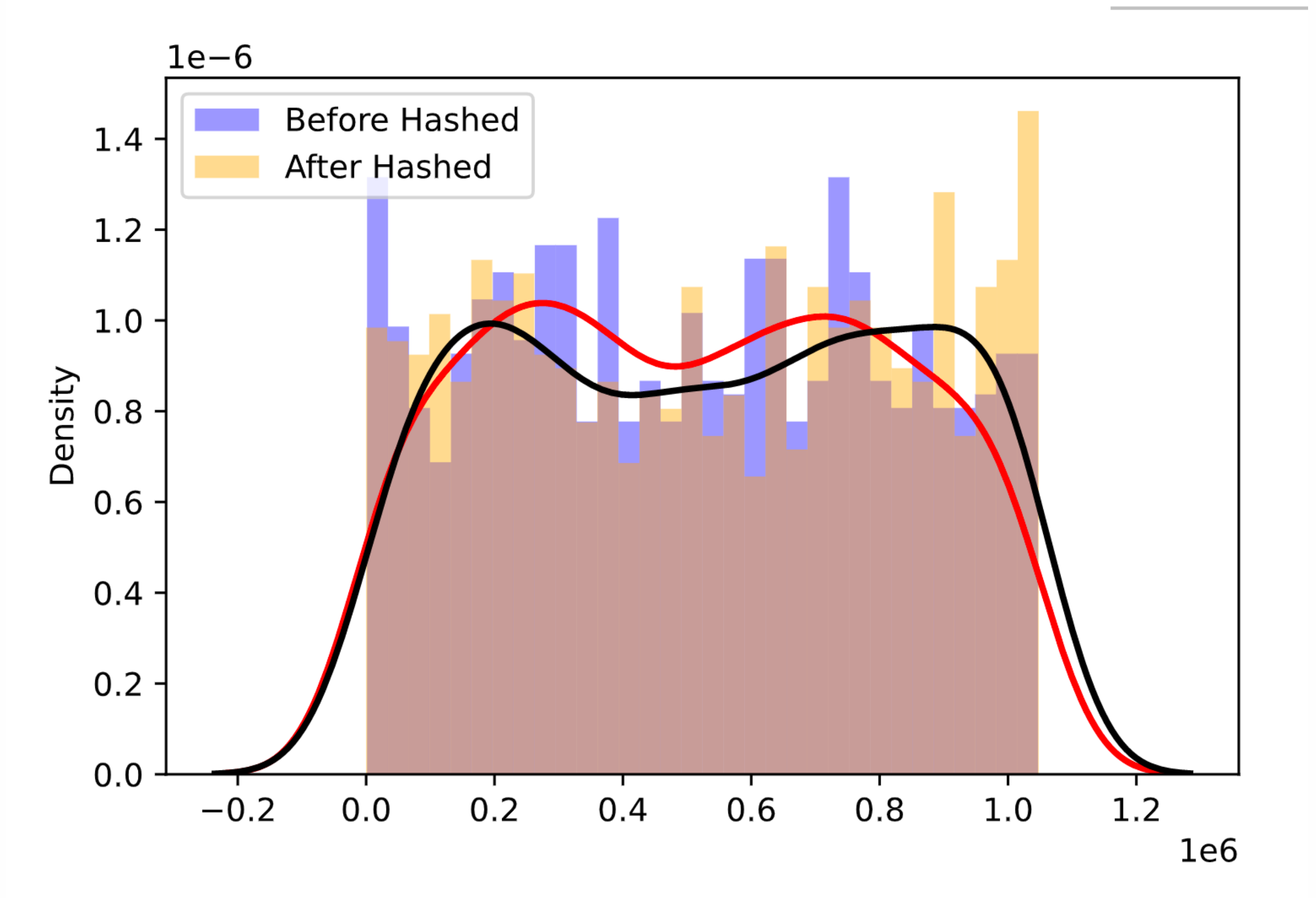
1. *Set – up*—**A description of the experimental set-up to aid reproducibility. For example, include the CPU frequency, memory, operating system, programming language, compiler (if applicable).** [1 mark]**

   I implemented all algorithms in Java, compiled with `OpenJdk javac 14.0.1`. The executables are tested under macOS 10.15.6 on a MacBook Pro with a 2.3GHz dual-core CPU, 64MB of eDRAM and 8GB memory.

