Balaji P V

AI&DS

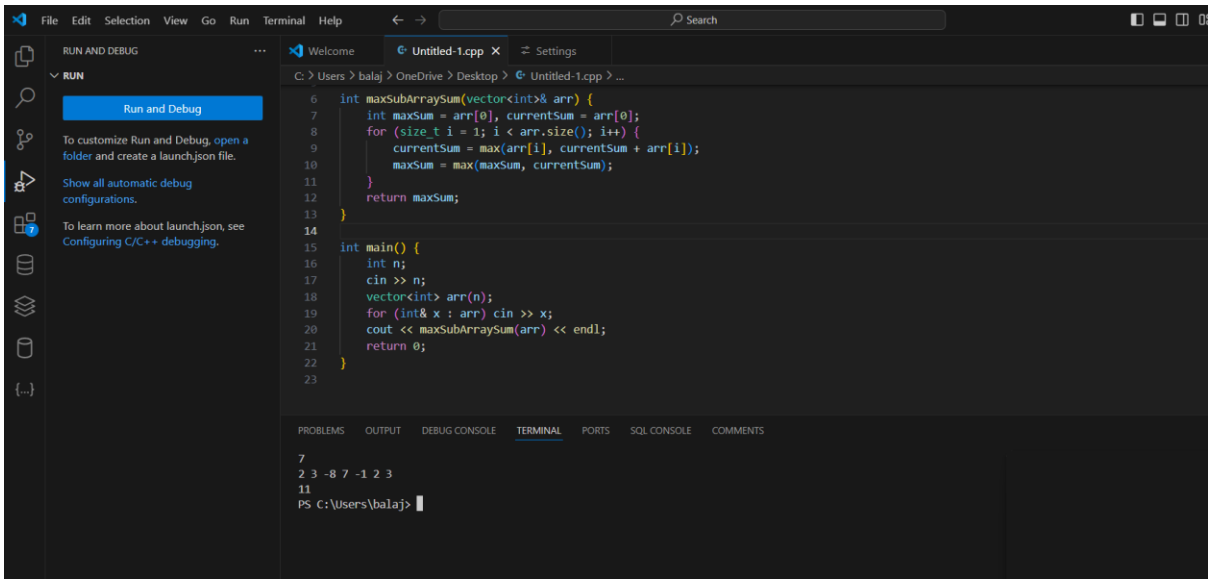
Coding Practise Problems

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.

Code:

```
#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

int maxSubArraySum(vector<int>& arr) {

   int maxSum = arr[0], currentSum = arr[0];

   for (size_t i = 1; i < arr.size(); i++) {

      currentSum = max(arr[i], currentSum + arr[i]);

      maxSum = max(maxSum, currentSum);

   }

   return maxSum;

}


int main() {

   int n;

   cin >> n;

   vector<int> arr(n);

   for (int& x : arr) cin >> x;

   cout << maxSubArraySum(arr) << endl;

   return 0;

}
```

Output:



Time complexity: O(n)

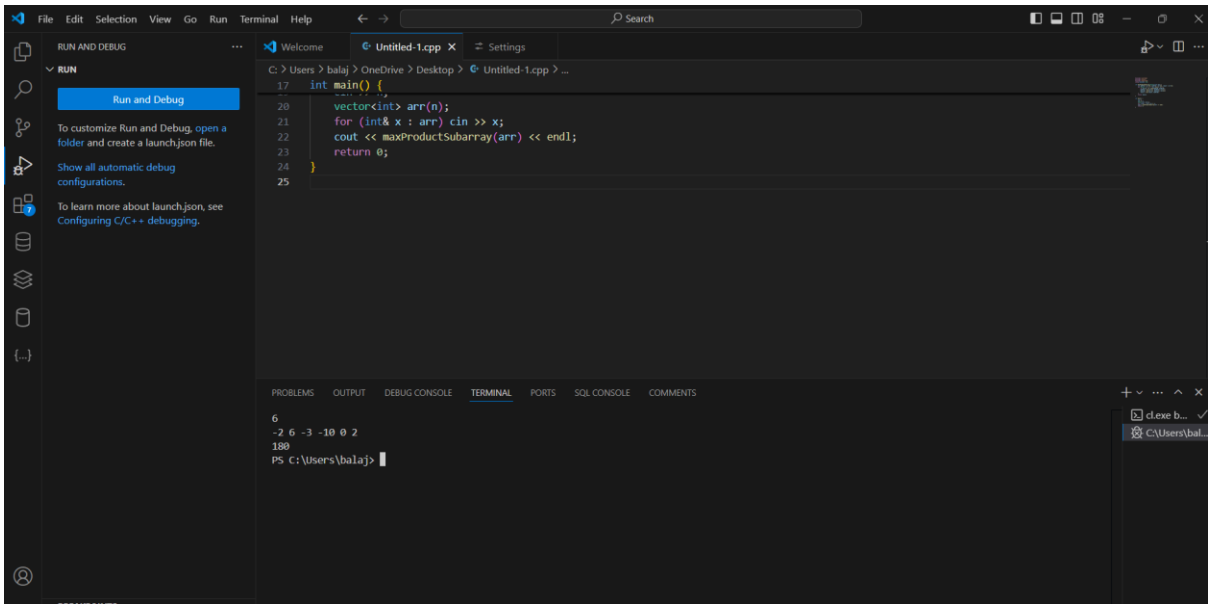
2) Maximum Product Subarray Given an integer array, the task is to find the maximum product of any subarray.

Code:

```
#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;


int maxProductSubarray(vector<int>& arr) {

   int maxProd = arr[0], minProd = arr[0], result = arr[0];

   for (size_t i = 1; i < arr.size(); i++) {

      if (arr[i] < 0) swap(maxProd, minProd);

      maxProd = max(arr[i], maxProd * arr[i]);

      minProd = min(arr[i], minProd * arr[i]);

      result = max(result, maxProd);

   }

   return result;

}
```

```
int main() {
    int n;
    cin >> n;
    vector<int> arr(n);
    for (int& x : arr) cin >> x;
    cout << maxProductSubarray(arr) << endl;
    return 0;
}
```

Output:



