**CSE 373 QuickCheck 7** Name: Evil Kevin Student ID: 1234567890

Suppose you work on a droid assembly line. You have a supposedly sorted array of $N$ `Droid` objects that implement `Comparable`. However, when looking through your array, you realize these aren't the droids you're looking for! The machine malfunctioned and made at most $k$ mistakes: there are no more than $k$ inversions, where we define an inversion as a pair of droids that is not in the right order.

*Hint*: The array [0 1 1 2 3 4 8 6 9 5 7] has 6 inversions: 8–6, 8–5, 8–7, 6–5, 9–5, 9–7.

For the questions below, give the typical or expected runtime. For example, for quicksort, assume that the pivot choices result in $O(\log N)$ recursive depth.

1. For each $k$, give the most efficient sorting algorithm and its simplified asymptotic runtime.

   (a) $k \in O(N)$ Algorithm: Insertion sort Runtime: $\Theta(N)$

   Insertion sort has a worst-case runtime of $\Theta(N^2)$ on general input, but remember that it works quite well if there are relatively few inversions: $\Theta(N + k)$.

   Java merge sort (adaptive merge sort with natural runs) might also work if the $k \in O(N)$ inversions are, for example, due to the smallest item moving to the end of the array. In this scenario, there are two natural runs, one with the first N - 1 items and the other with the last 1 item. But we can imagine a worst case where each adjacent pair of items in the array are swapped: in this case, there are about $k \approx N/2$ inversions but N natural runs, leading to $\Theta(N \log N)$ runtime.

   (b) $k \in O(N^2)$ Algorithm: Merge sort, quicksort, or heapsort Runtime: $\Theta(N \log N)$

   Any expected $\Theta(N \log N)$ runtime comparison sorting algorithm would work here: merge sort, quicksort, or heapsort. Insertion sort isn't the best sort for this problem since the runtime depends on the number of inversions, or $O(N + N^2) = O(N^2)$.

   Does stability matter? It's not well-defined in this problem. Since the machine mixes up the order of the droids, we're not sure if the relative ordering of equal items is any good.

   (c) $k \in O(\log N)$ Algorithm: Insertion sort Runtime: $\Theta(N)$

   Insertion sort has a worst-case runtime of $\Theta(N^2)$ on general input, but remember that it works quite well if there are relatively few inversions: $\Theta(N + k)$.

   Java merge sort (adaptive merge sort with natural runs) might also work, though the complete analysis is out of scope.

2. Two weeks later, you're given another batch of droids that are supposed to be sorted on a 32-bit `int ID`, an instance variable of `Droid`. The machine hasn't been fixed and again made at most $k$ mistakes. For each $k$, give the most efficient sorting algorithm and its simplified asymptotic runtime.

   (a) $k \in O(N^2)$ Algorithm: Radix sort Runtime: $\Theta(N)$

   Sorting on integer ID allows us to use counting sorts. Radix sort is the most realistic choice to make sure our count array is a reasonable size. That said, in theory, counting sort could work since we could initialize (one or more) count arrays to cover all 4 billion or so possible integers. The runtime for both radix sort and counting sort on integers is $\Theta(N)$ since there's a limit on the size of Java integers, so there are constant number of digits to compare.

(b) $k \leq 5$ Algorithm: Insertion sort or radix sort Runtime: $\Theta(N)$

Insertion sort works well because the array is almost sorted, so the runtime is in $\Theta(N + 5) = \Theta(N)$.

Counting sort and radix sorts work well for the same reason as part (a).

Adaptive merge sort is also appropriate here. Imagine an array with exactly 5 adjacent pairs (anywhere in the array) swapped out of order. There are thus at most 6 natural runs in the array.