2. **A description of steps taken to gain confidence in the implementations (i.e., testing) and ensuring a fair comparison between the algorithms. [1 mark]**

   1. Selecting elements randomly from universe by calling `StdRandom.uniform()`
   2. For BJKST, choose $2^{61}-1$ as prime in hash function so that the size of universe could extend to $2^{20}$, but the trade-off is that use `BigInteger` to do calculation in hash function which could increase the memory usage. For HyperLogLog, choose a prime $\in [2^{32}, 2^{33}]$. We set a fixed universe to $[0, 2^{20}]$ in all questions except for Q7 (detailed explanation is in Q7).
   3. When calculate the memory usage and runtime consumption, do 3 independent calculcations and take the mean of them as result.
   4. Assumed that the distinct count equals to the size of stream.

3. **Verification that the hash function has uniformly distributed output values. [1 mark]****

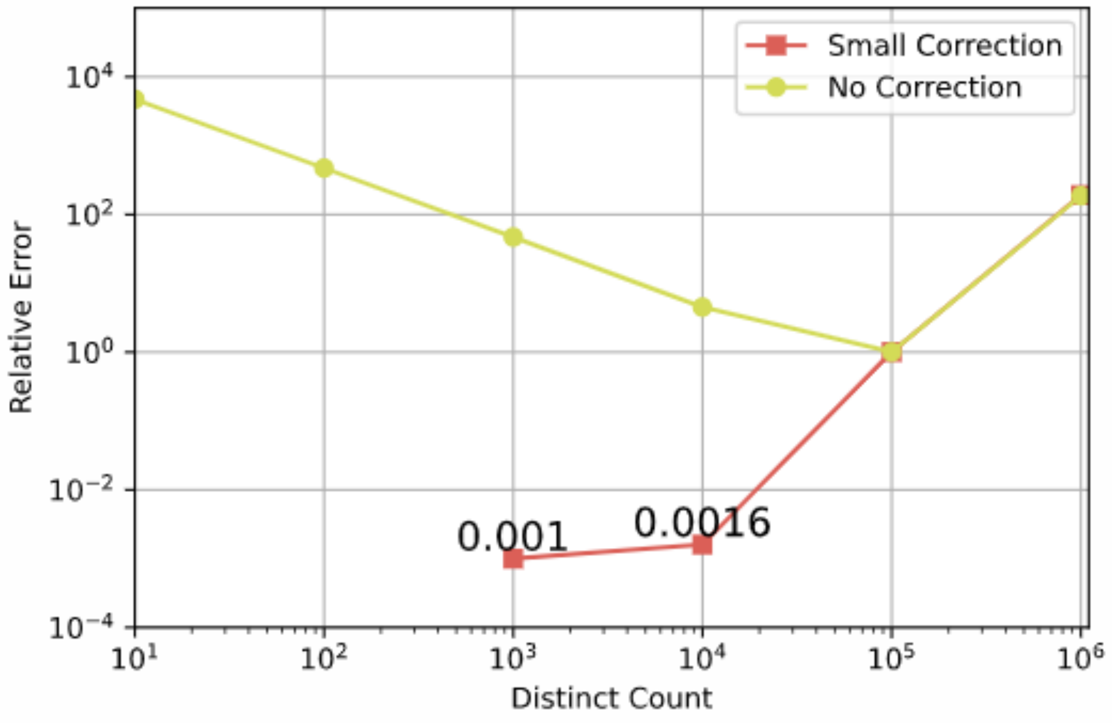


   We choose 1024 elements uniformly from $[0, 2^{20}]$ and hashed them. As the figure shown, hashed value distributes relatively uniformly. And also, the hash function doesn't change the distribution of randomly chosen numbers. Therefore, the hash function has uniformly distributed output values.