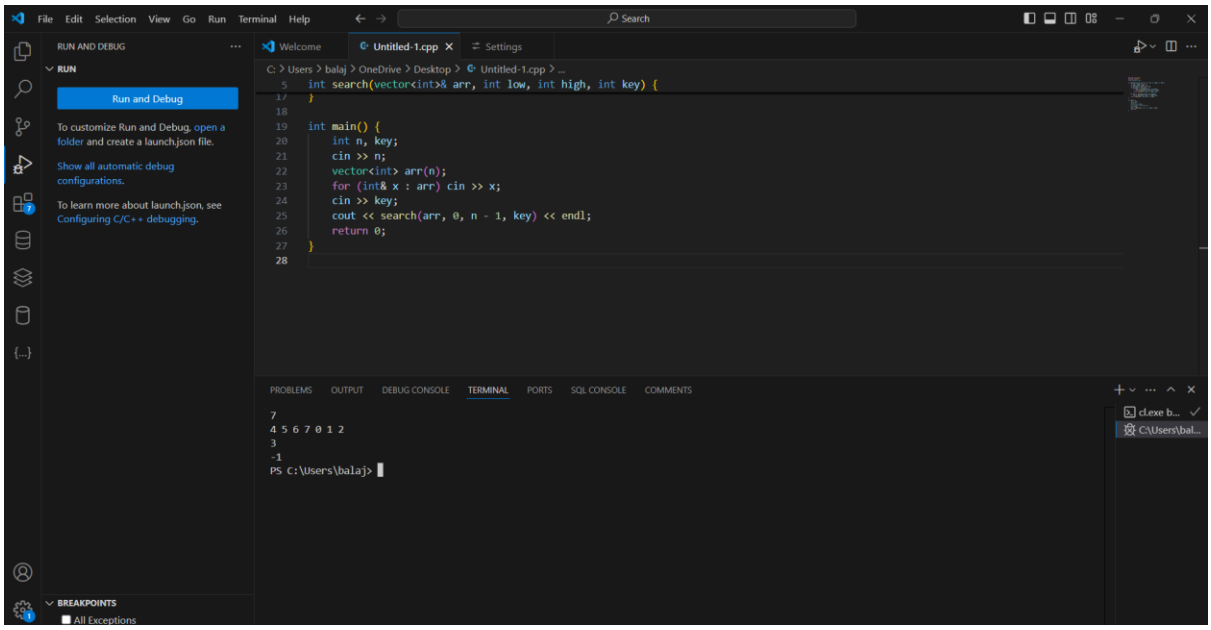
Time complexity: O(n)

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.

Code:

```
#include <iostream>

#include <vector>

using namespace std;


int search(vector<int>& arr, int low, int high, int key) {

   if (low > high) return -1;

   int mid = (low + high) / 2;

   if (arr[mid] == key) return mid;

   if (arr[low] <= arr[mid]) {

      if (key >= arr[low] && key <= arr[mid])

         return search(arr, low, mid - 1, key);

      return search(arr, mid + 1, high, key);

   }

   if (key >= arr[mid] && key <= arr[high])

      return search(arr, mid + 1, high, key);

   return search(arr, low, mid - 1, key);

}


int main() {

   int n, key;

   cin >> n;

   vector<int> arr(n);

   for (int& x : arr) cin >> x;

   cin >> key;

   cout << search(arr, 0, n - 1, key) << endl;

   return 0;

}
```

Output:



Time complexity: O(log n)

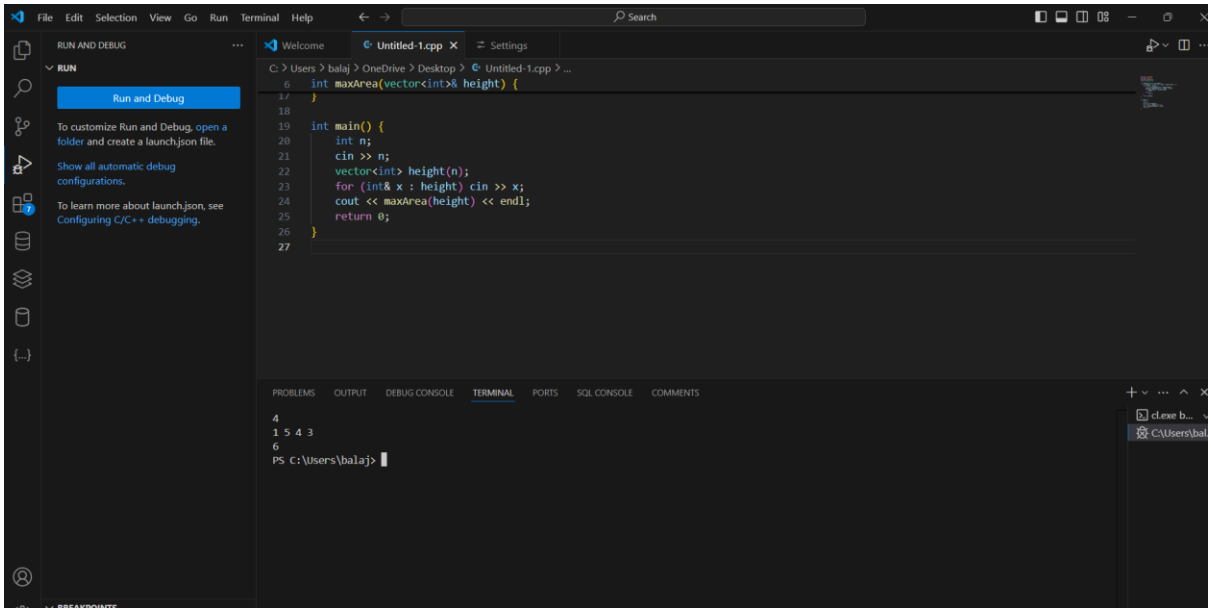
4) Container with Most Water

Code:

```
#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;


int maxArea(vector<int>& height) {

   int maxArea = 0, left = 0, right = height.size() - 1;

   while (left < right) {

      int h = min(height[left], height[right]);

      maxArea = max(maxArea, h * (right - left));

      if (height[left] < height[right])

         left++;

      else

         right--;
```

```
    }
    return maxArea;
}

int main() {
    int n;
    cin >> n;
    vector<int> height(n);
    for (int& x : height) cin >> x;
    cout << maxArea(height) << endl;
    return 0;
}
```

Output:



Time complexity: O(n)

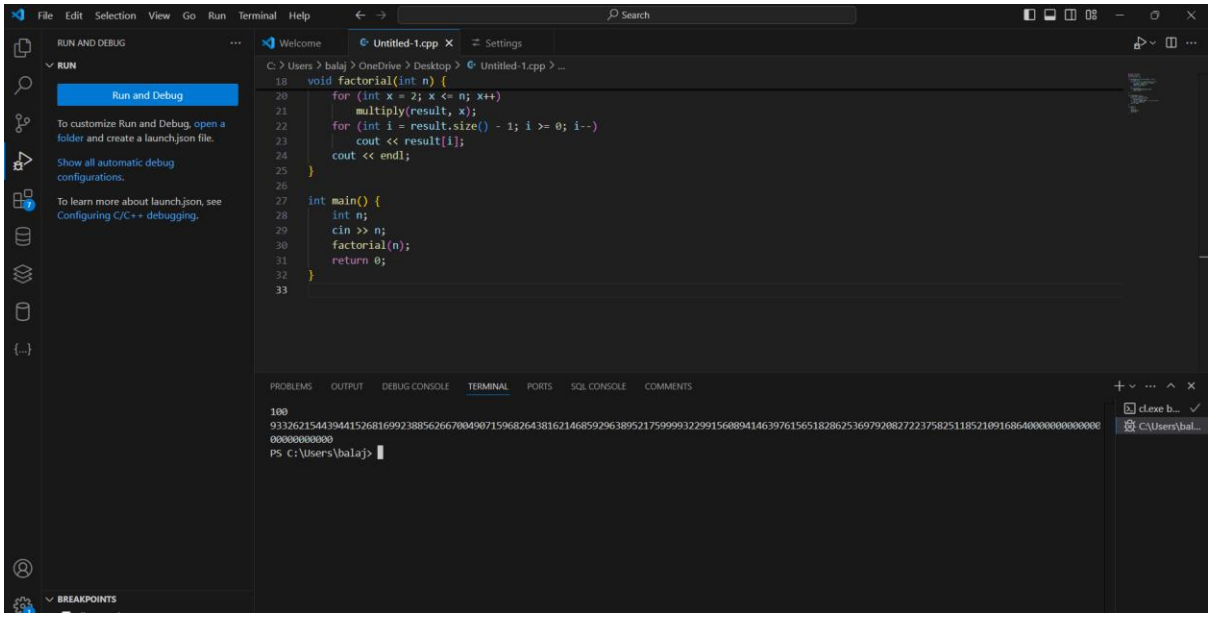
5) Find the Factorial of a large number

Code:

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

void multiply(vector<int>& result, int x) {
    int carry = 0;
    for (int i = 0; i < result.size(); i++) {
        int prod = result[i] * x + carry;
        result[i] = prod % 10;
        carry = prod / 10;
    }
    while (carry) {
        result.push_back(carry % 10);
        carry /= 10;
    }
}

void factorial(int n) {
    vector<int> result(1, 1);
    for (int x = 2; x <= n; x++)
        multiply(result, x);
    for (int i = result.size() - 1; i >= 0; i--)
        cout << result[i];
    cout << endl;
}

int main() {
    int n;
    cin >> n;
```

```
    factorial(n);
    return 0;
}
```

Output:



Time complexity: O(n)

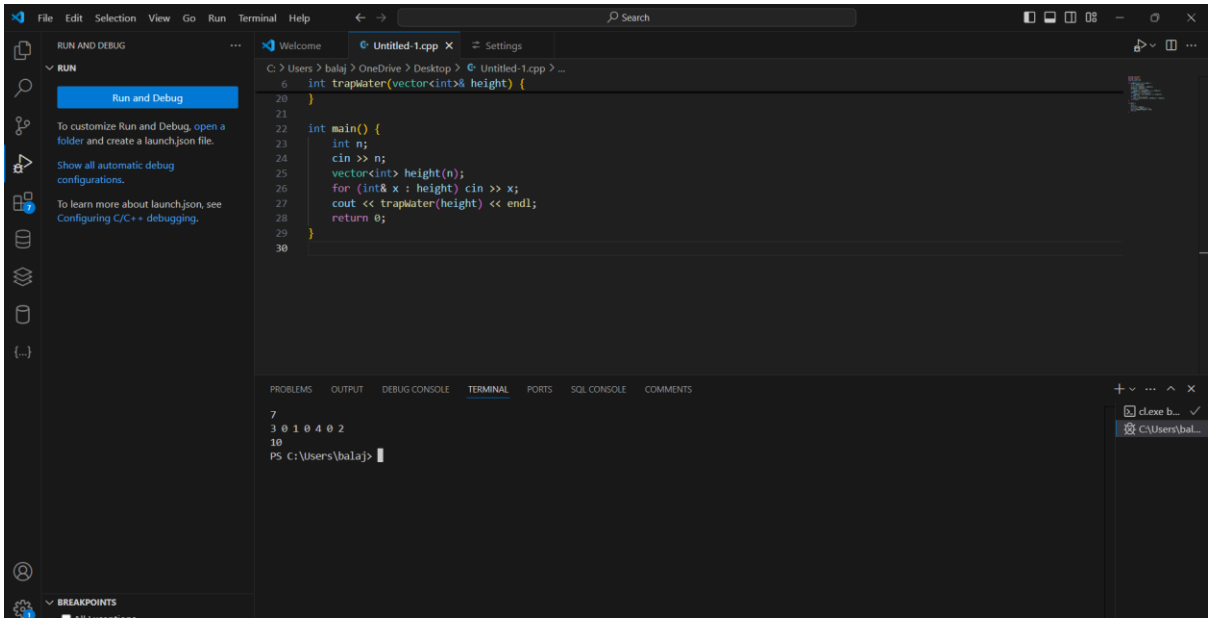
6)Trapping Rainwater Problem states that given an array of n non-negative integers arr[] representing an elevation map where the width of each bar is 1, compute how much water it can trap after rain.

Code:

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

int trapWater(vector<int>& height) {
    int n = height.size();
    if (n == 0) return 0;
    vector<int> leftMax(n), rightMax(n);
    leftMax[0] = height[0];
```

```
    for (int i = 1; i < n; i++)
        leftMax[i] = max(leftMax[i - 1], height[i]);
    rightMax[n - 1] = height[n - 1];
    for (int i = n - 2; i >= 0; i--)
        rightMax[i] = max(rightMax[i + 1], height[i]);
    int water = 0;
    for (int i = 0; i < n; i++)
        water += min(leftMax[i], rightMax[i]) - height[i];
    return water;
}

int main() {
    int n;
    cin >> n;
    vector<int> height(n);
    for (int& x : height) cin >> x;
    cout << trapWater(height) << endl;
    return 0;
}
```

Output:



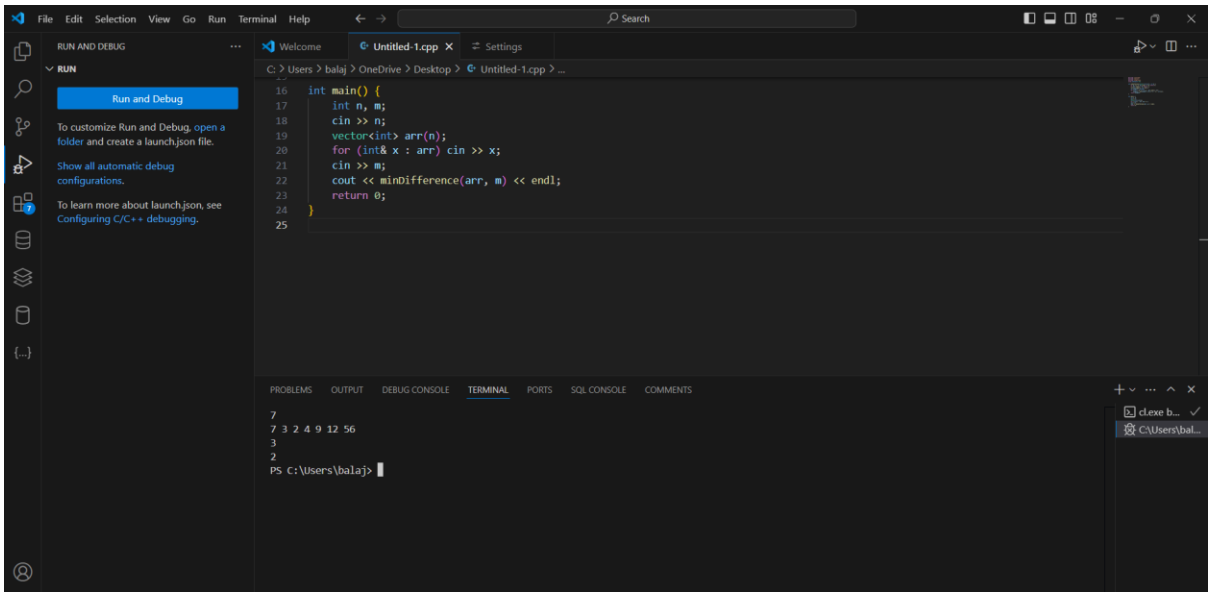
Time complexity: O(n)

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.

