nums)/sizeof(nums[0]);
	int i = 0;
	int j = n-1;
	int maxArea = 0;
	while(i<j)
	{
		int area = min(nums[i], nums[j]) * (j-i);
		maxArea = max(maxArea,area);
		if(nums[i] < nums[j])
		{
			i++;
		}
		else
		{
			j--;
		}

	}
	cout << "Result: " << maxArea;
}
```

Time Complexity : O(n)
Output:



5. **Find the Factorial of a large number**

```
#include <iostream>
#include <vector>
using namespace std;
vector<int> factorialDP(int n) {
    vector<int> result(1, 1);
    for (int i = 1; i <= n; ++i) {
        int carry = 0;
        for (int j = 0; j < result.size(); ++j) {
            int prod = result[j] * i + carry;
            result[j] = prod % 10;
            carry = prod / 10;
        }
        while (carry) {
            result.push_back(carry % 10);
            carry /= 10;
        }
    }
    return result;
}
int main() {
    vector<int> fact;
    fact = factorialDP(100);
    for (auto it = fact.rbegin(); it != fact.rend(); ++it) {
        cout << *it;
    }
    cout << endl;
    fact = factorialDP(50);
    for (auto it = fact.rbegin(); it != fact.rend(); ++it) {
        cout << *it;
    }
    cout << endl;
    return 0;
}
```

Time Complexity : O(n)
Output:

```
/tmp/hBpGMXQOQE.o
93326215443944152681699238856266700490715968264381621468592963895217599993229915608941463976156518286253697920827223758251185210916864000000000000000000000000
30414093201713378043612608166064768844377641568960512000000000000
```

6. **Trapping Rainwater Problem**

```
#include <bits/stdc++.h>
using namespace std;
int main() {
    int height[] = {3, 0, 1, 0, 4, 0, 2};
    int n = sizeof(height) / sizeof(height[0]);
    int left = 0;
    int right = n - 1;
    int leftMax = height[left];
    int rightMax = height[right];
    int water = 0;
    while (left < right) {
        if (leftMax < rightMax) {
            left++;
            leftMax = max(leftMax, height[left]);
            water += leftMax - height[left];
        } else {
            right--;
            rightMax = max(rightMax, height[right]);
            water += rightMax - height[right];
        }
    }
    cout << "Total Water Trapped: " << water << endl;
    return 0;
}
```

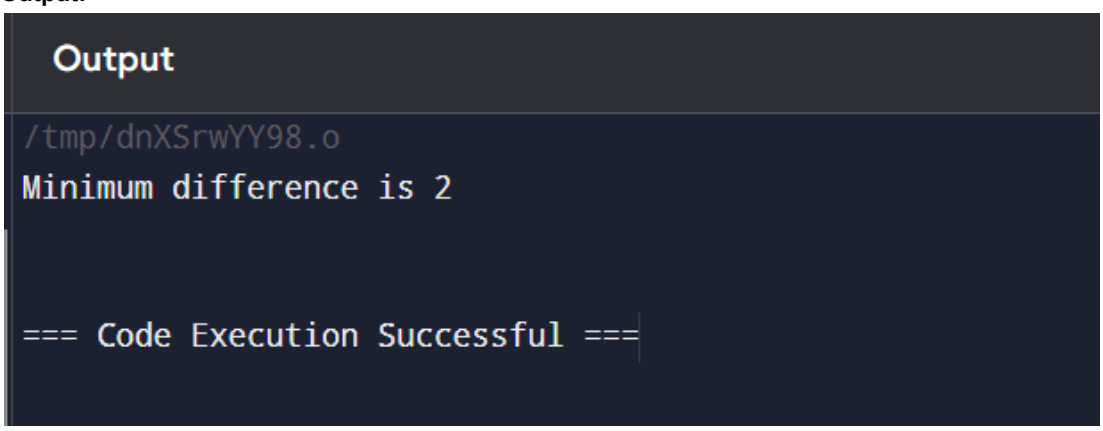
Time Complexity : O(n)
Output:

```
Output
/tmp/Ncgwyuioh4.o
Total Water Trapped: 10


=== Code Execution Successful ===
```

7. **Chocolate Distribution Problem**

```
#include <bits/stdc++.h>
using namespace std;
int findMinDifference(vector<int>& arr, int n, int m) {
    if (m == 0 || n == 0 || n < m) {
        return -1;
    }
    sort(arr.begin(), arr.end());
    int minDifference = INT_MAX;
    for (int i = 0; i + m - 1 < n; i++) {
        int difference = arr[i + m - 1] - arr[i];
        minDifference = min(minDifference, difference);
    }
    return minDifference;
}
int main() {
    vector<int> arr = {7, 3, 2, 4, 9, 12, 56};
    int m = 3;
    int n = arr.size();
    int result = findMinDifference(arr, n, m);
    if (result != -1) {
        cout << "Minimum difference is " << result << endl;
    } else {
        cout << "Not enough packets for the given number of students." << endl;
    }
    return 0;
}
```

Time Complexity : O(n)
Output:



8. **Merge Overlapping Intervals**

```
#include <bits/stdc++.h>
using namespace std;
vector<vector<int>> mergeIntervals(vector<vector<int>>& intervals) {
    if (intervals.empty()) return {};
    sort(intervals.begin(), intervals.end());
    vector<vector<int>> res;
    res.push_back(intervals[0]);
    for (int i = 1; i < intervals.size(); i++) {
        vector<int>& last = res.back();
        if (intervals[i][0] <= last[1]) {
            last[1] = max(last[1], intervals[i][1]);
        } else {
            res.push_back(intervals[i]);
        }
    }
    return res;
}
int main() {
    vector<vector<int>> arr1 = {{1, 3}, {2, 4}, {6, 8}, {9, 10}};
    vector<vector<int>> res1 = mergeIntervals(arr1);
    for (auto& interval : res1) {
        cout << "[" << interval[0] << ", " << interval[1] << "] ";
    }
    cout << endl;
    return 0;
}
```

Time Complexity : O(n log n)
Output:

```
/tmp/Ex3biX3Mxq.o
[1, 4] [6, 8] [9, 10]


=== Code Execution Successful ===
```

9. <u>A Boolean Matrix Question</u>

```
#include <iostream>
#include <vector>
using namespace std;
void modifyMatrix(vector<vector<int>>& mat) {
    int m = mat.size(), n = mat[0].size();
    vector<int> row(m, 0), col(n, 0);
    for (int i = 0; i < m; i++)
        for (int j = 0; j < n; j++)
            if (mat[i][j] == 1) {
                row[i] = 1;
                col[j] = 1;
            }
    for (int i = 0; i < m; i++)
        for (int j = 0; j < n; j++)
            if (row[i] == 1 || col[j] == 1)
                mat[i][j] = 1;
}
int main() {
    vector<vector<int>> mat1 = {{1, 0}, {0, 0}};
    vector<vector<int>> mat2 = {{0, 0, 0}, {0, 0, 1}};
    vector<vector<int>> mat3 = {{1, 0, 0, 1}, {0, 0, 1, 0}, {0, 0, 0, 0}};
    modifyMatrix(mat1);
    modifyMatrix(mat2);
    modifyMatrix(mat3);
    for (auto& row : mat1) {
        for (int cell : row) cout << cell << " ";
        cout << endl;
    }
    cout << endl;
    for (auto& row : mat2) {
        for (int cell : row) cout << cell << " ";
        cout << endl;
    }
    cout << endl;
    for (auto& row : mat3) {
        for (int cell : row) cout << cell << " ";
        cout << endl;
    }
    return 0;
}
```

Time Complexity : O(n*m)

Output:

```
/tmp/tmkVjLoK2Z.o
1 1
1 0

0 0 1
1 1 1

1 1 1 1
1 1 1 1
1 0 1 1
```

10. <u>Print a given matrix in spiral form</u>

```
#include <iostream>
#include <vector>
using namespace std;
void printSpiral(const vector<vector<int>>& mat) {
    int m = mat.size();
    if (m == 0) return;
    int n = mat[0].size();
    int t = 0, b = m - 1, l = 0, r = n - 1;
    while (t <= b && l <= r) {
        for (int i = l; i <= r; i++) cout << mat[t][i] << " ";
        t++;
        for (int i = t; i <= b; i++) cout << mat[i][r] << " ";
        r--;
        if (t <= b) {
            for (int i = r; i >= l; i--) cout << mat[b][i] << " ";
            b--;
        }
        if (l <= r) {
            for (int i = b; i >= t; i--) cout << mat[i][l] << " ";
            l++;
        }
    }
    cout << endl;
}

int main() {
    vector<vector<int>> mat1 = {{1, 2, 3, 4},
                                {5, 6, 7, 8},
                                {9, 10, 11, 12},
                                {13, 14, 15, 16}};
    vector<vector<int>> mat2 = {{1, 2, 3, 4, 5, 6},
                                {7, 8, 9, 10, 11, 12},
                                {13, 14, 15, 16, 17, 18}};
    printSpiral(mat1);
    printSpiral(mat2);
    return 0;
}
```

Time Complexity : O(n*m)

Output:

```
/tmp/rNYs6Cy1ak.o
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
```

11. **Check if given Parentheses expression is balanced or not**

```
#include <bits/stdc++.h>
using namespace std;
int main()
{
	stack<char> st;
	string str = "((()))()()";
	for(auto s : str)
	{
		if(!st.empty() && st.top() == '(' && s==')')
		{
			st.pop();
		}
		else
		{
			st.push(s);
		}
	}
	if(st.empty())
	{
		cout << "Balanced" <<endl;
	}
	else
	{
		cout << "Not Balanced";
	}
	return 0;
}
```

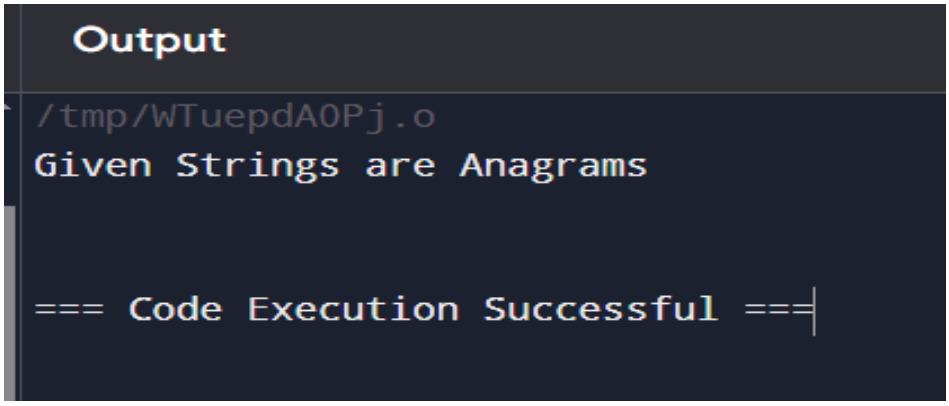
Time Complexity : O(n)
Output:

```
Output
/tmp/rwj9mi6kqX.o
Balanced


=== Code Execution Successful ===
```

12. **Check if two Strings are Anagrams of each other**

```
#include <bits/stdc++.h>
using namespace std;
bool findanagram(string s1, string s2)
{
        if(s1.size() != s2.size()) return false;
        unordered_map<char, int>m1;

        for(int i=0; i<s1.length(); i++)
        {
                m1[s1[i]]++;
                m1[s2[i]]--;
        }
         for (auto &pair : m1) {
    if (pair.second != 0) {
        return false;
    }
  }
  return true;
}
int main()
{
  string s1 = "allergy" ;
        string s2 = "allergy" ;
        if (findanagram(s1, s2)) {
    cout << "Given Strings are Anagrams" << endl;
  } else {
    cout << "Given Strings are not Anagrams" << endl;
  }
}
```

Time Complexity : O(n)
Output:



13. **Longest Palindromic Substring**

```
#include <iostream>
#include <string>
using namespace std;
string longestPalindrome(string s) {
    int n = s.size();
    if (n < 2) return s;
    int start = 0, maxLen = 1;
    for (int i = 0; i < n; ++i) {
        int l = i, r = i;
        while (r < n - 1 && s[r] == s[r + 1]) ++r;
        i = r;
        while (l > 0 && r < n - 1 && s[l - 1] == s[r + 1]) {
            --l;
            ++r;
        }
        if (r - l + 1 > maxLen) {
            start = l;
            maxLen = r - l + 1;
        }
    }
    return s.substr(start, maxLen);
}
int main() {
    cout << longestPalindrome("forgeeksskeegfor") << endl;
    cout << longestPalindrome("Geeks") << endl;
    cout << longestPalindrome("abc") << endl;
    cout << longestPalindrome("") << endl;
    return 0;
}
```

Time Complexity : O(n^2)
Output:

```
/tmp/mUaSGhS9wS.o
geeksskeeg
ee
a
```

14. Longest Common Prefix using Sorting

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;
string longestCommonPrefix(vector<string>& arr) {
    int n = arr.size();
    if (n == 0) return "-1";
    sort(arr.begin(), arr.end());
    string first = arr[0], last = arr[n - 1];
    int len = min(first.size(), last.size());
    int i = 0;
    while (i < len && first[i] == last[i]) i++;
    return i ? first.substr(0, i) : "-1";
}
int main() {
    vector<string> arr1 = {"geeksforgeeks", "geeks", "geek", "geezer"};
    vector<string> arr2 = {"hello", "world"};
    cout << longestCommonPrefix(arr1) << endl;
    cout << longestCommonPrefix(arr2) << endl;
    return 0;
}
```

Time Complexity : O(n*m)

Output:

```
/tmp/EBCRDKhwJB.o
gee
-1
```

15. <u>Delete middle element of a stack</u>

```
#include <iostream>
#include <stack>
using namespace std;
void deleteMiddle(stack<int>& s, int k) {
    if (k == 1) {
        s.pop();
        return;
    }
    int temp = s.top();
    s.pop();
    deleteMiddle(s, k - 1);
    s.push(temp);
}
void deleteMiddleElement(stack<int>& s) {
    int middle = s.size() / 2 + 1;
    deleteMiddle(s, middle);
}
int main() {
    stack<int> s;
    s.push(1);
    s.push(2);
    s.push(3);
    s.push(4);
    s.push(5);
    deleteMiddleElement(s);
    while (!s.empty()) {
        cout << s.top() << " ";
        s.pop();
    }
    return 0;
}
```

Time Complexity: O(N)
Output:

```
/tmp/yzdgDBIB1s.o
5 4 2 1
```

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

```
#include <iostream>
#include <vector>
#include <stack>
using namespace std;
vector<int> nextGreaterElement(vector<int>& arr) {
    int n = arr.size();
    vector<int> nge(n, -1);
    stack<int> s;
    for (int i = 0; i < n; i++) {
        while (!s.empty() && arr[s.top()] < arr[i]) {
            nge[s.top()] = arr[i];
            s.pop();
        }
        s.push(i);
    }
    return nge;
}
int main() {
    vector<int> arr = {4, 5, 2, 25};
    vector<int> result = nextGreaterElement(arr);
    for (int i = 0; i < arr.size(); i++) {
        cout << arr[i] << " -> " << result[i] << endl;
    }
    return 0;
}
```

Time Complexity: O(N)
Output:

```
/tmp/1HeQ5ZgE
4 -> 5
5 -> 25
2 -> 25
25 -> -1
```

17. **Print Right View of a Binary Tree**

```
#include <iostream>
#include <queue>
#include <vector>
using namespace std;
struct Node {
    int data;
    Node* left;
    Node* right;
    Node(int val) : data(val), left(nullptr), right(nullptr) {}
};
void rightView(Node* root) {
    if (root == nullptr) return;
    queue<Node*> q;
    q.push(root);
    while (!q.empty()) {
        int n = q.size();
        for (int i = 1; i <= n; i++) {
            Node* node = q.front();
            q.pop();
            if (i == n) {
                cout << node->data << " ";
            }
            if (node->left) q.push(node->left);
            if (node->right) q.push(node->right);
        }
    }
}

int main() {
    Node* root = new Node(1);
    root->left = new Node(2);
    root->right = new Node(3);
    root->left->left = new Node(4);
    root->left->right = new Node(5);
    root->right->right = new Node(6);
    root->left->right->left = new Node(7);
    root->left->right->right = new Node(8);
    rightView(root);
    return 0;
}
```

Time Complexity: O(N)
Output:

```
/tmp/pxryhildtl.o
1 3 6 8
```

18. **Maximum Depth or Height of Binary Tree**

```
#include <iostream>
#include <algorithm>
using namespace std;
struct Node {
    int data;
    Node* left;
    Node* right;
    Node(int val) : data(val), left(nullptr), right(nullptr) {}
};
int maxDepth(Node* root) {
    if (root == nullptr) {
        return 0;
    }
    int leftDepth = maxDepth(root->left);
    int rightDepth = maxDepth(root->right);
    return max(leftDepth, rightDepth) + 1;
}

int main() {
    Node* root = new Node(1);
    root->left = new Node(2);
    root->right = new Node(3);
    root->left->left = new Node(4);
    root->left->right = new Node(5);
    root->right->right = new Node(6);
    cout << "Maximum Depth of the Binary Tree: " << maxDepth(root) << endl;
    return 0;
}
```

Time Complexity: O(N)
Output:

```
/tmp/byHn9HASQH.o
Maximum Depth of the Binary Tree: 3
```