Rahul Nama GC63170

Question: 2

Bucket Sort Scalability Graph:
X-Axis: Size of Arrays from 1,00,000 to 6,00,000,000
Y-Axis: Execution time for each size

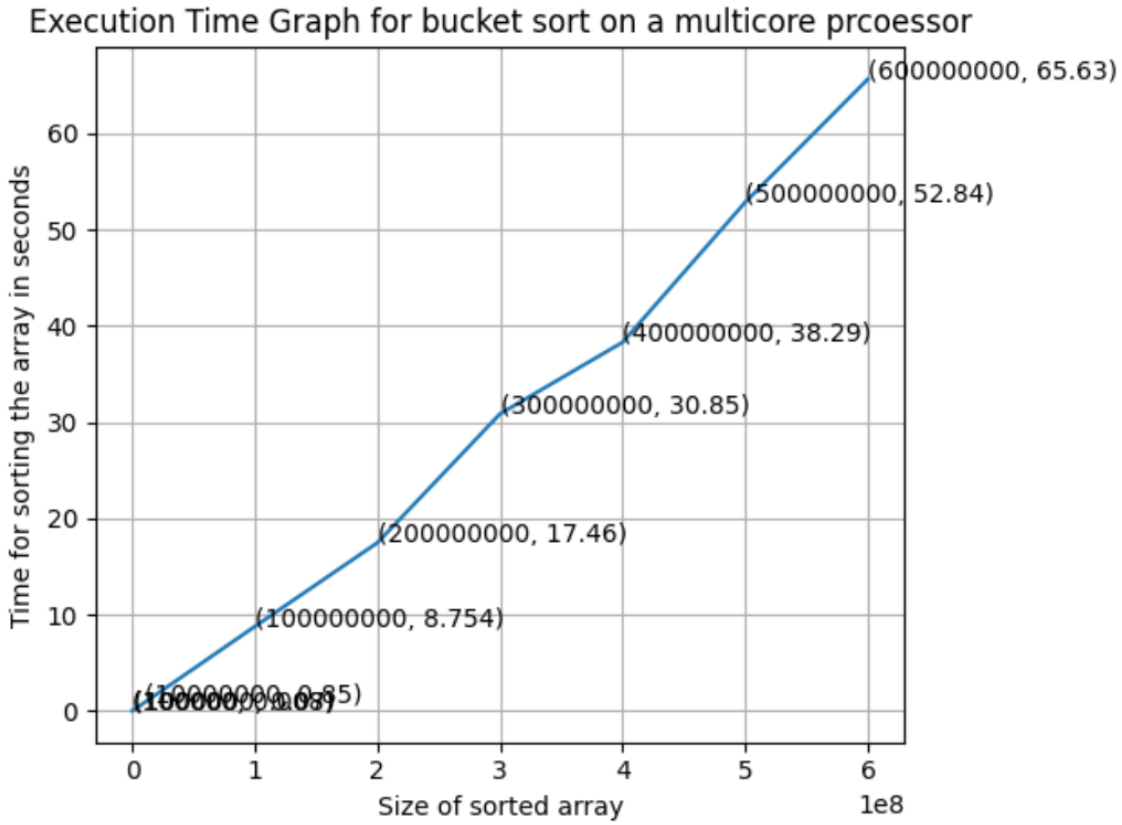
Number of Processes used: 4 (for first graph)

