# CSE222 HW6

## System Requirements

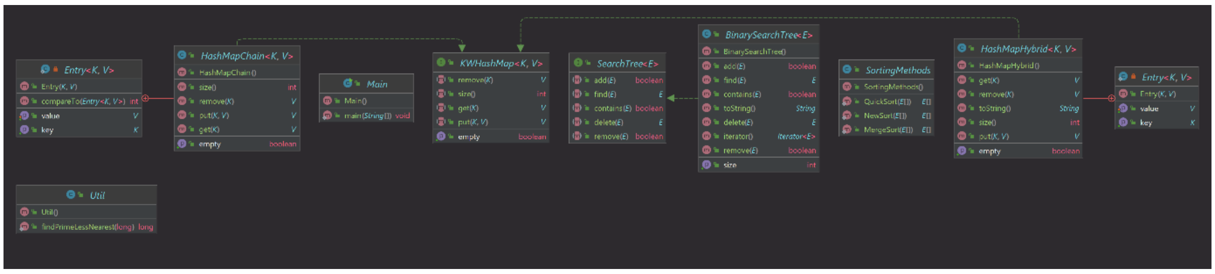
### Chained Hashmap

The hashmap should implement the chaining method using binary search trees. So that the addition, deletion, and search methods are faster compared to usual linked lists.

### Hybrid Hashmap

The hashmap should implement both coalesced hashing and double hashing. So that it can search and remove elements faster with coalesced hashing method and decrease the collision rate with double hashing.

## Class Diagram



## Q1

2.

a. In coalesced hashing you still implement open addressing but also put a next indicator to show where the new key is with same hash is put. This is does not necessarily cut down time taken to put entries but surely shortens the time taken for removing and getting operations. It is still inferior to chaining in means of adding entries.

b. In double hashing you use a formula that basically uses two hashing modulo numbers. This method breaks the linearity and makes hashing more randomly, so the probing is more randomized, and the collision rate is reduced. In means of getting and/or removing entries it may be inferior to chaining since different key values may lead to same hash indexes resulting in greater number of comparisons.

## Q2

1. Merge Sort

Let $T(n)$ be the function that gives the time complexity of merge sort for an array with n elements. We know that left and right subarrays are created in each recursive call and therefore there is n many elements copy operation in total, which is $\theta(n)$. Then for both of left and right subarrays the function is called again recursively, in total sorting of both arrays will take $2T\left(\frac{n}{2}\right)$ time. Then sorted left and right subarrays are compared and copied back to main array. Number of comparisons is at least $\frac{n}{2}$ since both arrays at least this long, for the copying of the arrays it's n. So, it is clear that $T(n) = 2T\left(\frac{n}{2}\right) + n$, and for an array of one element there's no need for comparisons, copy operations, or recursive calls, so $T(1) = 1$.

$$T(n) = 2T\left(\frac{n}{2}\right) + n$$

$$
\begin{aligned}
&= 2[2T\left(\frac{n}{4}\right) + \frac{n}{2}] + n \\
&= 4T\left(\frac{n}{4}\right) + 2n \\
&= 2^k T\left(\frac{n}{2^k}\right) + kn, k \geq 1 \\
\text{let } k &= log_2 n \\
T(n) &= nT(1) + n\, log_2 n \\
&= n + n\, log_2 n = \theta(n\, log\, n)
\end{aligned}
$$

2. Quick Sort

Let $T(n)$ be the function that gives the time complexity of quick sort for an array with n elements. In each recursive call the function is called recursively again for the part that is prior to the pivot and after it. We can't really tell how many elements there are before and after the pivot so at the worst case one of the sub arrays is always empty making it $T(1)$ and the other $T(n-1)$, for the best case they are both $T\left(\frac{n}{2}\right)$. And there's also partition function which is only consists of comparisons and swaps. There's a comparison for every swap so the number of comparisons over number swaps. So, number of comparisons determine the time complexity and there is exactly n many comparisons since every element is checked whether it's greater or less than pivot value. So, the worst time complexity of quick sort is

$$
\begin{aligned}
T_w(n) &= T_w(n-1) + n \\
&= (T_w(n-2) + n - 1) + n \\
&= T_w(n-k) + n + n - 1 + n - 2 + \cdots + n - k \\
\text{let } k &= n - 1 \\
T_w(n) &= T_w(1) + n - 1 + n - 2 + \cdots + 2 + 1 \\
&= 1 + \frac{n(n+1)}{2} = \theta(n^2)
\end{aligned}
$$

Where $T(1) = 1$ since an array with one element is sorted itself. And the best time complexity of it is

$$
\begin{aligned}
T_b(n) &= 2T_b\left(\frac{n}{2}\right) + n \\
&= \theta(n\, log\, n)
\end{aligned}
$$

Since we did the same math for merge sort too.

