***NAME : Himanshu Dixit***

***ENROLL NO. : 21103262***

***BATCH : B11***

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

***Lab - Week 3***

**Solution 1:**

#include <bits/stdc++.h>

using namespace std;

int cuberoot(int n)

{

    int low = 0;

    int high = n;

    int ans = 0;

    while (low < high)

    {

        int mid = (low + high) / 2;

        if (pow(mid, 3) == n)

            return mid;

        if (pow(mid, 3) < n)

        {

            low = mid + 1;

            ans = mid;

        }

        else

            high = high - 1;

    }

    return ans;

}

int main()

{

    int n;

    cout << "Enter the number : ";

    cin >> n;

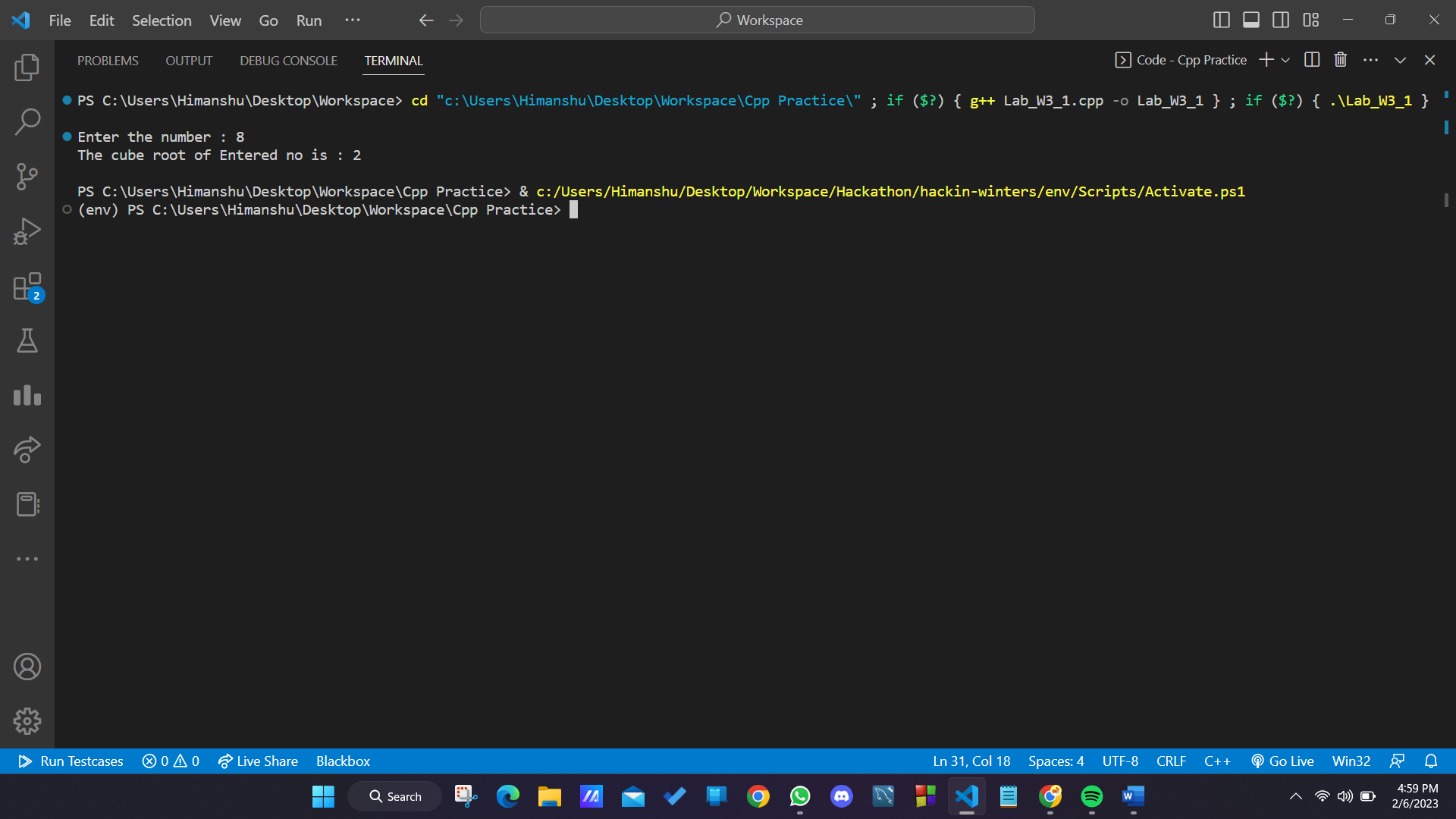
    int res = cuberoot(n);

    cout << "The cube root of Entered no is : " << res;

    return 0;

