***NAME : Himanshu Dixit***

***ENROLL NO. : 21103262***

***BATCH : B11***

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

***Lab - Week 1***

**Solution a)**

#include <iostream>

#include <algorithm>

using namespace std;

void selectionsort(int arr[], int size, int &comp)

{

    for (int i = 0; i < size; i++)

    {

        int min = i;

        for (int j = i + 1; j < size; j++)

        {

            if (arr[min] > arr[j])

            {

                min = j;

            }

            comp++;

        }

        swap(arr[i], arr[min]);

    }

}

void insertionsort(int arr[], int size, int &comp)

{

    for (int i = 1; i < size; i++)

    {

        for (int j = i; j > 0; j--)

        {

            comp++;

            if (arr[j] < arr[j - 1])

            {

                swap(arr[j], arr[j - 1]);

            }

            else

            {

                break;

            }

        }

    }

}

void bubblesort(int arr[], int size, int &comp)

{

    for (int i = 0; i < size; i++)

    {

        int flag = 1;

        for (int j = 0; j < size - i - 1; j++)

        {

            comp++;

            if (arr[j] > arr[j + 1])

            {

                swap(arr[j], arr[j + 1]);

                flag = 0;

            }

        }

        if (flag == 1)

        {

            break;

        }

    }

}

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

{

    int l1 = mid - l + 1;

    int l2 = r - mid;

    int arr1[l1], arr2[l2];

    for (int i = 0; i < l1; i++)

    {

        arr1[i] = arr[l + i];

    }

    for (int i = 0; i < l2; i++)

    {

        arr2[i] = arr[mid + i + 1];

    }

    int i = 0, j = 0, k = l;

    while (i < l1 && j < l2)

    {

        if (arr1[i] <= arr2[j])

        {

            arr[k] = arr1[i];

            i++;

            comp++;

        }

        else

        {

            arr[k] = arr2[j];

            j++;

            comp++;

        }

        k++;

    }

    while (i < l1)

    {

        arr[k] = arr1[i];

        i++;

        k++;

    }

    while (j < l2)

    {

        arr[k] = arr2[j];

        j++;

        k++;

    }

}

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

{

    if (l < r)

    {

        int mid = (l + r) / 2;

        mergesort(arr, l, mid, comp);

        mergesort(arr, mid + 1, r, comp);

        merge(arr, l, mid, r, comp);

    }

}

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);

    }

}

int main()

{

    cout << "\nCount Number of comparison of unsorted array : \n";

    int arr1[] = {4, 2, 3, 1, 7, 8, 9, 6};

    int size = sizeof(arr1) / sizeof(int);

    int comp = 0;

    cout << "Array is : ";

    for (int i = 0; i < size; i++)

    {

        cout << arr1[i] << " ";

    }

    cout << endl

         << endl;

    cout << "Bubble sort : ";

    bubblesort(arr1, size, comp);

    for (int i = 0; i < size; i++)

    {

        cout << arr1[i] << " ";

    }

    cout << "\nComparison : " << comp << endl

         << endl;

    comp = 0;

    int arr2[] = {4, 2, 3, 1, 7, 8, 9, 6};

    cout << "Insertion sort : ";

    insertionsort(arr2, size, comp);

    for (int i = 0; i < size; i++)

    {

        cout << arr2[i] << " ";

    }

    cout << "\nComparison : " << comp << endl

         << endl;

    comp = 0;

    int arr3[] = {4, 2, 3, 1, 7, 8, 9, 6};

    cout << "Selection sort : ";

    selectionsort(arr3, size, comp);

    for (int i = 0; i < size; i++)

    {

        cout << arr3[i] << " ";

    }

    cout << "\nComparison : " << comp << endl

         << endl;

    comp = 0;

    int arr4[] = {4, 2, 3, 1, 7, 8, 9, 6};

    cout << "Merge sort : ";

    mergesort(arr4, 0, size - 1, comp);

    for (int i = 0; i < size; i++)

    {

        cout << arr4[i] << " ";

    }

    cout << "\nComparison : " << comp << endl

         << endl;

    comp = 0;

    int arr5[] = {4, 2, 3, 1, 7, 8, 9, 6};

    cout << "Quick sort : ";

    quicksort(arr5, 0, size - 1, comp);

    for (int i = 0; i < size; i++)

    {

        cout << arr5[i] << " ";

    }

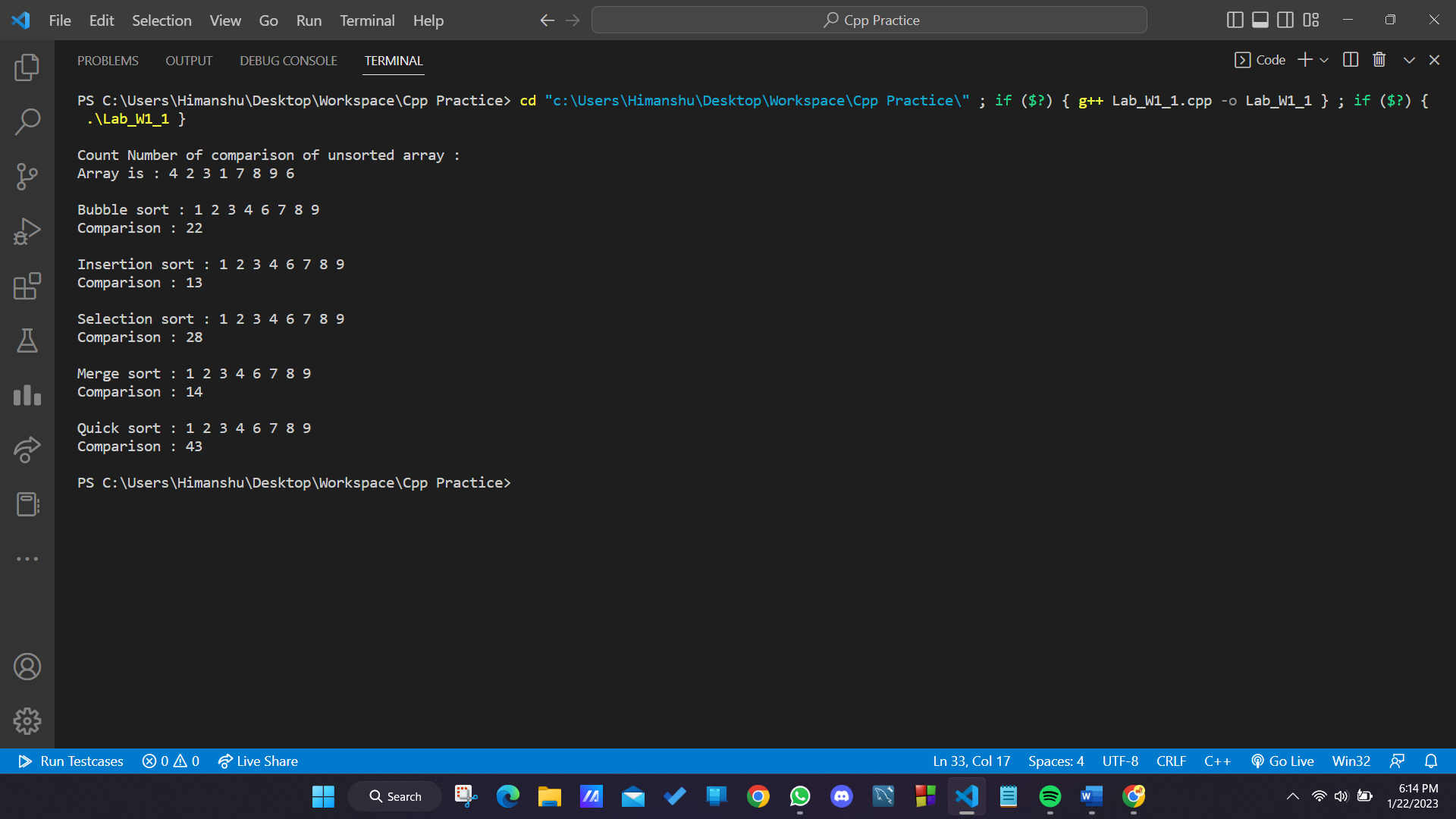
    cout << "\nComparison : " << comp << endl

