

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);

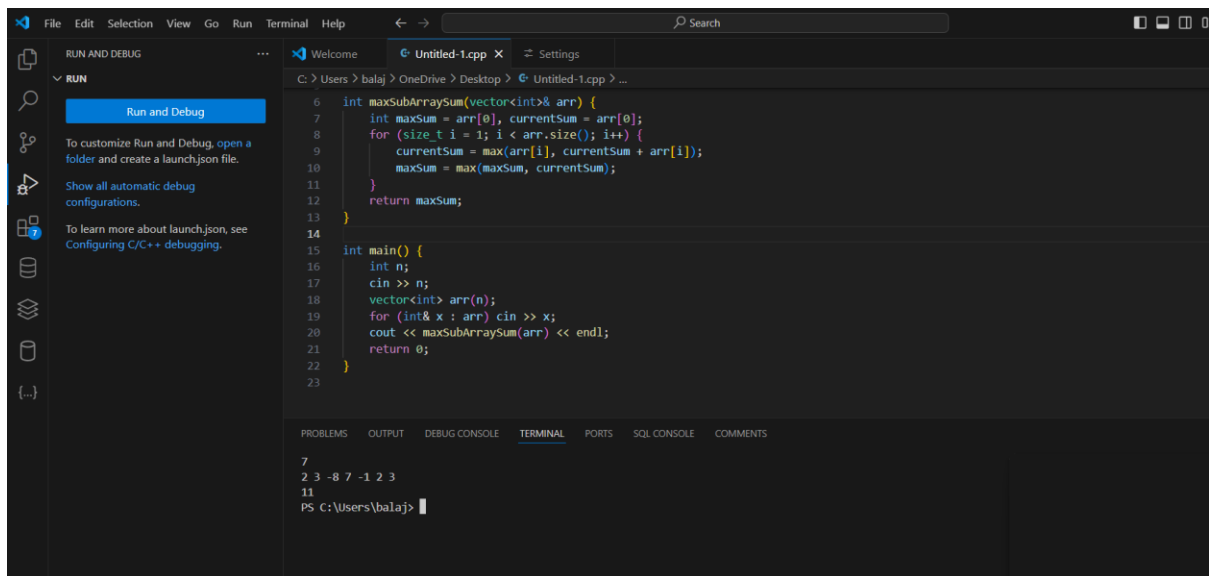
    for (int& x : arr) cin >> x;

    cout << maxSubArraySum(arr) << endl;

    return 0;

}
```

Output:



```
File Edit Selection View Go Run Terminal Help
Welcome Untitled-1.cpp X Settings
C:\Users\balaj> OneDrive\ Desktop> Untitled-1.cpp> ...
6 int maxSubArraySum(vector<int>& arr) {
7   int maxSum = arr[0], currentSum = arr[0];
8   for (size_t i = 1; i < arr.size(); i++) {
9     currentSum = max(arr[i], currentSum + arr[i]);
10    maxSum = max(maxSum, currentSum);
11  }
12  return maxSum;
13 }
14
15 int main() {
16   int n;
17   cin >> n;
18   vector<int> arr(n);
19   for (int& x : arr) cin >> x;
20   cout << maxSubArraySum(arr) << endl;
21   return 0;
22 }
23
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS SQL CONSOLE COMMENTS
7
2 3 -8 7 -1 2 3
11
PS C:\Users\balaj>
```

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>
```

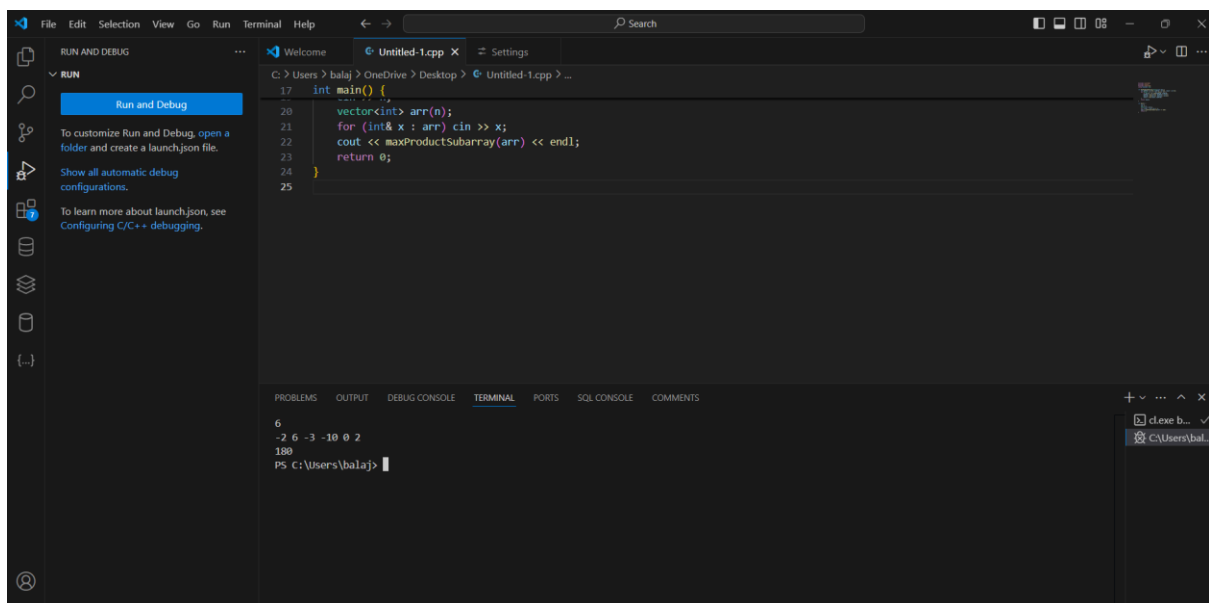
```
#include <algorithm>
```

```
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:



```
File Edit Selection View Go Run Terminal Help  
Welcome Untitled-1.cpp X Settings  
C:\Users\balaj> OneDrive > Desktop > Untitled-1.cpp > ...  
17 int main() {  
18  
19     vector<int> arr(n);  
20  
21     for (int& x : arr) cin >> x;  
22     cout << maxProductSubarray(arr) << endl;  
23     return 0;  
24 }  
25  
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS SQL CONSOLE COMMENTS  
6  
-2 6 -3 -10 0 2  
180  
PS C:\Users\balaj>
```

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>

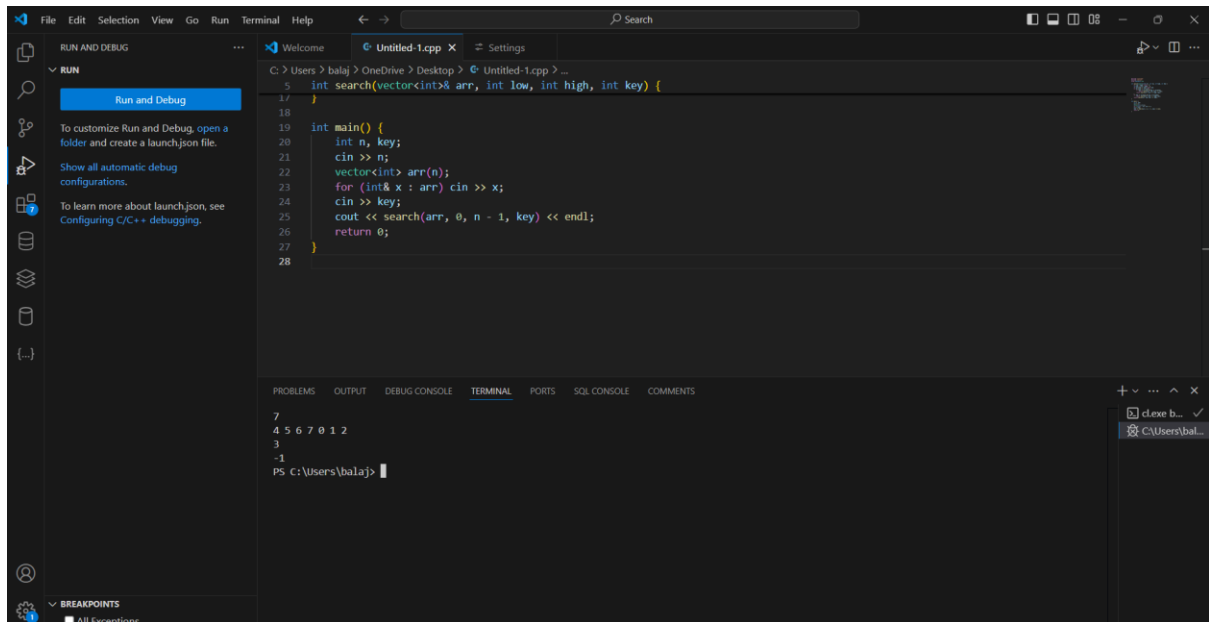
#include <vector>

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:

A screenshot of the Visual Studio Code editor. The main editor window shows a C++ file named 'Untitled-1.cpp'. The code defines a function 'int search(vector<int>& arr, int low, int high, int key)' and a 'main' function. The 'main' function reads an integer 'n' and a 'key', creates a vector 'arr' of size 'n', and fills it with values from 1 to 'n'. It then calls 'search(arr, 0, n - 1, key)' and prints the result. The 'TERMINAL' pane at the bottom shows the output: '7', '4 5 6 7 0 1 2', '3', '-1', and the prompt 'PS C:\Users\balaj>'. The 'RUN AND DEBUG' sidebar on the left is visible, showing the 'Run and Debug' button and some configuration instructions.

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);

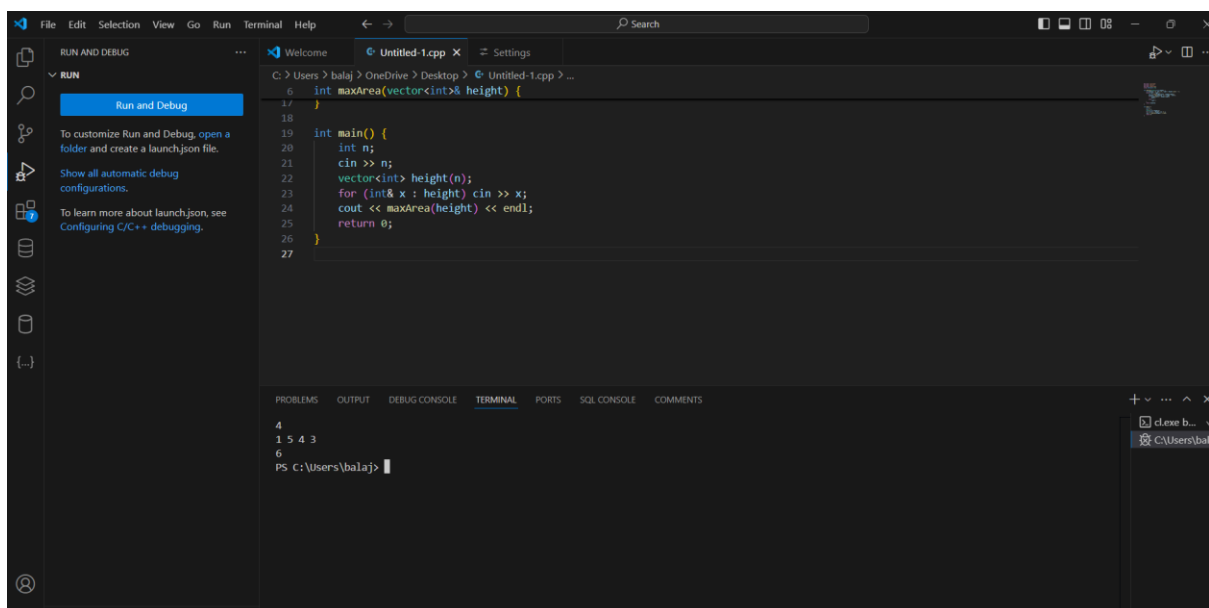
    for (int& x : height) cin >> x;

    cout << maxArea(height) << endl;

    return 0;
}