}

**Output:**



**Complexity: O(log(n))**

**Solution 2:**

#include <bits/stdc++.h>

using namespace std;

void findmissing(int arr[], int low, int high)

{

    if (low > high)

        cout << -1;

    if (low == high)

    {

        cout << arr[low];

        return;

    }

    int mid = low + (high - low) / 2;

    if (mid % 2 == 0)

    {

        if (arr[mid] == arr[mid + 1])

            findmissing(arr, mid + 2, high);

        else

            findmissing(arr, low, mid);

    }

    else

    {

        if (arr[mid] == arr[mid - 1])

            findmissing(arr, mid + 1, high);

        else

            findmissing(arr, low, mid - 1);

    }

}

int main()

{

    int n;

    cout << "Enter the no of elements : ";

    cin >> n;

    int arr[n];

    cout << "Enter the elements : ";

    for (int i = 0; i < n; i++)

    {

        cin >> arr[i];

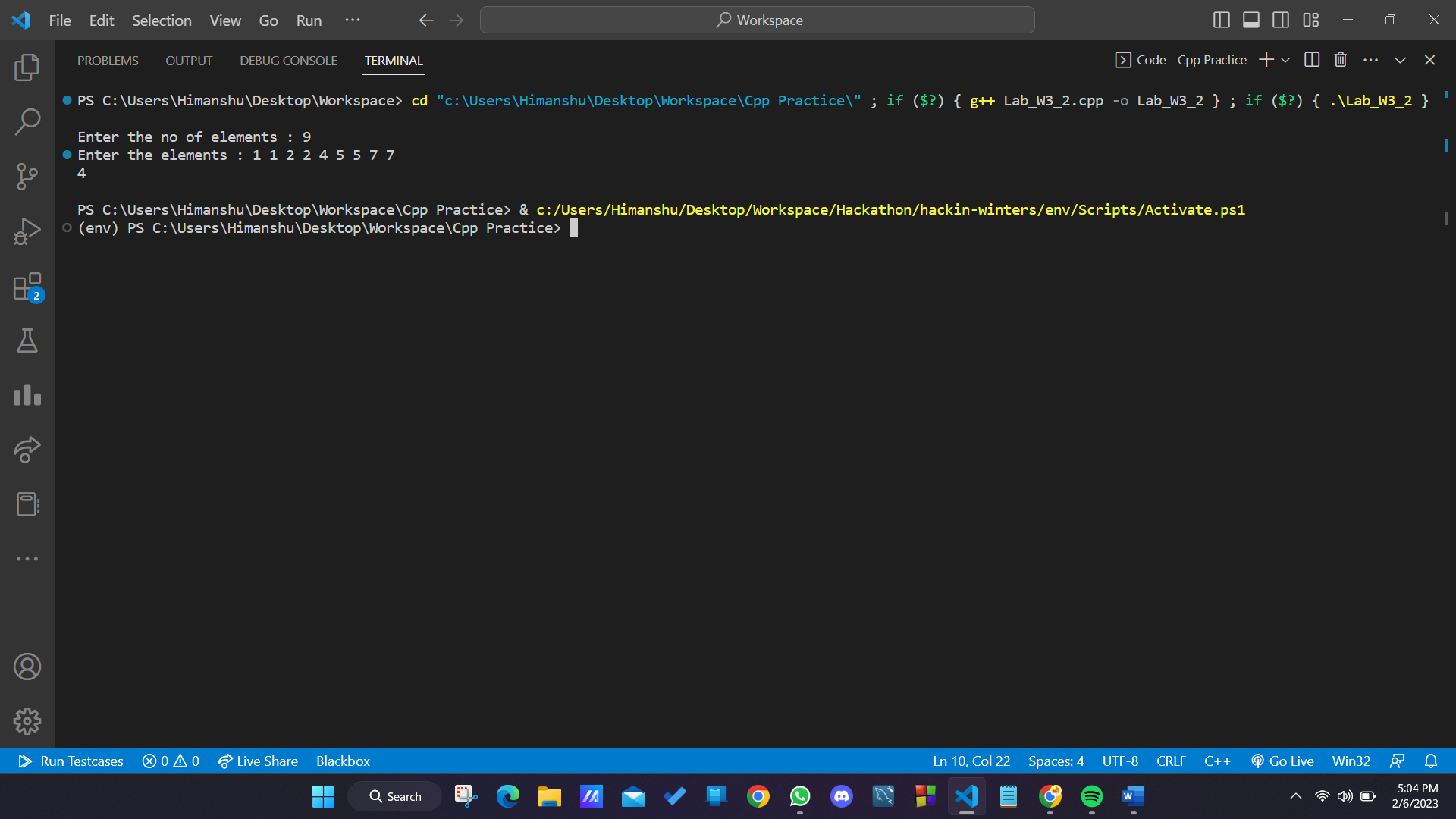
    }

    findmissing(arr, 0, n - 1);

    return 0;

}

**Output:**



**Solution 3:**

#include <bits/stdc++.h>

using namespace std;

void kclosest(int \*x, int \*y, int n, int k)

{

    priority\_queue<pair<long double, int>, vector<pair<long double, int>>, greater<pair<long double, int>>> pq;

    for (int i = 0; i < n; i++)

    {

        pq.push(make\_pair((x[i] \* x[i] + y[i] \* y[i]), i));

    }

    while (k--)

    {

        auto p = pq.top();

        pq.pop();

        cout << x[p.second] << " " << y[p.second] << endl;

    }

}

int main()

{

    int n;

    cout << "Total points : ";

    cin >> n;

    int x[n], y[n];

    for (int i = 0; i < n; i++)

    {

        cin >> x[i] >> y[i];

    }

    int k;

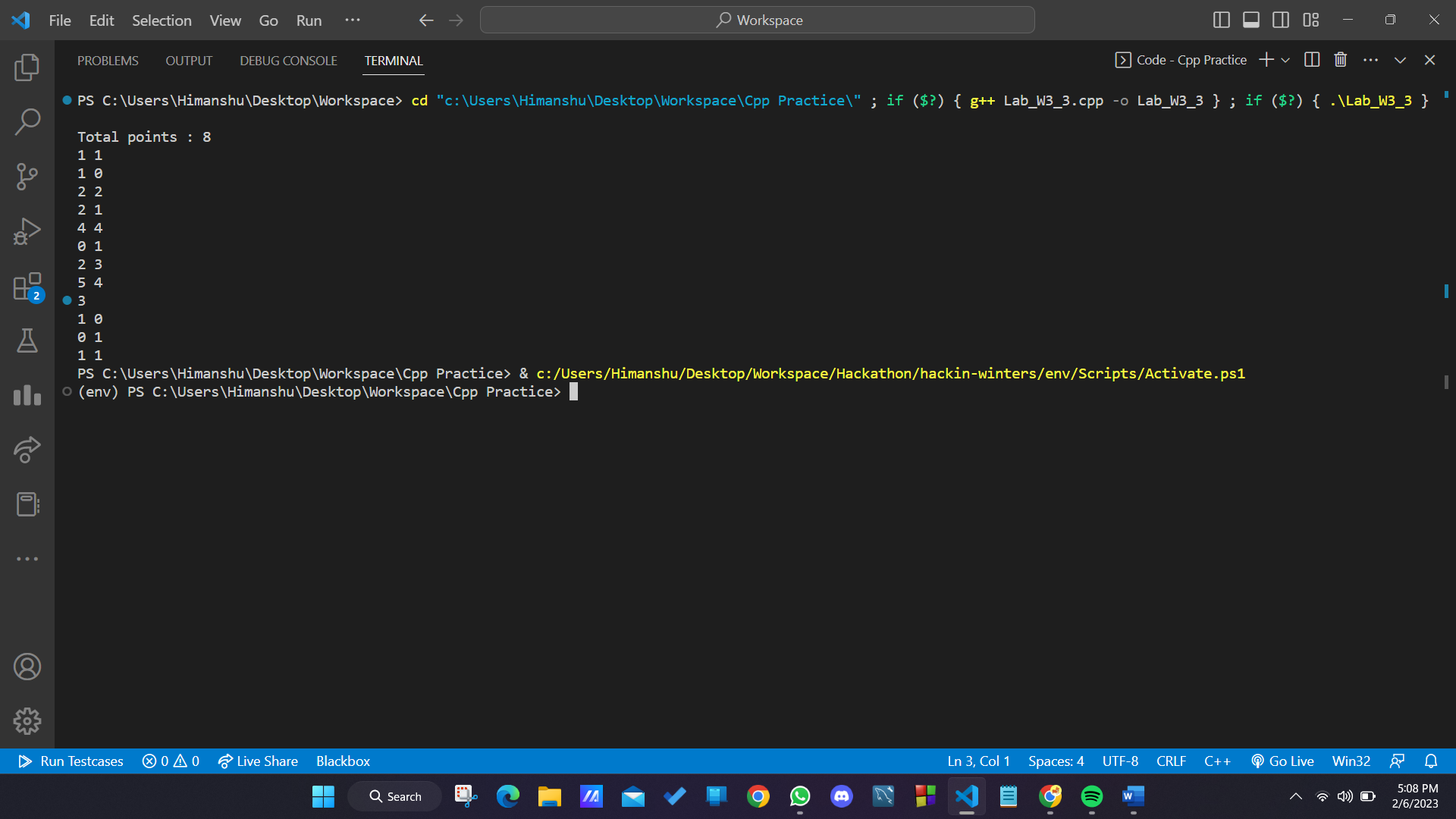
    cin >> k;