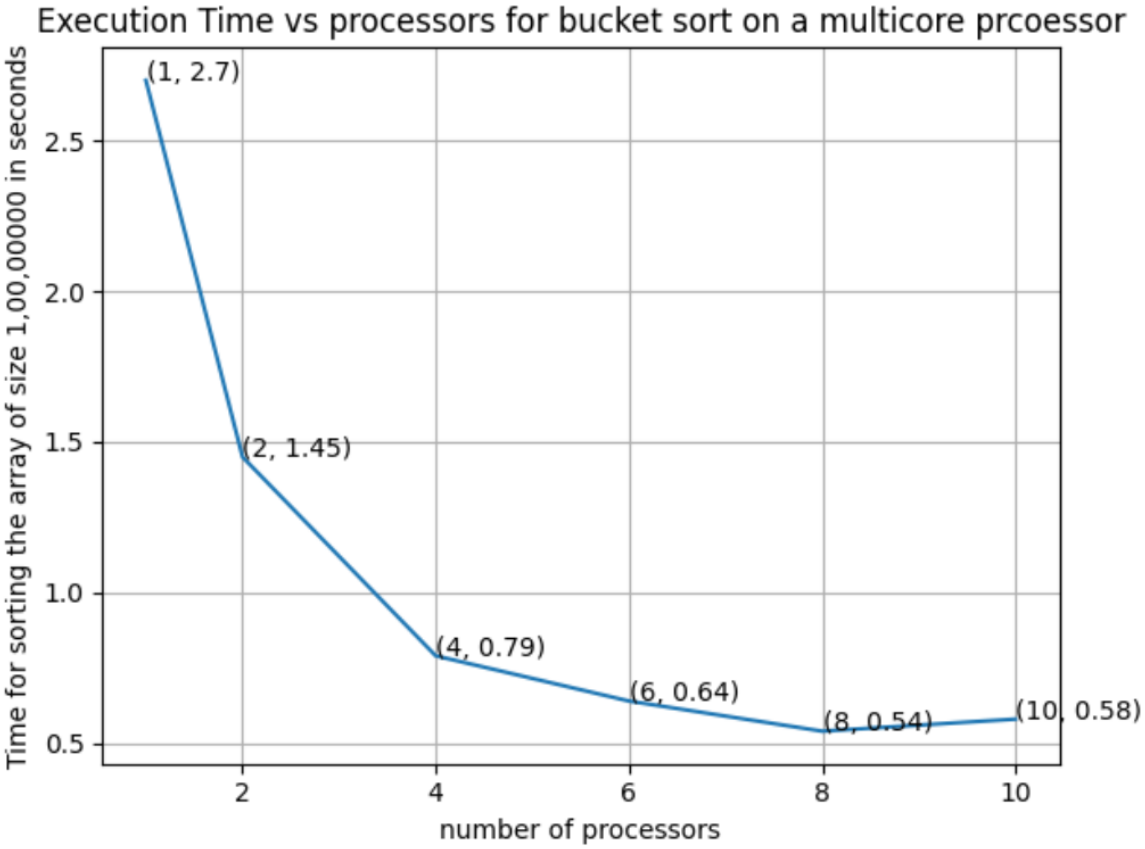
OS: Mac OS X
Time taken to generate and sort 6,00,000,000 is 65 seconds using 4 processors

Array Size	Execution Time
100000	0.008
1000000	0.07
10000000	0.850
100000000	8.754
200000000	17.46
300000000	30.85
400000000	38.29
500000000	52.84
600000000	65.63

Rahul Nama GC63170

Time took to generate and sort 6,00,000,000 elements is 53 seconds using 4 processors





Rahul Nama GC63170

Question: 3

Merge Sort Scalability Graph:

X-Axis: Size of Arrays from 1,00,000 to 6,00,000,000
Y-Axis: Execution time for each size

