**Q1: Based on the problem description, should distribution sort have low or high load imbalance?**

**Ans:** If the data is uniformly distributed (as in Programming activity#1), there should be no load imbalance. On the other hand, non-uniformly distributed data (as in Programming activity#2) will cause load imbalance.

**Q2: Is there anything different about the algorithm when p == 1?**

**Ans:** Yes, when p =1, that is there’s only one process, the algorithm essentially becomes sequential in nature. Also, there will be no exchange of buckets between processes as there’s only one process (p == 1).

| **# of Ranks (p)** | **Total Time (s)** | **Time to Distribute (s)** | **Time to Sort (s)** | **Parallel Speedup** | **Parallel Efficiency** | **Global Sum** | **Job Script Name** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 160.517614 | 4.423213 | 154.115881 |  |  | 499939833260556 | CS599\_HW3\_mh2752\_act1\_nprocs\_1.sh |
| 2 | 81.890555 | 7.178106 | 74.041471 | 1.96 | 0.98 | 499934584339325 | CS599\_HW3\_mh2752\_act1\_nprocs\_2.sh |
| 4 | 41.285711 | 5.603185 | 35.428834 | 3.89 | 0.97 | 499937155109296 | CS599\_HW3\_mh2752\_act1\_nprocs\_4.sh |
| 8 | 20.778549 | 3.604157 | 17.063612 | 7.73 | 0.97 | 499925398982548 | CS599\_HW3\_mh2752\_act1\_nprocs\_8.sh |
| 12 | 13.941407 | 2.806217 | 11.050261 | 11.51 | 0.96 | 499924679946291 | CS599\_HW3\_mh2752\_act1\_nprocs\_12.sh |
| 16 | 10.670937 | 2.433403 | 8.182177 | 15.04 | 0.94 | 499932728507365 | CS599\_HW3\_mh2752\_act1\_nprocs\_16.sh |
| 20 | 8.656845 | 2.178560 | 6.423371 | 18.54 | 0.93 | 499937769104586 | CS599\_HW3\_mh2752\_act1\_nprocs\_20.sh |

**Table-1: Programming activity 1**

**Q3: Does the time to distribute the data vary as a function of p?**

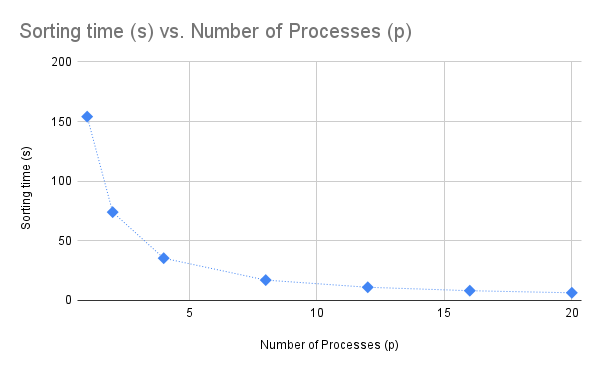
**Ans:** Yes, for p =1 to 2, it increases. For p = 2 to 20, the time decreases. As this time includes the times required to calculate the data ranges, preparing and populating the bins, sending the bins to other processes, and receiving bins from other processes, this total time to distribute varies. When p is 2, the time required to distribute is almost double the time required to distribute when p is 1. For p =2, both of the processes have to send and receive almost *localN/2* number of data points from each other which causes the distribution time to increase for p =2 compared to p =1. As p increases from 2 to 20, the number of data points shared among all pairs of processes decreases as *localN* decreases. As a result, even though the number of communications increases with p increasing, the total time to distribute keeps decreasing with increasing p values.

**Q4: Does the time to sort the data vary as a function of p?**

**Ans:** Yes, as p increases, time to sort decreases.

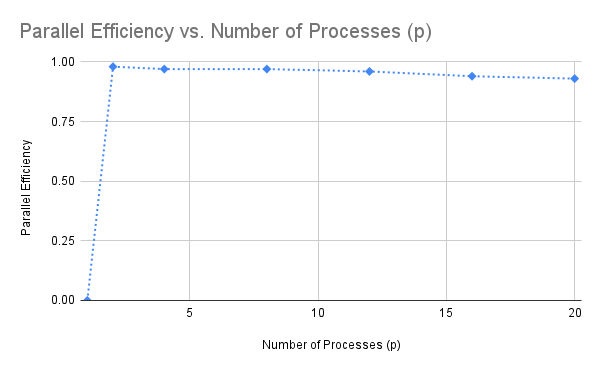
**Q5: How does distribution sort scale with increasing p?**

**Ans:** The time required to sort decreases almost exponentially with the increasing value of p as shown in the plot below.



