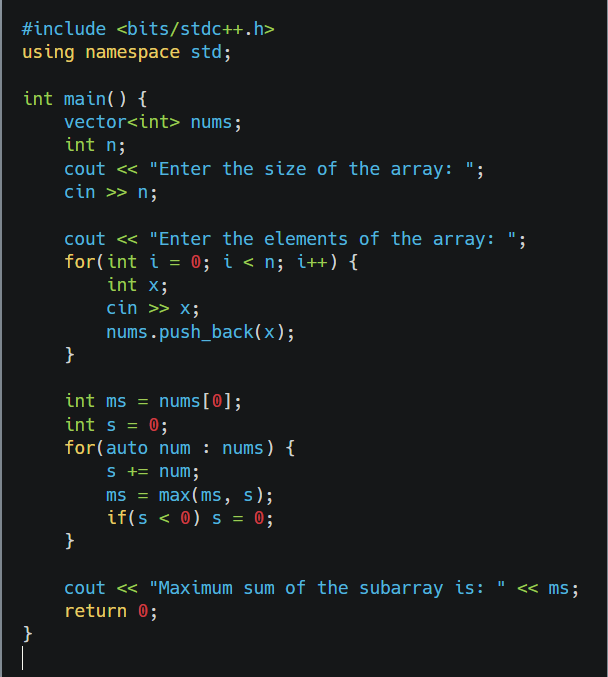
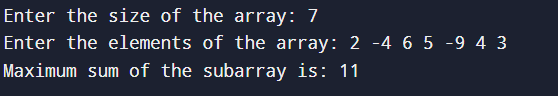
DATE: 10th November 2024

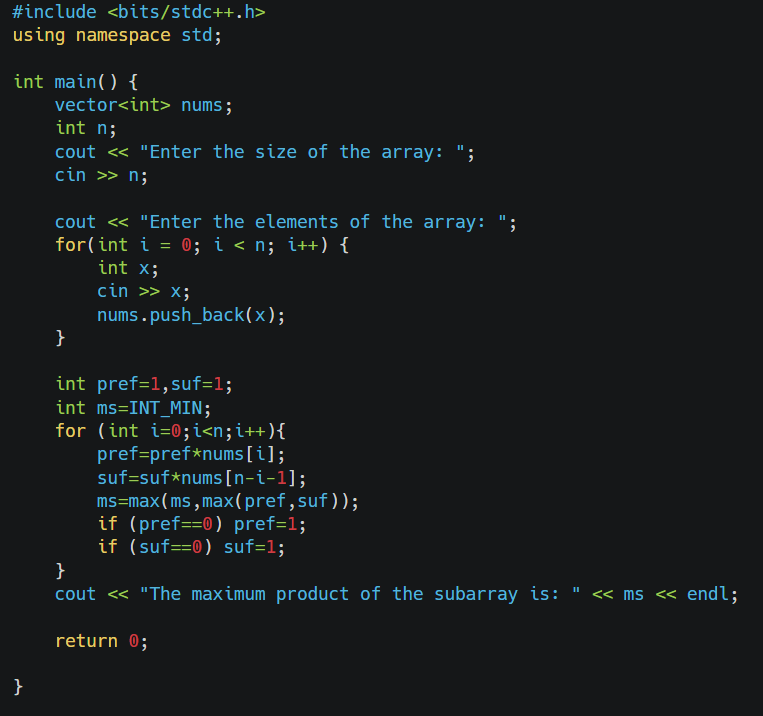
1. Kaden’s Algorithm  
     
   

TIME COMPLEXITY: O(N)

Output:

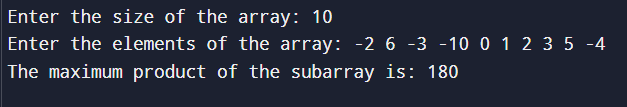


1. Maximum Product Subarray



TIME COMPLEXITY: O(N)

Output:

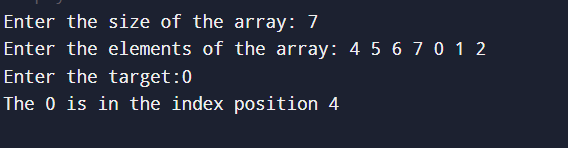


1. Search in a sorted and rotated Array

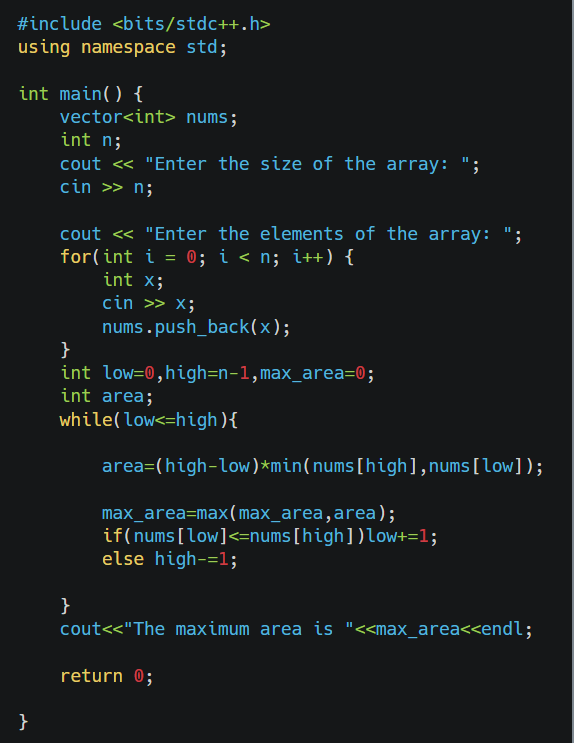


TIME COMPLEXITY: O(log(N))

Output:

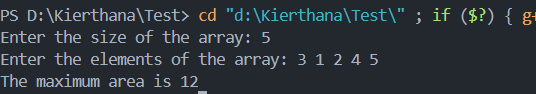


1. Container with Most Water

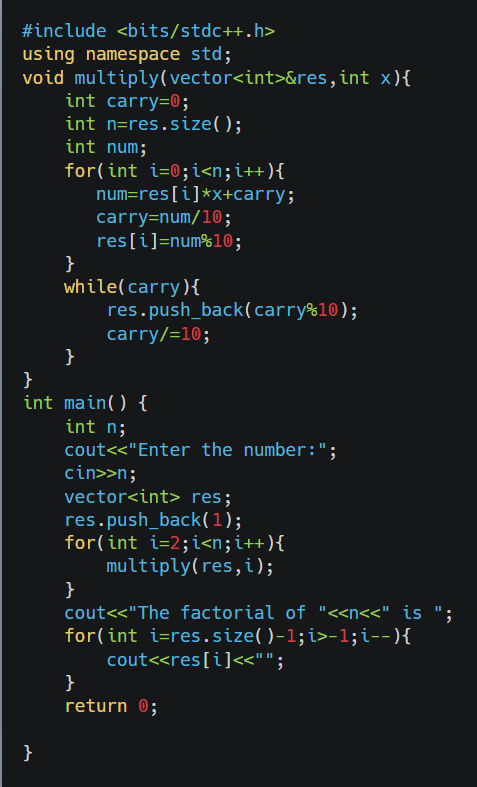


TIME COMPLEXITY: O(n)

Output:

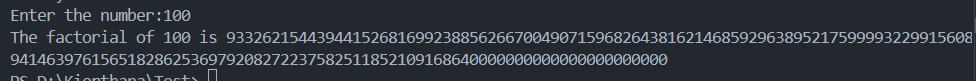


1. Find the Factorial of a large number

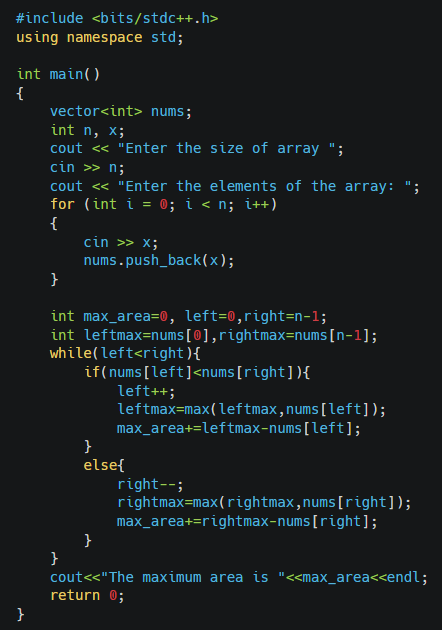


Time Complexity: O(N Log N!)

