Felicity Escarzaga
CS 599: HPC
Assignment 3

Activity 1:
In this program, the buckets are determined by starting at the rank number times the maximum value divided by the number of ranks (`my_start = my_rank * MAXVAL / nprocs`) and ending at the start plus the max value divided by the number of ranks (`my_end = my_start + MAXVAL / nprocs`). It then iterates for each process rank (`i < nprocs`) then for each value in the local data set (`j < oldData_len`) and determines if a value belongs to it's own bucket or to the bucket of the indexed process rank; then adds the value to either it's updated data list (`newData`) or to the send buffer (`sendDataSetBuffer`).
Once it has removed all values not in the current ranks bucket; it iterates through all process ranks twice, once to send the data (`MPI_Isend`) to each of the other ranks, then to receive data from each of the other ranks. When all the received data has been saved to the current rank's data set (`myDataSet`) it is sorted using quick sort (`qsort()`).

Question 1: *Based on the problem description, should distribution sort have low or high load imbalance?*

The distribution sort should have a low load imbalance since the data is uniformly distributed when generated and split equally in the buckets before being sorted by each rank.

Question 2: *Is there anything different about the algorithm when p=1?*

The algorithm does not split the data and therefore must sort all the data on one process.

Table 1: Activity 1

# of Ranks (p)	Total Time (s)	Time to Distribute (s)	Time to Sort (s)	Parallel Speedup	Parallel Efficiency	Global Sum	Job Script Name (*.sh)
1	155.631943	2.640046	152.991897	1	1	499939833260556	sort_act1_fhe2
2	78.64891	5.174835	73.474075	1.978818816	0.9894094082	499934584339325	sort_act1_fhe2
4	40.843034	5.544557	35.302445	3.810489275	0.9526223187	499937155109296	sort_act1_fhe2
8	21.81217	4.863945	16.986676	7.135096737	0.8918870922	499925398982548	sort_act1_fhe2
12	15.293141	4.294266	11.03896	10.17658459	0.8480487157	499924679946291	sort_act1_fhe2
16	12.003844	3.86736	8.197294	12.96517541	0.8103234628	499932728507365	sort_act1_fhe2
20	10.057753	3.677298	6.426014	15.4738283	0.773691415	499937769104586	sort_act1_fhe2

Question 3: *Does the time to distribute the data vary as a function of p?*

The time to distribute the data does vary as a function of p. It increases first, likely since with 1 process it does not actually send or receive any data, which is why it is lowest with 1 rank. However, after 4 ranks the time decreases since with more ranks generating less local data each rank does not need to iterate through its own data as many times.

Felicity Escarzaga
CS 599: HPC
Assignment 3

Question 4: *Does the time to sort the data vary as a function of p?*

The time to sort decreases as a function of p, since with more ranks the data is already split and therefore quicksort has less data to sort for each rank.

Question 5: *How does distribution sort scale with increasing p?*

As p increases the time decreases therefore the distribution sort benefits as p increases and scales well.

Question 6: *How does the parallel efficiency scale with increasing p?*

The parallel efficiency decreases as p increases therefore the parallel efficiency does not scale well with increasing p.

Question 7: *What is the bottleneck in distribution sort as p increases?*

The time to distribute does not decrease as fast as the rate at which time for sorting decreases. This means that as p increases the time to distribute will eventually be greater than the time to sort and likely continue to increase. Therefore the bottleneck of distribution sorting is the time to distribute.

Felicity Escarzaga
CS 599: HPC
Assignment 3

Activity 2:
This activity was programmed with the same bucket method as the first activity. The only difference is that this program generates exponential data whereas the first program generated uniform data.

Table 2: Activity 2

# of Ranks (p)	Total Time (s)	Time to Distribute (s)	Time to Sort (s)	Parallel Speedup	Parallel Efficiency	Global Sum	Job Script Name (*.sh)
1	153.506704	2.634551	150.872153	1	1	231347569226388	sort_act2_fhe2
2	136.041677	4.357459	131.684218	1.128379974	0.5641899872	231345039566716	sort_act2_fhe2
4	97.564423	4.48727	93.077153	1.573388119	0.3933470298	231348974505153	sort_act2_fhe2
8	59.813843	4.02062	55.793223	2.566407646	0.3208009557	231346368407353	sort_act2_fhe2
12	42.91398	3.43144	39.48254	3.577079171	0.2980899309	231344540852148	sort_act2_fhe2
16	33.409113	2.950175	30.458937	4.594755449	0.2871722156	231346601181947	sort_act2_fhe2
20	27.11185	2.649371	24.462478	5.661978212	0.2830989106	231347819319091	sort_act2_fhe2

Question 8: *Does performance differ between the uniformly distributed (Programming Activity #1) and exponentially distributed data? If so, explain the performance discrepancy.*

The performance decreases greatly since the data is no longer evenly distributed between the ranks. Given that the data is exponentially distributed, some buckets and therefore ranks will have much more data whereas others will have little to none to sort.

Felicity Escarzaga
CS 599: HPC
Assignment 3

Question 9: *Based on the description above, are there any additional overheads to the algorithm?*

The algorithm will require precomputing the bucket sizes by first putting the ranks 0's data into bins to create a histogram then evenly distributing the bins into buckets for each rank.

Activity 3:
As stated above this algorithm precomputes rank 0's data into bins then iterates through all the data to create a count for each bin. The counts are divided into buckets until they are at or just below the local data size divided by the number of ranks (`counts = localN/nprocs`). The bucket size for all ranks is then broadcast (`MPI_ Bcast`) to all other ranks.

Table 3: Activity 3

# of Ranks (p)	Total Time (s)	Time to Distribute (s)	Time to Sort (s)	Parallel Speedup	Parallel Efficiency	Global Sum	Job Script Name (*.sh)
1	182.231058	3.81416	178.416899	1	1	231347569226388	sort_act3_fhe2
2	93.243799	5.79717	87.44663	1.954350423	0.9771752114	231345039566716	sort_act3_fhe2
4	48.316212	5.820185	42.806414	3.771633794	0.9429084486	231348974505153	sort_act3_fhe2
8	25.890305	5.034303	20.995065	7.038582898	0.8798228623	231346368407353	sort_act3_fhe2
12	18.140529	4.436997	13.807646	10.04552061	0.8371267177	231344540852148	sort_act3_fhe2
16	14.445222	4.037877	10.411913	12.61531723	0.7884573269	231346601181947	sort_act3_fhe2
20	12.145803	3.733125	8.456347	15.00362372	0.750181186	231347819319091	sort_act3_fhe2

Question 10: *How does the histogram solution compare to the performance achieved in Programming Activity #2?*

The histogram solution is much faster than the program for activity 2 and has a much higher parallel efficiency though it does continue to decrease. Though it is the better solution to sorting exponential data.

Question 11: *How does the histogram solution compare to the performance achieved in Programming Activity #1?*

The histogram solution is neither as fast as activity 1 nor does it have a better parallel efficiency making it the poor solution between the two for sorting data.

Question 12: *Do you think all distribution sort implementations (e.g., libraries) based on bucketing should use a histogram?*

I think that it depends on the data whether or not a distribution sort should use a histogram since it is slower then the previous program meant for uniform data (activity 1).

Felicity Escarzaga
CS 599: HPC
Assignment 3

Question 13: *Can you think of a method that can be used to reduce the overhead of the histogram procedure?*

Increasing the bin size based on the max value may help decrease the overhead of the histogram solution.
Another solution might be to parallelize the binding process since the other ranks won't be able to continue until after the histogram procedure, but this will have diminishing returns.