         << endl;

    return 0;

}

**Output :**



**Solution b)**

#include <iostream>

#include <algorithm>

using namespace std;

void selectionsort(int arr[], int size, int &comp)

{

    for (int i = 0; i < size; i++)

    {

        int min = i;

        for (int j = i + 1; j < size; j++)

        {

            if (arr[min] > arr[j])

            {

                min = j;

            }

            comp++;

        }

        swap(arr[i], arr[min]);

    }

}

void insertionsort(int arr[], int size, int &comp)

{

    for (int i = 1; i < size; i++)

    {

        for (int j = i; j > 0; j--)

        {

            comp++;

            if (arr[j] < arr[j - 1])

            {

                swap(arr[j], arr[j - 1]);

            }

            else

            {

                break;

            }

        }

    }

}

void bubblesort(int arr[], int size, int &comp)

{

    for (int i = 0; i < size; i++)

    {

        int flag = 1;

        for (int j = 0; j < size - i - 1; j++)

        {

            comp++;

            if (arr[j] > arr[j + 1])

            {

                swap(arr[j], arr[j + 1]);

                flag = 0;

            }

        }

        if (flag == 1)

        {

            break;

        }

    }

}

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

{

    int l1 = mid - l + 1;

    int l2 = r - mid;

    int arr1[l1], arr2[l2];

    for (int i = 0; i < l1; i++)

    {

        arr1[i] = arr[l + i];

    }

    for (int i = 0; i < l2; i++)

    {

        arr2[i] = arr[mid + i + 1];

    }

    int i = 0, j = 0, k = l;

    while (i < l1 && j < l2)

    {

        if (arr1[i] <= arr2[j])

        {

            arr[k] = arr1[i];

            i++;

            comp++;

        }

        else

        {

            arr[k] = arr2[j];

            j++;

            comp++;

        }

        k++;

    }

    while (i < l1)

    {

        arr[k] = arr1[i];

        i++;

        k++;

    }

    while (j < l2)

    {

        arr[k] = arr2[j];

        j++;

        k++;

    }

}

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

{

    if (l < r)

    {

        int mid = (l + r) / 2;

        mergesort(arr, l, mid, comp);

        mergesort(arr, mid + 1, r, comp);

        merge(arr, l, mid, r, comp);

    }

}

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);

    }

}

int main()

{

    cout << "\nCount Number of comparison of sorted array : \n";

    int arr[] = {1, 2, 3, 4, 6, 7, 8, 9};

    int size = sizeof(arr) / sizeof(int);

    int comp = 0;

    cout << "Array is : ";

    for (int i = 0; i < size; i++)

    {

        cout << arr[i] << " ";

    }

    cout << endl

         << endl;

    cout << "Bubble sort : ";

    bubblesort(arr, size, comp);

    for (int i = 0; i < size; i++)

    {

        cout << arr[i] << " ";

    }

    cout << "\nComparison : " << comp << endl

         << endl;

    comp = 0;

    cout << "Insertion sort : ";

    insertionsort(arr, size, comp);

    for (int i = 0; i < size; i++)

    {

        cout << arr[i] << " ";

    }

    cout << "\nComparison : " << comp << endl

         << endl;

    comp = 0;

    cout << "Selection sort : ";

    selectionsort(arr, size, comp);

    for (int i = 0; i < size; i++)

    {

        cout << arr[i] << " ";

    }

    cout << "\nComparison : " << comp << endl

         << endl;

    comp = 0;

    cout << "Merge sort : ";

    mergesort(arr, 0, size - 1, comp);

    for (int i = 0; i < size; i++)

    {

        cout << arr[i] << " ";

    }

    cout << "\nComparison : " << comp << endl

         << endl;

    comp = 0;

    cout << "Quick sort : ";

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

    for (int i = 0; i < size; i++)

    {

        cout << arr[i] << " ";