**Q6: How does the parallel efficiency scale with increasing p?**

**Ans:** The parallel efficiency remains almost the same with an increasing value of p as shown in the plot below.



**Q7: What is the bottleneck in distribution sort as p increases?**

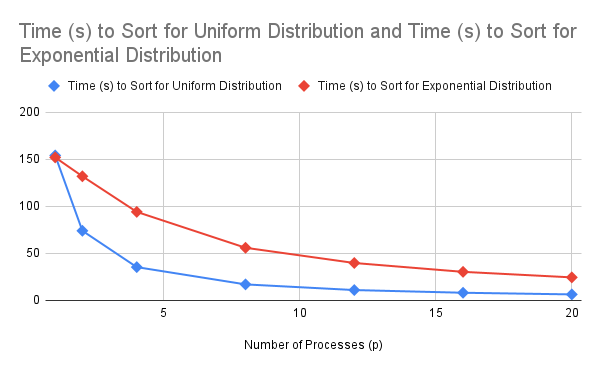
**Ans:** As p increases, the time required to sort will decrease but the number of communications between pairs of processes will increase. Eventually, with large enough p >> 20, the communication between process pairs will become the bottleneck.

| **# of Ranks (p)** | **Total Time (s)** | **Time to Distribute (s)** | **Time to Sort (s)** | **Parallel Speedup** | **Parallel Efficiency** | **Global Sum** | **Job Script Name** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 159.054836 | 5.034207 | 152.028257 |  |  | 231347569226388 | CS599\_HW3\_mh2752\_act2\_nprocs\_1.sh |
| 2 | 136.939259 | 3.811943 | 131.972076 | 1.16 | 0.58 | 231345039566716 | CS599\_HW3\_mh2752\_act2\_nprocs\_2.sh |
| 4 | 98.945549 | 4.174594 | 94.136591 | 1.61 | 0.40 | 231348974505153 | CS599\_HW3\_mh2752\_act2\_nprocs\_4.sh |
| 8 | 59.895343 | 3.566605 | 55.985885 | 2.66 | 0.33 | 231346368407353 | CS599\_HW3\_mh2752\_act2\_nprocs\_8.sh |
| 12 | 42.708413 | 2.647618 | 39.816202 | 3.72 | 0.31 | 231344540852148 | CS599\_HW3\_mh2752\_act2\_nprocs\_12.sh |
| 16 | 32.684274 | 2.050252 | 30.399434 | 4.87 | 0.30 | 231346601181947 | CS599\_HW3\_mh2752\_act2\_nprocs\_16.sh |
| 20 | 26.441527 | 1.771038 | 24.519276 | 6.02 | 0.30 | 231347819319091 | CS599\_HW3\_mh2752\_act2\_nprocs\_20.sh |

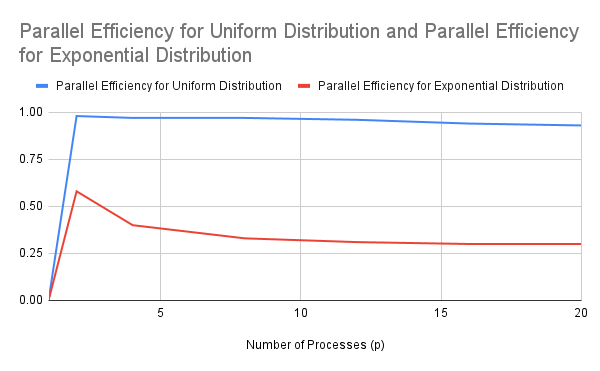
**Table-2: Programming activity 2**

**Q8: Does performance differ between the uniformly distributed (Programming Activity #1) and exponentially distributed data? If so, explain the performance discrepancy.**

**Ans:** Yes, we see a significant decrease in terms of performance between Activity #1 and Activity #2. First, the sorting time in Activity #2 is significantly higher than the corresponding sorting times in Activity #1 for the same p values as shown in the plot below:



Also, the parallel efficiencies in Activity #2 decreases significantly compared to the parallel efficiencies in Activity #1 as shown below:



As the data distribution and the computation load in Activity #2 are imbalanced among processes, we see these significant performance drops in terms of sorting times and parallel efficiencies.

**Q9: Based on the description above, are there any additional overheads to the algorithm?**

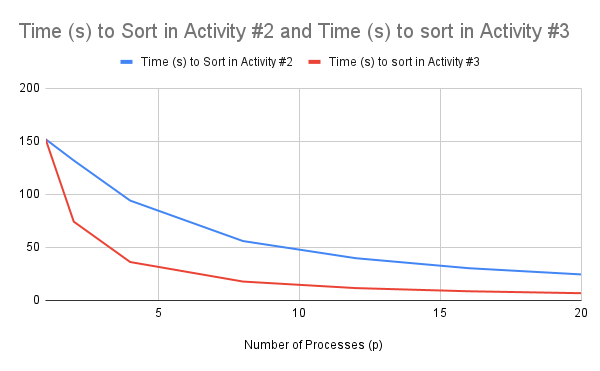
**Ans:** Yes, the calculation of the histogram and CDF incur additional computation overhead compared to Activity #1 and Activity #2.