```

Output:



The screenshot shows the Visual Studio Code interface. The editor displays the C++ code from the previous block. The terminal at the bottom shows the output of the program, which is the sequence of numbers entered by the user: 4, 1, 5, 4, 3. The terminal prompt is PS C:\Users\balaj>.

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:

The screenshot shows the Visual Studio Code interface. The editor displays a C++ file named 'Untitled-1.cpp' with the following code:

```

18 void factorial(int n) {
19     for (int x = 2; x <= n; x++)
20         multiply(result, x);
21     for (int i = result.size() - 1; i >= 0; i--)
22         cout << result[i];
23     cout << endl;
24 }
25
26
27 int main() {
28     int n;
29     cin >> n;
30     factorial(n);
31     return 0;
32 }
33

```

The bottom panel shows the 'TERMINAL' tab with the following output:

```

100
933262154439441526816992388562667004907159682643816214685929638952175999932299156689414639761565182862536979208272723758251185210916864000000000000000
000000000000
PS C:\Users\balaj>

```

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);

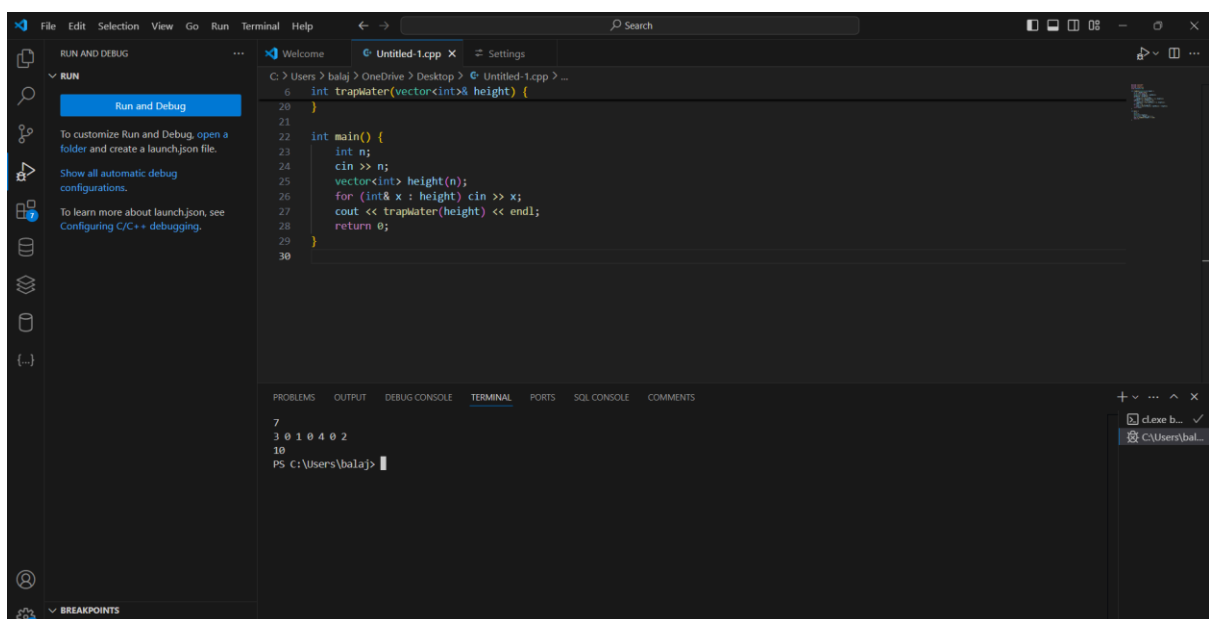
    for (int& x : height) cin >> x;

    cout << trapWater(height) << endl;

    return 0;
}

```

Output:



The screenshot shows the Visual Studio Code interface. The editor window displays a C++ program with the following code:

```

6  int trapWater(vector<int>& height) {
20 }
21
22 int main() {
23     int n;
24     cin >> n;
25     vector<int> height(n);
26     for (int& x : height) cin >> x;
27     cout << trapWater(height) << endl;
28     return 0;
29 }
30

```

The terminal window at the bottom shows the output of the program:

```

7  3 0 1 0 4 0 2
10
PS C:\Users\balaj>

```

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>

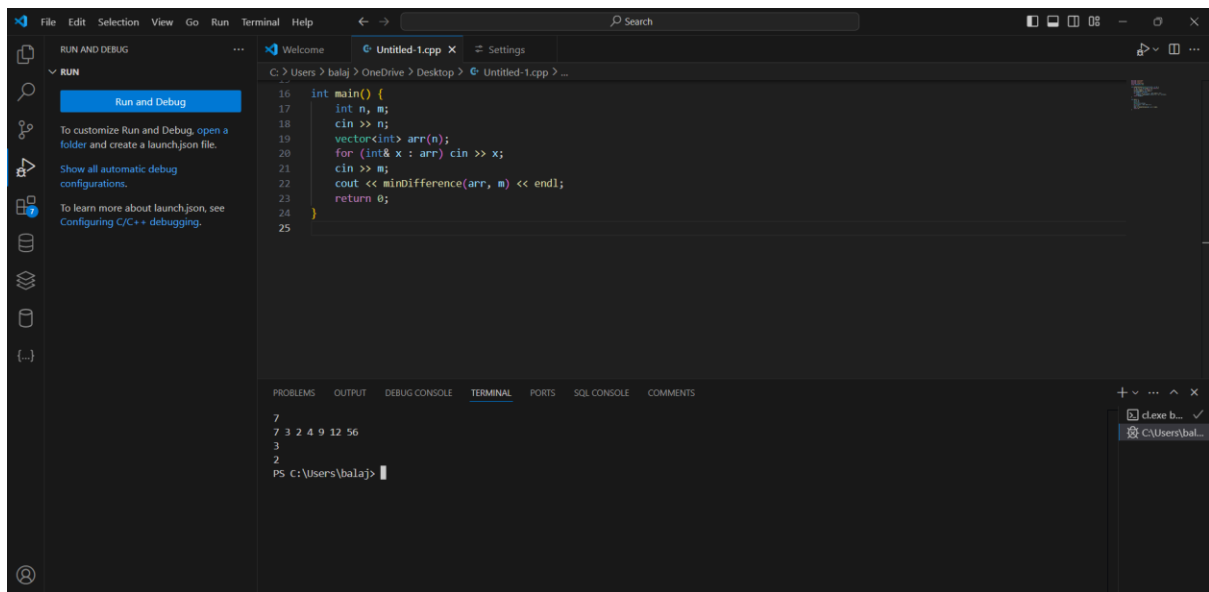
#include <algorithm>

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:



```
16 int main() {
17     int n, m;
18     cin >> n;
19     vector<int> arr(n);
20     for (int& x : arr) cin >> x;
21     cin >> m;
22     cout << minDifference(arr, m) << endl;
23     return 0;
24 }
25
```

7
7 3 2 4 9 12 56
3
2
PS C:\Users\balaj>

Time complexity: $O(n \log n)$

8) Merge Overlapping Intervals Given an array of time intervals where $arr[i] = [start_i, end_i]$, 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)

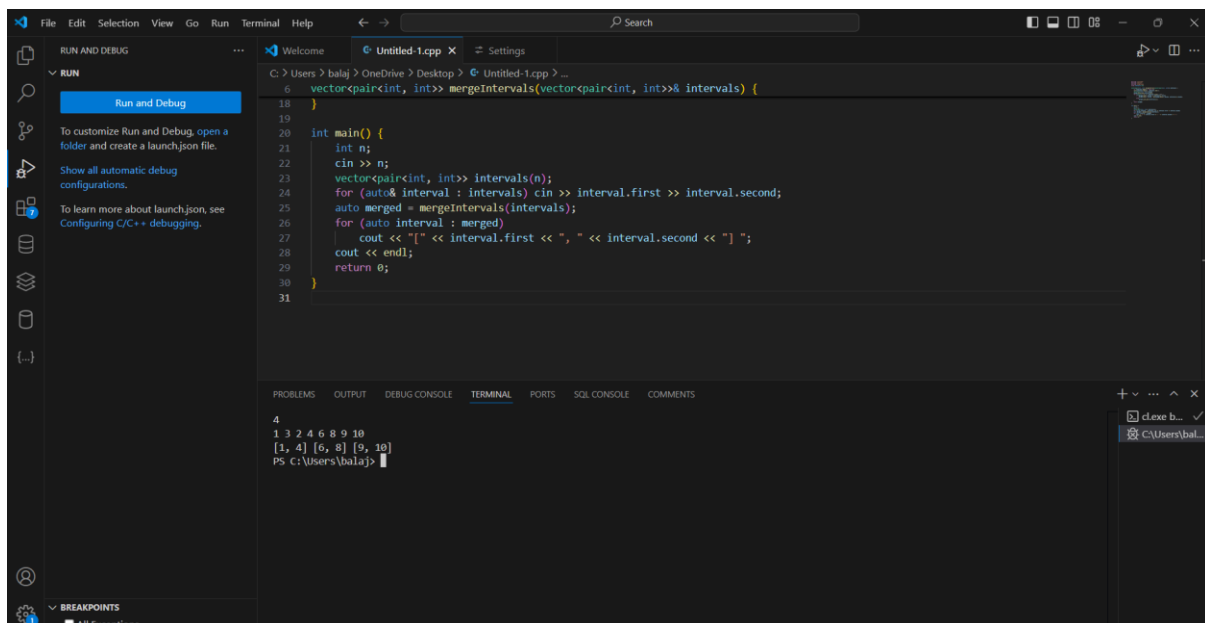
        cout << "[" << interval.first << " , " << interval.second << "]" ";

    cout << endl;

    return 0;
}

```

Output:



The screenshot shows the Visual Studio Code interface. The editor displays the C++ code for merging intervals. The terminal at the bottom shows the output of the program, which is the merged intervals: [1, 4], [6, 8], and [9, 10].

```

4
1 3 2 4 6 8 9 10
[1, 4] [6, 8] [9, 10]
PS C:\Users\balaj>

```

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 `i`th row and `j`th 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;

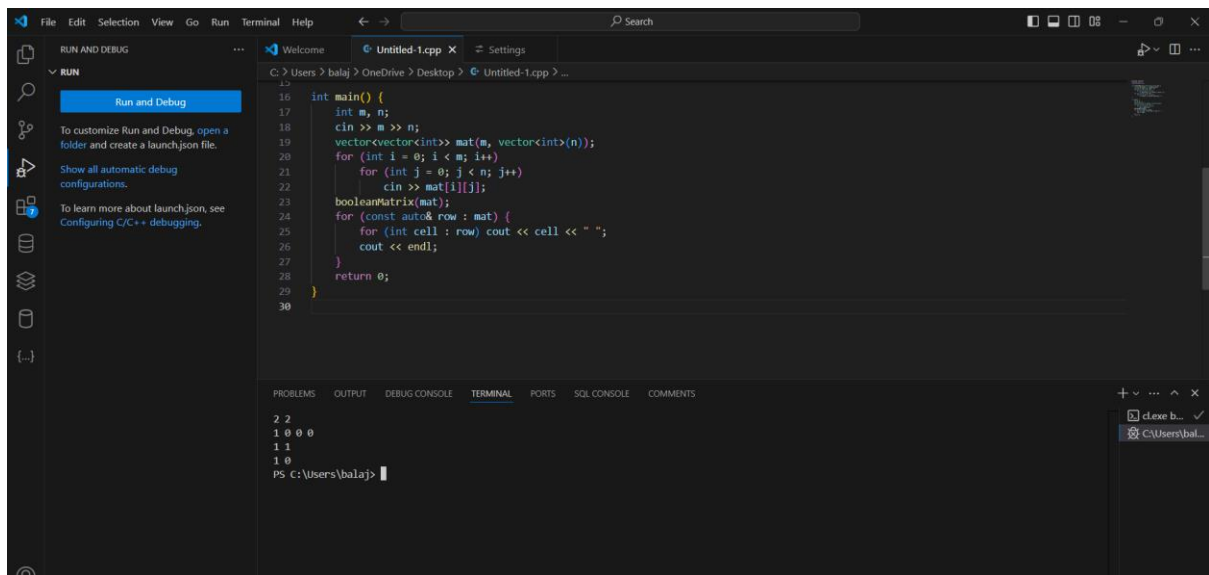
    }

    return 0;

}
```

}

Output:



```
16 int main() {
17     int m, n;
18     cin >> m >> n;
19     vector<vector<int>> mat(m, vector<int>(n));
20     for (int i = 0; i < m; i++)
21         for (int j = 0; j < n; j++)
22             cin >> mat[i][j];
23     booleanMatrix(mat);
24     for (const auto& row : mat) {
25         for (int cell : row) cout << cell << " ";
26         cout << endl;
27     }
28     return 0;
29 }
30
```

2 2
1 0 0 0
1 1
1 0
PS C:\Users\balaj>

Time complexity: $O(m * n)$

10) Print a given matrix in spiral form Given an $m \times n$ 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;
}

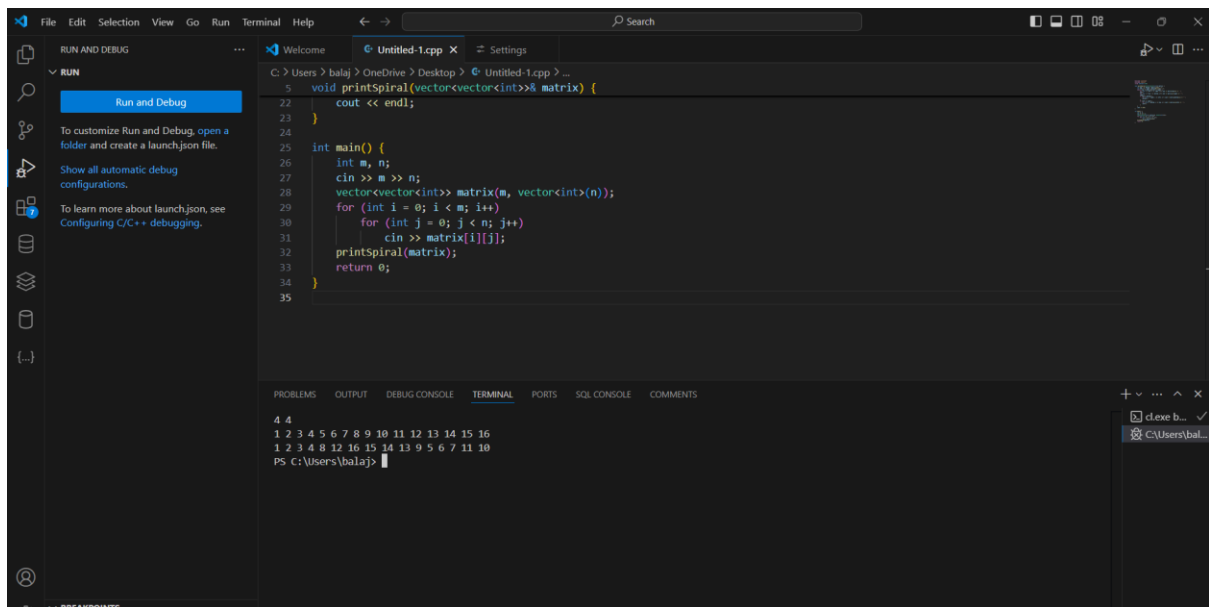
```

```

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:

A screenshot of the Visual Studio Code editor. The main editor window shows a C++ file named 'Untitled-1.cpp'. The code defines a function 'printSpiral' that takes a vector of vectors and prints its elements in a spiral order. The 'main' function reads two integers 'm' and 'n', creates a matrix, and calls 'printSpiral'. The 'TERMINAL' panel at the bottom shows the output of the program: '4 4', '1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16', '1 2 3 4 8 12 16 15 14 13 9 5 6 7 11 10', and the prompt 'PS C:\Users\balaj>'. The 'RUN AND DEBUG' sidebar on the left is visible, showing the 'Run and Debug' button and some instructions.

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();
```

```
        }
```

```
    }
```

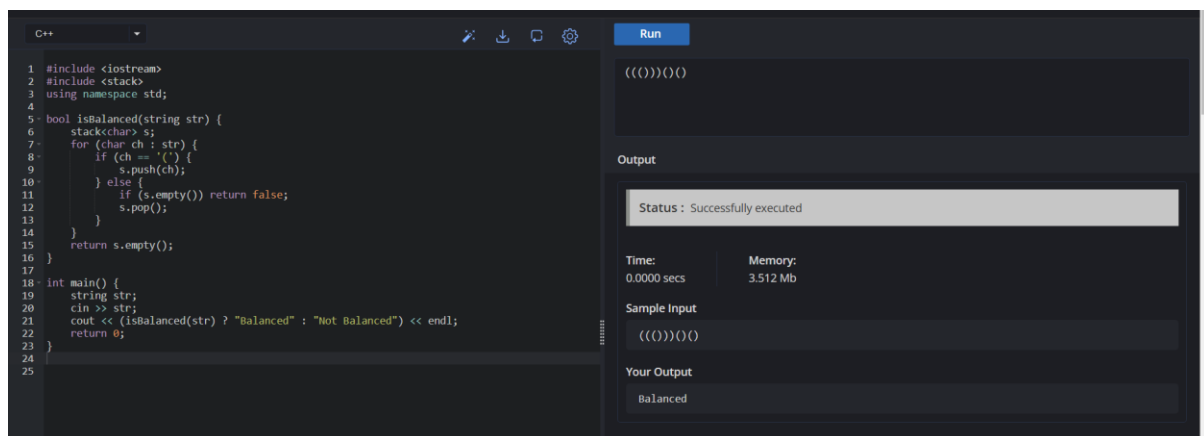
```
    return s.empty();
```



```
}
```

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

Output:



The screenshot shows a C++ IDE with a dark theme. The left pane contains the source code for the 'isBalanced' function and the 'main' function. The right pane shows the execution output. The code defines a stack-based function to check if a string of parentheses is balanced. The main function reads a string from the user and prints the result. The output pane shows the input string '((()))()' and the output 'Balanced'.

```
1 #include <iostream>  
2 #include <stack>  
3 using namespace std;  
4  
5 bool isBalanced(string str) {  
6     stack<char> s;  
7     for (char ch : str) {  
8         if (ch == '(') {  
9             s.push(ch);  
10        } else {  
11            if (s.empty()) return false;  
12            s.pop();  
13        }  
14    }  
15    return s.empty();  
16 }  
17  
18 int main() {  
19     string str;  
20     cin >> str;  
21     cout << (isBalanced(str) ? "Balanced" : "Not Balanced") << endl;  
22     return 0;  
23 }  
24  
25
```

Run

((()))()

Output

Status : Successfully executed

Time: 0.0000 secs Memory: 3.512 Mb

Sample Input

((()))()

Your Output

Balanced

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:

The screenshot shows a C++ IDE with the following code on the left:

```

1 #include <iostream>
2 #include <algorithm>
3 using namespace std;
4
5 bool areAnagrams(string s1, string s2) {
6     if (s1.length() != s2.length()) return false;
7     sort(s1.begin(), s1.end());
8     sort(s2.begin(), s2.end());
9     return s1 == s2;
10 }
11
12 int main() {
13     string s1, s2;
14     cin >> s1 >> s2;
15     cout << (areAnagrams(s1, s2) ? "true" : "false") << endl;
16     return 0;
17 }
18

```

On the right, the output section shows:

geeks kseeg

Output

Status : Successfully executed

Time: 0.0000 secs Memory: 3.352 Mb

Sample Input

geeks kseeg

Your Output

true

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);
}

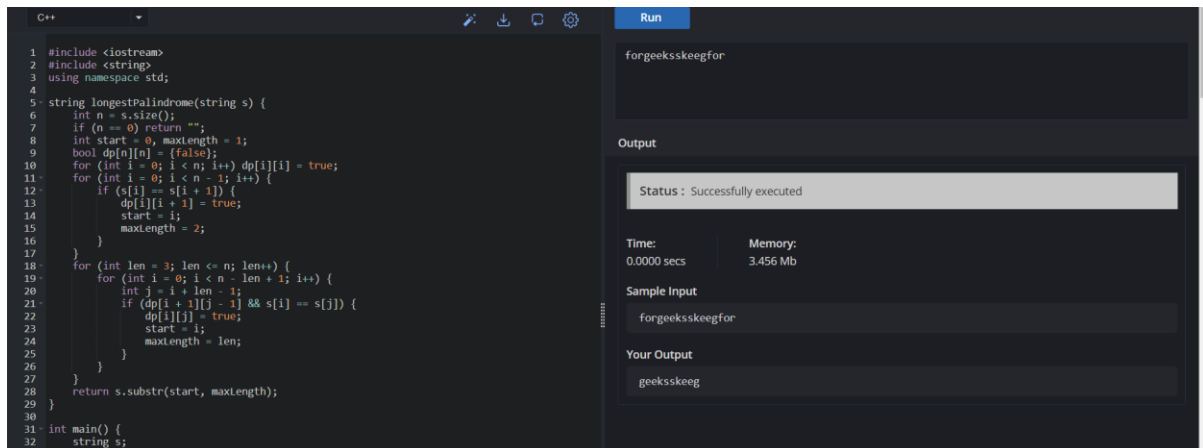
```

```

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

```

Output:



```
1 #include <iostream>
2 #include <string>
3 using namespace std;
4
5 string longestPalindrome(string s) {
6     int n = s.size();
7     if (n == 0) return "";
8     int start = 0, maxLength = 1;
9     bool dp[n][n] = {false};
10    for (int i = 0; i < n; i++) dp[i][i] = true;
11    for (int i = 0; i < n - 1; i++) {
12        if (s[i] == s[i + 1]) {
13            dp[i][i + 1] = true;
14            start = i;
15            maxLength = 2;
16        }
17    }
18    for (int len = 3; len <= n; len++) {
19        for (int i = 0; i < n - len + 1; i++) {
20            int j = i + len - 1;
21            if (dp[i + 1][j - 1] && s[i] == s[j]) {
22                dp[i][j] = true;
23                start = i;
24                maxLength = len;
25            }
26        }
27    }
28    return s.substr(start, maxLength);
29 }
30
31 int main() {
32     string s;
33     s = "forgeeksskeegfor";
34     cout << longestPalindrome(s) << endl;
35 }
```

forgeeksskeegfor

Output

Status : Successfully executed

Time: 0.0000 secs Memory: 3.456 Mb

Sample Input

forgeeksskeegfor

Your Output

geeksskeeg

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:

The screenshot shows a C++ IDE with the following code on the left:

```

1 #include <iostream>
2 #include <vector>
3 #include <algorithm>
4 using namespace std;
5
6 string longestCommonPrefix(vector<string>& arr) {
7     sort(arr.begin(), arr.end());
8     string prefix = "";
9     int n = min(arr[0].length(), arr[arr.size() - 1].length());
10    for (int i = 0; i < n; i++) {
11        if (arr[0][i] == arr[arr.size() - 1][i]) {
12            prefix += arr[0][i];
13        } else {
14            break;
15        }
16    }
17    return prefix.empty() ? "-1" : prefix;
18 }
19
20 int main() {
21     vector<string> arr;
22     string temp;
23     while (cin >> temp) arr.push_back(temp);
24     cout << longestCommonPrefix(arr) << endl;
25 }
26

```

On the right, the IDE's output window shows the input "geeksforgeeks geeks geek geezer" and the output "gee". Below the output, it shows "Status: Successfully executed", "Time: 0.0000 secs", "Memory: 3.46 Mb", "Sample Input: geeksforgeeks geeks geek geezer", and "Your Output: gee".

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:

```
1 #include <iostream>
2 #include <stack>
3 using namespace std;
4
5 void deleteMiddle(stack<int>& s, int size, int curr = 0) {
6     if (curr == size / 2) {
7         s.pop();
8         return;
9     }
10    int temp = s.top();
11    s.pop();
12    deleteMiddle(s, size, curr + 1);
13    s.push(temp);
14 }
15
16 int main() {
17     stack<int> s;
18     int num;
19
20     while (cin >> num) {
21         s.push(num);
22     }
23
24     int size = s.size();
25     deleteMiddle(s, size);
26
27     stack<int> temp;
28
29     while (!s.empty()) {
30         temp.push(s.top());
31         s.pop();
32     }
33
34     while (!temp.empty()) {
35         cout << temp.top() << " ";
36         temp.pop();
37     }
38     cout << endl;
39 }
40
```

1 2 3 4 5

Output

Status : Successfully executed

Time: 0.0000 secs Memory: 3.416 Mb

Sample Input

1 2 3 4 5

Your Output

1 2 4 5

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:

The screenshot shows a C++ IDE with the following code on the left and its output on the right.

```

1 #include <iostream>
2 #include <vector>
3 #include <stack>
4 using namespace std;
5
6 void nextGreaterElement(vector<int>& arr) {
7     stack<int> s;
8     vector<int> result(arr.size(), -1);
9
10    for (int i = 0; i < arr.size(); i++) {
11        while (!s.empty() && arr[s.top()] < arr[i]) {
12            result[s.top()] = arr[i];
13            s.pop();
14        }
15        s.push(i);
16    }
17
18    for (int i = 0; i < arr.size(); i++) {
19        cout << arr[i] << " -> " << result[i] << endl;
20    }
21 }
22
23 int main() {
24     vector<int> arr;
25     int num;
26     while (cin >> num) arr.push_back(num);
27     nextGreaterElement(arr);
28 }
29
30

```

The output on the right shows the sample input "4 5 2 25" and the resulting output:

```

4 -> 5
5 -> 25
2 -> 25
25 -> -1

```

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:

```

20     return root;
21 }
22
23 vector<int> rightView(Node* root) {
24     vector<int> result;
25     if (!root) return result;
26     queue<Node*> q;
27     q.push(root);
28
29     while (!q.empty()) {
30         int level_size = q.size();
31         for (int i = 0; i < level_size; i++) {
32             Node* node = q.front();
33             q.pop();
34             if (i == level_size - 1) {
35                 result.push_back(node->value);
36             }
37             if (node->left) q.push(node->left);
38             if (node->right) q.push(node->right);
39         }
40     }
41     return result;
42 }
43
44 int main() {
45     vector<int> arr;
46     int val;
47     while (cin >> val) {
48         arr.push_back(val);
49     }
50
51     int n = arr.size();
52     Node* root = NULL;
53     root = insertLevelOrder(arr, root, 0, n);
54     vector<int> right_view_result = rightView(root);
55
56     for (int i = 0; i < right_view_result.size(); i++) {
57         cout << right_view_result[i] << " ";
58     }
59     cout << endl;
60 }
61
62

```

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:

```
1  #include <vector>
2  using namespace std;
3
4  struct Node {
5      int value;
6      Node* left;
7      Node* right;
8      Node(int x) : value(x), left(NULL), right(NULL) {}
9  };
10
11 Node* insertLevelOrder(vector<int>& arr, Node* root, int i, int n) {
12     if (i < n) {
13         Node* temp = new Node(arr[i]);
14         root = temp;
15         root->left = insertLevelOrder(arr, root->left, 2 * i + 1, n);
16         root->right = insertLevelOrder(arr, root->right, 2 * i + 2, n);
17     }
18     return root;
19 }
20
21 int maxDepth(Node* root) {
22     if (!root) return 0;
23     else {
24         int left_depth = maxDepth(root->left);
25         int right_depth = maxDepth(root->right);
26         return max(left_depth, right_depth) + 1;
27     }
28 }
29
30 int main() {
31     vector<int> arr;
32     int val;
33     while (cin >> val) {
34         arr.push_back(val);
35     }
36
37     int n = arr.size();
38     Node* root = NULL;
39     root = insertLevelOrder(arr, root, 0, n);
40
41     int max_depth_result = maxDepth(root);
42
43     cout << max_depth_result << endl;
44 }
45
```

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: $O(N)$