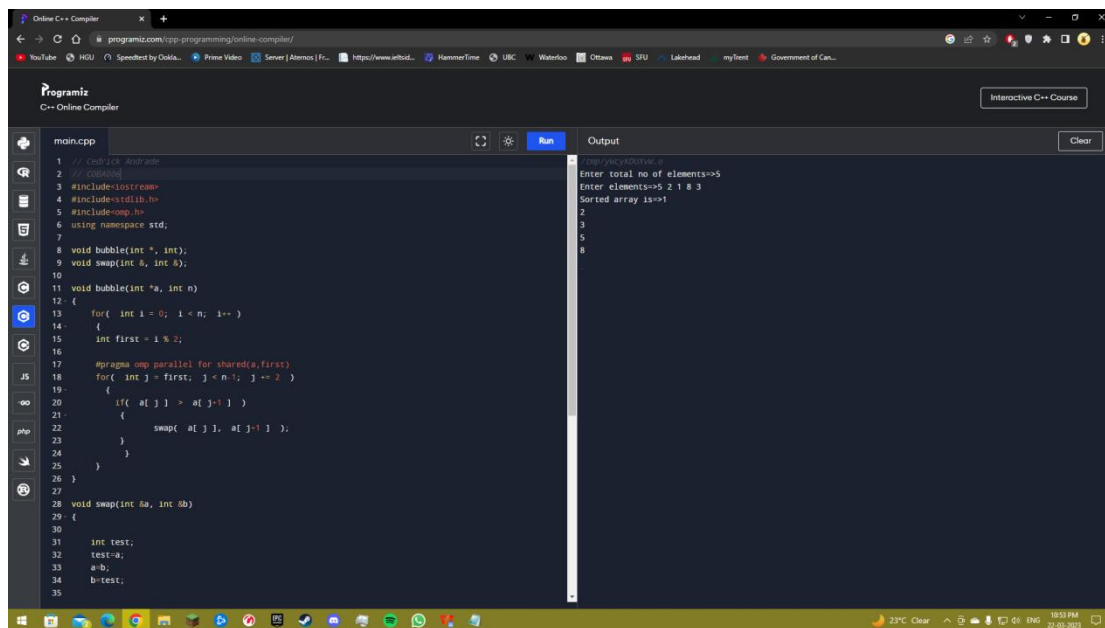


## Assignment 2A

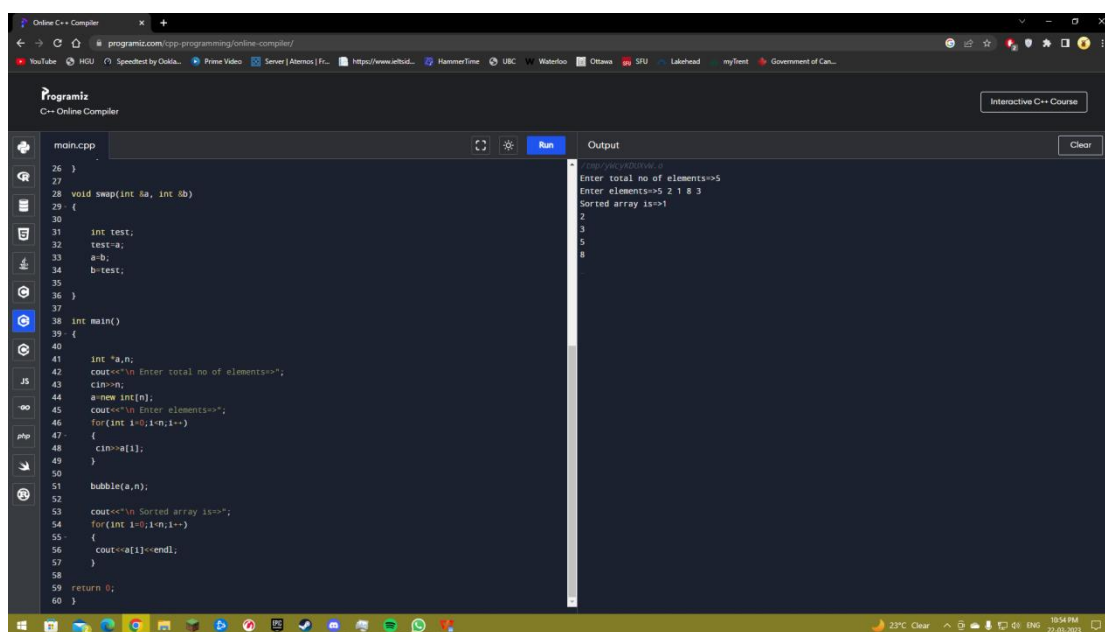


The screenshot shows the Programiz C++ Online Compiler interface. The code in `main.cpp` implements a bubble sort algorithm. It includes `<omp.h>` and uses `using namespace std;`. The `bubble` function is defined with parameters `int *`, `int`, and `int n`. It uses a `for` loop for `i` from 0 to `n-1`, and an inner `for` loop for `j` from `first` to `n-1`. The inner loop is parallelized using `#pragma omp parallel for`. The `swap` function is defined to swap two integers. The `main` function tests the `bubble` function with an array `test`.

```
1 // Cedrick Andrade
2 // COBA006
3 #include <iostream>
4 #include <vector>
5 #include <omp.h>
6 using namespace std;
7
8 void bubble(int *, int);
9 void swap(int &, int &);
10
11 void bubble(int *a, int n)
12 {
13     for( int i = 0; i < n; i++)
14     {
15         int first = i % 2;
16
17         #pragma omp parallel for shared(a, first)
18         for( int j = first; j < n-1; j += 2 )
19         {
20             if( a[j] > a[j+1] )
21             {
22                 swap( a[j], a[j+1] );
23             }
24         }
25     }
26 }
27
28 void swap(int &a, int &b)
29 {
30     int test;
31     test = a;
32     a = b;
33     b = test;
34 }
35
```

The output shows the program execution results:

```
Enter total no of elements=>5
Enter elements=>5 2 1 8 3
Sorted array is=>1
2
3
5
8
```



The screenshot shows the Programiz C++ Online Compiler interface. The code in `main.cpp` implements a bubble sort algorithm. It includes `<omp.h>` and uses `using namespace std;`. The `bubble` function is defined with parameters `int *`, `int`, and `int n`. It uses a `for` loop for `i` from 0 to `n-1`, and an inner `for` loop for `j` from `first` to `n-1`. The inner loop is parallelized using `#pragma omp parallel for`. The `swap` function is defined to swap two integers. The `main` function tests the `bubble` function with an array `test`.

```
26 }
27
28 void swap(int &a, int &b)
29 {
30     int test;
31     test = a;
