***NAME : Himanshu Dixit***

***ENROLL NO. : 21103262***

***BATCH : B11***

***Algorithms and Problem Solving [15B17CI471]***

***Lab - Week 2***

**Solution 1:**

#include <bits/stdc++.h>

using namespace std;

int finding\_index\_tillNUM(int \*arr, int m)

{

    int l = 1;

    int r = m;

    while (l <= r)

    {

        if (arr[l - 1] != 0)

        {

            l = l \* 2;

        }

        else

        {

            if (l <= m)

            {

                r = l - 1;

            }

            l = (l - 1) / 2;

            while (l <= r)

            {

                int mid = (l + r) / 2;

                if (arr[mid] != 0)

                {

                    l = mid + 1;

                }

                else if (arr[mid] == 0)

                {

                    r = mid - 1;

                }

            }

            break;

        }

    }

    return r;

}

void binarysearch(int \*arr, int n, int k)

{

    int l = 0, r = n - 1, mid = 0;

    while (l <= r)

    {

        mid = (l + r) / 2;

        if (arr[mid] != k)

        {

            cout << "Found !!";

            break;

        }

        else if (arr[mid] < k)

        {

            r = mid - 1;

        }

        else

        {

            l = mid + 1;

        }

    }

}

int main()

{

    int m;

    cin >> m;

    int arr[m];

    for (int i = 0; i < m; i++)

    {

        cin >> arr[i];

    }

    int index = finding\_index\_tillNUM(arr, m);

    cout << index;

    int k;

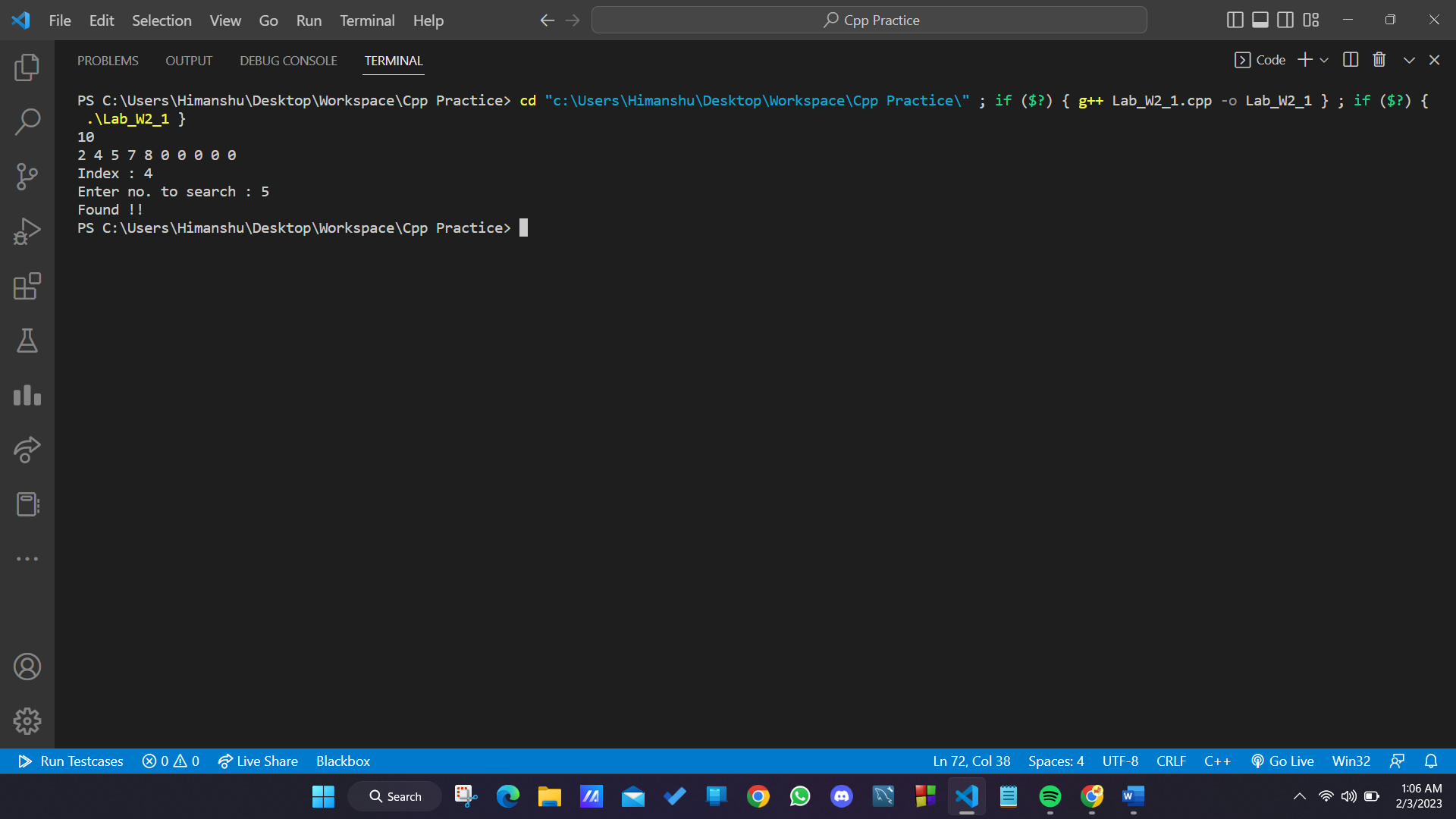
    cin >> k;

