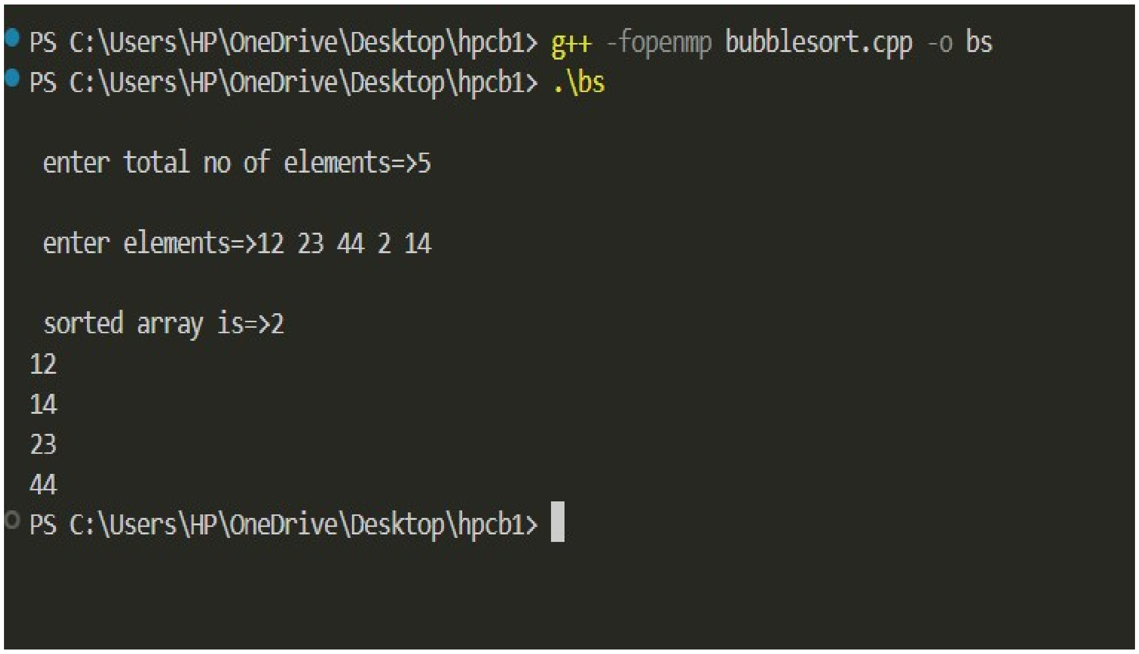
Problem Statement: Write a program to implement Parallel Bubble Sort and Merge sort using OpenMP. Use existing algorithms and measure the performance of sequential and parallel algorithms.

Output-1 (Parallel Bubble Sort)

```
#include<iostream>
#include<stdlib.h>
#include<omp.h>
using namespace std;

void bubble(int *, int);
void swap(int &, int &);

void bubble(int *a, int n)
{
   for(  int i = 0;  i < n;  i++ )
    {
    int first = i % 2;

    #pragma omp parallel for shared(a,first)
    for(  int j = first;  j < n-1;  j += 2  )
      {
       if(  a[ j ]  >  a[ j+1 ]  )
        {
              swap(  a[ j ],  a[ j+1 ]  );
        }
         }
    }
}

void swap(int &a, int &b)
{

   int test;
   test=a;
   a=b;
   b=test;

}

int main()
{

   int *a,n;
   cout<<"\n enter total no of elements=>";
   cin>>n;
   a=new int[n];
```