4. **An investigation of the effect of the small-range correction to the estimate in the HyperLogLog algorithm. Plot average absolute error vs exact distinct count (up to $10^8$ or more) to compare these effects and comment on any differences observed. [2 marks]***

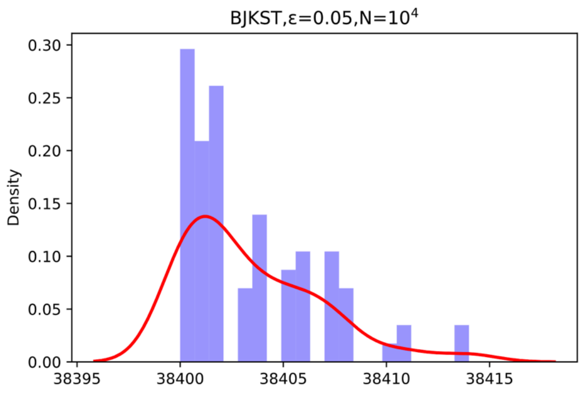
   When b = 16, we could see that HyperLogLog has small range correction when N, the number of distinct count, is less than $10^5$. When N rises up to $10^5$, there is no correction and E* := E because HyperLogLog produces E with 211118 which is larger than 163840; finally, N keeps increasing, when it reaches $10^6$, E exceeds $\frac{1}{30}2^{32}$, HyperLogLog will have large range correction.

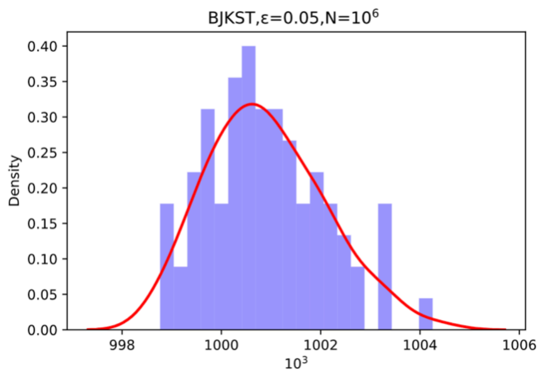
As for the relative error of small range correction, we could see when N = 10 and N = 100, the estimate of the number of distinct elements equals to N and the relative error is 0. As N grows, we could see the relative error keeps less than 0.01. By comparing small range correction and no correction, we could see there is a large gap between the relative error of small range correction and the relative error of no correction when N is small. As N grows, the gap narrows sharply. When N is small, small range correnction has a significant effect for estimate, but the effect becomes weaker as N increases. And the result of our test is in line with the coclusion of the paper which shows that HyperLogLog with small range correnction returns cardinality estimate E* with typical relative error $\pm 1.04/\sqrt{m}$ (equals to 0.4%).

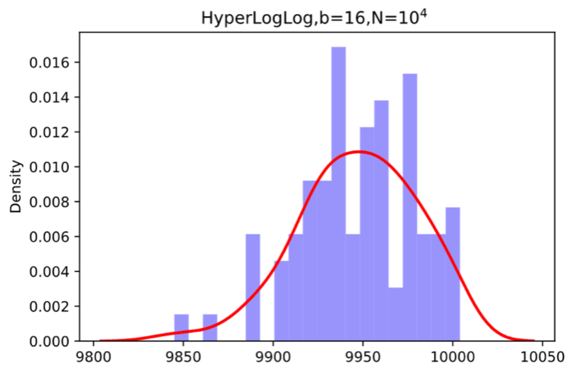


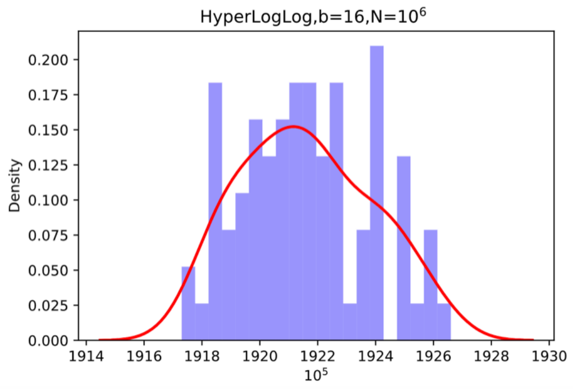
5. **Histograms for the BJKST and HyperLogLog algorithms similar to those given in Figure 4 of [2] showing the distribution of estimates for values of n = $10^4$ and $10^6$ and for each case specify parameters guaranteeing an estimate within ±5% of the true value with probability 0.9. [2 marks]**