    kclosest(x, y, n, k);

    return 0;

}

**Output:**



**Solution 4:**

#include <bits/stdc++.h>

using namespace std;

void insertionsort(int arr[], int l, int r, int &comp)

{

    for (int i = l + 1; i <= r; i++)

    {

        int val = arr[i];

        int j = i;

        while (j > l && arr[j - 1] > val)

        {

            comp++;

            arr[j] = arr[j - 1];

            j -= 1;

        }

        comp++;

        arr[j] = val;

    }

}

int partition(int \*arr, int l, int r, int &comp)

{

    int pivot = arr[r];

    int i = l, j = l;

    while (j < r)

    {

        if (arr[j] <= pivot)

        {

            swap(arr[i], arr[j]);

            i++;

        }

        j++;

        comp++;

    }

    swap(arr[i], arr[r]);

    return i;

}

void quicksort(int \*arr, int l, int r, int &comp)

{

    if (l < r)

    {

        int p = partition(arr, l, r, comp);

        quicksort(arr, 0, p - 1, comp);

        quicksort(arr, p + 1, r, comp);

    }

}

void hybridised\_quicksort(int \*arr, int l, int r, int &comp)

{

    if (l < r)

        while (l < r)

        {

            comp++;

            if (r - l + 1 <= 10)

            {

                insertionsort(arr, l, r, comp);

                break;

            }

            else

            {

                int pivot = partition(arr, l, r, comp);

                if (pivot - l < r - pivot)

                {

                    hybridised\_quicksort(arr, l, pivot - 1, comp);

                    l = pivot + 1;

                }

                else

                {

                    hybridised\_quicksort(arr, pivot + 1, r, comp);

                    r = pivot - 1;

                }

            }

        }

}

int main()

{

    int n;

    cin >> n;

    int arr[n];

    for (int i = 0; i < n; i++)

    {

        cin >> arr[i];

    }

    int \*temp = arr;

    int comp = 0;

    quicksort(arr, 0, n - 1, comp);

    for (int i = 0; i < n; i++)

    {

        cout << arr[i] << " ";

    }

    cout << endl

         << comp << " comparison takes is normal quicksort" << endl;

    comp = 0;

    hybridised\_quicksort(temp, 0, n - 1, comp);

    for (int i = 0; i < n; i++)

    {

        cout << temp[i] << " ";

