

Name: Roshan Vijey

Dept: EEE

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.

Input: arr[] = {2, 3, -8, 7, -1, 2, 3}

Output: 11

Explanation: The subarray {7, -1, 2, 3} has the largest sum 11.

Input: arr[] = {-2, -4}

Output: -2

Explanation: The subarray {-2} has the largest sum -2.

Input: arr[] = {5, 4, 1, 7, 8}

Output: 25

Explanation: The subarray {5, 4, 1, 7, 8} has the largest sum 25.

Question 1 : Maximum Subarray Sum – Kadane"s Algorithm

```
#include <bits/stdc++.h>

using namespace std;

int MaxSubarraySum(vector<int>&nums){

    int Maxsum = INT_MIN,sum = 0;

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

        sum+=nums[i];

        Maxsum = max(Maxsum,sum);

        if(sum < 0){

            sum = 0;

        }

    }

    return Maxsum;
```

```

}

int main()
{
    vector<int>nums= {-2,-5,10,-2,7,18,-18};

    cout << MaxSubarraySum(nums);
}

```

Time Complexity : $O(n)$

Space Complexity : $O(1)$

The screenshot shows the OneCompiler website interface. The code editor contains the following C++ code:

```

1 //Question 1 : Maximum Subarray Sum - Kadane's Algorithm
2
3 #include <bits/stdc++.h>
4 using namespace std;
5 int MaxSubarraySum(vector<int>&nums){
6     int Maxsum = INT_MIN, sum = 0;
7     for(int i = 0; i < nums.size(); i++){
8         sum += nums[i];
9         Maxsum = max(Maxsum, sum);
10        if(sum < 0){
11            sum = 0;
12        }
13    }
14    return Maxsum;
15 }
16 int main()
17 {
18     vector<int>nums= {-2,-5,10,-2,7,18,-18};
19     cout << MaxSubarraySum(nums);
20 }
21
22
23 // Time Complexity : O(n)
24 // Space Complexity : O(1)

```

On the right side, the 'STDIN' section shows 'Input for the program (Optional)' and the 'Output' section shows '33'.

2. Maximum Product Subarray

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

Input: arr[] = {-2, 6, -3, -10, 0, 2}

Output: 180

Explanation: The subarray with maximum product is {6, -3, -10} with product = $6 * (-3) * (-10) = 180$

Input: arr[] = {-1, -3, -10, 0, 60}

Output: 60

Explanation: The subarray with maximum product is {60}.

Question 2: Maximum Product Subarray

```
#include <bits/stdc++.h>

using namespace std;

int MaxProductSubarray(vector<int>&nums){
    int large = nums[0],small = nums[0],maxProduct = nums[0];

    for(int i = 1;i<nums.size();i++){
        int curr_large = large,curr_small = small;
        large = max({nums[i],curr_large*nums[i],curr_small*nums[i]});
        small = min({nums[i],curr_large*nums[i],curr_small*nums[i]});
        maxProduct = max(maxProduct,large);
    }

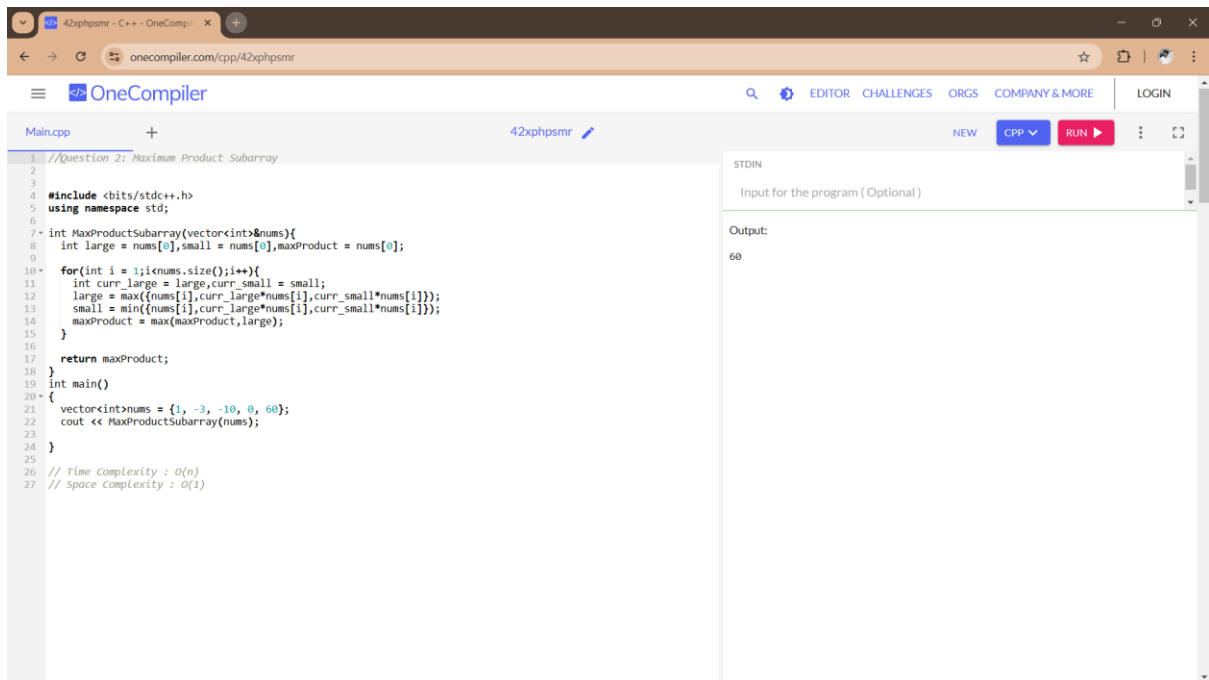
    return maxProduct;
}

int main()
{
    vector<int>nums = {1, -3, -10, 0, 60};
    cout << MaxProductSubarray(nums);

}
```

Time Complexity : $O(n)$

Space Complexity : $O(1)$



```
1 //Question 2: Maximum Product Subarray
2
3
4 #include <bits/stdc++.h>
5 using namespace std;
6
7 int MaxProductSubarray(vector<int>&nums){
8     int large = nums[0], small = nums[0], maxProduct = nums[0];
9
10    for(int i = 1; i < nums.size(); i++){
11        int curr_large = large, curr_small = small;
12        large = max({nums[i], curr_large*nums[i], curr_small*nums[i]});
13        small = min({nums[i], curr_large*nums[i], curr_small*nums[i]});
14        maxProduct = max(maxProduct, large);
15    }
16    return maxProduct;
17 }
18
19 int main()
20 {
21     vector<int> nums = {1, -3, -10, 0, 60};
22     cout << MaxProductSubarray(nums);
23 }
24
25
26 // Time Complexity : O(n)
27 // Space Complexity : O(1)
```

STDIN

Input for the program (Optional)

Output:

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`.