By running $36log\frac{1}{\delta}$ independent estimates and take the median as our final estimate, we could boost the success probability to 1-$\delta$. In this case, $\delta$ equals to 0.1. In order to have a guarantee of estimating within ±5%, $\epsilon$ of BJKST is set to 0.05 and the algorithm maintains a max heap of size 38400 which is larger than $10^4$, while m of HyperLogLog is set to 16. Therefore, when N, the distinct count, is $10^4$, we could see that BJKST overestimates N while HyperLogLog works fine. And when the distinct count rises up to $10^6$, BJKST give an estimate with $(1 \pm 0.05)F_0$, while HyperLogLog wildly overestimates the number of distinct elements.





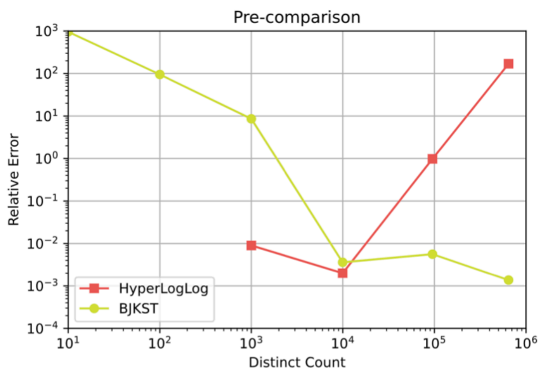


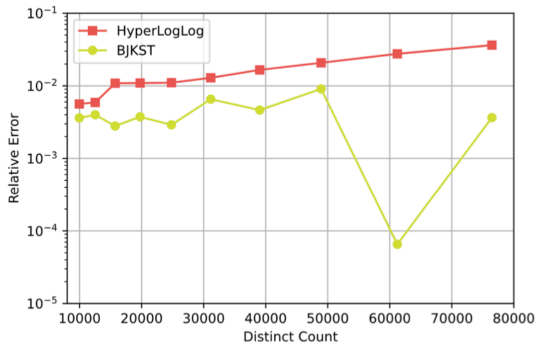


6. **A plot of average relative error vs exact distinct count comparing the BJKST and HyperLogLog algorithms with a choice of parameters to enable a fair comparison. [2 marks]**

As we could see, BJKST works when the number of distinct elements is larger than t, and HyperLogLog works well when it has small range correction. And from Q4 we could know HyperLogLog has small range correction when m equals to 16 and the distinct count is less than $10^5$. Considering constraints above, I decided to set $\epsilon$ of BJKST to 0.1 (so that t equals to 9600) and set m of HyperLogLog to 16, so that they could have a precise comparison where the exact distinct count $\in [10^4, 10^5]$. Furthermore, after having a rough comparison, where exact distinct count is $\{10,100,1000,9955,95316,644380\}$, I found that the relative error of BJKST is less than 1% when the distinct count is larger than $10^4$ (larger than t as well), while the relative error of HyperLogLog is less than 1% when the distinct count is less than $10^5$. Therefore, I decided to compare these two algorithms where the number of distinct elements $\in [10^4,$
$8 \times 10^4]$.

In the second figure, we can see BJKST shows maller relative error than HyperLogLog does in range $[10^4, 8 \times 10^4]$.





7. **A plot of average memory consumption (kB) as a function of actual distinct count (up to $10^8$) comparing the three algorithms. (If unable to compute memory usage for your program you may do a theoretical analysis for half marks.) Use a legend to combine plots having multiple input parameters. Point out where the algorithms differ and possible reasons for this. [2 marks]**

For this question, I'd like to have two comparisons. One is amongst three algorithms with universe {0,1,...,$2^{20}$} and the exact distinct count $\in \{10, 10^1, 10^2, \ldots, 10^6\}$. The other is between baseline and HyperLogLog with universe {0,1,…, $2^{32}$} and the exact distinct count $\in \{10, 10^1, 10^2, \ldots, 10^8\}$. Reason is shown below.