    }

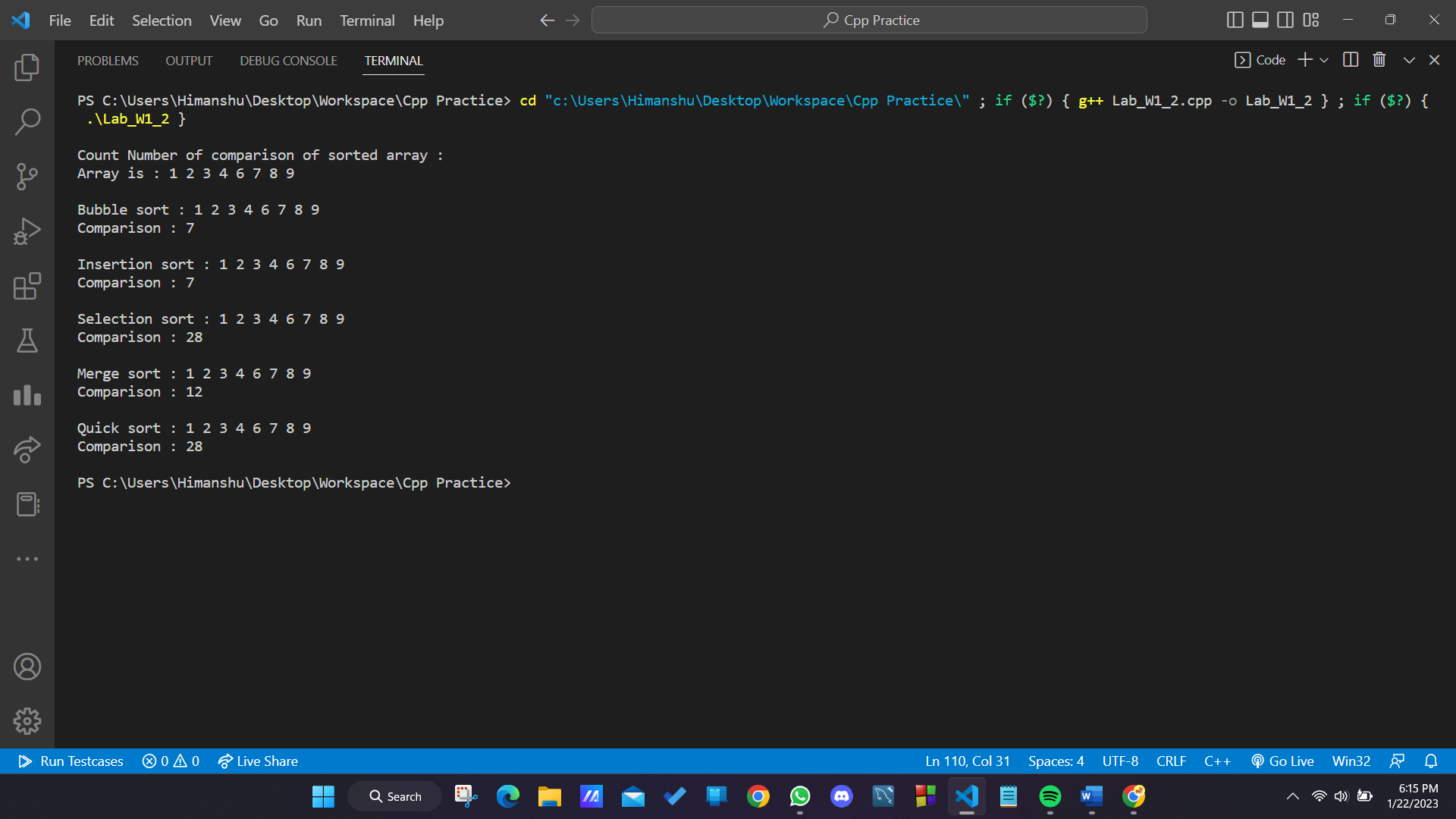
    cout << "\nComparison : " << comp << endl

         << endl;

    return 0;

}

**Output :**



**Solution c)**

#include <iostream>

using namespace std;

void ascendingdescending(int \*arr, int size, char flag)

{

    for (int i = 0; i < size; i++)

    {

        int flag = 1;

        for (int j = 0; j < size - i - 1; j++)

        {

            if (flag == 'a')

            {

                if (arr[j] > arr[j + 1])

                {

                    swap(arr[j], arr[j + 1]);

                    flag = 0;

                }

            }

            else

            {

                if (arr[j] < arr[j + 1])

                {

                    swap(arr[j], arr[j + 1]);

                    flag = 0;

                }

            }

        }

        if (flag == 1)

        {

            break;

        }

    }

}

int main()

{

    int n;

    cin >> n;

    int arr[n];

    for (int i = 0; i < n; i++)

    {

        cin >> arr[i];

    }

    char flag;

    cin >> flag;

    ascendingdescending(arr, n, flag);

    for (int i = 0; i < n; i++)

    {

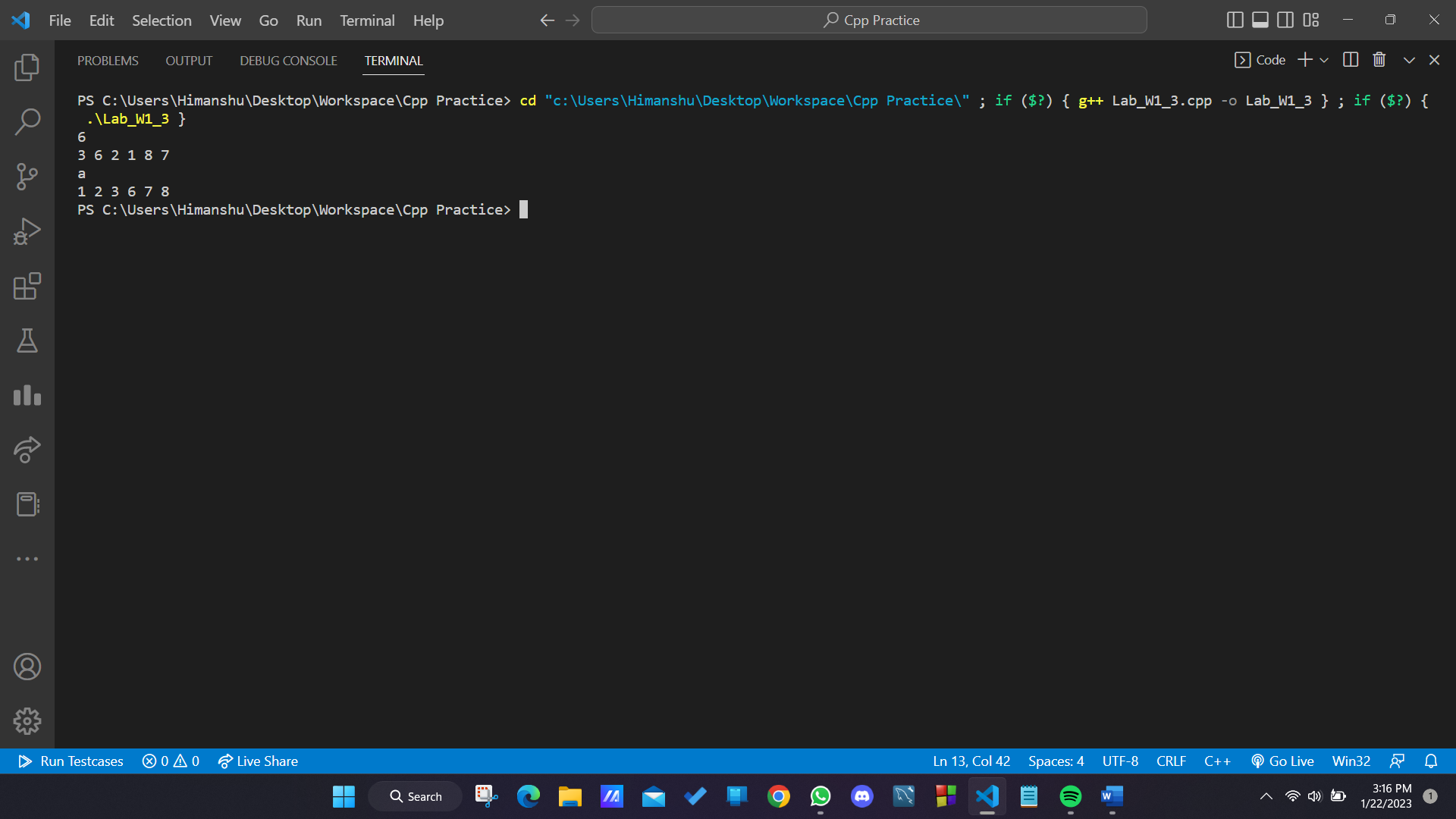
        cout << arr[i] << " ";