Input : `arr[] = {4, 5, 6, 7, 0, 1, 2}`, key = 0

Output : 4

Input : `arr[] = { 4, 5, 6, 7, 0, 1, 2 }`, key = 3

Output : -1

Input : `arr[] = {50, 10, 20, 30, 40}`, key = 10

Output : 1

Question 3: Search in a sorted and rotated Array

```
#include <bits/stdc++.h>
```

```
using namespace std;
```

```
int Search(vector<int>&nums,int key){
```

```
    int low = 0, high = nums.size()-1;
```

```

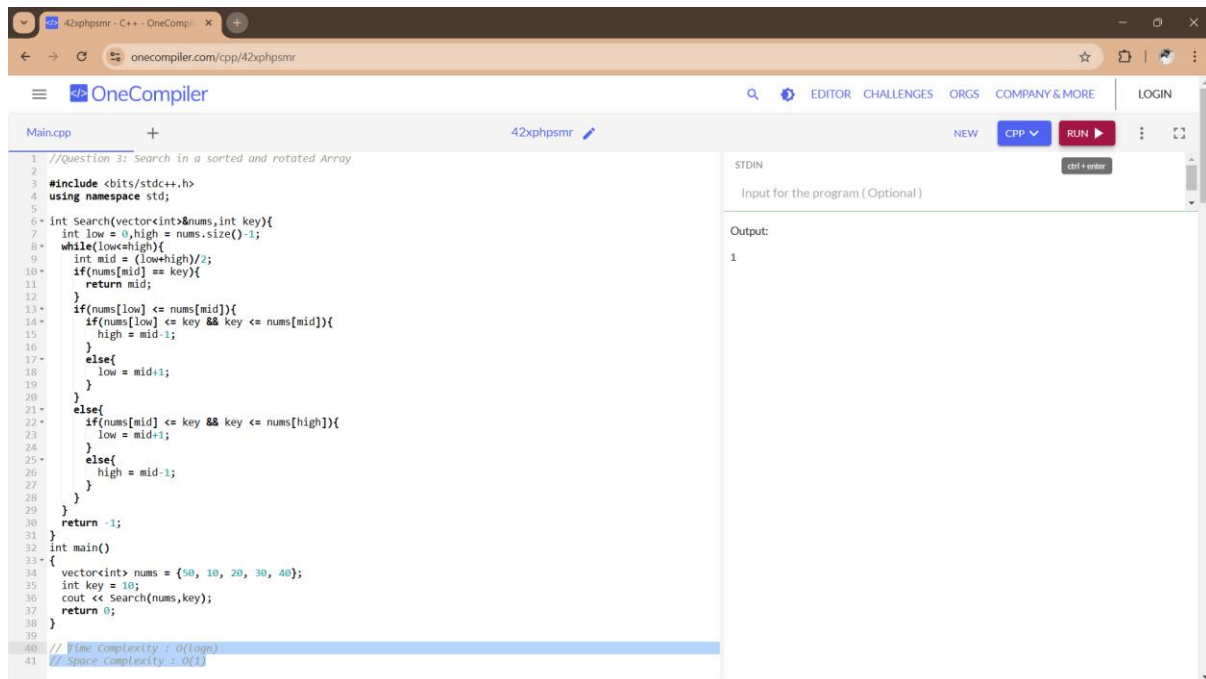
while(low<=high){
    int mid = (low+high)/2;
    if(nums[mid] == key){
        return mid;
    }
    if(nums[low] <= nums[mid]){
        if(nums[low] <= key && key <= nums[mid]){
            high = mid-1;
        }
        else{
            low = mid+1;
        }
    }
    else{
        if(nums[mid] <= key && key <= nums[high]){
            low = mid+1;
        }
        else{
            high = mid-1;
        }
    }
}
return -1;
}

int main()
{
    vector<int> nums = {50, 10, 20, 30, 40};
    int key = 10;
    cout << Search(nums,key);
    return 0;
}

```

Time Complexity : $O(\log n)$

Space Complexity : $O(1)$



```
1 //Question 3: Search in a sorted and rotated Array
2
3 #include <bits/stdc++.h>
4 using namespace std;
5
6 int Search(vector<int>&nums,int key){
7     int low = 0,high = nums.size()-1;
8     while(low<=high){
9         int mid = (low+high)/2;
10        if(nums[mid] == key){
11            return mid;
12        }
13        if(nums[low] <= nums[mid]){
14            if(nums[low] <= key && key <= nums[mid]){
15                high = mid-1;
16            }
17            else{
18                low = mid+1;
19            }
20        }
21        else{
22            if(nums[mid] <= key && key <= nums[high]){
23                low = mid+1;
24            }
25            else{
26                high = mid-1;
27            }
28        }
29    }
30    return -1;
31 }
32 int main()
33 {
34     vector<int> nums = {50, 10, 20, 30, 40};
35     int key = 10;
36     cout << Search(nums,key);
37     return 0;
38 }
39
40 // Time complexity : O(logn)
41 // Space Complexity : O(1)
```

STDIN
Input for the program (Optional)

Output:
1

4. Container with Most Water

Input: arr = [1, 5, 4, 3]

Output: 6

Explanation:

5 and 3 are distance 2 apart. So the size of the base = 2.

Height of container = $\min(5, 3) = 3$. So total area = $3 * 2 = 6$

Input: arr = [3, 1, 2, 4, 5]

Output: 12

Explanation:

5 and 3 are distance 4 apart. So the size of the base = 4.

Height of container = $\min(5, 3) = 3$. So total area = $4 * 3 = 12$

Question 4: Container with Most Water

```
#include <bits/stdc++.h>

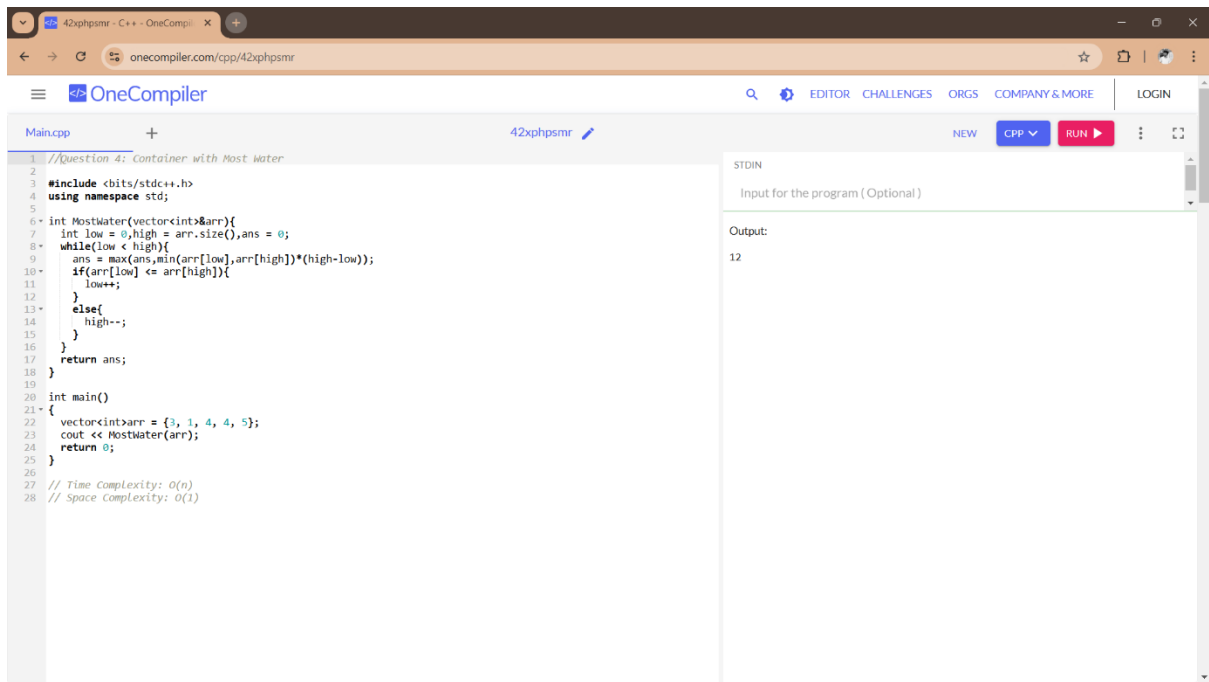
using namespace std;

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

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

Time Complexity: $O(n)$

Space Complexity: $O(1)$



5. Find the Factorial of a large number

Input: 100

Output:

93326215443944152681699238856266700490715968264381621468592963895217599993229915
6089414639761565182862536979208272237582511852109168640000000000000000000000

Input: 50

Output: 30414093201713378043612608166064768844377641568960512000000000000

Question 5: Factorial of a large number