    }

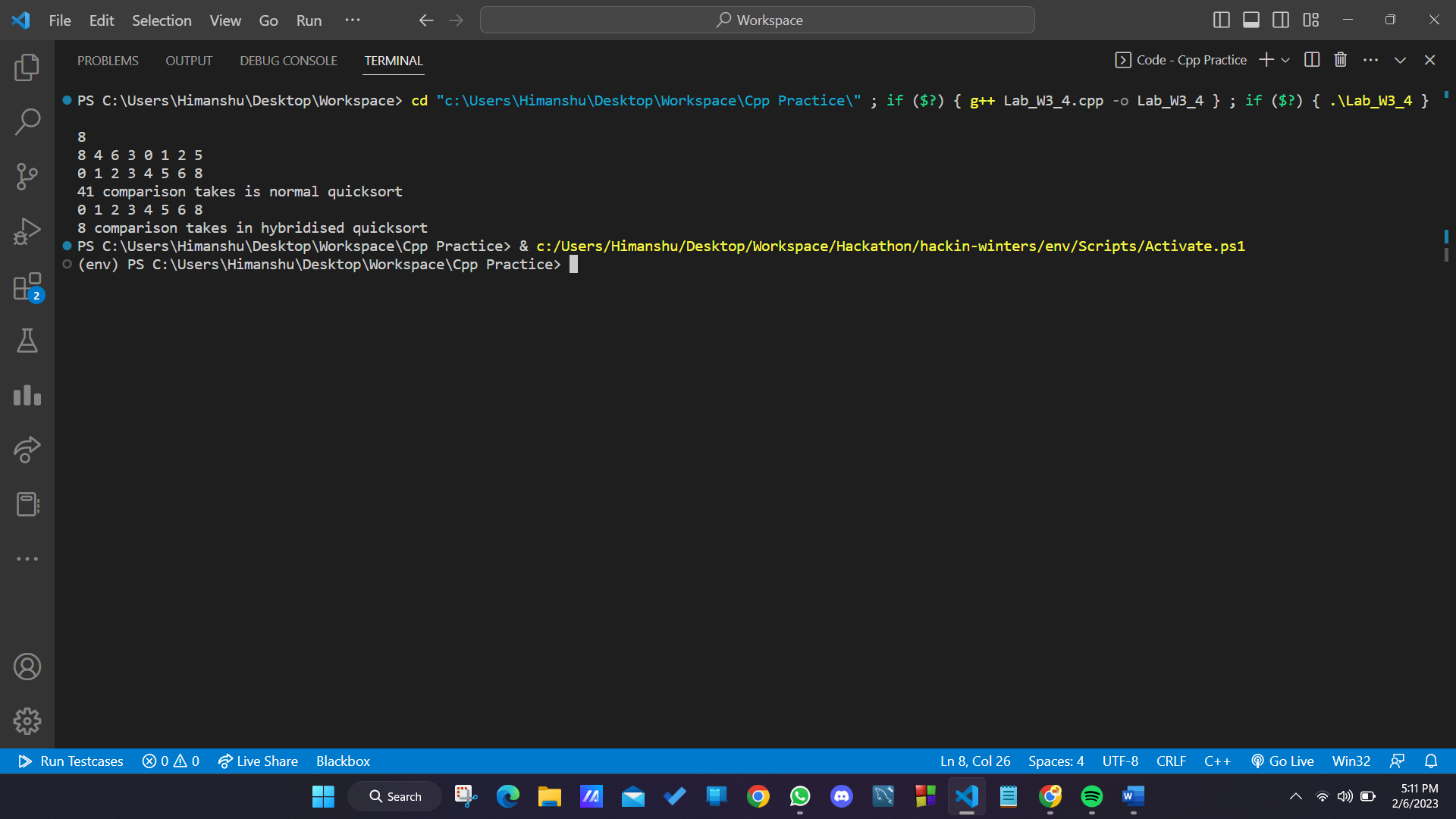
    cout << endl

         << comp << " comparison takes in hybridised quicksort";

    return 0;

}

**Output:**



**Solution 5:**

#include <bits/stdc++.h>

using namespace std;

void search(int \*arr, int n, int k, int \*a)

{

    int low = 0;

    int high = n - 1;

    int mid;

    while (low < high)

    {

        int mid = (low + high) / 2;

        if (arr[mid] == k)

        {

            int x = mid;

            while (arr[x] == k and x >= 0)

            {

                x--;

            }

            a[0] = x + 1;

            x = mid;

            while (arr[x] == k and x < n)

            {

                x++;

            }

            a[1] = x - 1;

        }

        if (arr[mid] < k)

        {

            low = mid + 1;

        }

        else

            high = mid - 1;

    }

}

int main()

{

    int n;

    cout << "Enter the size : ";

    cin >> n;

    int arr[n];

    cout << "Enter the elements : ";

    for (int i = 0; i < n; i++)

    {

        cin >> arr[i];

    }

    int k;

    cout << "Enter the element to be searched : ";

    cin >> k;

    int ans[2] = {-1, -1};

    search(arr, n, k, ans);

    if (ans[0] != -1 and ans[1] != -1)

    {

        cout << "The First occurence of element " << k << " is Located at index " << ans[0] << endl;

        cout << "The Last occurence of element " << k << " is Located at index " << ans[1] << endl;

    }

    else

    {

        cout << "Element not fornd in the array";

    }

    return 0;

}

**Output:**