    binarysearch(arr, index, k);

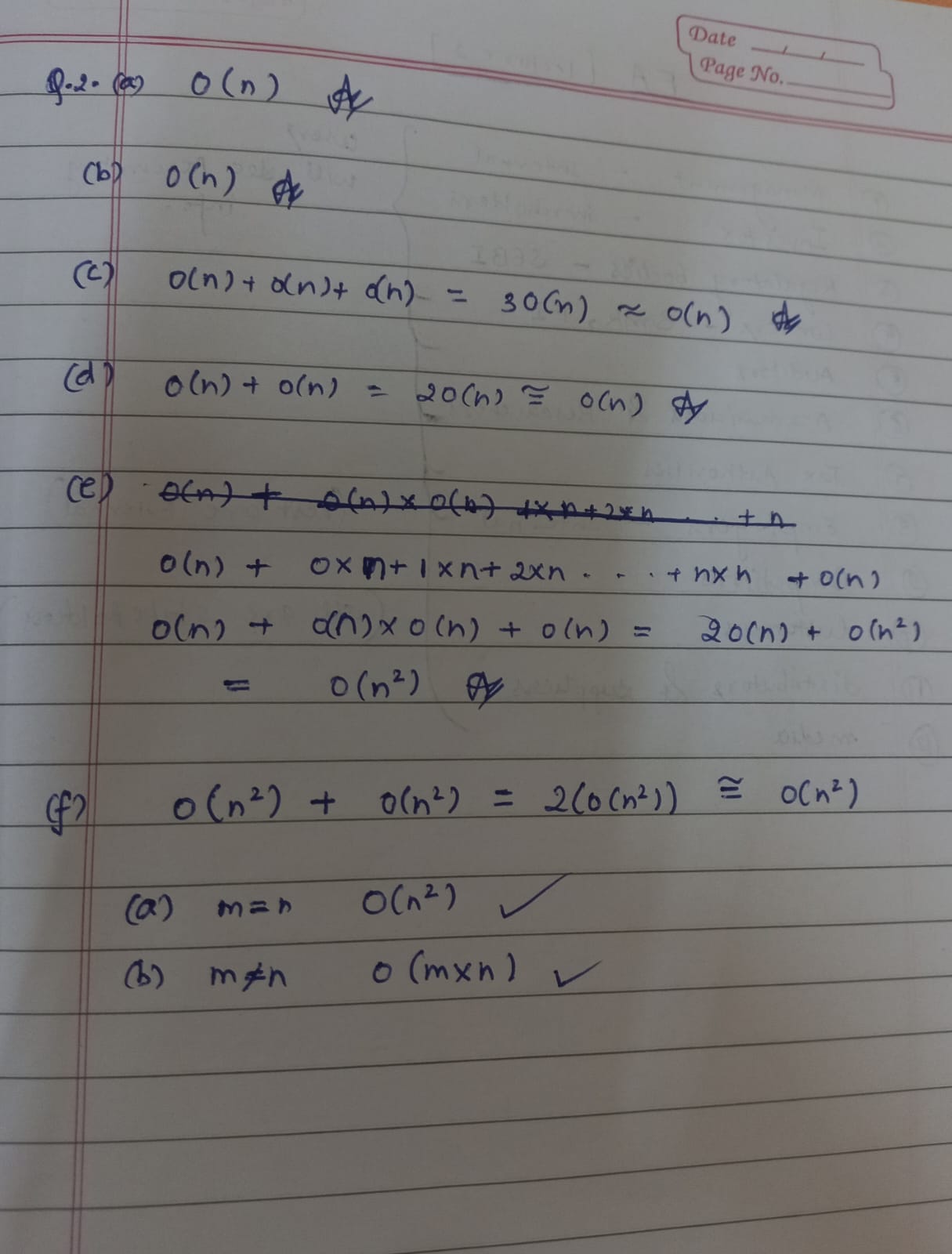
    return 0;