    }

    return 0;

}

**Output :**



**Solution d)**

#include <iostream>

#include <algorithm>

using namespace std;

void selectionsort(int arr[], int size, int &comp, char flag)

{

    for (int i = 0; i < size; i++)

    {

        int min = i;

        int max = i;

        for (int j = i + 1; j < size; j++)

        {

            if (flag == 'a' and arr[min] > arr[j])

            {

                min = j;

            }

            if (flag == 'd' and arr[max] < arr[j])

            {

                max = j;

            }

            comp++;

        }

        if (flag == 'a')

            swap(arr[i], arr[min]);

        if (flag == 'd')

            swap(arr[i], arr[max]);

    }

}

void insertionsort(int arr[], int size, int &comp, char flag)

{

    for (int i = 1; i < size; i++)

    {

        for (int j = i; j > 0; j--)

        {

            if (flag == 'a')

            {

                if (arr[j] < arr[j - 1])

                {

                    swap(arr[j], arr[j - 1]);

                    comp++;

                }

                else

                {

                    comp++;

                    break;

                }

            }

            if (flag == 'd')

            {

                if (arr[j] > arr[j - 1])

                {

                    swap(arr[j], arr[j - 1]);

                    comp++;

                }

                else

                {

                    comp++;

                    break;

                }

            }

        }

    }

}

void bubblesort(int arr[], int size, int &comp, char flag)

{

    for (int i = 0; i < size; i++)

    {

        int f = 1;

        for (int j = 0; j < size - i - 1; j++)

        {

            if (flag == 'a')

            {

                if (arr[j] > arr[j + 1])

                {

                    swap(arr[j], arr[j + 1]);

                    f = 0;

                }

                comp++;

            }

            else

            {

                if (arr[j] < arr[j + 1])

                {

                    swap(arr[j], arr[j + 1]);

                    f = 0;

                }

                comp++;

            }

        }

        if (f == 1)

        {

            break;

        }

    }

}

void merge(int \*arr, int l, int mid, int r, int &comp, char flag)

{

    int l1 = mid - l + 1;

    int l2 = r - mid;

    int arr1[l1], arr2[l2];

    for (int i = 0; i < l1; i++)

    {

        arr1[i] = arr[l + i];

    }

    for (int i = 0; i < l2; i++)

    {

        arr2[i] = arr[mid + i + 1];

    }

    int i = 0, j = 0, k = l;

    while (i < l1 && j < l2)

    {

        if (flag == 'a')

        {

            if (arr1[i] <= arr2[j])

            {

                arr[k] = arr1[i];

                i++;

                comp++;

            }

            else

            {

                arr[k] = arr2[j];

                j++;

                comp++;

            }

            k++;

        }

        if (flag == 'd')

        {

            if (arr1[i] >= arr2[j])

            {

                arr[k] = arr1[i];

                i++;

                comp++;

            }

            else

            {

                arr[k] = arr2[j];

                j++;

                comp++;

            }

            k++;

        }

    }

    while (i < l1)

    {

        arr[k] = arr1[i];

        i++;

        k++;

    }

    while (j < l2)

    {

        arr[k] = arr2[j];

        j++;

        k++;

    }

}

void mergesort(int \*arr, int l, int r, int &comp, char flag)

{

    if (l < r)

    {

        int mid = (l + r) / 2;

        mergesort(arr, l, mid, comp, flag);

        mergesort(arr, mid + 1, r, comp, flag);

        merge(arr, l, mid, r, comp, flag);

    }

}

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

{

    int pivot = arr[r];

    int i = l, j = l;

    while (j < r)

    {

        if (flag == 'a' and arr[j] <= pivot)

        {

            swap(arr[i], arr[j]);

            i++;

        }

        if (flag == 'd' and arr[j] >= pivot)

        {

            swap(arr[i], arr[j]);

            i++;

        }

        comp++;

        j++;

    }

    swap(arr[i], arr[r]);

    return i;

}

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

{

    if (l < r)

    {

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

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

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

    }

}

int main()

{

    cout << "\nCount Number of comparison of Sorted array : \n";

    int arr1[] = {1, 2, 3, 4, 6, 7, 8, 9};

    int size = sizeof(arr1) / sizeof(int);

    int comp = 0;

    char flag = 'd';

    cout << "Array is : ";

    for (int i = 0; i < size; i++)

    {

        cout << arr1[i] << " ";

    }

    cout << endl

         << endl;

    cout << "Bubble sort : ";

    bubblesort(arr1, size, comp, flag);

    for (int i = 0; i < size; i++)