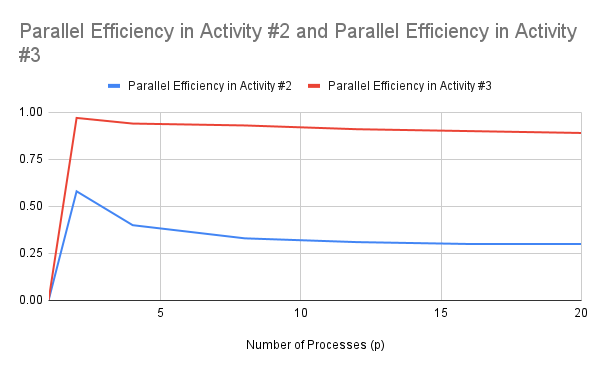
| **# of Ranks (p)** | **Total Time (s)** | **Time to Distribute (s)** | **Time to Sort (s)** | **Parallel Speedup** | **Parallel Efficiency** | **Global Sum** | **Job Script Name** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 161.680187 | 8.909002 | 152.129262 |  |  | 231347569226388 | CS599\_HW3\_mh2752\_act3\_nprocs\_1.sh |
| 2 | 83.568981 | 8.947580 | 74.292277 | 1.93 | 0.97 | 231345039566716 | CS599\_HW3\_mh2752\_act3\_nprocs\_2.sh |
| 4 | 43.151357 | 6.764360 | 36.223362 | 3.75 | 0.94 | 231348974505153 | CS599\_HW3\_mh2752\_act3\_nprocs\_4.sh |
| 8 | 21.834069 | 3.956330 | 17.805790 | 7.40 | 0.93 | 231346368407353 | CS599\_HW3\_mh2752\_act3\_nprocs\_8.sh |
| 12 | 14.738494 | 3.062513 | 11.640068 | 10.97 | 0.91 | 231344540852148 | CS599\_HW3\_mh2752\_act3\_nprocs\_12.sh |
| 16 | 11.267948 | 2.575877 | 8.651316 | 14.35 | 0.90 | 231346601181947 | CS599\_HW3\_mh2752\_act3\_nprocs\_16.sh |
| 20 | 9.074534 | 2.212080 | 6.828769 | 17.82 | 0.89 | 231347819319091 | CS599\_HW3\_mh2752\_act3\_nprocs\_20.sh |

**Table-3: Programming activity 3**

**Q10: How does the histogram solution compare to the performance achieved in Programming Activity #2?**

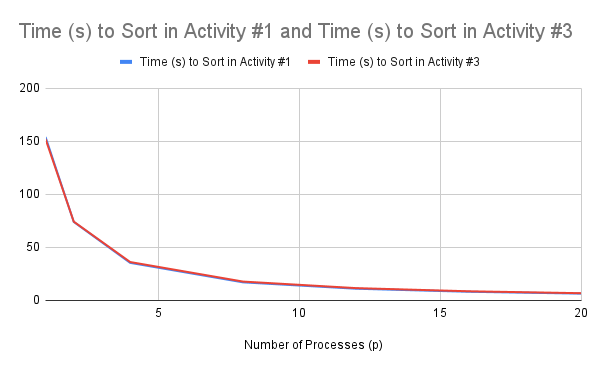
**Ans:** Compared to the performance of Activity #2, the histogram solution here in Activity #3 performs much better in terms of time to sort and parallel efficiency as shown in the plots below:

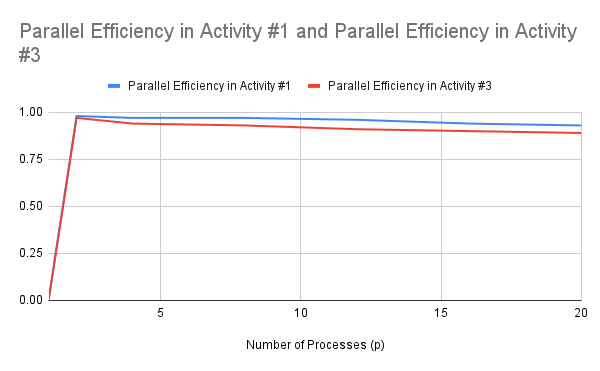




**Q11: How does the histogram solution compare to the performance achieved in Programming Activity #1?**

**Ans:** The performances of Activity #1 and Activity #3 are very similar as shown by the following plots:





**Q12: Do you think all distribution sort implementations (e.g., libraries) based on bucketing should use a histogram?**

**Ans:** If the distribution of values is not uniform across the process ranks, then the bucket-based sorting should use a histogram-based approach to avoid load imbalance.

**Q13: Can you think of a method that can be used to reduce the overhead of the histogram procedure?**

**Ans:** Yes. The overhead can possibly be reduced by parallelizing the histogram computation, e.g., using a #pragma omp parallel for … directive.