```
#include <bits/stdc++.h>
```

```
using namespace std;
```

```
#define MAX 1000
```

```
int multiply(int x, int res[], int res_size);
```

```
void factorial(int n)
```



```

{
    int res[MAX];
    res[0] = 1;
    int res_size = 1;

    for (int x = 2; x <= n; x++)
        res_size = multiply(x, res, res_size);

    for (int i = res_size - 1; i >= 0; i--)
        cout << res[i];
}

int multiply(int x, int res[], int res_size)
{
    int carry = 0;
    for (int i = 0; i < res_size; i++) {
        int prod = res[i] * x + carry;

        res[i] = prod % 10;

        carry = prod / 10;
    }

    while (carry) {
        res[res_size] = carry % 10;
        carry = carry / 10;
        res_size++;
    }

    return res_size;
}

```

```
int main()
{
    factorial(100);
    return 0;
}
```

Time Complexity : $O(N \log N!)$

Space Complexiy : $O(\text{Max})$

A screenshot of the OneCompiler web interface. The browser address bar shows "onecompiler.com/cpp/42xphpsmr". The page has a navigation bar with links like EDITOR, CHALLENGES, ORGS, COMPANY & MORE, and LOGIN. Below the navigation bar, there's a header area with "Main.cpp" and "42xphpsmr". The main editor displays C++ code for calculating factorial using recursion. The code includes headers, defines MAX as 1000, and implements multiply and factorial functions. On the right side, there are tabs for STDIN, Input for the program (Optional), and Output. The Output tab shows a long numerical result.

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.

Input: arr[] = {3, 0, 1, 0, 4, 0, 2}

Output: 10

Explanation: The expected rainwater to be trapped is shown in the above image.

Input: arr[] = {3, 0, 2, 0, 4}

Output: 7

Explanation: We trap $0 + 3 + 1 + 3 + 0 = 7$ units.

Input: arr[] = {1, 2, 3, 4}

Output: 0

Explanation : We cannot trap water as there is no height bound on both sides

Input: arr[] = {10, 9, 0, 5}

Output: 5

Explanation : We trap $0 + 0 + 5 + 0 = 5$

Question 6 : Trapping Rainwater Problem

```
#include<bits/stdc++.h>
```

```
using namespace std;
```

```
int trap(vector <int>&arr) {
```

```
    int n = arr.size();
```

```
    vector<int>prefix(n),suffix(n);
```

```
    prefix[0] = arr[0];
```

```
    for (int i = 1; i < n; i++) {
```

```
        prefix[i] = max(prefix[i - 1], arr[i]);
```

```
    }
```

```
    suffix[n - 1] = arr[n - 1];
```

```
    for (int i = n - 2; i >= 0; i--) {
```

```
        suffix[i] = max(suffix[i + 1], arr[i]);
```

```
    }
```

```
    int waterTrapped = 0;
```

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

```
        waterTrapped += min(prefix[i], suffix[i]) - arr[i];
```

```
    }
```

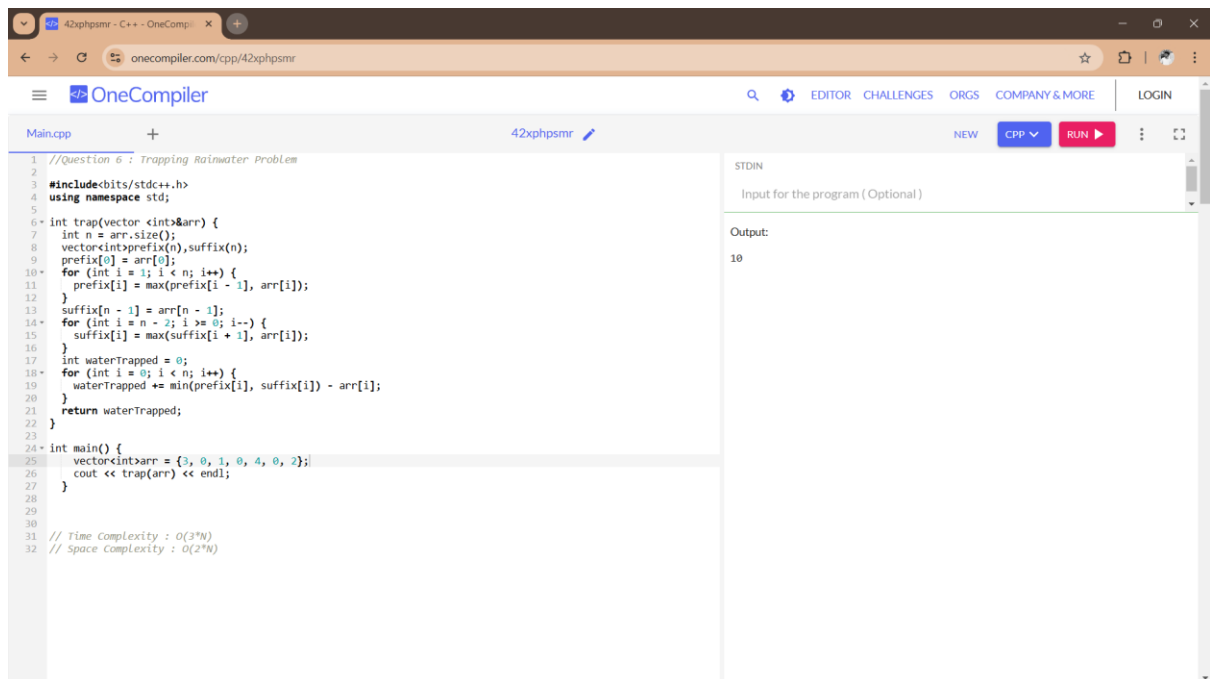
```
    return waterTrapped;
```

```
}
```

```
int main() {  
  
    vector<int>arr = {3, 0, 1, 0, 4, 0, 2}  
  
    cout << trap(arr) << endl;  
  
}  
}
```

Time Complexity : $O(3*N)$

Space Complexity : $O(2*N)$



The screenshot shows the OneCompiler website interface. The editor displays a C++ program for the Trapping Rainwater Problem. The code defines a function `trap` that takes a vector of integers and returns the total trapped water. It uses prefix and suffix arrays to calculate the maximum height to the left and right of each bar. The `main` function initializes the array `{3, 0, 1, 0, 4, 0, 2}` and prints the result of `trap(arr)`. Comments at the bottom indicate the time complexity is $O(3*N)$ and space complexity is $O(2*N)$. The output section shows the result `10`.

7. Chocolate Distribution Problem

Given an array `arr[]` of n integers where `arr[i]` represents the number of chocolates in i th 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.

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

Output: 2

Explanation: If we distribute chocolate packets {3, 2, 4}, we will get the minimum difference, that is 2.

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

Output: 7

Explanation: If we distribute chocolate packets {3, 2, 4, 9, 7}, we will get the minimum difference, that is $9 - 2 = 7$.

Question 7 : Chocolate Distribution Problem

```
#include <bits/stdc++.h>
```

```
using namespace std;
```

```
int findDiff(vector<int> &arr, int m) {
```

```
    int n = arr.size();
```

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

```
    int ans = INT_MAX;
```

```
    for (int i = 0; i + m - 1 < n; i++) {
```

```
        int diff = arr[i + m - 1] - arr[i];
```

```
        if (diff < ans){
```

```
            ans = diff;
```

```
        }
```

```
    }
```

```
    return ans;
```

```
}
```

```
int main() {
```

```
    vector<int> arr = {7, 3, 2, 4, 9, 12, 56};
```

```

int m = 3;

cout << findDiff(arr, m);

return 0;
}

```

Time Complexity : $O(N \log N)$

Space Complexity : $O(1)$

The screenshot shows the OneCompiler website interface. The editor contains the following C++ code:

```

1 //Question 7 : Chocolate Distribution Problem
2
3 #include <bits/stdc++.h>
4 using namespace std;
5
6 int findDiff(vector<int> &arr, int m) {
7     int n = arr.size();
8     sort(arr.begin(), arr.end());
9     int ans = INT_MAX;
10
11     for (int i = 0; i + m - 1 < n; i++) {
12         int diff = arr[i + m - 1] - arr[i];
13         if (diff < ans) {
14             ans = diff;
15         }
16     }
17     return ans;
18 }
19
20 int main() {
21     vector<int> arr = {7, 3, 2, 4, 9, 12, 56};
22     int m = 3;
23     cout << findDiff(arr, m);
24     return 0;
25 }
26
27 // Time Complexity : O(NlogN)
28 // Space Complexity : O(1)

```

The right-hand side of the interface shows the STDIN input field (empty) and the Output field, which displays the number 2.

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.

Input: $arr[] = [[1, 3], [2, 4], [6, 8], [9, 10]]$

Output: $[[1, 4], [6, 8], [9, 10]]$

Explanation: In the given intervals, we have only two overlapping intervals $[1, 3]$ and $[2, 4]$. Therefore, we will merge these two and return $[[1, 4], [6, 8], [9, 10]]$.

Input: $arr[] = [[7, 8], [1, 5], [2, 4], [4, 6]]$

Output: $[[1, 6], [7, 8]]$

Explanation: We will merge the overlapping intervals $[[1, 5], [2, 4], [4, 6]]$ into a single interval $[1, 6]$.

Question 8 : Merge Overlapping Intervals

```
#include <bits/stdc++.h>

using namespace std;

vector<vector<int>> mergeInterval(vector<vector<int>> &arr) {

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

    vector<vector<int>> ans;

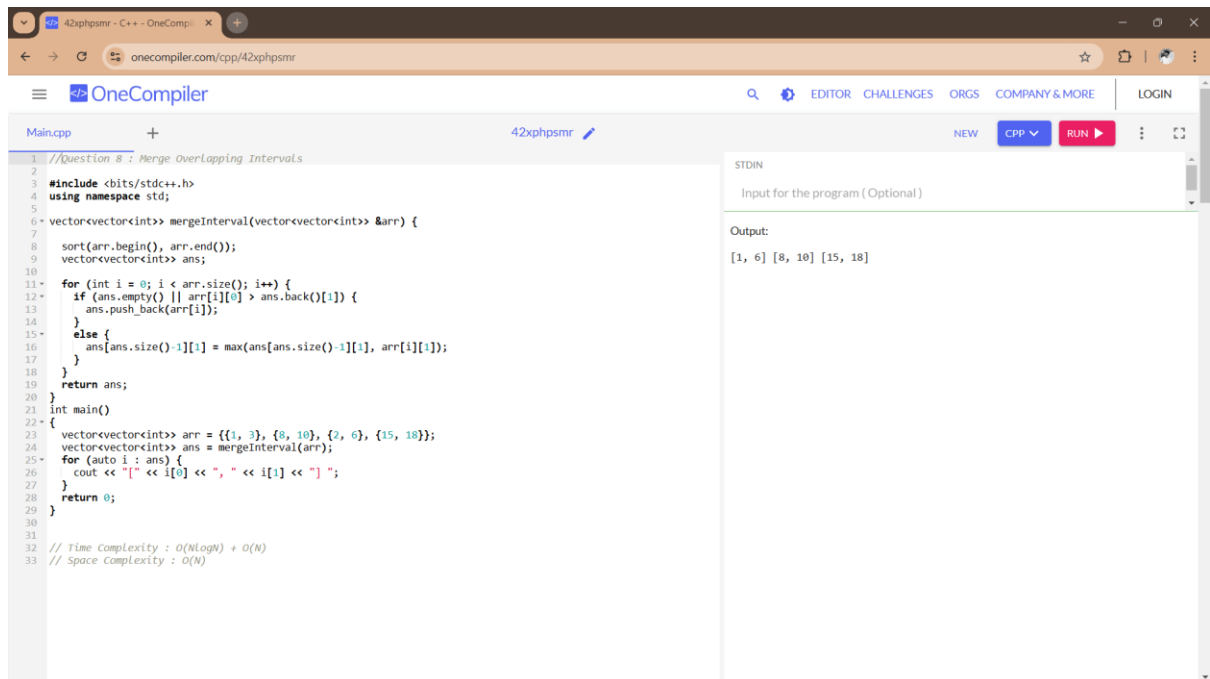
    for (int i = 0; i < arr.size(); i++) {
        if (ans.empty() || arr[i][0] > ans.back()[1]) {
            ans.push_back(arr[i]);
        }
        else {
            ans[ans.size()-1][1] = max(ans[ans.size()-1][1], arr[i][1]);
        }
    }

    return ans;
}

int main()
{
    vector<vector<int>> arr = {{1, 3}, {8, 10}, {2, 6}, {15, 18}};
    vector<vector<int>> ans = mergeInterval(arr);
    for (auto i : ans) {
        cout << "[" << i[0] << ", " << i[1] << "]" << " ";
    }
    return 0;
}
```

Time Complexity : $O(N \log N) + O(N)$

Space Complexity : $O(N)$



```
1 //Question 8 : Merge Overlapping Intervals
2
3 #include <bits/stdc++.h>
4 using namespace std;
5
6 vector<vector<int>> mergeInterval(vector<vector<int>> &arr) {
7
8     sort(arr.begin(), arr.end());
9     vector<vector<int>> ans;
10
11     for (int i = 0; i < arr.size(); i++) {
12         if (ans.empty() || arr[i][0] > ans.back()[1]) {
13             ans.push_back(arr[i]);
14         }
15         else {
16             ans[ans.size()-1][1] = max(ans[ans.size()-1][1], arr[i][1]);
17         }
18     }
19     return ans;
20 }
21 int main()
22 {
23     vector<vector<int>> arr = {{1, 3}, {8, 10}, {2, 6}, {15, 18}};
24     vector<vector<int>> ans = mergeInterval(arr);
25     for (auto i : ans) {
26         cout << "[" << i[0] << ", " << i[1] << "] ";
27     }
28     return 0;
29 }
30
31 // Time Complexity : O(NlogN) + O(N)
32 // Space Complexity : O(N)
```

STDIN
Input for the program (Optional)

Output:
[1, 6] [8, 10] [15, 18]

9. A Boolean Matrix Question

Given a boolean matrix $mat[M][N]$ of size $M \times N$, 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.

Input: $\{\{1, 0\},$

$\{0, 0\}\}$

Output: $\{\{1, 1\}$

$\{1, 0\}\}$

Input: $\{\{0, 0, 0\},$

$\{0, 0, 1\}\}$

Output: $\{\{0, 0, 1\},$

$\{1, 1, 1\}\}$

Input: $\{\{1, 0, 0, 1\},$

$\{0, 0, 1, 0\},$

$\{0, 0, 0, 0\}\}$

Output: $\{\{1, 1, 1, 1\},$

$\{1, 1, 1, 1\},$

{1, 0, 1, 1}}

Question 9: A Boolean Matrix Question

```
#include <bits/stdc++.h>
```

```
using namespace std;
```

```
void boolmatrix(vector<vector<int>>&mat){
```

```
    vector<int>row(mat.size(),0);
```

```
    vector<int>col(mat[0].size(),0);
```

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

```
        for(int j = 0;j<mat[0].size();j++){
```

```
            if(mat[i][j] == 1){
```

```
                row[i] = 1;
```

```
                col[j] = 1;
```

```
            }
```

```
        }
```

```
    }
```

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

```
        if(row[i] == 1){
```

```
            for(int j = 0;j<mat[0].size();j++){
```

```
                mat[i][j] = 1;
```

```
            }
```

```
        }
```

```
    }
```

```
    for(int j = 0;j<col.size();j++){
```

```
        if(col[j] == 1){
```

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

```
                mat[i][j] = 1;
```

```

    }
}
}

}

int main() {
    vector<vector<int>>>mat = {{1, 0, 0, 1},
                               {0, 0, 1, 0},
                               {0, 0, 0, 0}};

    boolmatrix(mat);
    for(int i = 0;i<mat.size();i++){
        for(int j = 0;j<mat[0].size();j++){
            cout << mat[i][j] << " ";
        }
        cout << endl;
    }
    return 0;
}

```

Time Complexity : $O(2*m*n)$

Space Complexity : $O(m+n)$

```

1 //Question 9: A Boolean Matrix Question
2 #include <bits/stdc++.h>
3 using namespace std;
4
5 void boolMatrix(vector<vector<int>>&mat){
6     vector<int>row(mat.size(),0);
7     vector<int>col(mat[0].size(),0);
8
9     for(int i = 0;i<mat.size();i++){
10        for(int j = 0;j<mat[0].size();j++){
11            if(mat[i][j] == 1){
12                row[i] = 1;
13                col[j] = 1;
14            }
15        }
16    }
17    for(int i = 0;i<row.size();i++){
18        if(row[i] == 1){
19            for(int j = 0;j<col[0].size();j++){
20                mat[i][j] = 1;
21            }
22        }
23    }
24    for(int j = 0;j<col.size();j++){
25        if(col[j] == 1){
26            for(int i = 0;i<mat.size();i++){
27                mat[i][j] = 1;
28            }
29        }
30    }
31 }
32
33
34
35 int main() {
36     vector<vector<int>>mat = {{1, 0, 0, 1},
37                             {0, 0, 1, 0},
38                             {0, 0, 0, 0}};
39     boolMatrix(mat);
40     for(int i = 0;i<mat.size();i++){
41         for(int j = 0;j<mat[0].size();j++){
42             cout << mat[i][j] << " ";
43         }
44     }
45 }

```

STDIN

Input for the program (Optional)

Output:

```

1 1 1 1
1 1 1 1
1 0 1 1

```

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.

Input: str = “((()))()”

Output: Balanced

Input: str = “()()()”

Output: Not Balanced

Question 13: Check if given Parentheses expression is balanced or not

```

#include <bits/stdc++.h>

using namespace std;

string Balanced(string &str){

    int open = 0;

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

        if(str[i] == '('){

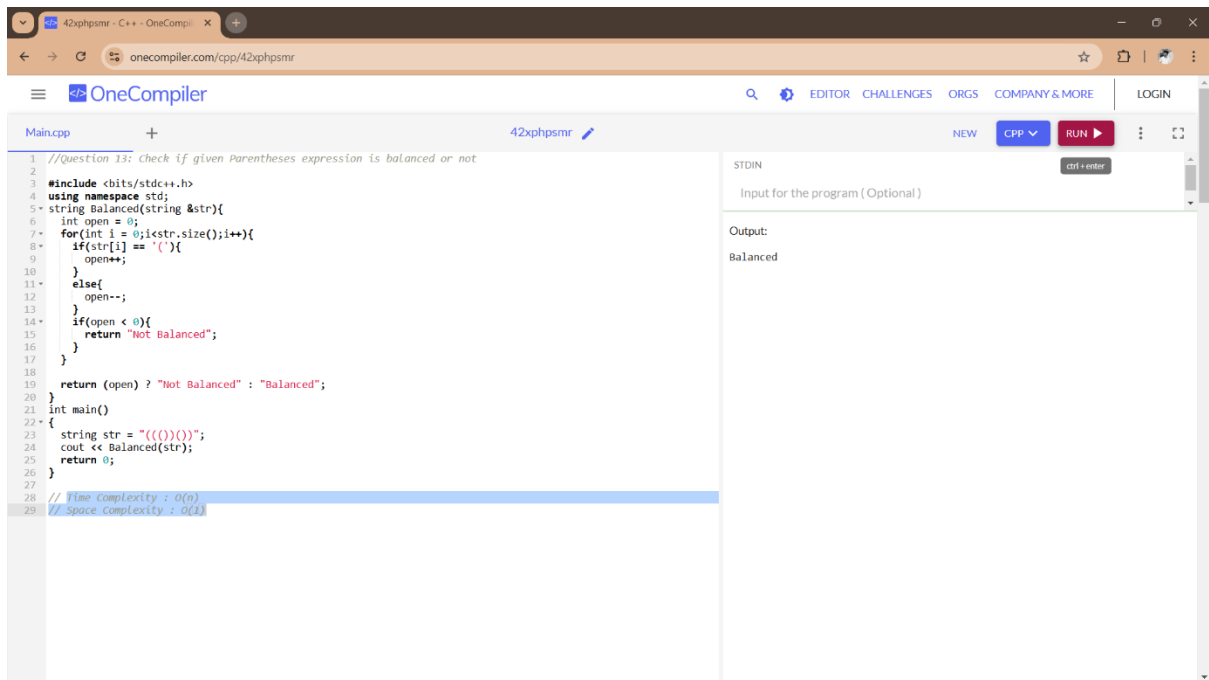
            open++;

```

```
    }  
    else{  
        open--;  
    }  
    if(open < 0){  
        return "Not Balanced";  
    }  
}  
  
return (open) ? "Not Balanced" : "Balanced";  
}  
int main()  
{  
    string str = "((()))()";  
    cout << Balanced(str);  
    return 0;  
}
```

Time Complexity : $O(n)$

Space Complexity : $O(1)$



```
1 //Question 13: Check if given Parentheses expression is balanced or not
2
3 #include <bits/stdc++.h>
4 using namespace std;
5 string Balanced(string &str){
6     int open = 0;
7     for(int i = 0; i<str.size(); i++){
8         if(str[i] == '('){
9             open++;
10        }
11        else{
12            open--;
13        }
14        if(open < 0){
15            return "Not Balanced";
16        }
17    }
18    return (open ? "Not Balanced" : "Balanced");
19 }
20
21 int main()
22 {
23     string str = "(()())";
24     cout << Balanced(str);
25     return 0;
26 }
27
28 // Time complexity : O(n)
29 // Space complexity : O(1)
```

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.

Question 14: Check if two Strings are Anagrams of each other

```
#include <bits/stdc++.h>
```

```

using namespace std;

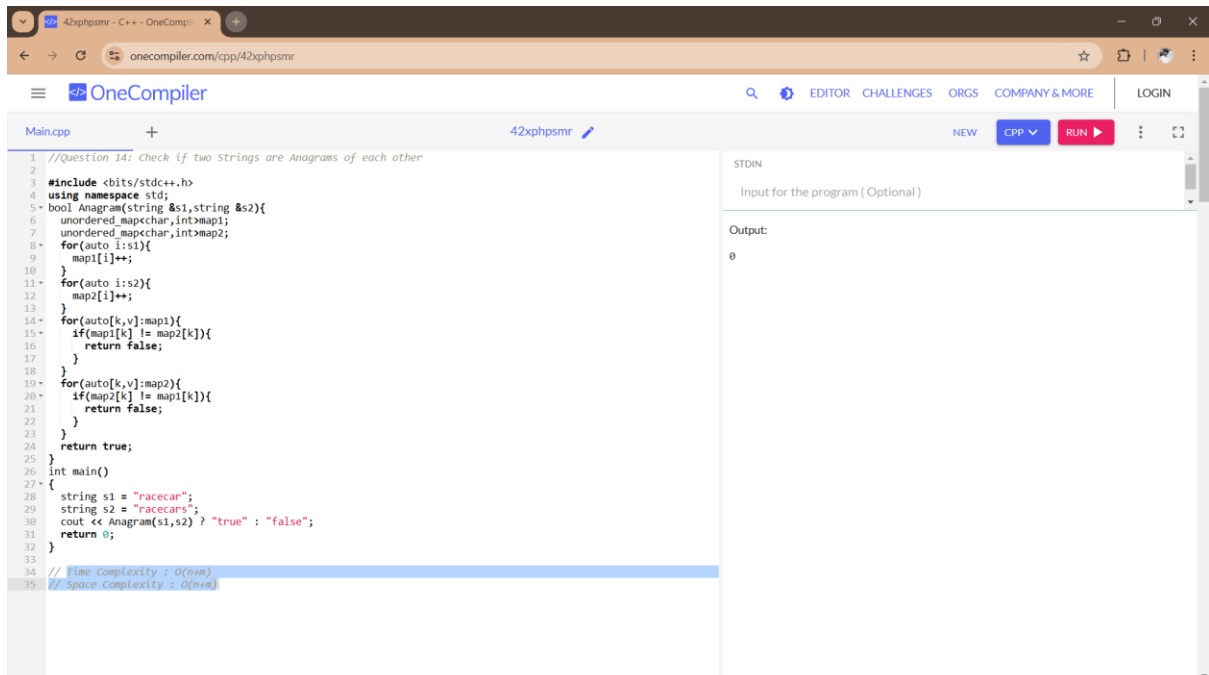
bool Anagram(string &s1,string &s2){
    unordered_map<char,int>map1;
    unordered_map<char,int>map2;
    for(auto i:s1){
        map1[i]++;
    }
    for(auto i:s2){
        map2[i]++;
    }
    for(auto[k,v]:map1){
        if(map1[k] != map2[k]){
            return false;
        }
    }
    for(auto[k,v]:map2){
        if(map2[k] != map1[k]){
            return false;
        }
    }
    return true;
}

int main()
{
    string s1 = "racecar";
    string s2 = "racecars";
    cout << Anagram(s1,s2) ? "true" : "false";
    return 0;
}

```

Time Complexity : $O(n+m)$

Space Complexity : $O(n+m)$



```
1 //Question 14: Check if two Strings are Anagrams of each other
2
3 #include <bits/stdc++.h>
4 using namespace std;
5 bool Anagram(string &s1,string &s2){
6     unordered_map<char,int>map1;
7     unordered_map<char,int>map2;
8     for(auto i:s1){
9         map1[i]++;
10    }
11    for(auto i:s2){
12        map2[i]++;
13    }
14    for(auto [k,v]:map1){
15        if(map2[k] != map1[k]){
16            return false;
17        }
18    }
19    for(auto [k,v]:map2){
20        if(map1[k] != map2[k]){
21            return false;
22        }
23    }
24    return true;
25 }
26 int main()
27 {
28     string s1 = "racecar";
29     string s2 = "racecars";
30     cout << Anagram(s1,s2) ? "true" : "false";
31     return 0;
32 }
33
34 // Time complexity : O(n+m)
35 // Space Complexity : O(n+m)
```

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.

Input: `str = "Geeks"`

Output: `"ee"`

Input: `str = "abc"`

Output: `"a"`

Input: `str = ""`

Output: `""`

Question 15: Longest Palindromic Substring

```

#include <bits/stdc++.h>

using namespace std;

string longPalStr(string &s) {

    if (s.size() == 0) {
        return "";
    }

    int start = 0, maxlen = 1;

    for (int i = 0; i < s.size(); i++) {
        for (int j = 0; j <= 1; j++) {
            int low = i;
            int high = i + j;

            while (low >= 0 && high < s.size() && s[low] == s[high]) {
                int currlen = high - low + 1;
                if (currlen > maxlen) {
                    start = low;
                    maxlen = currlen;
                }
                low--;
                high++;
            }
        }
    }

    return s.substr(start, maxlen);
}

```



```

int main() {

    string s = "goingtodriveracecar";

    cout << longPalStr(s) << endl;

    return 0;

}

```

The screenshot shows the OneCompiler website interface. The editor contains the following C++ code:

```

1 //Question 15: Longest Palindromic Substring
2
3 #include <bits/stdc++.h>
4 using namespace std;
5
6 string longPalStr(string &s) {
7
8     if (s.size() == 0) {
9         return "";
10    }
11
12    int start = 0, maxlen = 1;
13
14    for (int i = 0; i < s.size(); i++) {
15        for (int j = 0; j <= i; j++) {
16            int low = i;
17            int high = i + j;
18
19            while (low >= 0 && high < s.size() && s[low] == s[high]) {
20                int currlen = high - low + 1;
21                if (currlen > maxlen) {
22                    start = low;
23                    maxlen = currlen;
24                }
25                low--;
26                high++;
27            }
28        }
29    }
30
31    return s.substr(start, maxlen);
32 }
33
34 int main() {
35     string s = "goingtodriveracecar";
36     cout << longPalStr(s) << endl;
37     return 0;
38 }

```

On the right side, the 'Output' section displays 'racecar'.

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.

Input: `arr[] = ["hello", "world"]`

Output: `-1`

Explanation: There's no common prefix in the given strings.

Question 16: Longest Common Prefix using Sorting