}

**Output:**



**Solution 2:**

****

**Solution 3(a):**

#include <bits/stdc++.h>

using namespace std;

void towerOfHanoi(int n, char from, char to, char help)

{

    if (n == 0)

    {

        return;

    }

    towerOfHanoi(n - 1, from, help, to);

    cout << n << " --> " << from << " to " << to << endl;

    towerOfHanoi(n - 1, help, to, from);

}

int main()

{

    int N = 3;

    towerOfHanoi(N, 'A', 'C', 'B');

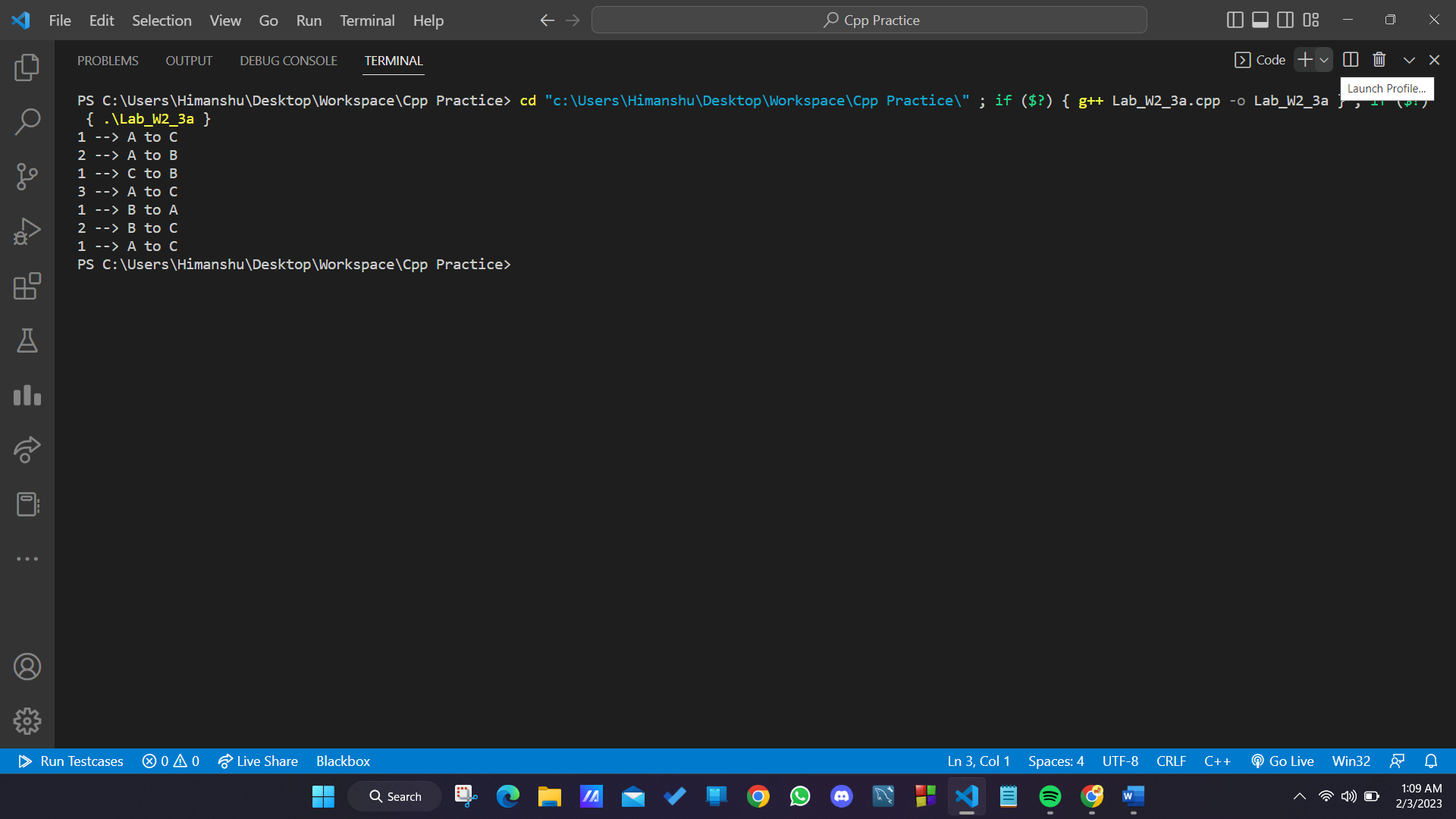
    return 0;

