Programming Assignment 1 - Rishab Dudhia (SID: 862141444)

a. First I read in the inputs and put the values of the scores into a vector. Then, I wrote the mergesort() algorithm to sort the vector of scores in $O(n \log n)$ time. Once the vector was sorted, I used a for loop to complete all queries. Once the rank was input, locating the grade and outputting the score was done in $O(1)$ time since I was just accessing the element of the rank which was one less than the rank.

b. I used the mergesort() algorithm from part a to sort the values read into the vector. Once the vector was sorted, I read in a grade to find the rank of and if it was the highest grade it would output rank 1 in $O(1)$ time. If the grade was not the highest, I used a for loop to iterate through the sorted vector to find the desired grade and I incremented the rank value for each iteration of the for loop giving a result in $O(n)$ time.

c. Modifying the code from part b slightly, in the final for loop, if the grade read in the current iteration was the same as the grade that came before it in the vector, the count value was not incremented. The count value acts the same as the rank value in part b, but is used to be compared against the actual rank value query.

d. Originally with submission id: 130319365, I wrote a basic bubble sort algorithm, but it ran in $O(n^2)$ time. To attempt to write an algorithm that runs in $O(n \log n)$ I wrote a variation of the mergesort() algorithm in the regards of recursively breaking down the vector to one element then merging the sub-vectors to determine if there needs to be adjacent swap(s). To merge the sub-vectors after the original vector was recursively split in half, I had pointers to the beginning of both sub-vectors. If the value pointed to by the left sub-vector pointer was less than or equal to the value pointed to by the right sub-vector pointer, no adjacent swap is necessary. The value pointed to by the left

sub-vector pointer is pushed onto the holder vector which is the vector of the correct order after adjacent swaps and then the left sub-vector pointer is incremented. However, if the value pointed to by the left sub-vector pointer is greater than the value pointed to by the right sub-vector pointer, one or more adjacent swaps are necessary. The way I determined the number of adjacent swaps necessary was based on the remaining length of the left sub-vector. Since the larger value on the left needs to propagate all the way to the back of its sub-vector, the count of adjacent swaps necessary is the remaining length of the left sub-vector. We can rely on the recursion to sort the sub-vectors so that it is ensured that in the merge function only sorted sub-vectors are being merged so that if a value needs to propagate backwards it will propagate all the way to the back of the sub-vector. Once the number of adjacent swaps is added to the count, the value pointed to by the right sub-vector pointer is pushed onto the holder vector and then the right sub-vector pointer is incremented and the process continues until one sub-vector has been sorted. The range (l to r) in the current merge() function call has the correct order placed into it as the holder vector is copied into the cars vector in the respective range.