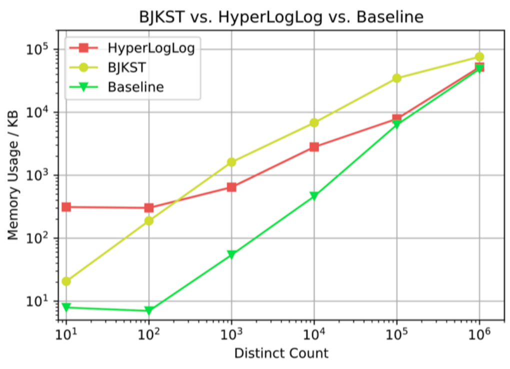
As the chosen prime in hash functions for BJKST is $2^{61} - 1$, and $2n^3$, where n is the size of universe, should be less than $2^{61} - 1$ so that the hash function could have the property of 2-Universial hash family. So I choose $2^{20}$ as the size of universe of BJKST. However, $2^{20}$ is close to $10^6$ and it is still kind of far from $10^8$, so that I decide to have two comparisons.

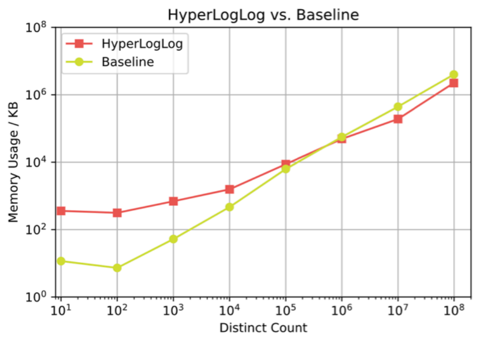
As both of two figures shown, the memory usage of baseline is less than the memory usage of HyperLogLog and BJKST when distinct count is less than $10^6$. Then baseline used more memory than HyperLogLog when the distinct count keeps growing.

There is a linear relationship between the memory usage of baseline and the distinct count, because baseline algorithm is implemented with `HashMap` , whose space complexity is O(N).

As for HyperLogLog , when the distinct count is small, integer arrays `M[]` makes the most contribution to the memory usage. So HyperLogLog used more memory than the other two algorithms when distinct count is less then 100. As the distinct number grows, String arrays used in hash function could also consume more memory .

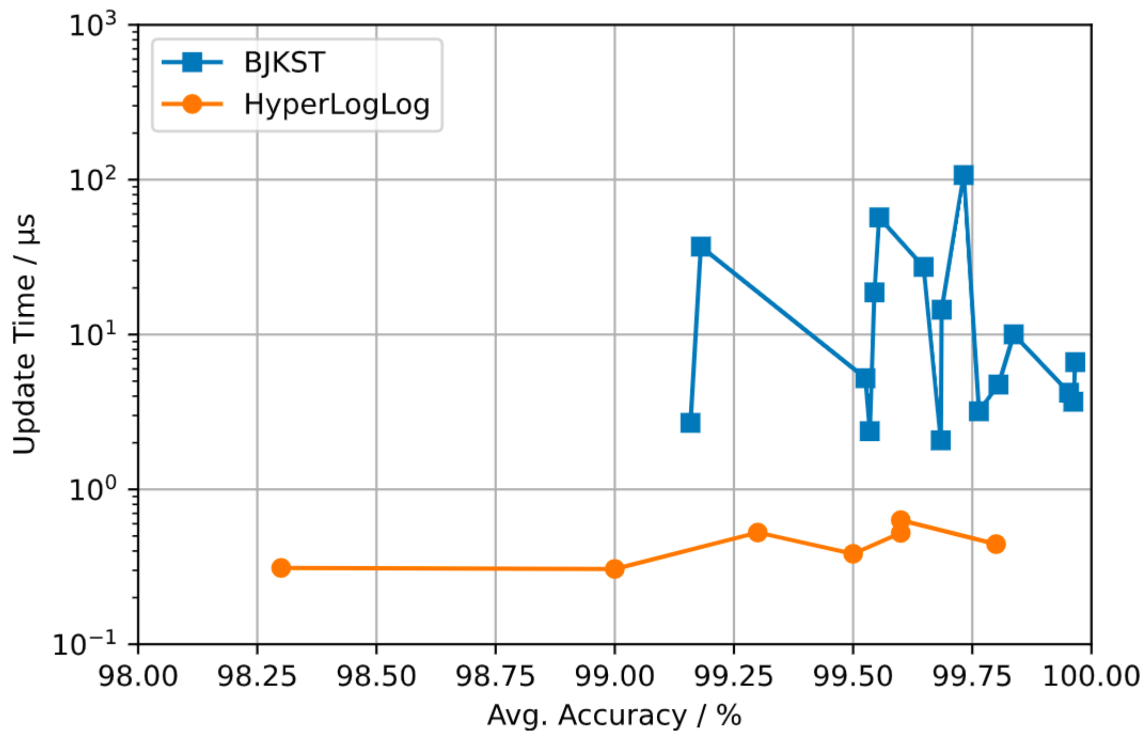
In terms of BJKST, as the distinct count grows, the size of max heap grows. The hash function of BJKST is also an important factor causing the increase in memory usage, becasue I use `BigInteger` in the implementation of hash function and the `BigInteger` store big number as `int[]` .