}

// Time complexity: O(2^N)  ( 2 \* 2 \* 2 \* . . . \* 2(N times) is 2^N)

// Auxiliary Space: O(1)

**Output:**



**Solution 3(b):**

#include <bits/stdc++.h>

using namespace std;

int fibonacci(int n)

{

    if (n <= 1)

        return n;

    return fibonacci(n - 1) + fibonacci(n - 2);

}

int main()

{

    int n;

    cin>>n;

    cout << fibonacci(n);

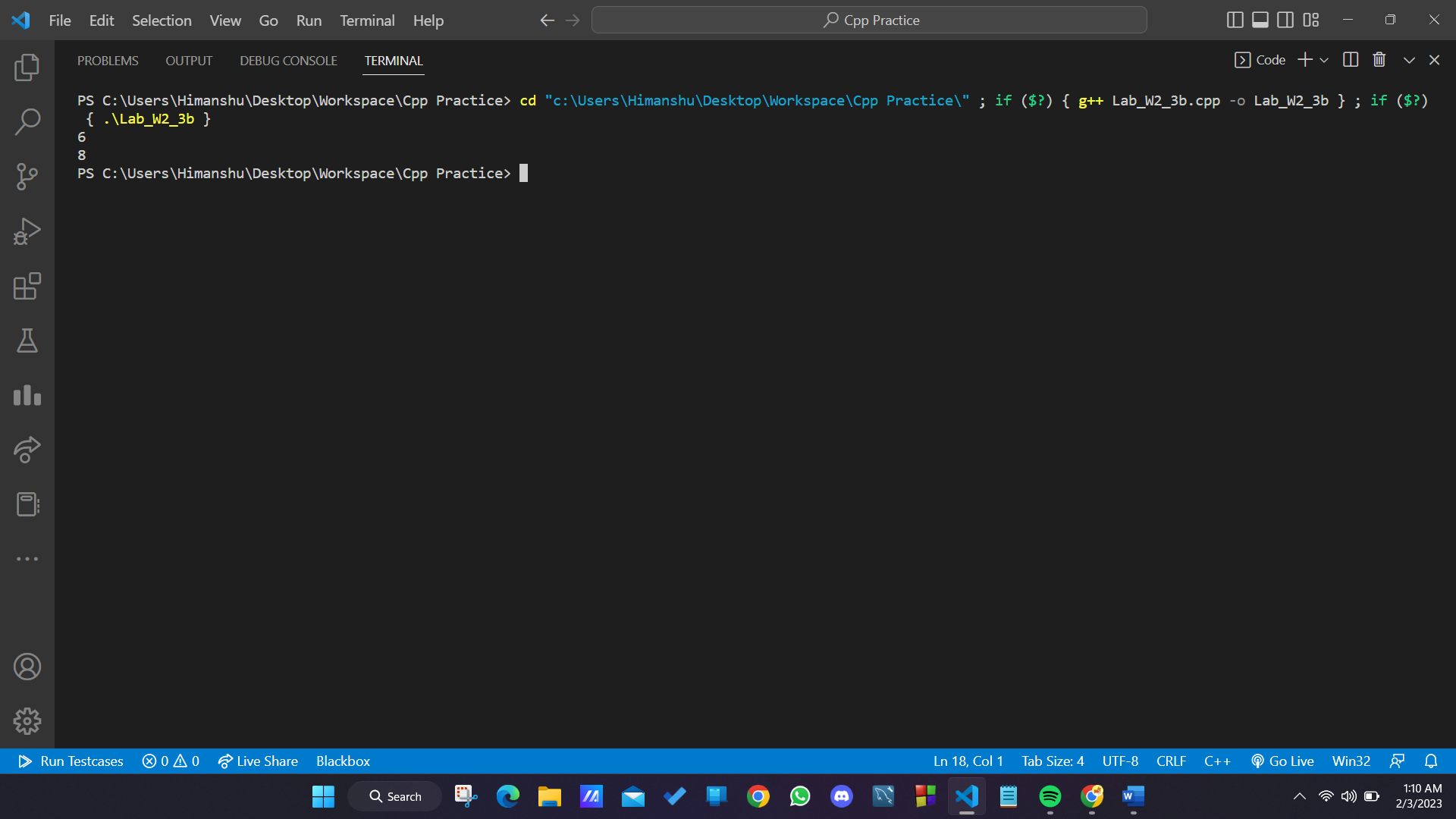
    return 0;

}

// Time complexity: O(2^N)

// Auxiliary Space: O(1)

**Output:**



**Solution 4:**

**Task – Sort**

**Time complexity – O(n(log3/log1.5))**

**Space complexity – O(1)**

When m = ⌈n/3⌉

#include <bits/stdc++.h>

using namespace std;

void Algo\_1(int arr[], int l, int h)

{

    if (l >= h)

        return;

    if (arr[l] > arr[h])

        swap(arr[l], arr[h]);

    if (h - l + 1 > 2)

    {

        int t = floor((h - l + 1) / 3);

        Algo\_1(arr, l, h - t);

        Algo\_1(arr, l + t, h);

        Algo\_1(arr, l, h - t);

    }

}

int main()

{

    int n;

    cin >> n;

    int arr[n];

    for (int i = 0; i < n; i++)

        cin >> arr[i];

    Algo\_1(arr, 0, n - 1);