Output:

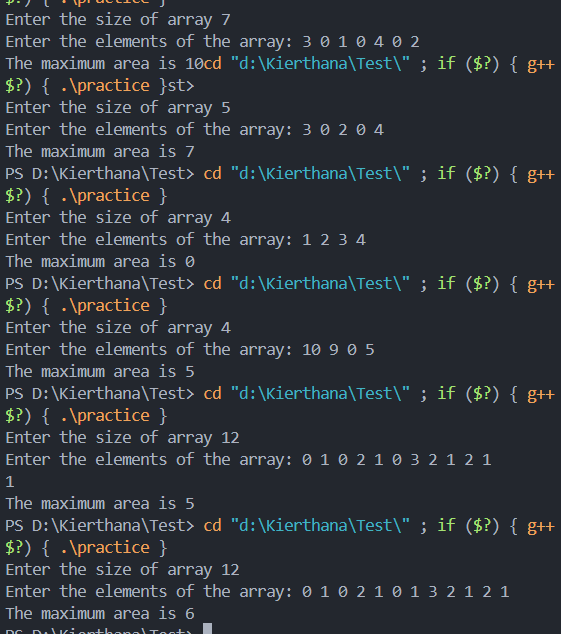


1. Trapping Rainwater

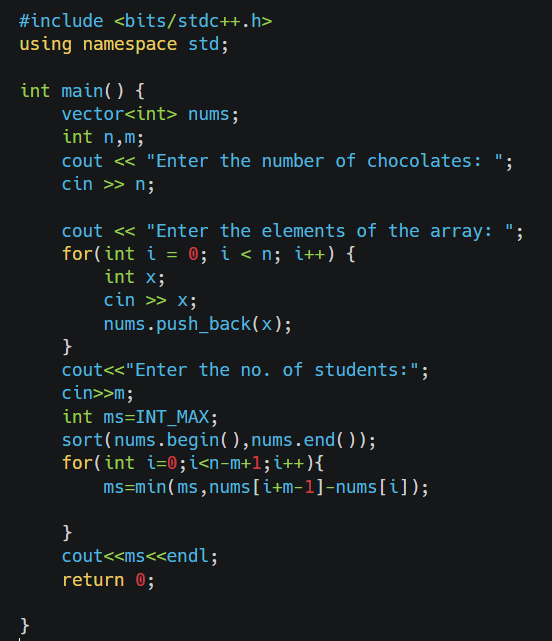


Time Complexity(n)

Output:

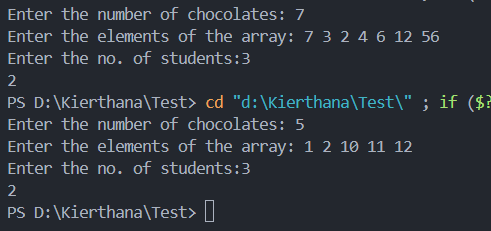


1. Chocolate Distribution Problem

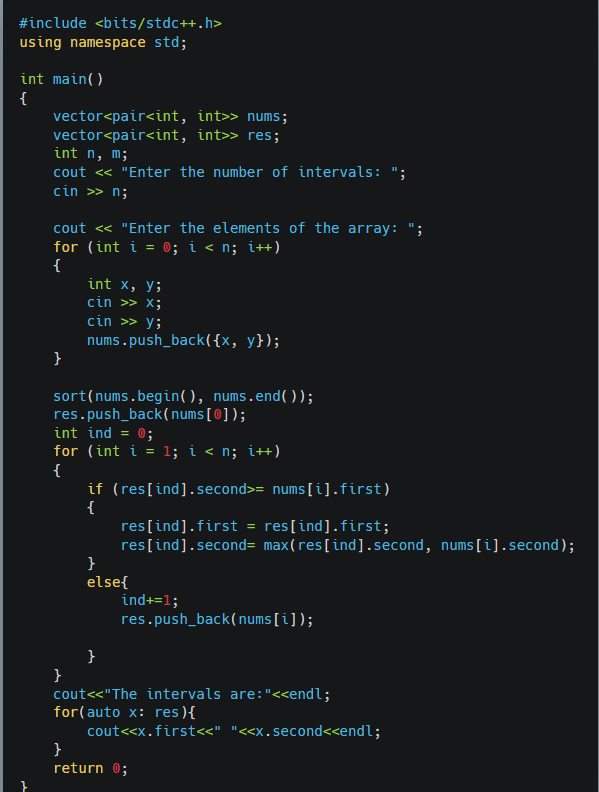


TIME COMPLEXITY: O(N LogN) [sliding Window]

Output:

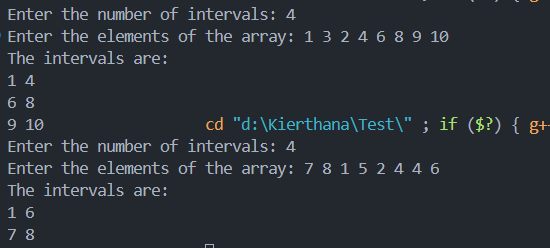


1. Merge Overlapping Intervals



TIME COMPLEXITY: O(N LogN)

Output:

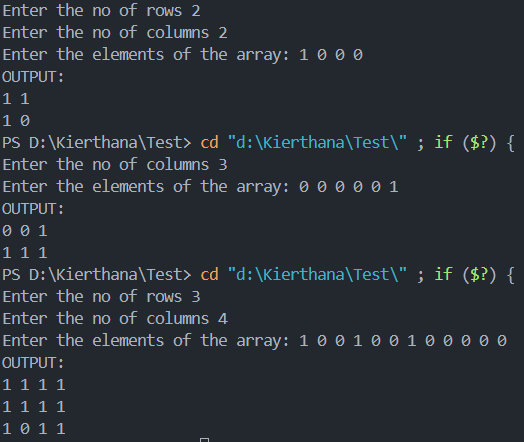


1. A Boolean Matrix Question

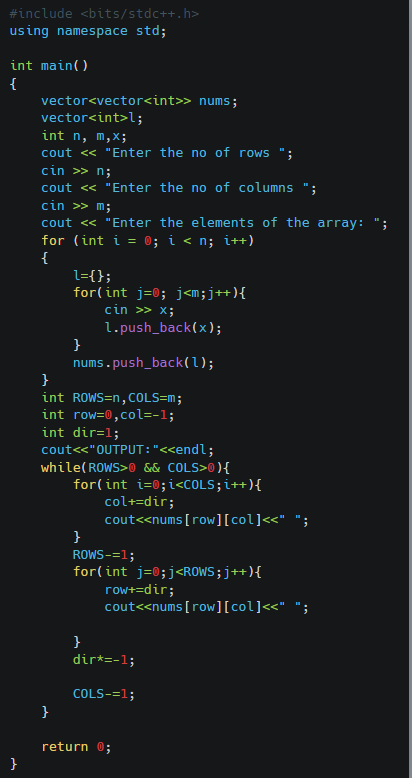


TIME COMPLEXITY: O(NM)

Output:

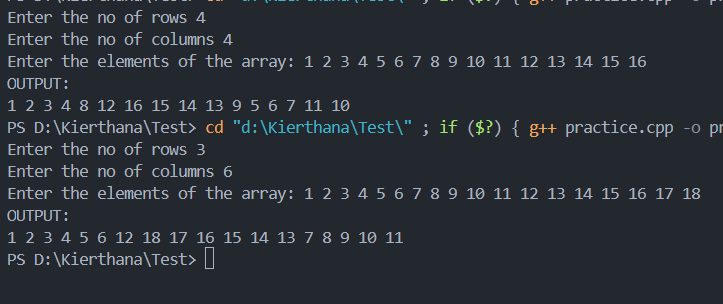


1. Print a given matrix in spiral form



TIME COMPLEXITY: O(N\*M)

Output:



1. Check if given Parentheses expression is balanced or not

#include <bits/stdc++.h>

using namespace std;

bool check(string s){

   vector<int> stack;

   for(auto ch:s){

        if (ch == '(') {

            stack.push\_back(ch);

        }

        else {

            if (stack.empty()){

                return false;

            }

            stack.pop\_back();

        }

    }

    return stack.empty();

}

int main()

{

    string s;

    cout<<"Enter the string"<<" ";

    cin>>s;

    if(check(s)) cout<<"Balanced"<<endl;

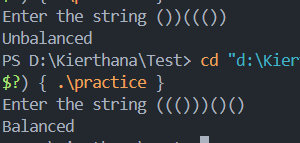
    else cout<<"Unbalanced"<<endl;

    return 0;

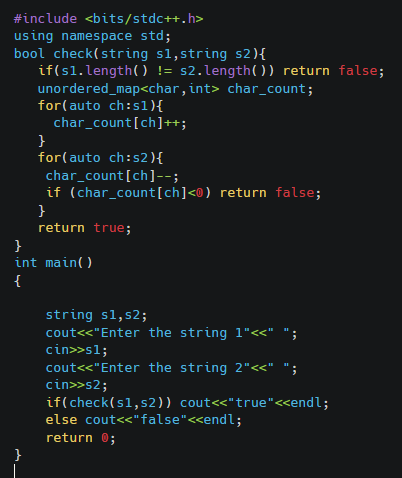
}

TIME COMPLEXITY: O(N)

Output:

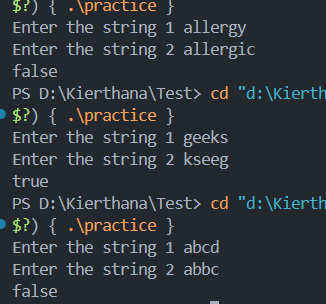


1. Check if two Strings are Anagrams of each other

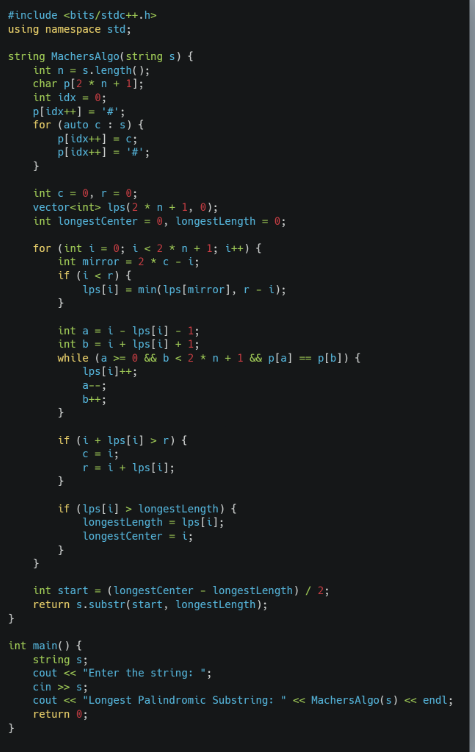


TIME COMPLEXITY: O(N)

Output:

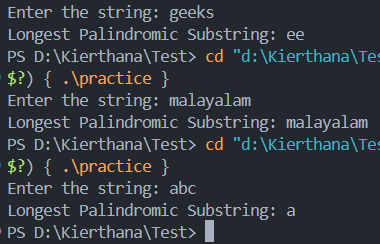


1. Longest Palindromic Substring



Time Complexity:O(N)

Output:

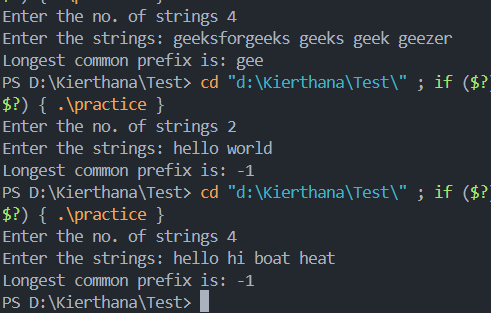


1. Longest Common Prefix using Sorting

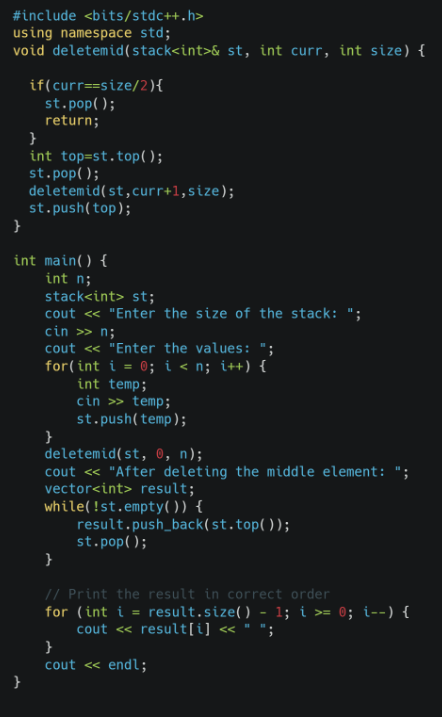


TIME COMPLEXITY: O(N LogN)

Output:

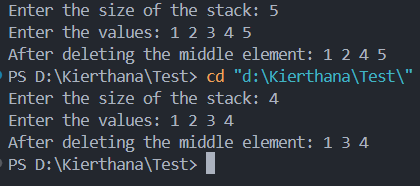


1. Delete middle element of a stack

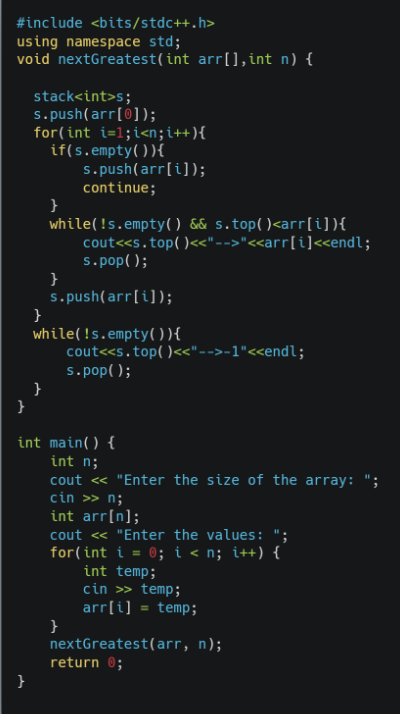


TIME COPLEXITY: O(N)

Output:

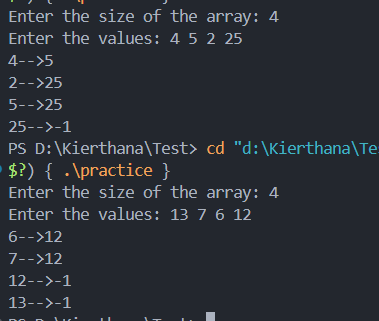


1. Next Greater Element (NGE) for every element in given Array



TIME COMPLEXITY: O(N)

Output:

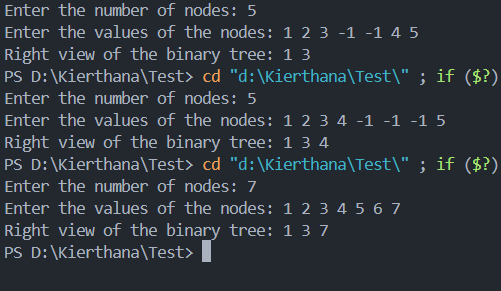


1. Print Right View of a Binary Tree
2. #include <bits/stdc++.h>
3. using namespace std;
4. class BinaryTree {
5. private:
6. struct Node {
7. int data;
8. Node\* left;
9. Node\* right;
10. Node(int val) : data(val), left(nullptr), right(nullptr) {}
11. };
12. Node\* root;
13. public:
14. BinaryTree() : root(nullptr) {}
15. void createTree(const vector<int>& arr) {
16. if (arr.empty()) return;
18. root = new Node(arr[0]);
19. queue<Node\*> q;
20. q.push(root);
22. int i = 1;
23. while (!q.empty() && i < arr.size()) {
24. Node\* current = q.front();
25. q.pop();
26. if (i < arr.size() && arr[i] != -1) {
27. current->left = new Node(arr[i]);
28. q.push(current->left);
29. }
30. i++;
31. if (i < arr.size() && arr[i] != -1) {
32. current->right = new Node(arr[i]);
33. q.push(current->right);
34. }
35. i++;
36. }
37. }
38. void rightView() {
39. if (root == nullptr) return;
40. queue<Node\*> q;
41. q.push(root);
42. while (!q.empty()) {
43. int n = q.size();
44. for (int i = 1; i <= n; i++) {
45. Node\* current = q.front();
46. q.pop();
48. if (i == n) {
49. cout << current->data << " ";
50. }
52. if (current->left) {
53. q.push(current->left);
54. }
55. if (current->right) {
56. q.push(current->right);
57. }
58. }
59. }
60. cout << endl;
61. }
63. };
64. int main() {
65. BinaryTree tree;
66. int n;
67. cout << "Enter the number of nodes: ";
68. cin >> n;
69. vector<int> arr;
70. cout << "Enter the values of the nodes: ";
71. for (int i = 0; i < n; i++) {
72. int x;
73. cin >> x;
74. arr.push\_back(x);
75. }
76. tree.createTree(arr);
77. cout << "Right view of the binary tree: ";
78. tree.rightView();

81. return 0;
82. }

TIME COMPLEXITY:O(N)

Output:



1. Maximum Depth or Height of Binary Tree
2. #include <bits/stdc++.h>
3. using namespace std;
4. class BinaryTree {
5. private:
6. struct Node {
7. int data;
8. Node\* left;
9. Node\* right;
10. Node(int val) : data(val), left(nullptr), right(nullptr) {}
11. };
12. Node\* root;
13. public:
14. BinaryTree() : root(nullptr) {}
15. void createTree(const vector<int>& arr) {
16. if (arr.empty()) return;
18. root = new Node(arr[0]);
19. queue<Node\*> q;
20. q.push(root);
22. int i = 1;
23. while (!q.empty() && i < arr.size()) {
24. Node\* current = q.front();
25. q.pop();
26. if (i < arr.size() && arr[i] != -1) {
27. current->left = new Node(arr[i]);
28. q.push(current->left);
29. }
30. i++;
31. if (i < arr.size() && arr[i] != -1) {
32. current->right = new Node(arr[i]);
33. q.push(current->right);
34. }
35. i++;
36. }
37. }
38. int maxDepth(Node\* node) {
39. if (node == nullptr) {
40. return 0;
41. } else {
43. int leftDepth = maxDepth(node->left);
44. int rightDepth = maxDepth(node->right);
45. return max(leftDepth, rightDepth) + 1;
46. }
47. }
49. int getMaxDepth() {
50. return maxDepth(root);
51. }
53. };
54. int main() {
55. BinaryTree tree;
56. int n;
57. cout << "Enter the number of nodes: ";
58. cin >> n;
59. vector<int> arr;
60. cout << "Enter the values of the nodes: ";
61. for (int i = 0; i < n; i++) {
62. int x;
63. cin >> x;
64. arr.push\_back(x);
65. }
66. tree.createTree(arr);
67. cout << "Maximum depth of the binary tree: " << tree.getMaxDepth() << endl;

70. return 0;
71. }

TIME COMPLEXITY: O(N)

Output:

