

# Date of Experiment: 24/01/2023

**Title:–**

1. Find maximum and minimum of an array.
2. Find maximum and second maximum in an array whose time complexity is better than (2n +1)

# Theory:–

1.

Initialize values of min and max as minimum and maximum of the first two elements respectively. Starting from 3rd, compare each element with max and min, and change max and min accordingly (i.e., if the element is smaller than min then change min, else if the element is greater than max then change max, else ignore the element)

2.

Initialize the first as 0(i.e, index of arr[0] element, Start traversing the array from array[1], If the current element in array say arr[i] is greater than first. Then update first and second as,

second = first

first = arr[i]

If the current element is in between first and second,then update second to store the value of current variable as second = arr[i]. Return the value stored in second.

# CODE :

# Find Max and Minimum

#include<iostream>

using *namespace* std;

*struct* Pair

{

*int* min;

*int* max;

};

Pair getMinMax(*int* *arr*[], *int* *n*)

{

*struct* Pair minmax;

*int* i;

    if (*n* == 1)

    {

        minmax.max = *arr*[0];

        minmax.min = *arr*[0];

        return minmax;

    }

    if (*arr*[0] > *arr*[1])

    {

        minmax.max = *arr*[0];

        minmax.min = *arr*[1];

    }

    else

    {

        minmax.max = *arr*[1];

        minmax.min = *arr*[0];

    }

    for(i = 2; i < *n*; i++)

    {

        if (*arr*[i] > minmax.max)

            minmax.max = *arr*[i];

        else if (*arr*[i] < minmax.min)

            minmax.min = *arr*[i];

    }

    return minmax;

}

*int* main()

{

*int* arr[] = { 1000, 11, 445,

                1, 330, 3000 };

*int* arr\_size = 6;

*struct* Pair minmax = getMinMax(arr, arr\_size);

    cout << "Minimum element is "

        << minmax.min << endl;

    cout << "Maximum element is "

        << minmax.max;

    return 0;

}

# Find Second Max

#include <iostream>

using *namespace* std;

*int* secondLargest(*int* *arr*[], *int* *n*) {

*int* largest = 0, secondLargest = -1;

    // finding the largest element in the array

    for (*int* i = 1; i < *n*; i++) {

        if (*arr*[i] > *arr*[largest])

            largest = i;

    }

    for (*int* i = 0; i < *n*; i++) {

        if (*arr*[i] != *arr*[largest]) {

            if (secondLargest == -1)

                secondLargest = i;

            else if (*arr*[i] > *arr*[secondLargest])

                secondLargest = i;

        }

    }

    return secondLargest;

}

*int* main() {

*int* arr[] = { 12, 35, 1, 10, 34, 1 };

*int* n = sizeof(arr)/sizeof(arr[0]);

*int* second\_Largest = secondLargest(arr, n);

    if (second\_Largest == -1)

        cout << "Second largest didn't exit\n";

    else

        cout << "Second largest : " << arr[second\_Largest];

}

# Analysis of Algorithm:

**1**

**Time Complexity:** O(n)

In this method, the total number of comparisons is **1 + 2(n-2)** in the worst case and **1 + n – 2** in the best case.   
In the above implementation, the worst case occurs when elements are sorted in descending order and the best case occurs when elements are sorted in ascending order.

**2.**

# **Time Complexity: O(n).** Only one traversal of the array is needed

# Experiment and Results:

# Max and Minimum :

# Normal case :

# 

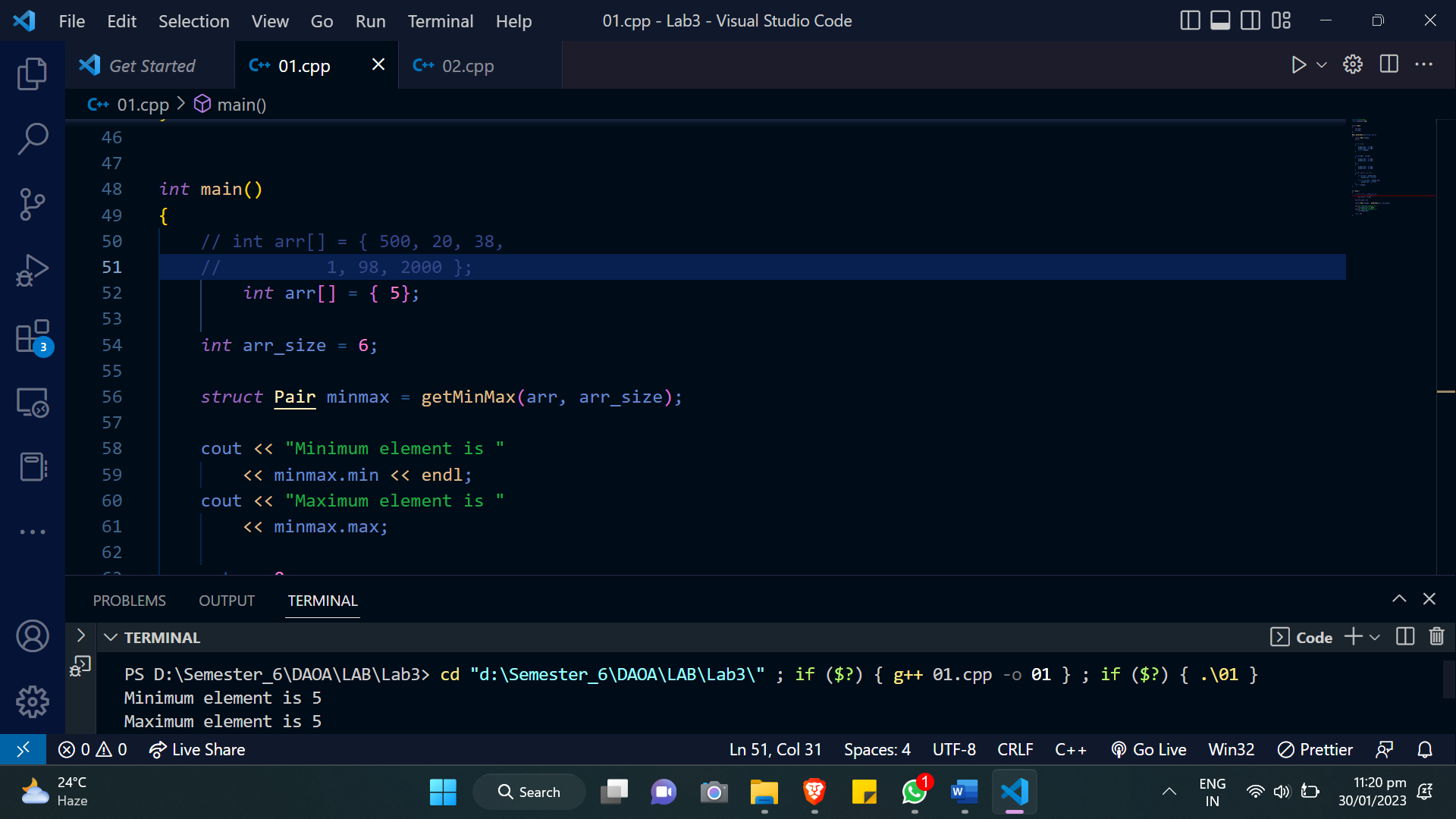
# When only one element is present:

# 

# When all elements are same:

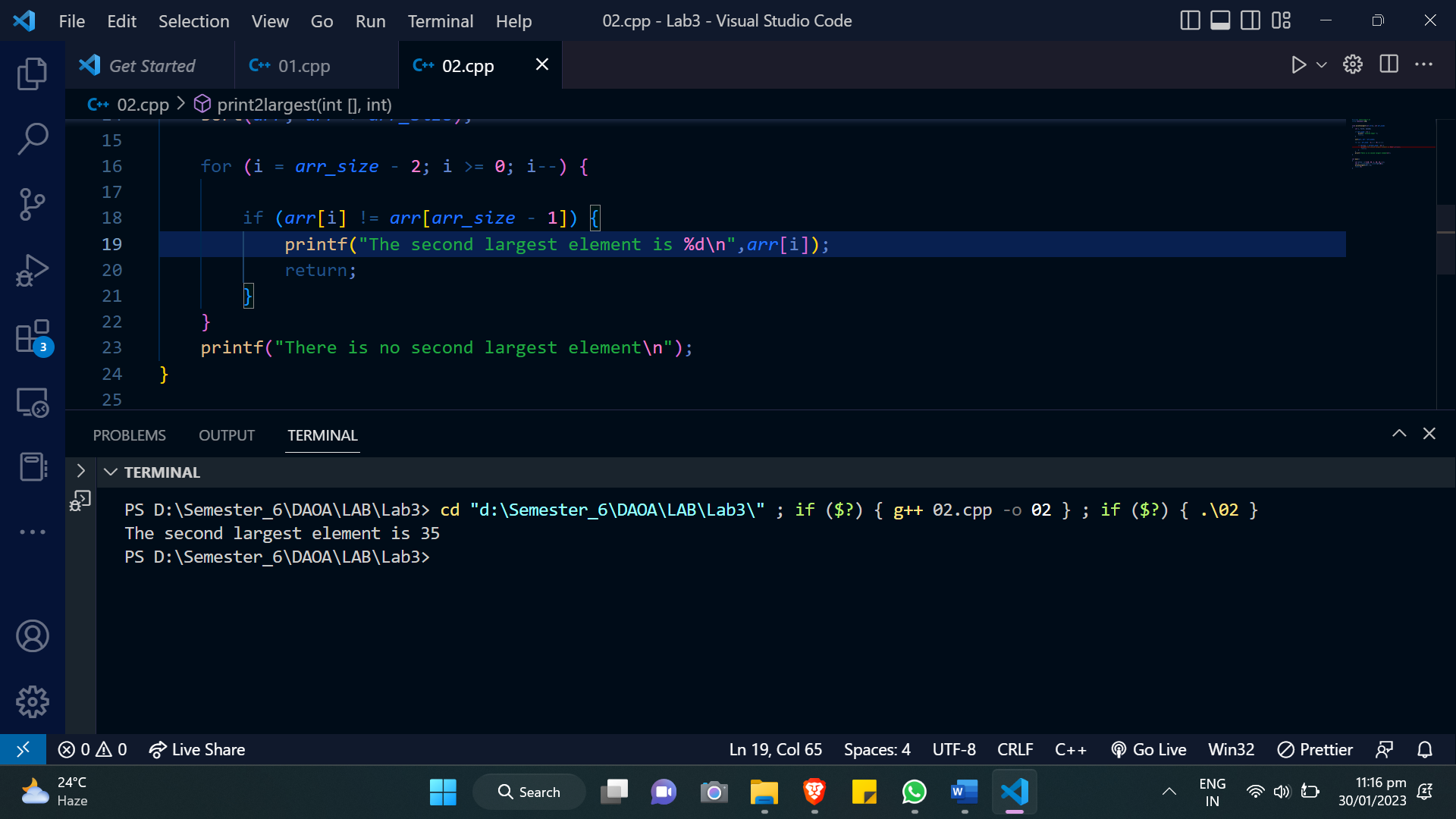
# 

# When only 1 element is present:

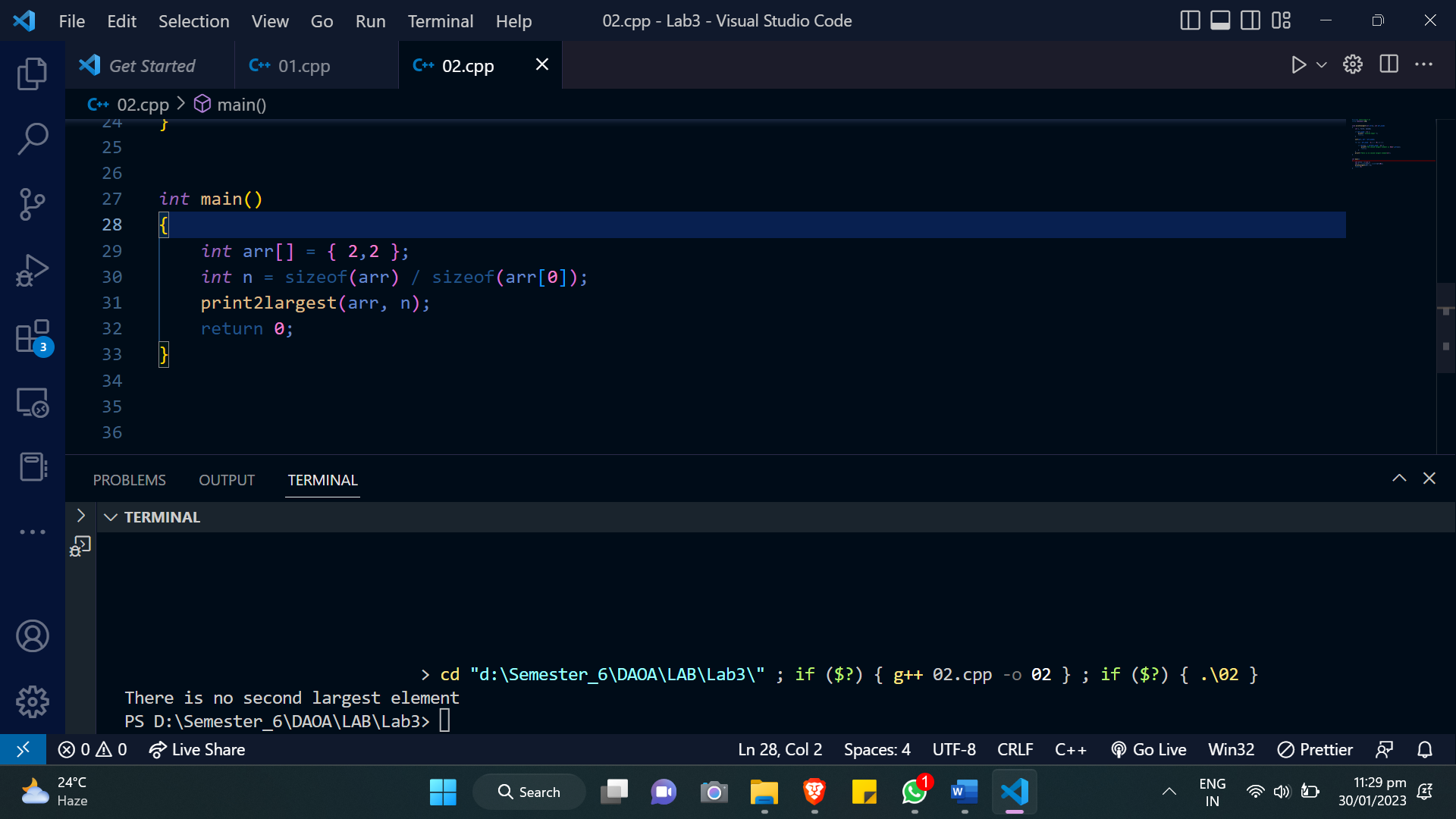


# Second maximum

# Normal case:



**When all elements are same:**



# Conclusion:

This experiment aims to find maximum, second maximum, and minimum values in an array. Also, we considered a lot of edge cases as far as array elements are concerned. In one single traversal, the second maximum element was also retrieved. All the programs were running in O(n) worst case time complexity.