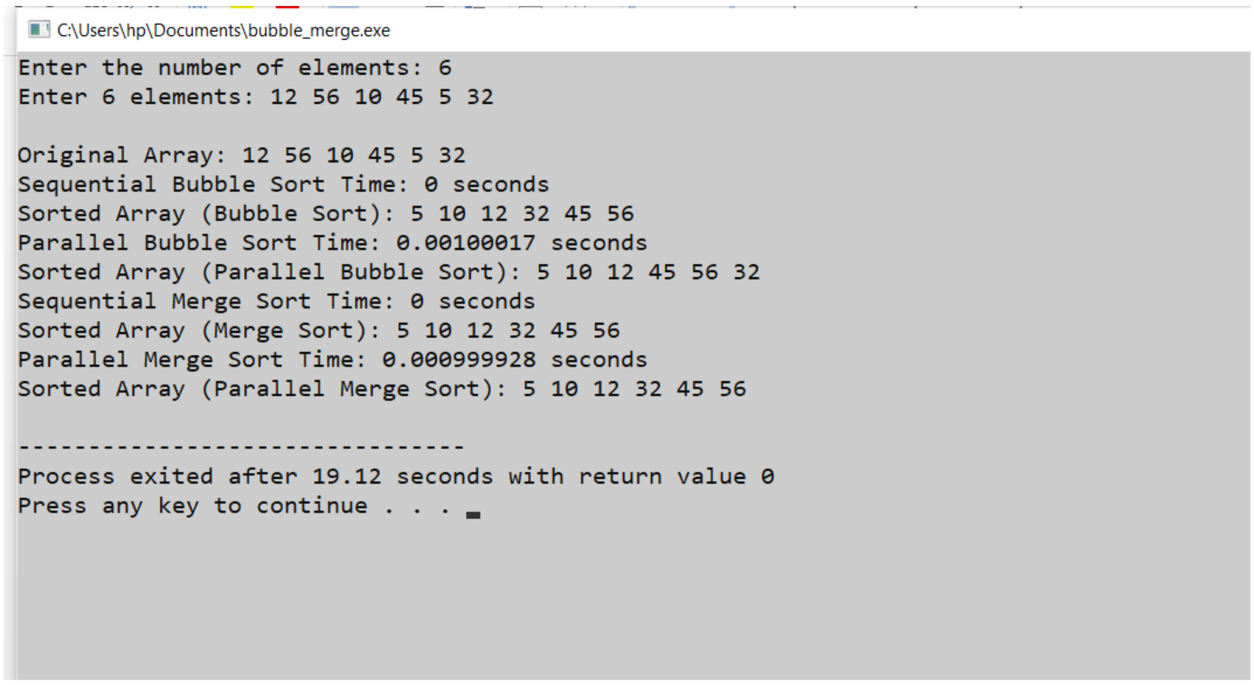
Program Code :

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

// Function to swap two elements
void swap(int &a, int &b) {
    int temp = a;
    a = b;
    b = temp;
}
// Sequential Bubble Sort
void bubbleSort(int arr[], int n) {
    for (int i = 0; i < n - 1; i++) {
        for (int j = 0; j < n - i - 1; j++) {
            if (arr[j] > arr[j + 1]) {
                swap(arr[j], arr[j + 1]);
            }
        }
    }
}
// Parallel Bubble Sort using OpenMP
void parallelBubbleSort(int arr[], int n) {
    for (int i = 0; i < n - 1; i++) {
        #pragma omp parallel for
        for (int j = 0; j < n - i - 1; j++) {
            if (arr[j] > arr[j + 1]) {
                swap(arr[j], arr[j + 1]);
            }
        }
    }
}
```