Code:

```
#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;


int minDifference(vector<int>& arr, int m) {

   if (m == 0 || arr.size() == 0) return 0;

   if (arr.size() < m) return -1;

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

   int minDiff = INT_MAX;

   for (int i = 0; i + m - 1 < arr.size(); i++)

      minDiff = min(minDiff, arr[i + m - 1] - arr[i]);

   return minDiff;

}


int main() {

   int n, m;

   cin >> n;

   vector<int> arr(n);

   for (int& x : arr) cin >> x;

   cin >> m;

   cout << minDifference(arr, m) << endl;

   return 0;

}
```

Output:



Time complexity: O(n log n)

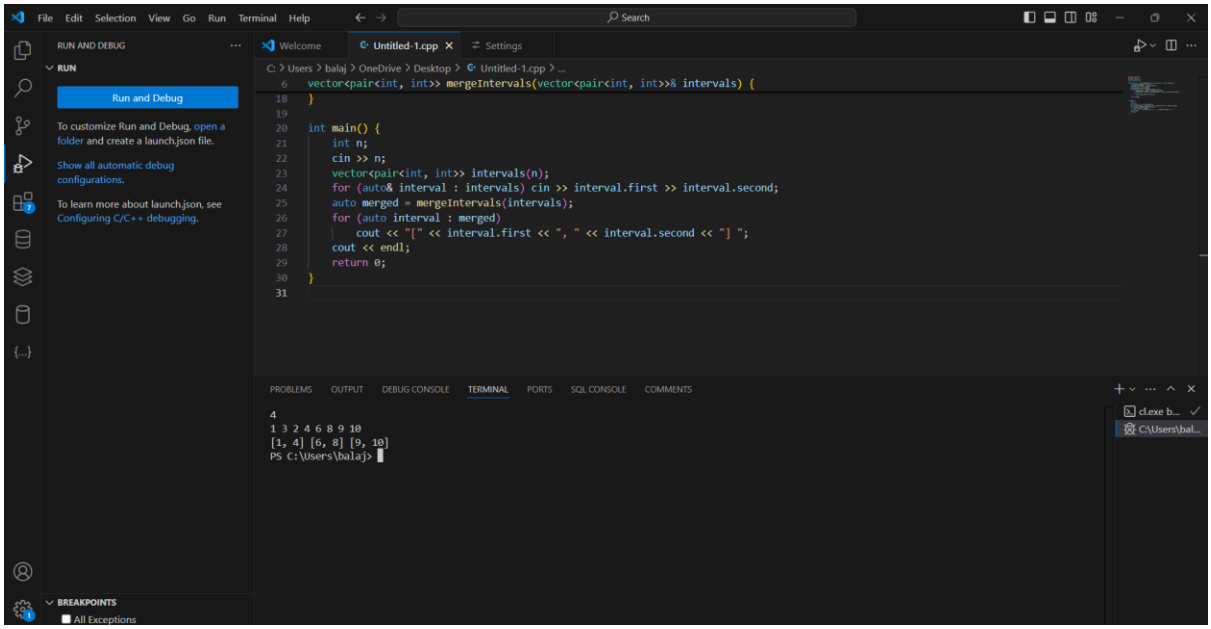
8) Merge Overlapping Intervals Given an array of time intervals where arr[i] = [starti, endi], the task is to merge all the overlapping intervals into one and output the result which should have only mutually exclusive intervals.

Code:

```
#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;


vector<pair<int, int>> mergeIntervals(vector<pair<int, int>>& intervals) {

    if (intervals.empty()) return {};

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

    vector<pair<int, int>> merged;

    merged.push_back(intervals[0]);

    for (size_t i = 1; i < intervals.size(); i++) {

        if (merged.back().second >= intervals[i].first)

            merged.back().second = max(merged.back().second, intervals[i].second);

        else
```

```
        merged.push_back(intervals[i]);
    }
    return merged;
}

int main() {
    int n;
    cin >> n;
    vector<pair<int, int>> intervals(n);
    for (auto& interval : intervals) cin >> interval.first >> interval.second;
    auto merged = mergeIntervals(intervals);
    for (auto interval : merged)
        cout << "[" << interval.first << ", " << interval.second << "] ";
    cout << endl;
    return 0;
}
```

Output:



Time complexity: O(n log n)

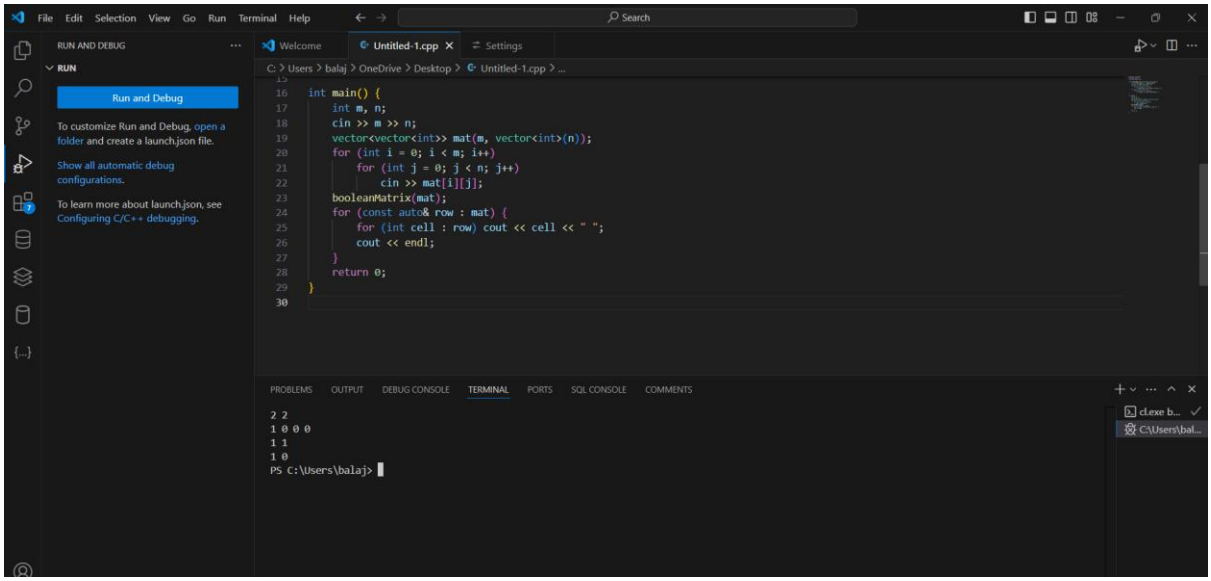
9) A Boolean Matrix Question Given a boolean matrix mat[M][N] of size M X N, modify it such that if a matrix cell mat[i][j] is 1 (or true) then make all the cells of ith row and jth column as 1.