```
#include <bits/stdc++.h>

using namespace std;

string longCommonPrefix(vector<string>& arr) {

    if (arr.empty()) {
        return "-1";
    }

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

    string first = arr[1];
    string last = arr[arr.size()-1];
    int minlen = min(first.size(), last.size());

    int i = 0;

    while (i < minlen && first[i] == last[i]) {
        i++;
    }

    if (i == 0) return "-1";

    return first.substr(0, i);
}

int main() {
    vector<string> arr = {"sam", "samuel", "samsung"};
    cout << longCommonPrefix(arr) << endl;
```

```

return 0;
}

```

Time Complexity : $O(N \log N + M)$

Space Complexiy : $O(1)$

```

1 #include <bits/stdc++.h>
2 using namespace std;
3
4 string longCommonPrefix(vector<string>& arr) {
5     if (arr.empty()) {
6         return "-1";
7     }
8     sort(arr.begin(), arr.end());
9
10    string first = arr[1];
11    string last = arr[arr.size()-1];
12    int minlen = min(first.size(), last.size());
13
14    int i = 0;
15    while (i < minlen && first[i] == last[i]) {
16        i++;
17    }
18    if (i == 0) return "-1";
19    return first.substr(0, i);
20 }
21
22 int main() {
23     vector<string> arr = {"sam", "samuel", "samsung"};
24     cout << longCommonPrefix(arr) << endl;
25     return 0;
26 }

```

STDIN
Input for the program (Optional)

Output:
sam

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]

Question 17: Delete middle element of a stack

```

#include <bits/stdc++.h>

using namespace std;

```

```
void deleteMid(stack<int> &st ,int N){
```

```
    int popped_elements = N-st.size();
```

```
    if(popped_elements == (N/2)){
```

```
        st.pop();
```

```
        return;
```

```
    }
```

```
    int x=st.top();
```

```
    st.pop();
```

```
    deleteMid(st,N);
```

```
    st.push(x);
```

```
}
```

```
int main()
```

```
{
```

```
    stack<int> st;
```

```
    st.push(5);
```

```
    st.push(4);
```

```
    st.push(3);
```

```
    st.push(2);
```

```
    st.push(1);
```

```
    int N = st.size();
```

```
    deleteMid(st,N);
```

```
    while (!st.empty()) {
```

```

    cout << st.top() << " ";

    st.pop();
}

return 0;
}

```

Time Complexity: $O(N)$

Space Complexity : $O(N)$

The screenshot shows the OneCompiler website interface. The main editor displays a C++ program titled "42xhpsmr". The code implements a function to delete the middle element of a stack. The output window shows the result: "1 2 4 5".

```

//Question 17: Delete middle element of a stack
#include <bits/stdc++.h>
using namespace std;

void deleteMid(stack<int> &st, int N){
    int popped_elements = N-st.size();
    if(popped_elements == (N/2)){
        st.pop();
        return;
    }
    int x=st.top();
    st.pop();
    deleteMid(st,N);
    st.push(x);
}

int main()
{
    stack<int> st;
    st.push(5);
    st.push(4);
    st.push(3);
    st.push(2);
    st.push(1);

    int N = st.size();
    deleteMid(st,N);

    while (!st.empty()) {
        cout << st.top() << " ";
        st.pop();
    }
    return 0;
}

```

Output: 1 2 4 5

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: $4 \rightarrow 5$

$5 \rightarrow 25$

$2 \rightarrow 25$

$25 \rightarrow -1$

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

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

```
#include <bits/stdc++.h>

using namespace std;

void NGE(vector<int>&nums,vector<int>&nge){
    stack<int>st;
    for(int i = nums.size()-1;i>=0;i--){
        while(!st.empty() && nums[st.top()]<=nums[i]){
            st.pop();
        }
        if(!st.empty()){
            nge[i] = nums[st.top()];
        }
        st.push(i);
    }
}
```

```
int main()
{
    vector<int>nums = {13,7,6,12};
    vector<int>nge(nums.size(),-1);
    NGE(nums,nge);
    for(int i = 0;i<nums.size();i++){
        cout << nums[i] << " -> " << nge[i] << endl;
```

```

}

return 0;

}

```

Time Complexity : $O(n)$

Space Complexity : $O(n)$

```

1 //Question 18: Next Greater Element (NGE) for every element in given Array
2
3 #include <bits/stdc++.h>
4 using namespace std;
5 void NGE(vector<int>&nums, vector<int>&nge){
6     stack<int>st;
7     for(int i = nums.size()-1; i>=0; i--){
8         while(!st.empty() && nums[st.top()]<nums[i]){
9             st.pop();
10        }
11        if(!st.empty()){
12            nge[i] = nums[st.top()];
13        }
14        st.push(i);
15    }
16 }
17
18 int main()
19 {
20     vector<int>nums = {13,7,6,12};
21     vector<int>nge(nums.size(), -1);
22     NGE(nums, nge);
23     for(int i = 0; i<nums.size(); i++){
24         cout << nums[i] << " -> " << nge[i] << endl;
25     }
26     return 0;
27 }
28
29 // Time Complexity : O(n)
30 // Space Complexity : O(n)

```

Output:

```

13 -> -1
7 -> 12
6 -> 12
12 -> -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.

Question 19: Right View of Binary Tree

```
#include <bits/stdc++.h>
```

```
using namespace std;
```

```
struct Node {
```

```
    int data;
```

```
    Node *left;
```

```
    Node *right;
```

```
Node(int val) {  
    data = val;  
    left = NULL;  
    right = NULL;  
}  
};
```

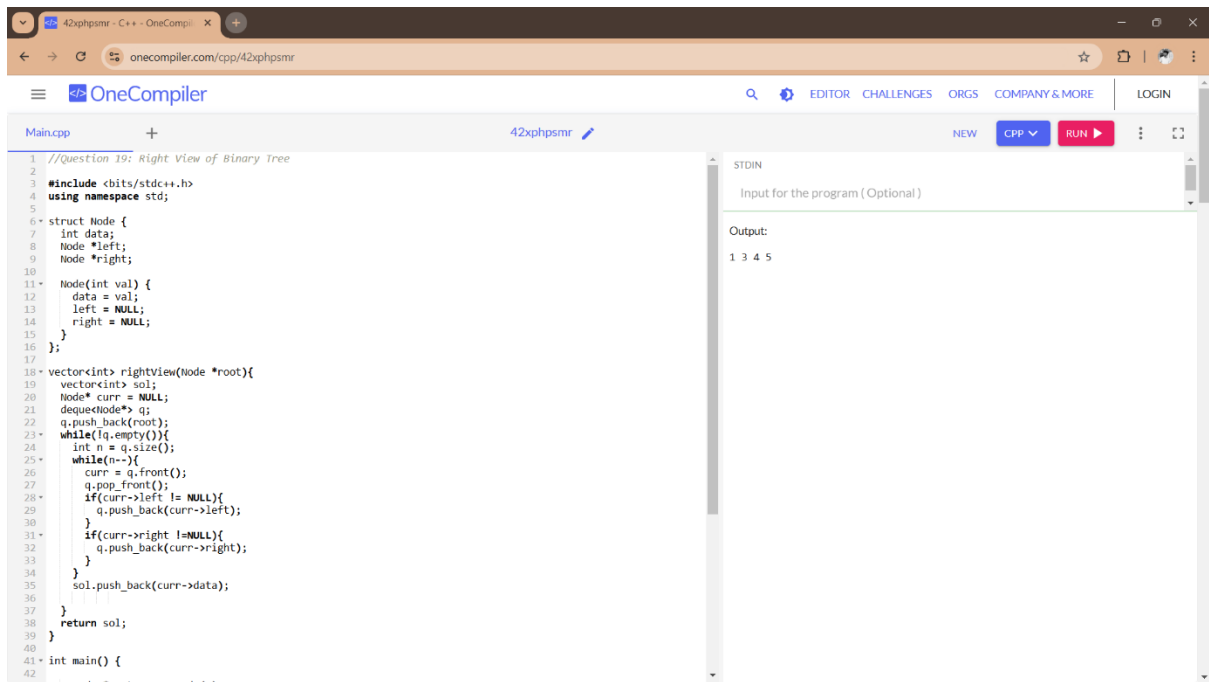
```
vector<int> rightView(Node *root){  
    vector<int> sol;  
    Node* curr = NULL;  
    deque<Node*> q;  
    q.push_back(root);  
    while(!q.empty()){  
        int n = q.size();  
        while(n--){  
            curr = q.front();  
            q.pop_front();  
            if(curr->left != NULL){  
                q.push_back(curr->left);  
            }  
            if(curr->right != NULL){  
                q.push_back(curr->right);  
            }  
        }  
        sol.push_back(curr->data);  
    }  
    return sol;  
}
```



```
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->left->right = new Node(5);  
  
    vector<int> sol;  
    sol = rightView(root);  
  
    for(auto i:sol){  
        cout << i << ' ';  
    }  
  
    return 0;  
}
```

