MergeSort

```
code = """
#include<bits/stdc++.h>
#include<omp.h>

using namespace std;

// reusable func. called recursively
void merge(vector<int> &arr, int start, int mid, int end) {
    int len = (end - start) + 1;

    // temp array used in merge sort
    int temp[len];
    int cur = 0;

    int i = start;
    int j = mid + 1;
    while(i <= mid && j <= end){
        if(arr[i] < arr[j]) {
            temp[cur] = arr[i];
            cur++;
            i++;
        }
        else {
            temp[cur] = arr[j];
            cur++;
            j++;
        }
    }
    if(i <= mid) {
        while(i <= mid) {
            temp[cur] = arr[i];
            i++;
            cur++;
        }
    }

    else if(j <= end) {
        while(j <= end) {
            temp[cur] = arr[j];
            j++;
            cur++;
        }
    }

    cur = 0;
    for(i=start; i<=end; i++) {
        arr[i] = temp[cur];
        cur++;
    }

}

// serial caller method
void serialMergeSort(vector<int> &arr, int start, int end) {
    if(start < end) {
        // avoid overflow
        int mid = start + (end-start) / 2;

        serialMergeSort(arr, start, mid);
        serialMergeSort(arr, mid+1, end);

        merge(arr, start, mid, end);
    }
}

// parallel caller method
void parallelMergeSort(vector<int> &arr, int start, int end) {
    if(start < end) {
        // avoid overflow
        int mid = (start + end) / 2;

        #pragma omp parallel sections
        {
            #pragma omp section
            serialMergeSort(arr, start, mid);

            #pragma omp section
            serialMergeSort(arr, mid+1, end);
        }

        merge(arr, start, mid, end);

    }
}

int main(int argc, char *argv[]) {
    int size = 50000;
    vector<int> a;

    double start, end;

    omp_set_num_threads(2);

    for(int i=0; i<size; i++) {
        a.push_back(rand()% 1000);
    }

    vector<int> ar1, ar2;
```

```
ar1 = a;
ar2 = a;

//int a[]= {7,33,5,5,23,111,75,34,77,121,120};

cout<<"Input list:  ";
for(int i=0; i<size; i++)
    cout<<a[i]<<" ";
cout<<endl<<endl;

start = omp_get_wtime();
serialMergeSort(ar1, 0, size-1);
end = omp_get_wtime();

/*
cout<<"Merge Sorted List(serial):  ";
for(int i=0; i<size; i++)
  cout<<ar1[i]<<" ";
cout<<endl<<endl;
*/

// in seconds
cout<<"Execution time(serial) = "<<(end-start)<<" seconds"<<endl;


start = omp_get_wtime();
parallelMergeSort(ar2, 0, size-1);
end = omp_get_wtime();

/*
cout<<"Merge Sorted List(parallel):  ";
for(int i=0; i<size; i++)
  cout<<ar2[i]<<" ";
   cout<<endl<<endl;
   */

   // in seconds
   cout<<"Execution time(parallel) = "<<(end-start)<<" seconds"<<endl;

   return 0;
}

"""
```

```
text_file = open("merge11.cpp", "w")
text_file.write(code)
text_file.close()
```

```
!g++ -fopenmp merge11.cpp
```

```
!./a.out
```

BubbleSort

```
code = """
#include<omp.h>
#include<iostream>
using namespace std;

void serialBubbleSort(int arr[], int n) {

   for(int i = 0; i < n; i++) {
      for(int j = 0; j < n-i-1; j++) {
         if(arr[j] > arr[j+1]) swap(arr[j], arr[j+1]);
      }
   }
}

void parallelBubbleSort(int arr[], int n) {
   for(int i = 0; i < n-1; i++) {
      int first = i%2;
      #pragma omp parallel for
      for(int j = first; j < n-1; j += 2) {
         if(arr[j] > arr[j+1]) swap(arr[j], arr[j+1]);
      }
   }
}

void swap(int *num1, int *num2) {
   int temp = *num1;
   *num1 = *num2;
   *num2 = temp;
}

int main() {
   int n = 20000;
   int a[n];

   omp_set_num_threads(2);

   for(int i=0; i<n; i++) {
      a[i] = rand()% 100;
   }

   cout<<"Input list:  ";
   for(int i=0; i<n; i++)
      cout<<a[i]<<" ";
   cout<<endl<<endl;
```

```
    double start, end;

    start = omp_get_wtime();
    serialBubbleSort(a, n);
    end = omp_get_wtime();

    /*
    cout<<"Bubble Sorted List(serial):  ";
    for(i=0; i<n; i++)
      cout<<a[i]<<" ";
    cout<<endl<<endl;
    */

    cout<<"Execution time(serial) = "<<(end-start)<<" seconds"<<endl;


    start = omp_get_wtime();
    parallelBubbleSort(a, n);
    end = omp_get_wtime();

    cout<<"Execution time(parallel) = "<<(end-start)<<" seconds"<<endl;

    return 0;

}
"""
```

```
text_file = open("code.cpp", "w")
text_file.write(code)
text_file.close()
```

```
!g++ -fopenmp code.cpp
```

```
!./a.out
```