3. New Sort

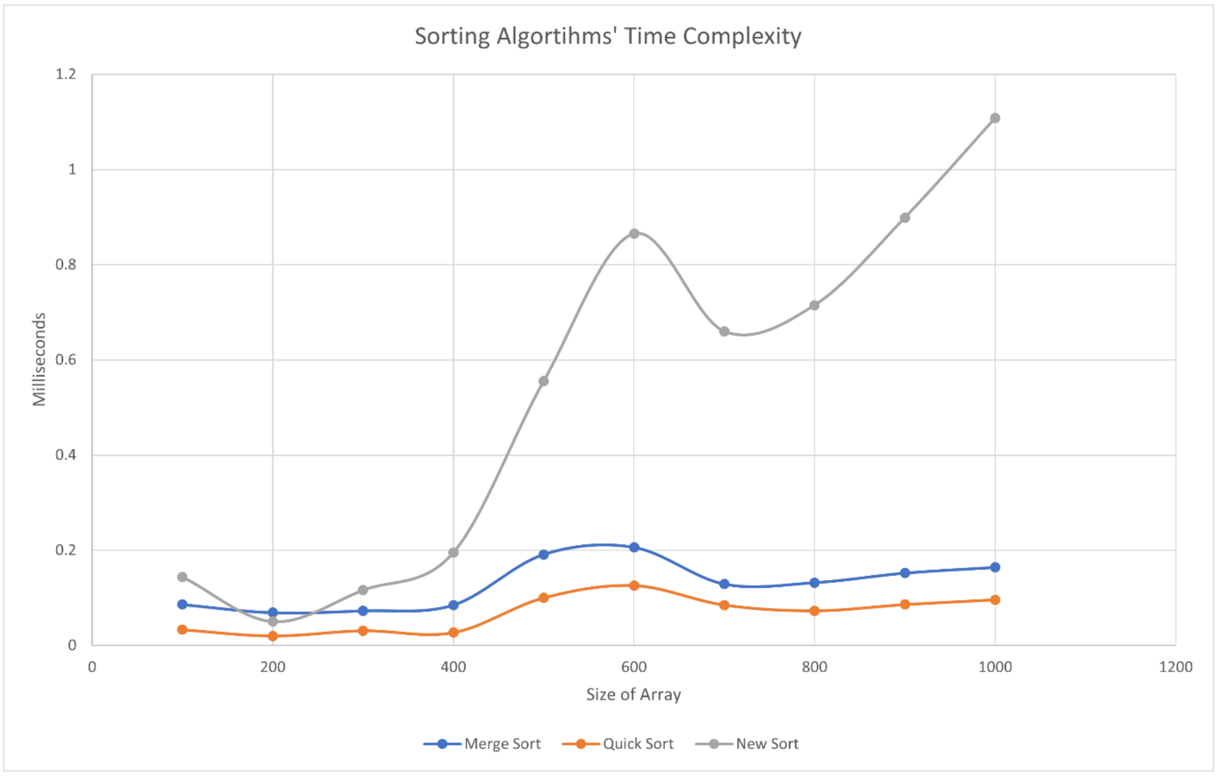
New sort is basically insertion sort but done on both ends. It's almost quadruple as fast but still quadratic.

$$
\begin{aligned}
T(n) &= T(n-2) + n \\
&= T(n-4) + n + n - 2 \\
&= T(n-2k) + 2\left(\frac{n}{2} + \frac{n}{2} - 1 + \frac{n}{2} - 2 + \cdots + \frac{n}{2} - (k-1)\right) \\
\text{let } k &= \frac{n}{2} \\
T(n) &= 2\left(\frac{n}{2} + \frac{n}{2} - 1 + \cdots + 1\right)
\end{aligned}
$$

$$= 2\left(\frac{\frac{n}{2}\left(\frac{n}{2}+1\right)}{2}\right)$$

$$= \frac{n^2}{4} + \frac{n}{2} = \theta(n^2)$$

Since we know $T(0) = \theta(1)$



I have tested and measured the sorting methods, and even though the results are bumpy, and I don’t really know why, it is clear that New Sort is quadratic, and Merge and Quick sort are $\theta(n\log n)$ on average but Quick sort’s coefficient seems to be smaller. The results were gotten by a thousand tries for each size of array.