Time Complexity : $O(N)$

Space Complexity : $O(N)$



```
1 //Question 19: Right View of Binary Tree
2
3 #include <bits/stdc++.h>
4 using namespace std;
5
6 struct Node {
7     int data;
8     Node *left;
9     Node *right;
10
11     Node(int val) {
12         data = val;
13         left = NULL;
14         right = NULL;
15     }
16 };
17
18 vector<int> rightView(Node *root){
19     vector<int> sol;
20     Node* curr = NULL;
21     deque<Node*> q;
22     q.push_back(root);
23     while(!q.empty()){
24         int n = q.size();
25         while(n--){
26             curr = q.front();
27             q.pop_front();
28             if(curr->left != NULL){
29                 q.push_back(curr->left);
30             }
31             if(curr->right != NULL){
32                 q.push_back(curr->right);
33             }
34             sol.push_back(curr->data);
35         }
36     }
37     return sol;
38 }
39
40 int main() {
```

STDIN

Input for the program (Optional)

Output:

1 3 4 5

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.

Question 20: Maximum Depth or Height of Binary Tree

```
#include <bits/stdc++.h>
```

```
using namespace std;
```

```
struct Node {
```

```
    int data;
```

```
    Node *left;
```

```
    Node *right;
```

```
Node(int val) {
```

```
    data = val;
```

```
    left = NULL;
```

```
    right = NULL;
```

```

    }
};

int maxDepth(Node *node) {
    if (node == nullptr){
        return 0;
    }

    int left = maxDepth(node->left);
    int right = maxDepth(node->right);

    return max(left, right) + 1;
}

int main() {

    Node *root = new Node(1);

    root->left = new Node(2);
    root->right = new Node(3);
    root->right->left = new Node(4);
    root->right->right = new Node(5);

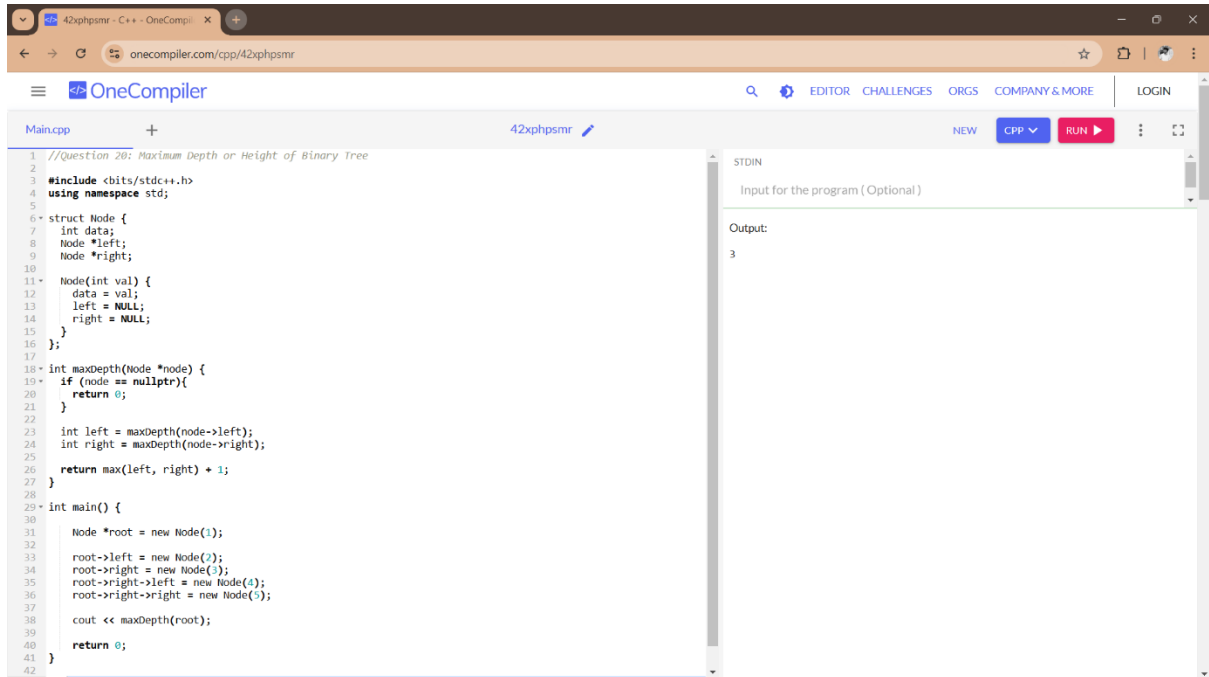
    cout << maxDepth(root);

    return 0;
}

```

Time Complexity : $O(N)$

Space Complexity: $O(N)$



The screenshot shows the OneCompiler C++ IDE interface. The editor displays a C++ program for finding the maximum depth of a binary tree. The program includes a `Node` struct with `data`, `left`, and `right` pointers. A recursive function `maxDepth` is defined to calculate the height of the tree. The `main` function constructs a binary tree with root 1, left child 2, right child 3, and further nodes 4 and 5. The output of the program is 3, which is displayed in the output window on the right.

```
1 //Question 20: Maximum Depth or Height of Binary Tree
2
3 #include <bits/stdc++.h>
4 using namespace std;
5
6 struct Node {
7     int data;
8     Node *left;
9     Node *right;
10 }
11
12 Node(int val) {
13     data = val;
14     left = NULL;
15     right = NULL;
16 }
17
18 int maxDepth(Node *node) {
19     if (node == nullptr) {
20         return 0;
21     }
22     int left = maxDepth(node->left);
23     int right = maxDepth(node->right);
24     return max(left, right) + 1;
25 }
26
27
28
29 int main() {
30     Node *root = new Node(1);
31
32     root->left = new Node(2);
33     root->right = new Node(3);
34     root->right->left = new Node(4);
35     root->right->right = new Node(5);
36
37     cout << maxDepth(root);
38
39     return 0;
40 }
41
42
```

Output: 3