Code:

```
#include <iostream>

#include <vector>

using namespace std;


void booleanMatrix(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] = col[j] = 1;

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

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

         if (row[i] || col[j]) mat[i][j] = 1;

}


int main() {

   int m, n;

   cin >> m >> n;

   vector<vector<int>> mat(m, vector<int>(n));

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

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

         cin >> mat[i][j];

   booleanMatrix(mat);

   for (const auto& row : mat) {

      for (int cell : row) cout << cell << " ";

      cout << endl;

   }

   return 0;
```

}

Output:



Time complexity: O(m * n)

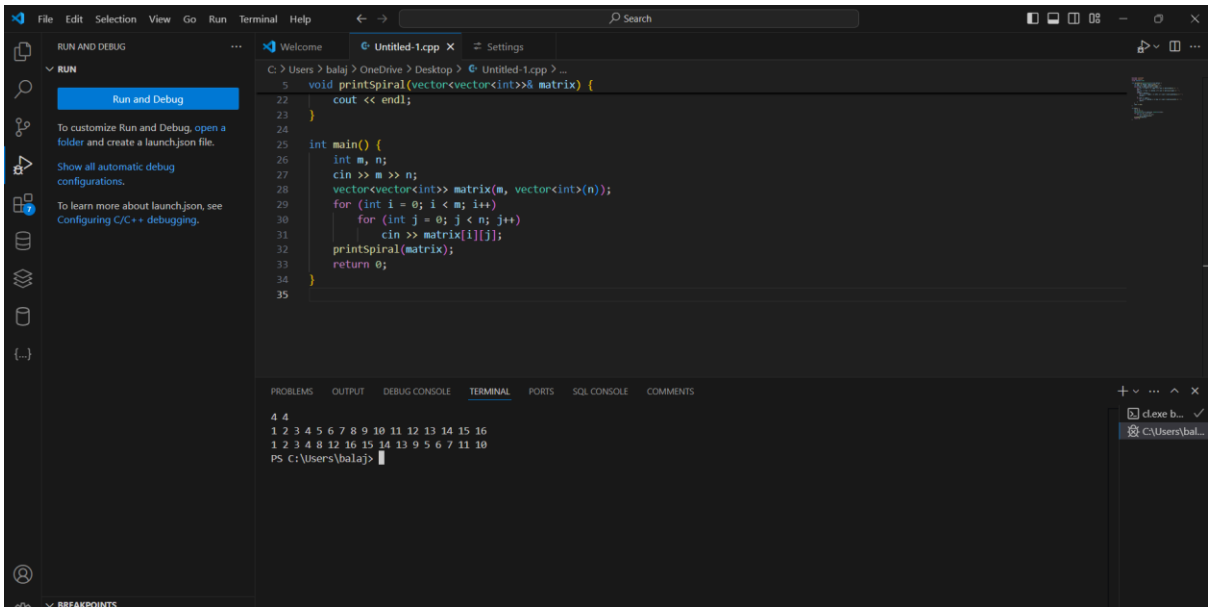
10) Print a given matrix in spiral form Given an m x n matrix, the task is to print all elements of the matrix in spiral form.

Code:

```
#include <iostream>

#include <vector>

using namespace std;


void printSpiral(vector<vector<int>>& matrix) {

   int top = 0, bottom = matrix.size() - 1;

   int left = 0, right = matrix[0].size() - 1;

   while (top <= bottom && left <= right) {

      for (int i = left; i <= right; i++) cout << matrix[top][i] << " ";

      top++;

      for (int i = top; i <= bottom; i++) cout << matrix[i][right] << " ";

      right--;
```

```
        if (top <= bottom) {
            for (int i = right; i >= left; i--) cout << matrix[bottom][i] << " ";
            bottom--;
        }
        if (left <= right) {
            for (int i = bottom; i >= top; i--) cout << matrix[i][left] << " ";
            left++;
        }
    }
    cout << endl;
}

int main() {
    int m, n;
    cin >> m >> n;
    vector<vector<int>> matrix(m, vector<int>(n));
    for (int i = 0; i < m; i++)
        for (int j = 0; j < n; j++)
            cin >> matrix[i][j];
    printSpiral(matrix);
    return 0;
}
```

Output:



Time complexity: O(m * n)

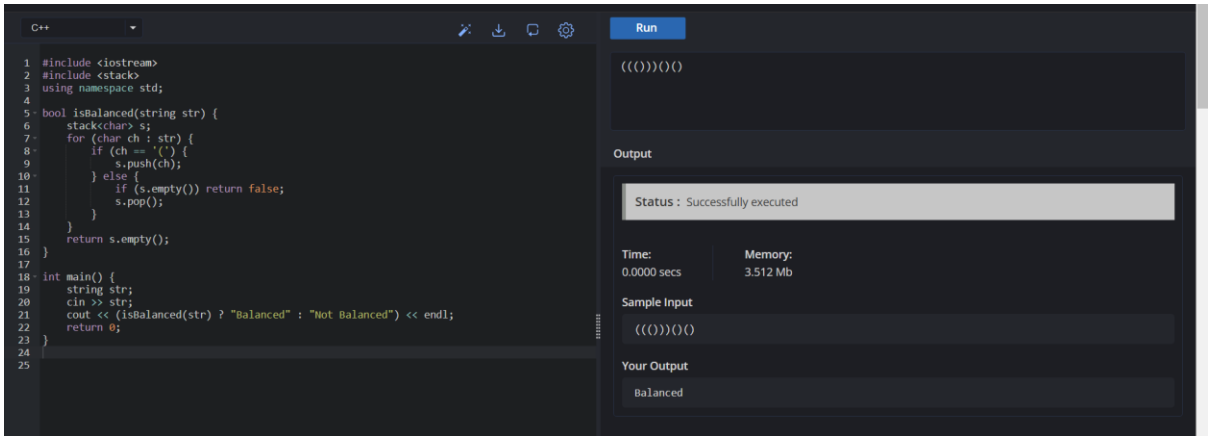
13) Check if given Parentheses expression is balanced or not Given a string str of length N, consisting of (and) only, the task is to check whether it is balanced or not.

