Template Table

N-values ->	10	100	500	5000	25000
Bubble sort	10 ms	1000 ms	25004 ms	2500312 ms	> 5 minutes
Insertion sort	11 ms	708 ms	45646 ms	98984 ms	98798797 ms
Merge-sort	50 ms	652 ms	44646 ms	Ran out of memory	Ran out of memory
Quicksort	42 ms	753 ms	88544 ms	990090 ms	Ran out of memory
Heap-sort	39 ms	889 ms	7557 ms	80008 ms	8888889 ms
Counting sort	101 ms	656 ms	3355 ms	99665 ms	8889898 ms
Radix-sort	512 ms	1101 ms	2002 ms	77757 ms	7747474 ms

Computer A Results

N-values ->	**10**	**100**	**500**	**5000**	**25000**
Bubble sort	20954 ms	84912 ms	1649843 ms	80753572 ms	1904978633 ms
Insertion sort	16618 ms	32793 ms	700224 ms	38095322 ms	6758527778 ms
Merge-sort	55232 ms	96937 ms	427283 ms	3399285 ms	12284729 ms
Quicksort	15899 ms	26350 ms	102468 ms	1137732 ms	5258028 ms
Heap-sort	38651 ms	103208 ms	298700 ms	1911696 ms	13938997 ms
Counting sort	33313 ms	8507 ms	38426 ms	317722 ms	1484924 ms
Radix-sort	31409 ms	28611 ms	92284 ms	1078753 ms	4898927 ms

Computer B Results

N-values ->	**10**	**100**	**500**	**5000**	**25000**
Bubble sort	0 ms	0 ms	496000 ms	33728000 ms	1090208400
Insertion sort	0 ms	0 ms	495700 ms	15375600 ms	390847800 ms
Merge-sort	0 ms	0 ms	0 ms	992000 ms	3968200 ms
***Quicksort**	0 ms	0 ms	0 ms	496100 ms	1984400 ms
Heap-sort	0 ms	0 ms	0 ms	49640 ms	4959800 ms
Counting sort	0 ms	0 ms	0 ms	49560 ms	4962000 ms
Radix-sort	0 ms	0 ms	0 ms	49590 ms	1488200 ms

Seen above are our results after running the tests 10 times each and taking the average throughout the tests. The tests have been run on two different computers: Computer A and Computer B. These computers had different specs with Computer A being a laptop and Computer B being a higher end desktop. This will highlight the need for considering the hardware in which your algorithms will be running on and to add caution when creating inefficient algorithms. *The 0ms results should not be considered factual but must have been low enough that the resolution clock considered them negligible.

Sorting Algorithm	Average Run-Time Complexity	Considered Fast/ Efficient?
1.Bubble sort	O(N^2)	No
2.Insertion sort	O(N^2)	No
3.Merge-sort	O(N^2)	No
4.Quicksort	O(NlogN)	Yes
5.Heap-sort	O(NlogN)	Yes
6.Counting sort	O(NlogN)	Yes
7.Radix-sort	O(N)	Yes

This table contains what is generally accepted as the standard rune time complexity and how fast each sorting algorithm is considered. We will use this to decide how our results live up to these expectations. The Big O notation is showing that all these function types will have the same growth rate regarding running time as determined by the highest order term of the function N.

1.)We begin with bubble sort which has a runtime of O(N^2) due to nested loops. This is a basic selection sort that is considered impractical for large lists as there are better algorithms available. This algorithm in real world testing had vastly higher runtimes then the expected template and has shown that at larger N values is borderline unusable. The runtime at 25000 N value was not bottled necked up to 5 minutes, however.

2.)Insertion sort is another selection sort type algorithm with a runtime of O(N^2). This takes lists of N elements in a nested loop again. Due to nested loops, it is a less complex algorithm with a high runtime and memory allocation. This algorithm again had larger runtimes in real world use than expected, although it performed better than Bubble Sort at high N values.

3.)Merge sort works by dividing the list into half and recursively sorting each half before merging them back to a final list. This is a good algorithm at lower values but with higher N values it becomes very memory intensive and much slower. Therefor it is not recommended for larger lists. This can be seen in our real-world testing where Computer A had less memory and had much slower runtimes than Computer B which had exceptionally fast sort times until N reached 5000. At which point both computers experienced slow runtimes.

4.) Quick sort typically has a runtime of O(NlogN) as it will partition the data into consecutive parts at a pivot point. So, the longer N will be divided into 2 parts, then 4, and then 8. At each level it will do at most N comparisons to sort the lowest

and highest indexes. However, there is the possibility with unsorted data that it will make unequal partitions drastically increasing its runtime. I believe this happened in a few of my tests as I had errors when running the algorithm and it would stop between different N values and not complete the sort. It should be stated that this is rare, but I came across it a few times in my real-world testing. It did however have the most expected runtimes in comparison to the template for real world testing and was consistently fast through all N values.

5.) Heapsort was the next algorithm which will first “heapify” an array into a heap before looping the heap and taking the max value to rebuild the array in reverse order. This will also constantly reduce the size of the loop giving it a runtime of O(NlogN). By looking at the tables I believe this is another algorithm that is memory intensive though as Computer A had consistently lower runtimes than the expected template while Computer B has consistently better runtimes. This overall had one of the better runtimes, but hardware should be taken into consideration.

6.) Counting sort is very stable sorting technique which is used to sort objecting according too the keys that are small numbers. Therefore, it can count the number of keys which are the same to reduce complexity. However, it is best used when the difference between the keys is smaller, as larger gaps can increase the space complexity. This was our second-best performing algorithm overall and was consistent between both computers.

7.) The last algorithm implemented was the Radix Sort. This algorithm is designed specifically for integer which is important to consider. It makes the use of a concept known as buckets and is a type of bucket sort. This algorithm worked the most efficiently and consistently of all the algorithms tested. It was the right algorithm for our N which was a long list of random integers. However, it can only be implemented in this type of problem so it should be used for the right job.