    {

        cout << arr1[i] << " ";

    }

    cout << "\nComparison : " << comp << endl

         << endl;

    comp = 0;

    int arr2[] = {1, 2, 3, 4, 6, 7, 8, 9};

    cout << "Insertion sort : ";

    insertionsort(arr2, size, comp, flag);

    for (int i = 0; i < size; i++)

    {

        cout << arr2[i] << " ";

    }

    cout << "\nComparison : " << comp << endl

         << endl;

    comp = 0;

    int arr3[] = {1, 2, 3, 4, 6, 7, 8, 9};

    cout << "Selection sort : ";

    selectionsort(arr3, size, comp, flag);

    for (int i = 0; i < size; i++)

    {

        cout << arr3[i] << " ";

    }

    cout << "\nComparison : " << comp << endl

         << endl;

    comp = 0;

    int arr4[] = {1, 2, 3, 4, 6, 7, 8, 9};

    cout << "Merge sort : ";

    mergesort(arr4, 0, size - 1, comp, flag);

    for (int i = 0; i < size; i++)

    {

        cout << arr4[i] << " ";

    }

    cout << "\nComparison : " << comp << endl

         << endl;

    comp = 0;

    int arr5[] = {1, 2, 3, 4, 6, 7, 8, 9};

    cout << "Quick sort : ";

    quicksort(arr5, 0, size - 1, comp, flag);

    for (int i = 0; i < size; i++)

    {

        cout << arr5[i] << " ";

    }

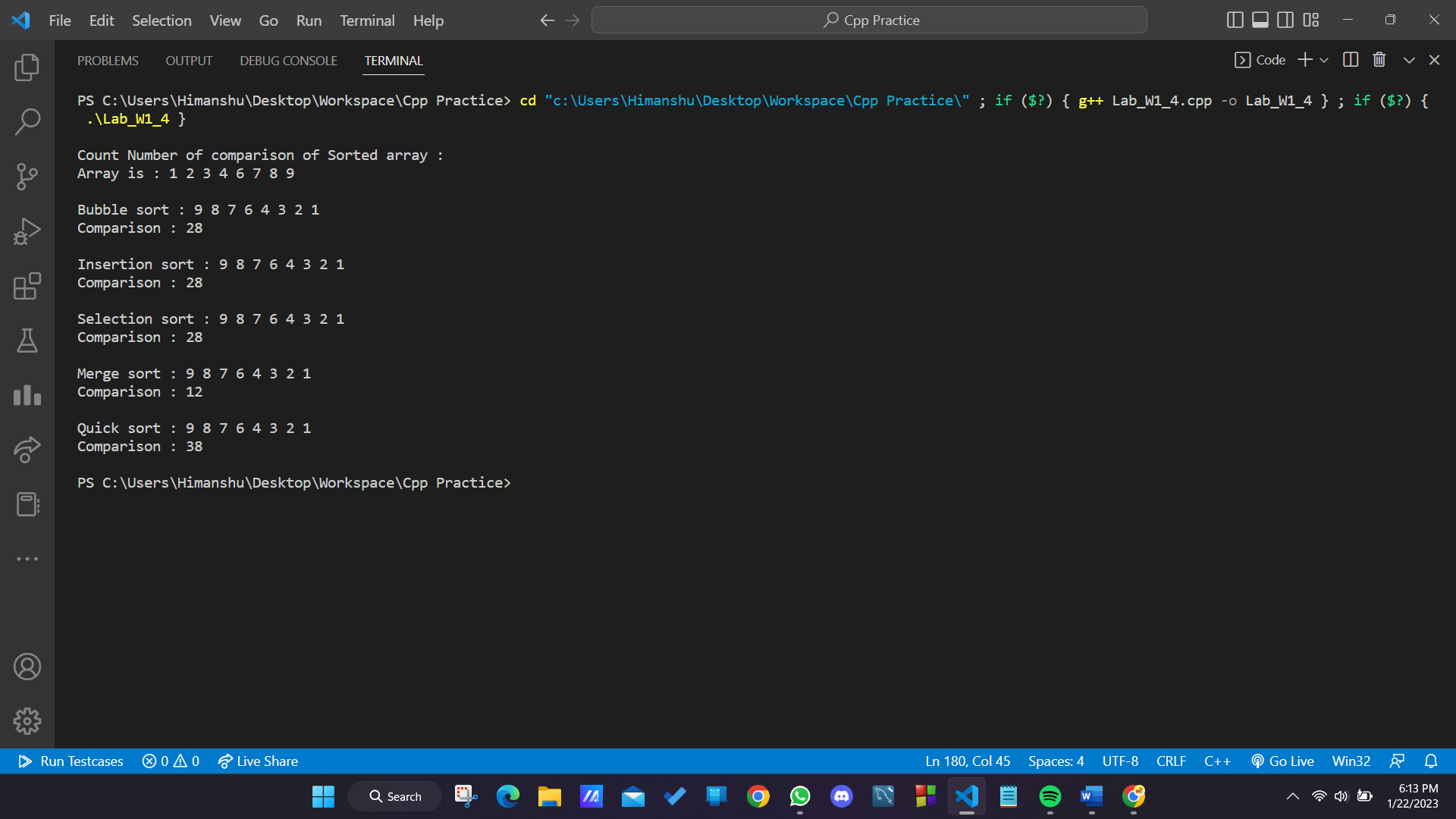
    cout << "\nComparison : " << comp << endl

         << endl;

    return 0;

}

**Output :**



**Solution e)**

#include <iostream>

#include <algorithm>

using namespace std;

int Lpartition(int \*arr, int l, int r)

{

    int pivot = arr[l];

    int i = l + 1, j = l + 1;

    while (j <= r)

    {

        if (arr[j] <= pivot)

        {

            swap(arr[i], arr[j]);

            i++;

        }

        j++;

    }

    swap(arr[i - 1], arr[l]);

    return i - 1;

}

int Hpartition(int \*arr, int l, int r)

{

    int pivot = arr[r];

    int i = l, j = l;

    while (j < r)

    {

        if (arr[j] <= pivot)

        {

            swap(arr[i], arr[j]);

            i++;

        }

        j++;

    }

    swap(arr[i], arr[r]);

    return i;

}

int Mpartition(int \*arr, int l, int r)

{

    int p = (l + r) % 2 == 0 ? (l + r) / 2 : (l + r + 1) / 2;

    int pivot = arr[p];

    int i = l, j = l;

    while (j <= r)