Code:

```
#include <iostream>

#include <stack>

using namespace std;


bool isBalanced(string str) {

   stack<char> s;

   for (char ch : str) {

      if (ch == '(') {

         s.push(ch);

      } else {

         if (s.empty()) return false;

         s.pop();

      }

   }

   return s.empty();
```

```
}

int main() {
    string str;
    cin >> str;
    cout << (isBalanced(str) ? "Balanced" : "Not Balanced") << endl;
    return 0;
}
```

Output:



Time complexity: O(N)

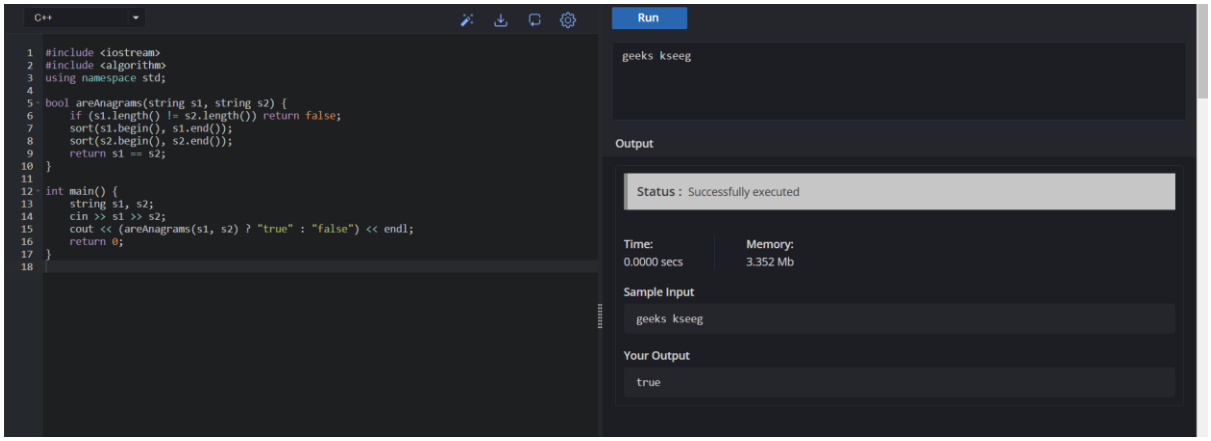
14) Check if two Strings are Anagrams of each other Given two strings s1 and s2 consisting of lowercase characters, the task is to check whether the two given strings are anagrams of each other or not. An anagram of a string is another string that contains the same characters, only the order of characters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

Code:

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

bool areAnagrams(string s1, string s2) {
    if (s1.length() != s2.length()) return false;
```

```
    sort(s1.begin(), s1.end());
    sort(s2.begin(), s2.end());
    return s1 == s2;
}

int main() {
    string s1, s2;
    cin >> s1 >> s2;
    cout << (areAnagrams(s1, s2) ? "true" : "false") << endl;
    return 0;
}
```

Output:



Time complexity: O(N log N)

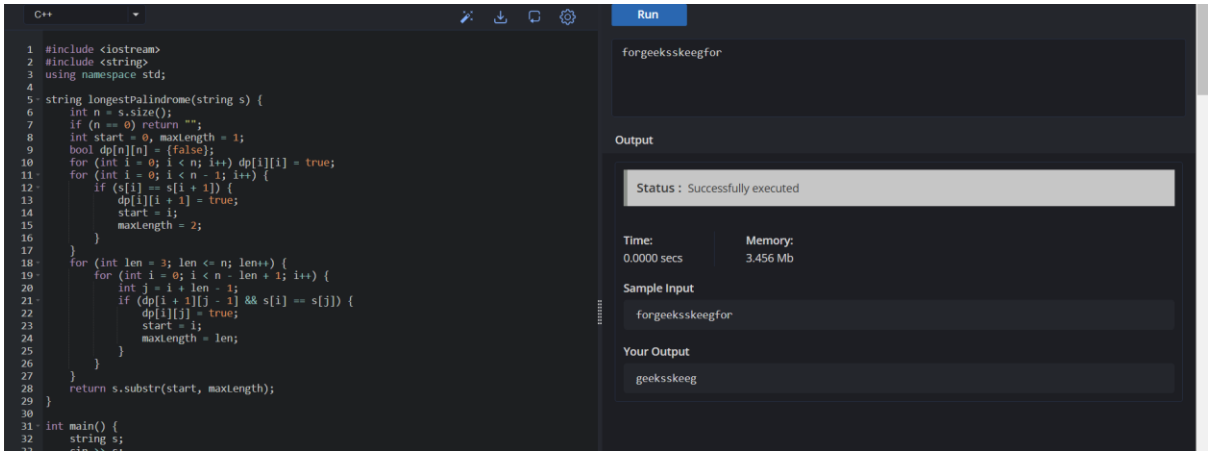
15) Longest Palindromic Substring Given a string str, the task is to find the longest substring which is a palindrome. If there are multiple answers, then return the first appearing substring.

Code:

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

string longestPalindrome(string s) {
```

```
    int n = s.size();
    if (n == 0) return "";
    int start = 0, maxLength = 1;
    bool dp[n][n] = {false};
    for (int i = 0; i < n; i++) dp[i][i] = true;
    for (int i = 0; i < n - 1; i++) {
        if (s[i] == s[i + 1]) {
            dp[i][i + 1] = true;
            start = i;
            maxLength = 2;
        }
    }
    for (int len = 3; len <= n; len++) {
        for (int i = 0; i < n - len + 1; i++) {
            int j = i + len - 1;
            if (dp[i + 1][j - 1] && s[i] == s[j]) {
                dp[i][j] = true;
                start = i;
                maxLength = len;
            }
        }
    }
    return s.substr(start, maxLength);
}

int main() {
    string s;
    cin >> s;
    cout << longestPalindrome(s) << endl;
    return 0;
}
```

Output:



Time complexity: O(N^2)

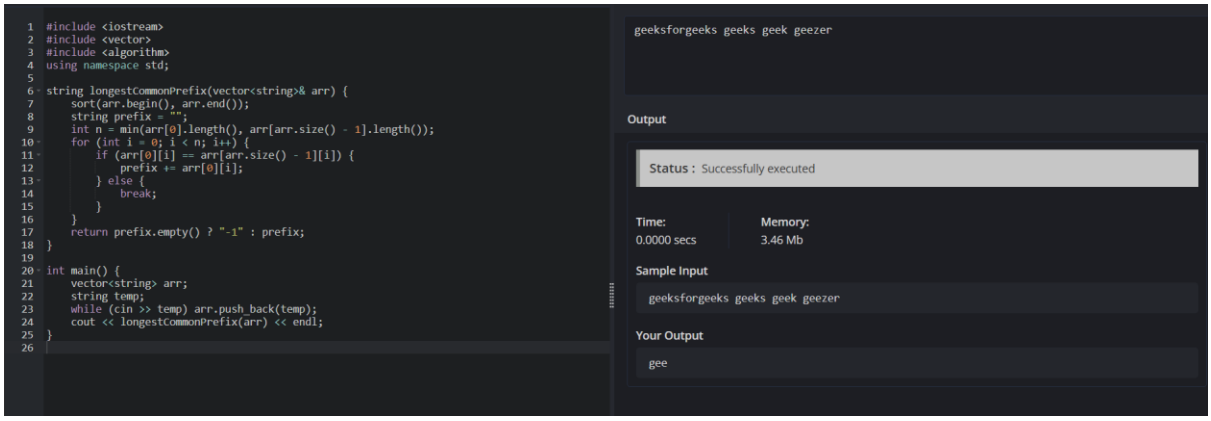
16) Longest Common Prefix using Sorting Given an array of strings arr[]. The task is to return the longest common prefix among each and every strings present in the array. If there‟s no prefix common in all the strings, return “-1”.

