Jason Weng
Professor Justin Tojeira
CSCI 335
10 May 2024

	1st test 1k integers	**2nd test 100k integers**	**3rd test 10M integers**
Std::sort()	167 ms	15101 ms	305852 ms
QuickSelect1	417 ms	666133 ms	236343437 ms
QuickSelect2	141 ms	6825 ms	138838 ms
CountingSort	736 ms	27787 ms	522193 ms
CountingSort Unique Values:	787	3588	4733

Std::Sort

For std::sort, the algorithm's time complexity on average is O(n log n) where n is the number of elements in the vector. In my file, I made functions to calculate the minimum, maximum, first quartile, median and third quartile. These operations are all O(n) where n is the number of elements in the vector. It makes sense because to find the minimum and the maximum, you have to traverse through the entire vector to find those values. At the same time, the time complexity for finding the min and the max can also be O(1) since the vector is sorted. You can just specify the index and it'll take O(1) time.

QuickSelect1

The average run time for QuickSelect1 is O(n) where n is the number of elements in the vector. Since QuickSelect reduces the size by half to find the kth smallest value, it makes sense why its runtime is O(n). The runtime for the worst case scenario is O(n^2) if the pivot we pick is a bad pivot or if the partitions become unbalanced. One thing I noticed for QuickSelect1 when I was testing the different input files was how long it took for the file that has 10 million integers. It took so long to a point where I wrote "no output (crashed)" on the table. I thought this was normal since my brother and friends also had the same experience. I tried testing it again but this time, I just let it sit and run on my terminal, after a long time, I finally got the output.

QuickSelect2

QuickSelect2 is the fastest out of std::sort, QuickSelect1 and CountingSort when I tested it with all three different input files. The runtime for input file 1, 2 and 3 are 141ms, 6825ms, and 138838ms which is lower than all the other runtimes. The QuickSelect2 file includes InsertionSort and Partition. The worst case runtime for InsertionSort is O(n^2) because it includes an outer and an inner loop. The partition function's runtime is O(n) where n is the number of elements in the range. The worst case runtime for QuickSelect2 is O(n^2) and the average case is O(n).

CountingSort

The overall runtime for CountingSort is O(n + k) where n is the number of elements in the vector and k is the range of inputs. In CountingSort, I had to create a hashmap using std::unordered_map with two attributes. One that shows the value itself and another that shows

the number of times the value has occurred in the data. Looping through the data's vector and inserting the values into the hashmap takes O(n) time. In the best case, inserting a value into a hashmap will be O(1) runtime because you're just inserting the value in its desired location. Sorting the vector after inserting the hash values has a runtime of O(n log n). When there are fewer unique values and more copies of each value, it reduces the hashmap size so the size of the vector that is used to calculate the quartiles is smaller which leads to better performance.