    {

        if (arr[j] <= pivot and j != p)

        {

            swap(arr[i], arr[j]);

            i++;

        }

        j++;

    }

    if (p < i and arr[p] > arr[i])

        swap(arr[i], arr[p]);

    else if (p > i and arr[p] < arr[i])

        swap(arr[i], arr[p]);

    return i;

}

void Lquicksort(int \*arr, int l, int r)

{

    if (l < r)

    {

        int p = Lpartition(arr, l, r);

        Lquicksort(arr, 0, p - 1);

        Lquicksort(arr, p + 1, r);

    }

}

void Hquicksort(int \*arr, int l, int r)

{

    if (l < r)

    {

        int p = Lpartition(arr, l, r);

        Lquicksort(arr, 0, p - 1);

        Lquicksort(arr, p + 1, r);

    }

}

void Mquicksort(int \*arr, int l, int r)

{

    if (l < r)

    {

        int p = Lpartition(arr, l, r);

        Lquicksort(arr, 0, p - 1);

        Lquicksort(arr, p + 1, r);

    }

}

int main()

{

    int n;

    cin >> n;

    int arr[n];

    for (int i = 0; i < n; i++)

    {

        cin >> arr[i];

    }

    char ch;

    cout << "Enter L , H, and M to make pivot : ";

    cin >> ch;

    if (ch == 'L')

    {

        Lquicksort(arr, 0, n - 1);

    }

    if (ch == 'H')

    {

        Hquicksort(arr, 0, n - 1);

    }

    if (ch == 'M')

    {

        Mquicksort(arr, 0, n - 1);

    }

    for (int i = 0; i < n; i++)

    {

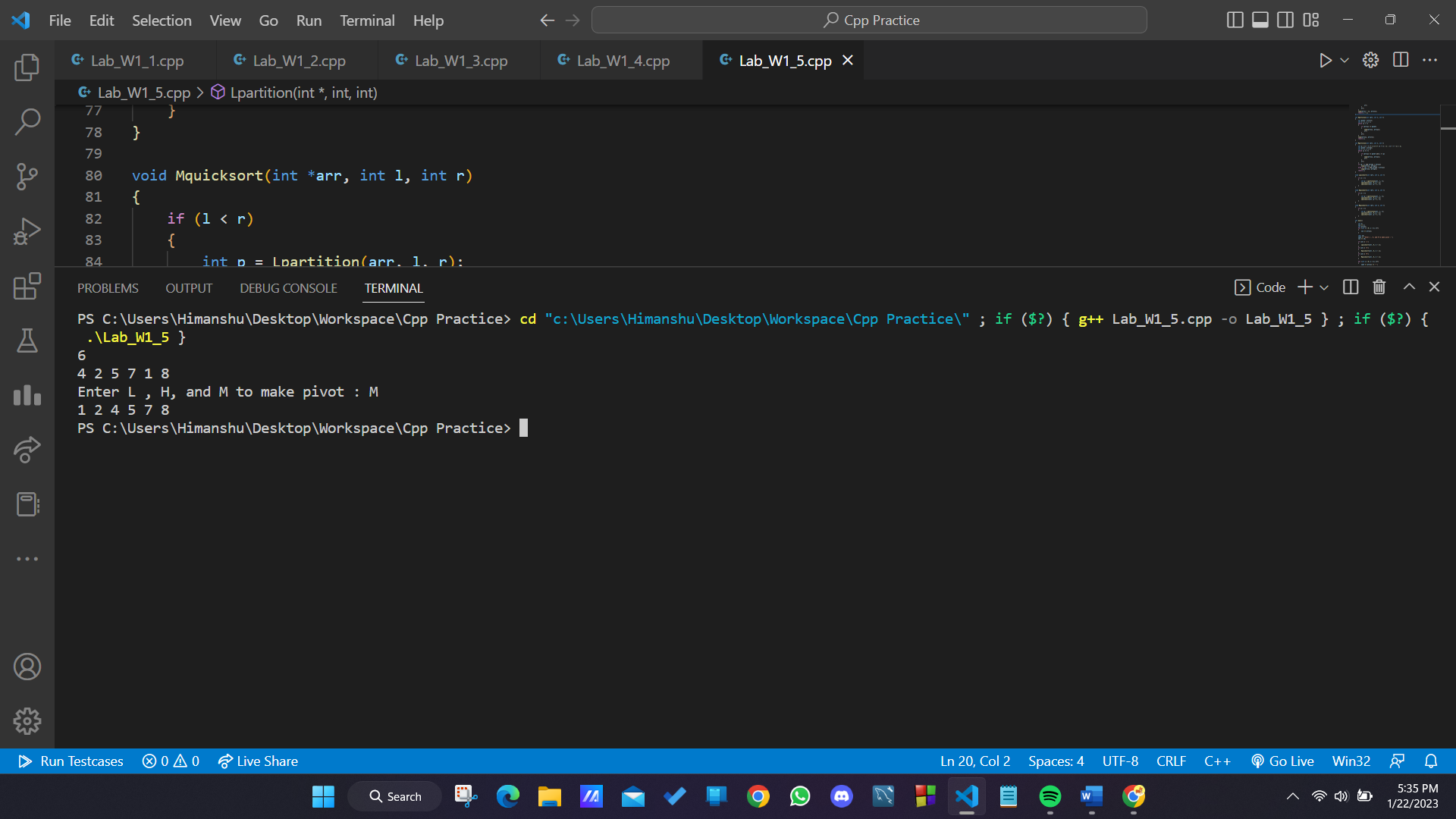
        cout << arr[i] << " ";

    }

    return 0;

}

**Output :**



**Solution f)**

Complexity of Quick sort is O(n(logn)).

Best Case : if pivot is always is in middle then complexity is O(n(logn)).

Worst Case : (when array is sorted or reverse sorted) then complexity is O(n2).

**Solution g)**

Complexity of Merge sort is O(n(logn)).

It is same for best and worst case because array is break from middle every time.

Best Case : O(n(logn)).

Worst Case : O(n(logn)).

**Solution h)**

Complexity of Bubble sort is O(n2).

In ordinary bubble sort means without break, it will always compare with next element.

For Best Case = O(n2)

For Worst Case = O(n2)

In Advanced Bubble sort means with break,

For Best Case = O(n) for sorted array

For Worst Case = O(n2)

**Solution i)**

Complexity of Selection sort is O(n2).

