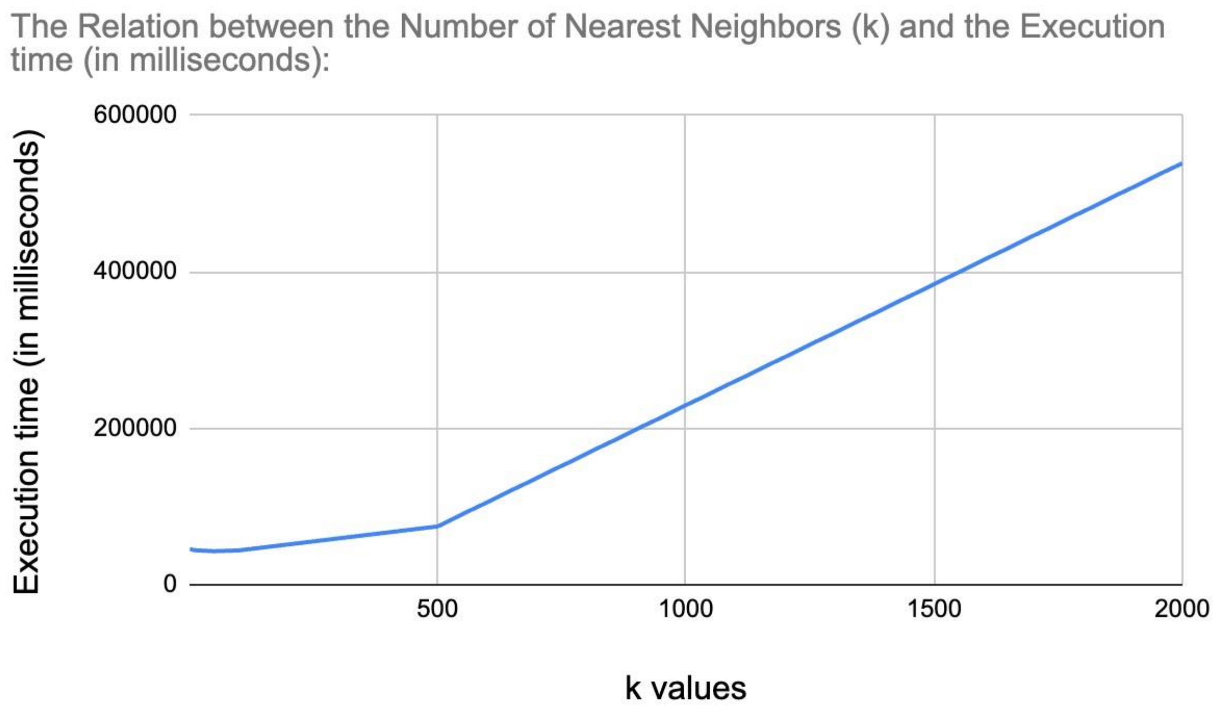
Project: k-Nearest Neighbor Algorithms
Michael Massaad

1. Finding k Nearest Neighbors Using PriorityQueue1 Implementation:

Table 1: Results of Experiment Using Dataset of 1,000,000 Points for Finding k Nearest Neighbors of 100 Query Points

k values	**Execution time (in milliseconds)**
1	45770.0
10	44103.0
50	42826.0
100	43838.0
500	74466.0
2000	538996.0

Graph 1: The Relation between the Number of Nearest Neighbors (k) and the Execution time (in milliseconds):

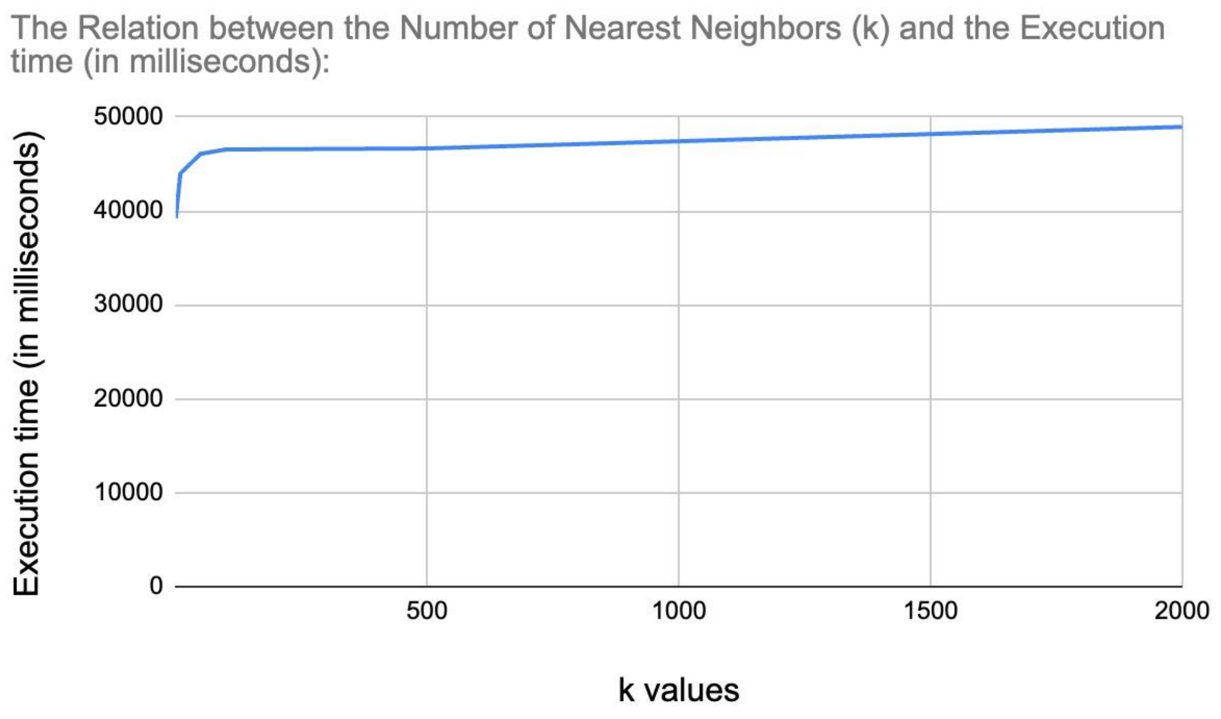


2. Finding k Nearest Neighbors Using PriorityQueue2 Implementation:

Table 2: Results of Experiment Using Dataset of 1,000,000 Points for Finding k Nearest Neighbors of 100 Query Points

k values	**Execution time (in milliseconds)**
1	39207.0
10	43979.0
50	46072.0
100	46541.0
500	46648.0
2000	48945.0

Graph 2: The Relation between the Number of Nearest Neighbors (k) and the Execution time (in milliseconds):

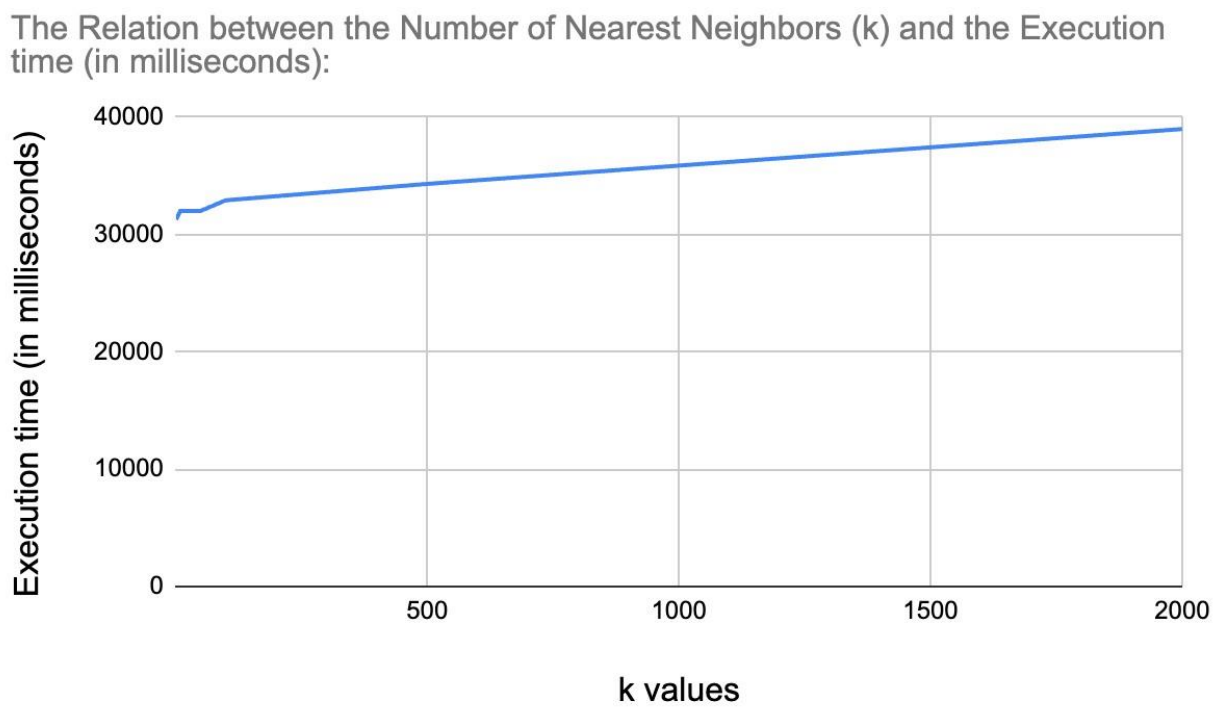


3. Finding k Nearest Neighbors Using PriorityQueue3 Implementation:

Table 3: Results of Experiment Using Dataset of 1,000,000 Points for Finding k Nearest Neighbors of 100 Query Points

k values	**Execution time (in milliseconds)**
1	31274.0
10	32011.0
50	32011.0
100	32914.0
500	34310.0
2000	38986.0

Graph 3: The Relation between the Number of Nearest Neighbors (k) and the Execution time (in milliseconds):



Comments on Results from Experiments:

From the results of the experiments, we can observe that the implementation of the PriorityQueue1 (PQ1) takes a longer time to execute the KNN algorithm for 100 query points if we are to compare it with the execution time of the implementations from PriorityQueue2 (PQ2) and PriorityQueue3 (PQ3).

This makes sense since in the implementation of PQ1, which uses an ArrayList, whenever we want to insert a new point into the priority queue (using offer()), we have to do a comparison of the distance from the current query with each of the points that are already in the priority queue. Therefore, if we are increasing the number of elements that are desired in the priority queue (increasing the value of k), then the amount of time to do the comparisons for the insertion with also increase. In observing Table 1 and Graph 1, we can see the increase in the execution time when we increase the value of k.

In addition, both PQ2, which uses an ArrayList, and PQ3, which uses the java.util.PriorityQueue<E>, both are based on a max heap (for PQ3, I consulted the Javadoc for a priority queue, https://docs.oracle.com/javase/8/docs/api/java/util/PriorityQueue.html). When we are doing an insertion of a new point into the priority queue, both implementations use the binary search algorithm to do the comparisons with the current elements in the queue. Therefore, they would both have a quicker execution time than the PQ1 implementation. In observing Tables 2 and 3, along with graphs 2 and 3, we can observe that the execution time still increases when the value of k increase, but by a small amount of time.