32     a = b;
33     b = test;
34 }
35
36 int main()
37 {
38     int *a, n;
39     cout << "Enter total no of elements=" << endl;
40     cin >> n;
41     a = new int[n];
42     cout << "Enter elements=" << endl;
43     for( int i = 0; i < n; i++)
44     {
45         cin >> a[i];
46     }
47     bubble(a, n);
48     cout << "Sorted array is=" << endl;
49     for( int i = 0; i < n; i++)
50     {
51         cout << a[i] << " ";
52     }
53     return 0;
54 }
55
```

The output shows the program execution results:

```
Enter total no of elements=>5
Enter elements=>5 2 1 8 3
Sorted array is=>1
2
3
5
8
```

## Assignment 2B

The image displays three sequential screenshots of a web-based C++ compiler (Programiz) showing the implementation of a parallel merge sort algorithm. The code is written in `main.cpp` and uses OpenMP for parallelization.

**First Screenshot:** Shows the initial setup of the merge sort algorithm. It includes headers for `<iostream>`, `<vector>`, and `<omp.h>`, and uses the `std` namespace. The `merge` function is defined, which takes a vector and indices `left`, `middle`, and `right`. It recursively splits the array into two halves and merges them back in sorted order. The `main` function initializes an array `arr = { 12, 11, 13, 5, 6, 7, 3 }` and calls `mergeSort(arr, 0, n - 1)`.

**Second Screenshot:** Shows the implementation of the `mergeSort` function. It uses a `while` loop to split the array into two halves and recursively calls `mergeSort` on each half. The `merge` function is also shown, which merges the two sorted halves back into a single sorted array.

**Third Screenshot:** Shows the final implementation of the `mergeSort` function. It uses a `while` loop to split the array into two halves and recursively calls `mergeSort` on each half. The `merge` function is also shown, which merges the two sorted halves back into a single sorted array. The `main` function is also shown, which initializes the array and calls `mergeSort`.

The output of the program, displayed in the right-hand pane, is: `Sorted array: 5 6 7 11 12 13`.