8. **A plot of average accuracy vs runtime comparing the two algorithms. Use a legend to incorporate multiple input parameters. Point out where the algorithms differ and possible reasons for this. [1 mark]**

   In BJKST, when a new element of the stream comes in, it needs to compare with the maximum value in the max heap, find out if the element is already in max heap, if not, delete the maximum value and insert the new element into the max heap. The time complexity is $O(1)$, $O(N)$, $O(logN)$, $O(logN)$, respectively. While in HyperLogLog, it only needs to update an integer in an array with time complexity $O(1)$. Therefore, the update time of HyperLogLog is far more less than the update time of BJKST. But as the figure shown, BJKST can achieve high accuracy easily than HyperLogLog. Here, $Accuracy = 1 - Relative\ Error$. So, this is a trade-off.



9. *Practical considerations*—**point out any differences between practice and theory and possible explanations for the difference. [1 mark]**

   In theory, the space usage of BJKST is $O(1/\epsilon^2 \cdot logm)$ and the space usage of HyperLogLog is $m$ bytes, that is, in Q7, the memory usage of BJKST should be stable when the distinct number exceeds 9600 and the memory usage of HyperLogLog should be around $2^{16} \times 4$ bytes (because I use integer array). However, in practical situation, the space usage of these two algorithms is linear to the distinct count. I think it is because:

   1. We calculate the memory usage of whole estimate of distinct count and it will inevitably include the memory usage of the part of stream in JVM heap.
   2. The size of an object in Java is the sum of the size of instance header, instance data and padding. The theorectical model only calculates the size of instance data.

10. *Conclusion and recommendation*–**which would be your recommended algorithm for practical use? Indicate how your answer depends on the application (e.g., speed, accuracy or memory requirements). [2 marks]**

    Based on the way I implemented these 3 algorithms and the results I got, I think baseline algorithm,based onhash map, is the best to use in practical use. It is becasue:

    1. Easy to implement. You could implement it with one line code.
    2. Compared with HyperLogLog, the baseline algorithm seems to use a little bit more memory when the distinct count is larger than $10^6$; however, HyperLogLog will have large range correction in this situation and the relative error is high, while baseline algorithm could give us exact distinct count.
    3. Compared with BJKST, the baseline algorithm use less memory and estimates faster. Also, the baseline algorithm is more flexible than BJKST on choosing the range of universe and the number of distinct elements.
    4. In theory, BJKST and HyperLogLog use less memory and runtime than hash map. However, in my practical implemetation and test, hash map is better than those two algorithms. I think in practical use, we should carefully investigate the perfomance of algorithms implemented in specific language, and take properties of specific language (like the concrete implementations of abstract data structure in a specific language, garbage collection in Java, etc) into account.