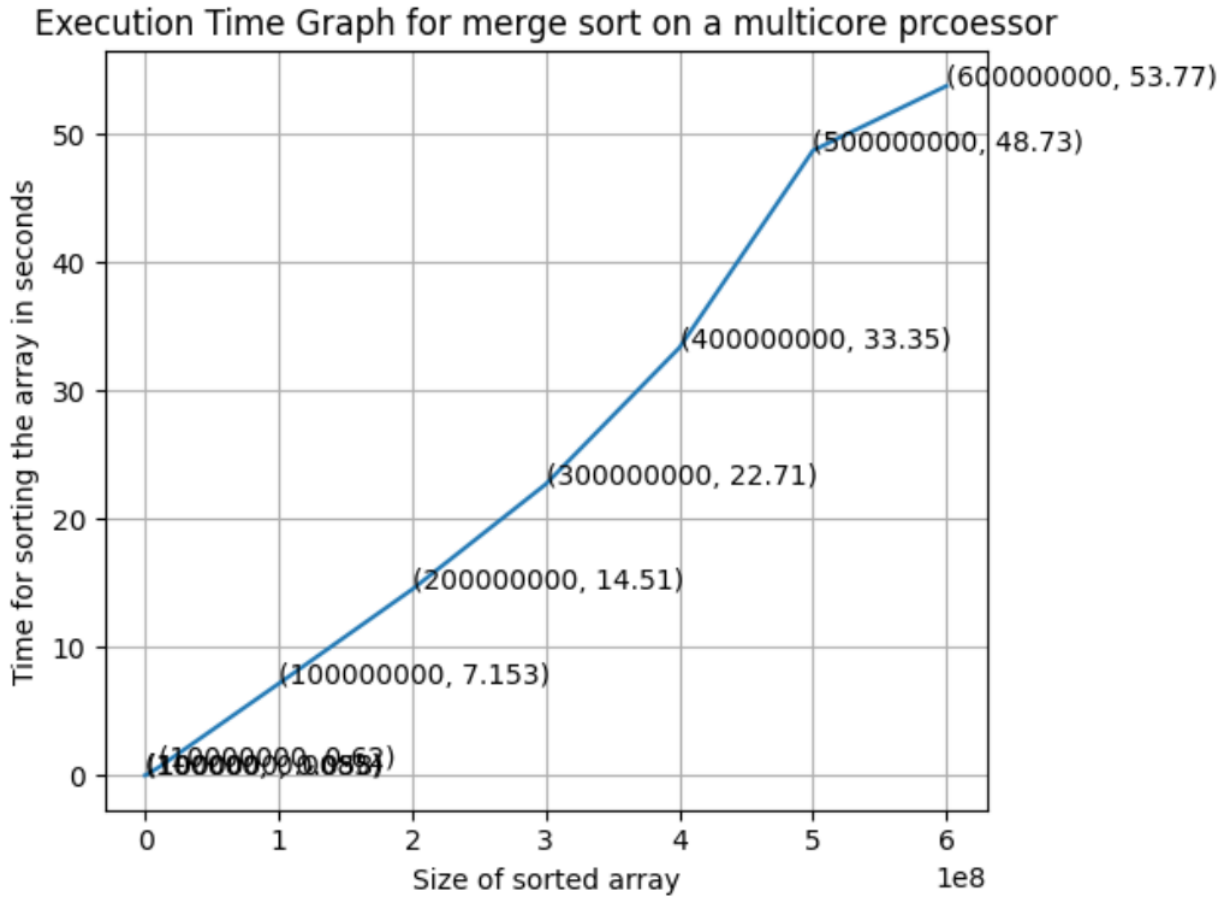
Number of Processes used: 4

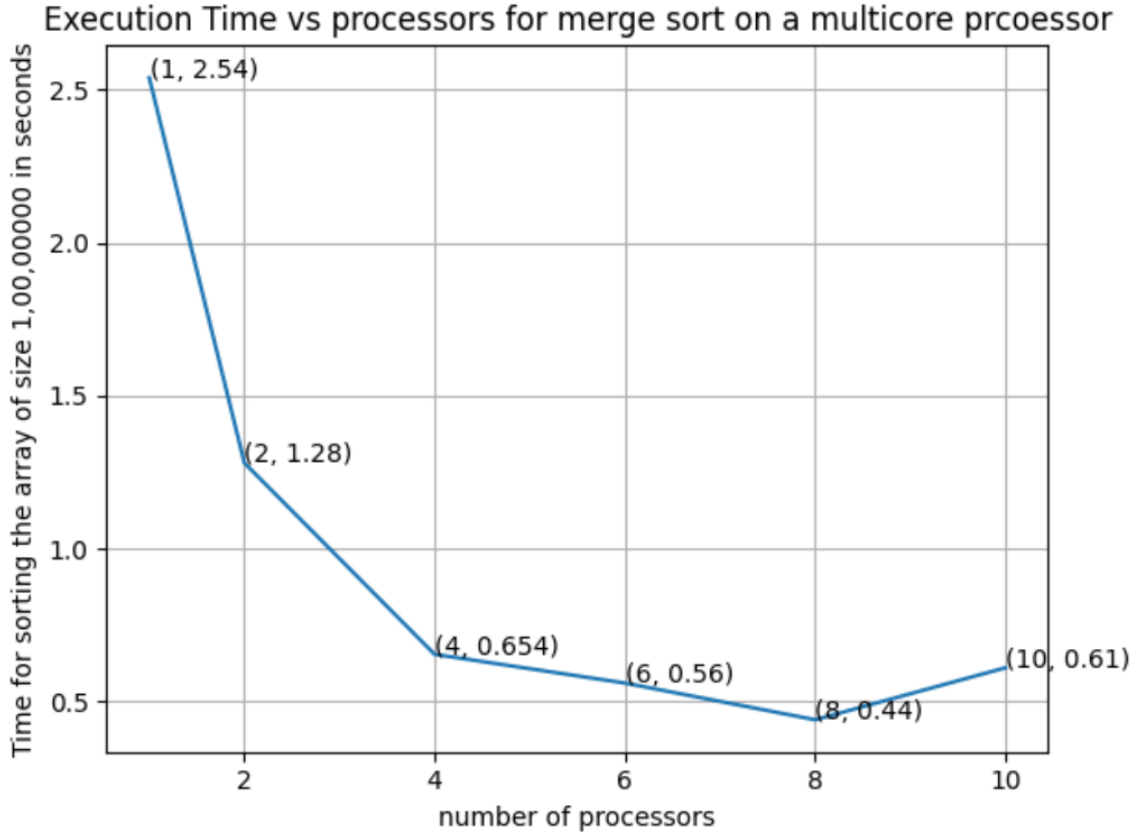
OS: Mac OS X

Time took to generate and sort 6,00,000,000 elements is 53 seconds using 4 processors

Array Size	Execution Time
100000	0.0085
1000000	0.058
10000000	0.62
100000000	7.153
200000000	14.51
300000000	22.71
400000000	33.35
500000000	48.73
600000000	53.77

Rahul Nama GC63170





Comparison:

Rahul Nama GC63170

From the graphs, we can say that the merge sort is faster than bucket sort. Though for small array sizes execution time of both algorithms is almost same, execution time differed as we increase the array size and processors. One of the main reasons for the slowness of bucket sort is the method it follows to assign the values into each sub-bucket. As all the buckets may not end up having equal elements. Most of the time, few buckets have many values and other buckets are empty which degrades the performance as it takes more time to sort each individual bucket if it has more elements.

Also, upto 8 processors' execution time of both algorithms is decreasing. However, after 8 processors it started increasing as processor communication overhead is adding up.

Execution:

```
compile: bucket_sort.o merge_sort.o

bucket_sort.o: bucket_sort.c
    mpicc -o bucketsort bucket_sort.c
merge_sort.o: merge_sort.c
    mpicc -o mergesort merge_sort.c
run_bucket_sort: bucket_sort.o
    mpiexec -np 4 ./bucketsort 10000
run_merge_sort: merge_sort.o
    mpiexec -np 4 ./mergesort 10000
clean:
    rm -rf bucketsort mergesort

```

Make compile: compiles both algorithms

Run_bucket_sort: to run bucket sort with 4 processors and 10,000 array size

Run_merge_sort: to run merge sort with 4 processors and 10,000 array size