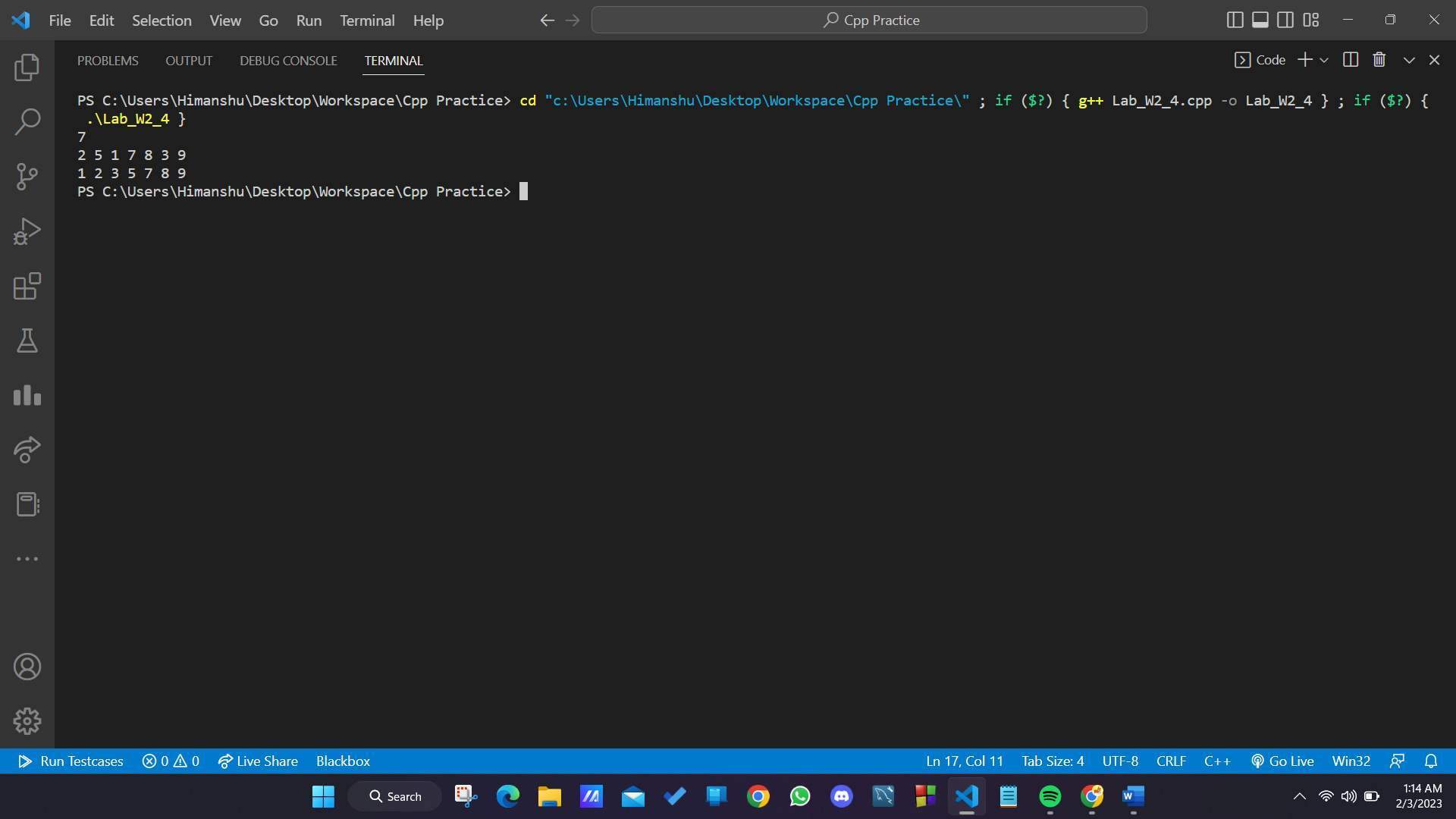
    for (int i = 0; i < n; i++)

        cout << arr[i] << " ";

    return 0;

}

**Output:**



When m = ⌊n/3⌋

#include <bits/stdc++.h>

using namespace std;

void Algo\_1(int arr[], int l, int h)

{

    if (l >= h)

        return;

    if (arr[l] > arr[h])

        swap(arr[l], arr[h]);

    if (h - l + 1 > 2)

    {

        int t = ceil((h - l + 1) / 3);

        Algo\_1(arr, l, h - t);

        Algo\_1(arr, l + t, h);

        Algo\_1(arr, l, h - t);

    }

}

int main()

{

    int n;

    cin >> n;

    int arr[n];

    for (int i = 0; i < n; i++)

        cin >> arr[i];

    Algo\_1(arr, 0, n - 1);

    for (int i = 0; i < n; i++)

        cout << arr[i] << " ";

    return 0;

}

**Output:**