It will find minimum in each loop either sorted or not so best and worst are same

For Best Case = O(n2)

For Worst Case = O(n2)

**Solution j)**

Complexity of Insertion sort is O(n2).

It will compare to the previous element if previous one is already smaller then it will not compare with more previous so it’s Best case is sorted array

For Best Case = O(n)

Worst case is unsorted array.

For Worst Case = O(n2)

**Solution k)**

#include <iostream>

using namespace std;

void merge(int \*arr, int l, int mid, int r)

{

    int l1 = mid - l + 1;

    int l2 = r - mid;

    int arr1[l1], arr2[l2];

    for (int i = 0; i < l1; i++)

    {

        arr1[i] = arr[l + i];

    }

    for (int i = 0; i < l2; i++)

    {

        arr2[i] = arr[mid + i + 1];

    }

    int i = 0, j = 0, k = l;

    while (i < l1 && j < l2)

    {

        if (arr1[i] <= arr2[j])

        {

            arr[k] = arr1[i];

            i++;

        }

        else

        {

            arr[k] = arr2[j];

            j++;

        }

        k++;

    }

    while (i < l1)

    {

        arr[k] = arr1[i];

        i++;

        k++;

    }

    while (j < l2)

    {

        arr[k] = arr2[j];

        j++;

        k++;

    }

}

void mergesort(int \*arr, int l, int r)

{

    if (l < r)

    {

        int mid = (l + r) / 2;

        mergesort(arr, l, mid);

        mergesort(arr, mid + 1, r);

        merge(arr, l, mid, r);

    }

}

void mergerev(int \*arr, int l, int mid, int r)

{

    int l1 = mid - l + 1;

    int l2 = r - mid;

    int arr1[l1], arr2[l2];

    for (int i = 0; i < l1; i++)

    {

        arr1[i] = arr[l + i];

    }

    for (int i = 0; i < l2; i++)

    {

        arr2[i] = arr[mid + i + 1];

    }

    int i = 0, j = 0, k = l;

    while (i < l1 && j < l2)

    {

        if (arr1[i] >= arr2[j])

        {

            arr[k] = arr1[i];

            i++;

        }

        else

        {

            arr[k] = arr2[j];

            j++;

        }

        k++;

    }

    while (i < l1)

    {

        arr[k] = arr1[i];

        i++;

        k++;

    }

    while (j < l2)

    {

        arr[k] = arr2[j];

        j++;

        k++;

    }

}

void mergesortrev(int \*arr, int l, int r)

{

    if (l < r)

    {

        int mid = (l + r) / 2;

        mergesortrev(arr, l, mid);

        mergesortrev(arr, mid + 1, r);

        mergerev(arr, l, mid, r);

    }

}

int main()

{

    int n;

    cin >> n;

    int arr[n];

    for (int i = 0; i < n; i++)

    {

        cin >> arr[i];

    }

    int arr1[n / 2 + 1];

    int arr2[n / 2 + 1];

    int x = 0, y = 0;

    for (int i = 0; i < n; i++)

    {

        if (i % 2 != 0)

        {

            arr2[y++] = arr[i];

        }

        else

        {

            arr1[x++] = arr[i];

        }

    }

    mergesort(arr1, 0, x - 1);

    mergesortrev(arr2, 0, y - 1);

    for (int i = 0; i < x; i++)

    {

        arr[i] = arr1[i];

    }

    for (int i = 0; i < y; i++)

    {

        arr[x + i] = arr2[i];

    }

    for (int i = 0; i < n; i++)

    {

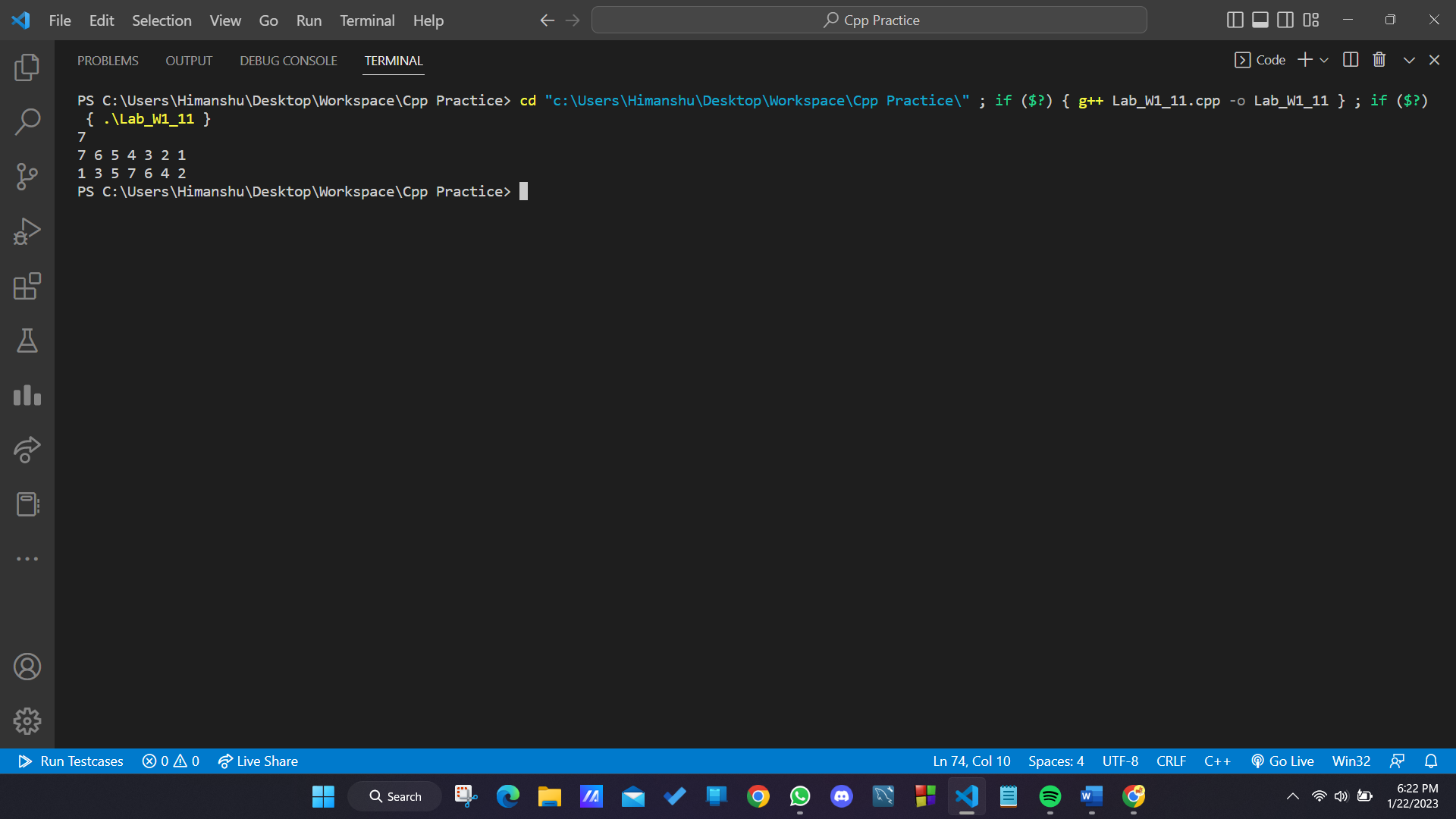
        cout << arr[i] << " ";