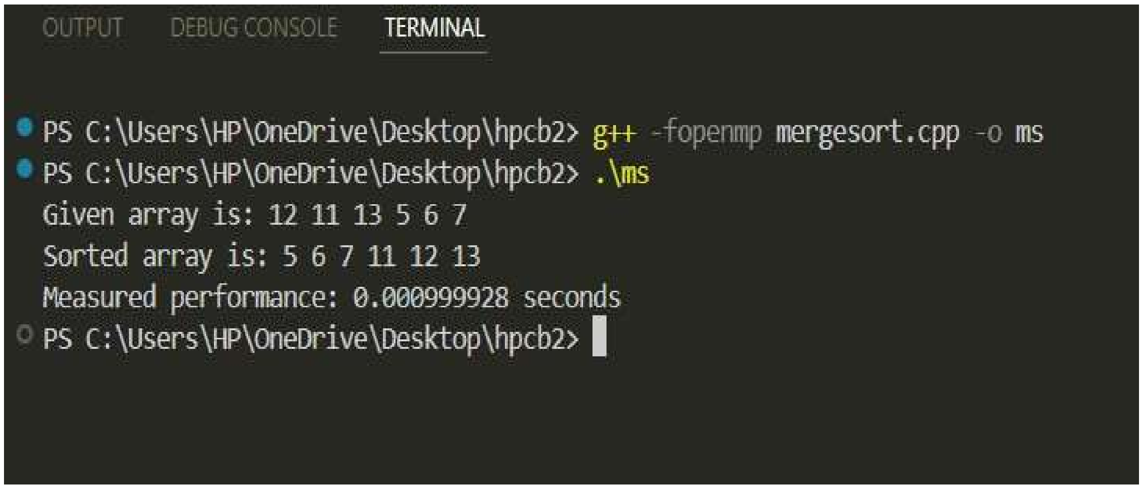
```
    cout<<"\n enter elements=>";
    for(int i=0;i<n;i++)
    {
     cin>>a[i];
    }

    bubble(a,n);

    cout<<"\n sorted array is=>";
    for(int i=0;i<n;i++)
    {
     cout<<a[i]<<endl;
    }

return 0;
}
```



```
PS C:\Users\HP\OneDrive\Desktop\hpcb1> g++ -fopenmp bubblesort.cpp -o bs
PS C:\Users\HP\OneDrive\Desktop\hpcb1> .\bs

 enter total no of elements=>5

 enter elements=>12 23 44 2 14

 sorted array is=>2
12
14
23
44
PS C:\Users\HP\OneDrive\Desktop\hpcb1> 
```

Output-2 (Parallel Merge Sort)

```
#include <iostream>
#include <omp.h>
void merge(int* arr, int l, int m, int r) {
    int i, j, k;
    int n1 = m - l + 1;
    int n2 = r - m;
    int* L = new int[n1];
    int* R = new int[n2];

    for (i = 0; i < n1; i++)
        L[i] = arr[l + i];
    for (j = 0; j < n2; j++)
        R[j] = arr[m + 1 + j];

    i = 0;
    j = 0;
    k = l;

    while (i < n1 && j < n2) {
        if (L[i] <= R[j]) {
            arr[k] = L[i];
            i++;
        }
        else {
            arr[k] = R[j];
            j++;
        }
        k++;
    }

    while (i < n1) {
        arr[k] = L[i];
        i++;
        k++;
    }

    while (j < n2) {
        arr[k] = R[j];
        j++;
        k++;
    }

    delete[] L;
    delete[] R;
}

void mergeSort(int* arr, int l, int r) {
    if (l < r) {
        int m = l + (r - l) / 2;

        #pragma omp parallel sections
        {
            #pragma omp section
            mergeSort(arr, l, m);

            #pragma omp section
            mergeSort(arr, m + 1, r);
        }

        merge(arr, l, m, r);
    }
}

int main() {
    int arr[] = {12, 11, 13, 5, 6, 7};
    int n = sizeof(arr) / sizeof(arr[0]);

    std::cout << "Given array is: ";
    for (int i = 0; i < n; i++)
        std::cout << arr[i] << " ";
    std::cout << std::endl;

    double start = omp_get_wtime();
    mergeSort(arr, 0, n - 1);
    double stop = omp_get_wtime();
    std::cout << "Sorted array is: ";
    for (int i = 0; i < n; i++)
        std::cout << arr[i] << " ";
    std::cout << std::endl;
    std::cout << "Measured performance: " << stop - start << " seconds" << std::endl;

    return 0;
}
```



```
PS C:\Users\HP\OneDrive\Desktop\hpcb2> g++ -fopenmp mergesort.cpp -o ms
PS C:\Users\HP\OneDrive\Desktop\hpcb2> .\ms
Given array is: 12 11 13 5 6 7
Sorted array is: 5 6 7 11 12 13
Measured performance: 0.000999928 seconds
PS C:\Users\HP\OneDrive\Desktop\hpcb2>
```