```
void merge(int arr[], int left, int mid, int right) {
    int n1 = mid - left + 1;
    int n2 = right - mid;
    int *L = new int[n1], *R = new int[n2];
    for (int i = 0; i < n1; i++) L[i] = arr[left + i];
    for (int i = 0; i < n2; i++) R[i] = arr[mid + 1 + i];
    int i = 0, j = 0, k = left;
    while (i < n1 && j < n2) {
        arr[k++] = (L[i] <= R[j]) ? L[i++] : R[j++];
    }
    while (i < n1) arr[k++] = L[i++];
    while (j < n2) arr[k++] = R[j++];
    delete[] L;
    delete[] R;
}
// Sequential Merge Sort
void mergeSort(int arr[], int left, int right) {
    if (left < right) {
        int mid = left + (right - left) / 2;
        mergeSort(arr, left, mid);
        mergeSort(arr, mid + 1, right);
        merge(arr, left, mid, right);
    }
}
void parallelMergeSort(int arr[], int left, int right)
{
    if (left < right) {
        int mid = left + (right - left) / 2;
        #pragma omp parallel sections
        {
            #pragma omp section
            parallelMergeSort(arr, left, mid);
            #pragma omp section
            parallelMergeSort(arr, mid + 1, right);
        }
        merge(arr, left, mid, right);
    }
}
void printArray(int arr[], int n) {
    for (int i = 0; i < n; i++) {
        cout << arr[i] << " ";
    }
}
void measurePerformance(int arr[], int n) {
    int *arr1 = new int[n];
    int *arr2 = new int[n];
    int *arr3 = new int[n];
    int *arr4 = new int[n];
    // Copy input array to ensure fair comparison
    copy(arr, arr + n, arr1);
    copy(arr, arr + n, arr2);
    copy(arr, arr + n, arr3);
    copy(arr, arr + n, arr4);
    cout << "\nOriginal Array: ";
    printArray(arr, n);
    double start, end;
    // Sequential Bubble Sort
    start = omp_get_wtime();
    bubbleSort(arr1, n);
    end = omp_get_wtime();
    cout << "Sequential Bubble Sort Time: " << (end - start) << " seconds\n";
    cout << "Sorted Array (Bubble Sort): ";
    printArray(arr1, n);
    // Parallel Bubble Sort
    start = omp_get_wtime();
    parallelBubbleSort(arr2, n);
    end = omp_get_wtime();
    cout << "Parallel Bubble Sort Time: " << (end - start) << " seconds\n";
    cout << "Sorted Array (Parallel Bubble Sort): ";
    printArray(arr2, n);
    // Sequential Merge Sort
    start = omp_get_wtime();
    mergeSort(arr3, 0, n - 1);
    end = omp_get_wtime();
    cout << "Sequential Merge Sort Time: " << (end - start) << " seconds\n";
    cout << "Sorted Array (Merge Sort): ";
    printArray(arr3, n);
    // Parallel Merge Sort
    start = omp_get_wtime();
    parallelMergeSort(arr4, 0, n - 1);
    end = omp_get_wtime();
    cout << "Parallel Merge Sort Time: " << (end - start) << " seconds\n";
    cout << "Sorted Array (Parallel Merge Sort): ";
    printArray(arr4, n);
    delete[] arr1;
    delete[] arr2;
    delete[] arr3;
    delete[] arr4;
}
int main() {
    int n;
    cout << "Enter the number of elements: ";
    cin >> n;
    int *arr = new int[n];
    cout << "Enter " << n << " elements: ";
    for (int i = 0; i < n; i++) {
        cin >> arr[i];
    }
    measurePerformance(arr, n);
    delete[] arr;
    return 0;
}
```

Output :



```
Enter the number of elements: 6
Enter 6 elements: 12 56 10 45 5 32

Original Array: 12 56 10 45 5 32
Sequential Bubble Sort Time: 0 seconds
Sorted Array (Bubble Sort): 5 10 12 32 45 56
Parallel Bubble Sort Time: 0.00100017 seconds
Sorted Array (Parallel Bubble Sort): 5 10 12 45 56 32
Sequential Merge Sort Time: 0 seconds
Sorted Array (Merge Sort): 5 10 12 32 45 56
Parallel Merge Sort Time: 0.000999928 seconds
Sorted Array (Parallel Merge Sort): 5 10 12 32 45 56

--------------------------------
Process exited after 19.12 seconds with return value 0
Press any key to continue . . .
```