Code:

```
#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;


string longestCommonPrefix(vector<string>& arr) {

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

   string prefix = "";

   int n = min(arr[0].length(), arr[arr.size() - 1].length());

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

      if (arr[0][i] == arr[arr.size() - 1][i]) {

         prefix += arr[0][i];

      } else {

         break;

      }
```

```
    }
    return prefix.empty() ? "-1" : prefix;
}

int main() {
    vector<string> arr;
    string temp;
    while (cin >> temp) arr.push_back(temp);
    cout << longestCommonPrefix(arr) << endl;
}
```

Output:



Time complexity: O(N * M)

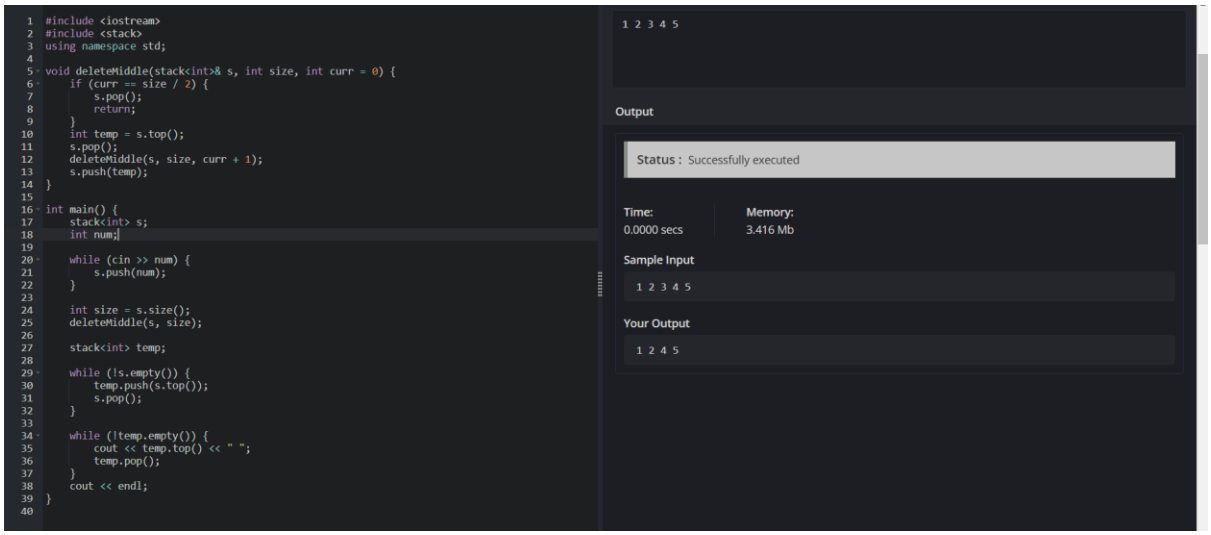
17) Delete middle element of a stack Given a stack with push(), pop(), and empty() operations, The task is to delete the middle element of it without using any additional data structure.

Code:

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

void deleteMiddle(stack<int>& s, int size, int curr = 0) {
    if (curr == size / 2) {
        s.pop();
        return;
```

```
    }
    int temp = s.top();
    s.pop();
    deleteMiddle(s, size, curr + 1);
    s.push(temp);
}

int main() {
    stack<int> s;
    int num;

    while (cin >> num) {
        s.push(num);
    }

    int size = s.size();
    deleteMiddle(s, size);

    stack<int> temp;

    while (!s.empty()) {
        temp.push(s.top());
        s.pop();
    }

    while (!temp.empty()) {
        cout << temp.top() << " ";
        temp.pop();
    }
    cout << endl;
}
```

Output:



Time complexity: O(N)

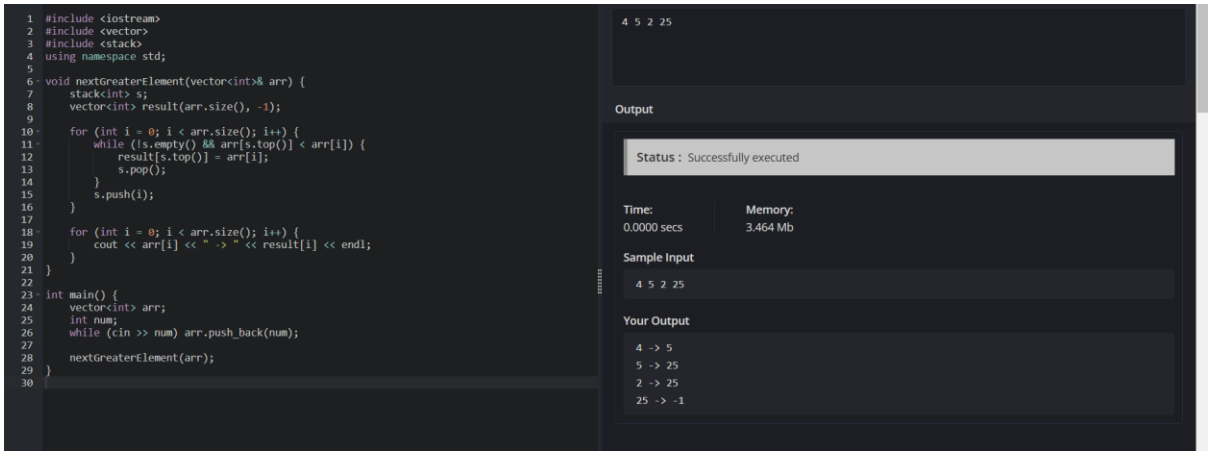
18) Next Greater Element (NGE) for every element in given Array Given an array, print the Next Greater Element (NGE) for every element. Note: The Next greater Element for an element x is the first greater element on the right side of x in the array. Elements for which no greater element exist, consider the next greater element as -1.

Code:

```
#include <iostream>

#include <vector>

#include <stack>

using namespace std;


void nextGreaterElement(vector<int>& arr) {

   stack<int> s;

   vector<int> result(arr.size(), -1);


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

      while (!s.empty() && arr[s.top()] < arr[i]) {

         result[s.top()] = arr[i];

         s.pop();