    }

    return 0;

}

**Output :**



**Complexity : O(n/2(log(n/2))) \* 2 = O(n(logn)) (approximately)**

**Calling two times merge sort**

**Solution l)**

#include <iostream>

using namespace std;

int main()

{

    int n;

    cin >> n;

    int arr[n];

    for (int i = 0; i < n; i++)

    {

        cin >> arr[i];

    }

    int arr1[n / 2 + 1];

    int arr2[n / 2 + 1];

    int x = 0, y = 0;

    for (int i = 0; i < n; i++)

    {

        if (i < n / 2)

        {

            arr1[x++] = arr[i];

        }

        else

        {

            arr2[y++] = arr[i];

        }

    }

    int i = 0, j = 0, k = 0;

    while (i < x && j < y)

    {

        if (arr1[i] <= arr2[j])

        {

            arr[k] = arr1[i];

            i++;

        }

        else

        {

            arr[k] = arr2[j];

            j++;

        }

        k++;

    }

    while (i < x)

    {

        arr[k] = arr1[i];

        i++;

        k++;

    }

    while (j < y)

    {

        arr[k] = arr2[j];

        j++;

        k++;

    }

    for (int i = 0; i < n; i++)

    {

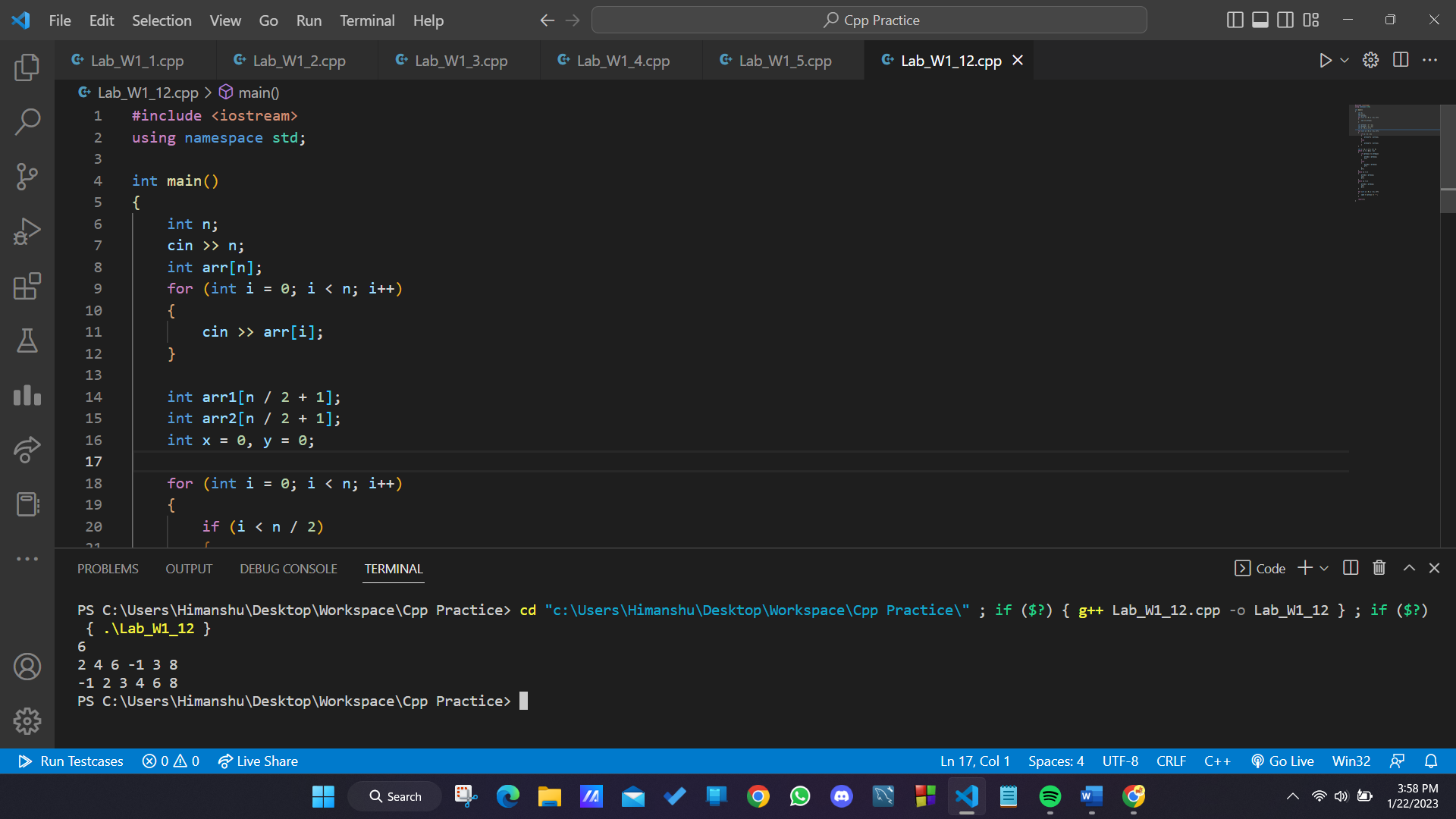
        cout << arr[i] << " ";

    }

    return 0;

}

**Output :**



**Complexity : O(n) simply merging is done in between two sorted array**