      }

      s.push(i);
```

```
    }

    for (int i = 0; i < arr.size(); i++) {
        cout << arr[i] << " -> " << result[i] << endl;
    }
}

int main() {
    vector<int> arr;
    int num;
    while (cin >> num) arr.push_back(num);

    nextGreaterElement(arr);
}
```

Output:



Time complexity: O(N)

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.

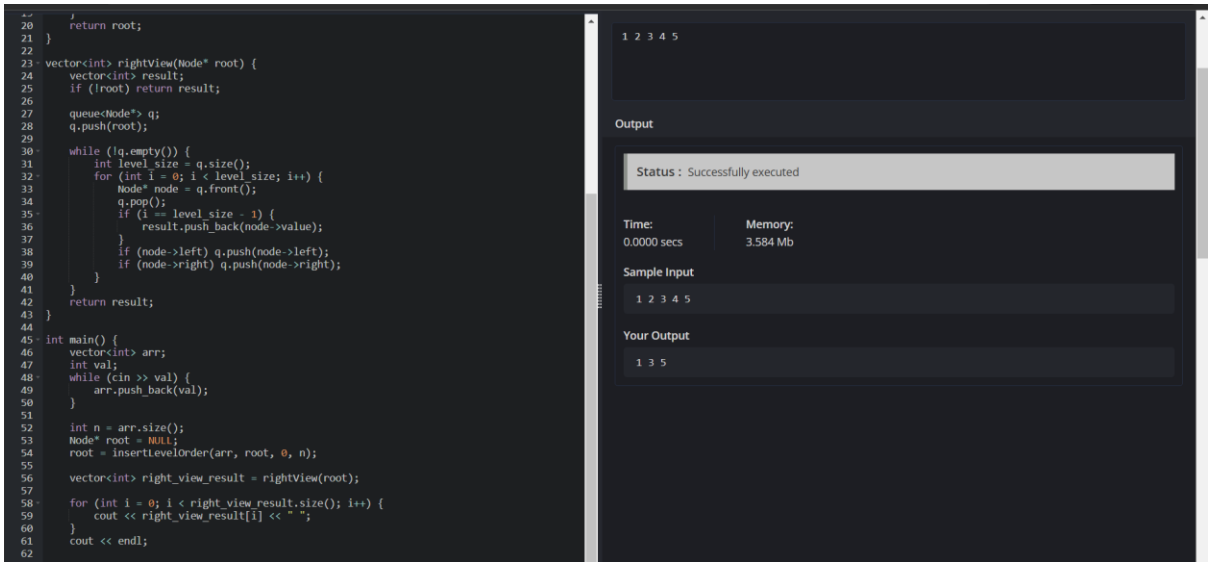
Code:

```
#include <iostream>

#include <queue>

#include <vector>
```

```
using namespace std;

struct Node {
    int value;
    Node* left;
    Node* right;
    Node(int x) : value(x), left(NULL), right(NULL) {}
};

Node* insertLevelOrder(vector<int>& arr, Node* root, int i, int n) {
    if (i < n) {
        Node* temp = new Node(arr[i]);
        root = temp;
        root->left = insertLevelOrder(arr, root->left, 2 * i + 1, n);
        root->right = insertLevelOrder(arr, root->right, 2 * i + 2, n);
    }
    return root;
}

vector<int> rightView(Node* root) {
    vector<int> result;
    if (!root) return result;

    queue<Node*> q;
    q.push(root);

    while (!q.empty()) {
        int level_size = q.size();
        for (int i = 0; i < level_size; i++) {
            Node* node = q.front();
            q.pop();
```

```
            if (i == level_size - 1) {
                result.push_back(node->value);
            }
            if (node->left) q.push(node->left);
            if (node->right) q.push(node->right);
        }
    }
    return result;
}

int main() {
    vector<int> arr;
    int val;
    while (cin >> val) {
        arr.push_back(val);
    }

    int n = arr.size();
    Node* root = NULL;
    root = insertLevelOrder(arr, root, 0, n);

    vector<int> right_view_result = rightView(root);

    for (int i = 0; i < right_view_result.size(); i++) {
        cout << right_view_result[i] << " ";
    }
    cout << endl;

    return 0;
}
```

Output:



```
    return root;
}

vector<int> rightView(Node* root) {
    vector<int> result;
    if (!root) return result;

    queue<Node*> q;
    q.push(root);

    while (!q.empty()) {
        int level_size = q.size();
        for (int i = 0; i < level_size; i++) {
            Node* node = q.front();
            q.pop();
            if (i == level_size - 1) {
                result.push_back(node->value);
            }
            if (node->left) q.push(node->left);
            if (node->right) q.push(node->right);
        }
    }
    return result;
}

int main() {
    vector<int> arr;
    int val;
    while (cin >> val) {
        arr.push_back(val);
    }

    int n = arr.size();
    Node* root = NULL;
    root = insertLevelOrder(arr, root, 0, n);

    vector<int> right_view_result = rightView(root);

    for (int i = 0; i < right_view_result.size(); i++) {
        cout << right_view_result[i] << " ";
    }
    cout << endl;
```

```
1 2 3 4 5
```

Output

Status : Successfully executed

Time: 0.0000 secs

Memory: 3.584 Mb

Sample Input

```
1 2 3 4 5
```

Your Output

```
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.

Code:

```
#include <iostream>

#include <vector>

using namespace std;


struct Node {

   int value;

   Node* left;

   Node* right;

   Node(int x) : value(x), left(NULL), right(NULL) {}

};


Node* insertLevelOrder(vector<int>& arr, Node* root, int i, int n) {

   if (i < n) {

      Node* temp = new Node(arr[i]);

      root = temp;

      root->left = insertLevelOrder(arr, root->left, 2 * i + 1, n);
```

```
        root->right = insertLevelOrder(arr, root->right, 2 * i + 2, n);
    }
    return root;
}

int maxDepth(Node* root) {
    if (!root) return 0;
    else {
        int left_depth = maxDepth(root->left);
        int right_depth = maxDepth(root->right);
        return max(left_depth, right_depth) + 1;
    }
}

int main() {
    vector<int> arr;
    int val;
    while (cin >> val) {
        arr.push_back(val);
    }

    int n = arr.size();
    Node* root = NULL;
    root = insertLevelOrder(arr, root, 0, n);

    int max_depth_result = maxDepth(root);

    cout << max_depth_result << endl;

    return 0;
}
```

Output:

```cpp
using namespace std;

struct Node {
    int value;
    Node* left;
    Node* right;
    Node(int x) : value(x), left(NULL), right(NULL) {}
};

Node* insertLevelOrder(vector<int>& arr, Node* root, int i, int n) {
    if (i < n) {
        Node* temp = new Node(arr[i]);
        root = temp;
        root->left = insertLevelOrder(arr, root->left, 2 * i + 1, n);
        root->right = insertLevelOrder(arr, root->right, 2 * i + 2, n);
    }
    return root;
}

int maxDepth(Node* root) {
    if (!root) return 0;
    else {
        int left_depth = maxDepth(root->left);
        int right_depth = maxDepth(root->right);
        return max(left_depth, right_depth) + 1;
    }
}

int main() {
    vector<int> arr;
    int val;
    while (cin >> val) {
        arr.push_back(val);
    }

    int n = arr.size();
    Node* root = NULL;
    root = insertLevelOrder(arr, root, 0, n);

    int max_depth_result = maxDepth(root);

    cout << max_depth_result << endl;
```

```
12 8 18 5 11
```

Output

Status : Successfully executed

Time: 0.0000 secs

Memory: 3.496 Mb

Sample Input

```
12 8 18 5 11
```

Your Output

```
3
```

Time complexity: O(N)