CSE 373 QuickCheck 8 Name: Evil Kevin Student ID: 1234567890

For answers that involve filling-in a ◯, fill-in the shape completely: ●.

1. Suppose we create a new sorting algorithm called PartitionHybridSort(input, lo, hi, subsort), where input is an array of integers, lo is the lowest index in the array to be sorted, hi is the highest index in the array to be sorted, and subsort is the sorting algorithm to use in step 3:

   1. If hi − lo ≤ 1, return.
   2. Partition around input[lo], i.e. the leftmost item of the current subproblem.
   3. Use subsort to sort the left and right subproblems.

   Give the **worst-case runtime** for PartitionHybridSort using different subsort algorithms in terms of N, the size of the input.

   PartitionHybridSort uses the partitioning idea from quicksort but changes what happens after partitioning the left and right subproblems. In the worst-case, partitioning around the lo item leaves us with $N - 1$ items in the array. Sorting $N - 1$ items is not asymptotically different from sorting N items.

   (a) PartitionHybridSort(input, 0, N, InsertionSort) Worst-case: $\Theta(N^2)$

   (b) PartitionHybridSort(input, 0, N, MergeSort) Worst-case: $\Theta(N \log N)$

   (c) PartitionHybridSort(input, 0, N, LSDRadixSort) Worst-case: $\Theta(N)$

   (d) PartitionHybridSort(input, 0, N, PartitionHybridSort) Worst-case: $\Theta(N^2)$

   PartitionHybridSort as the subsort results in naïve quicksort.

2. Suppose we replace the counting sort used in LSD radix sort with merge sort, resulting in a new radix-based sorting algorithm called LSD Radix Merge Sort. The merge sort used as a subsort by LSD Radix Merge Sort is exactly like regular merge sort, except that its merge operation compares only one digit of an input to decide which is larger. For example, the merge operation would ordinarily consider 361 to be less than 410, but if we're sorting on the final digit, it will consider 361 to be larger than 430 since $1 > 0$.

   Just like regular LSD radix sort, LSD Radix Merge Sort would sort by the last digit, then second to last digit, and so forth. We can define LSD Radix Quicksort in a similar way. Assume that LSD Radix Quicksort always picks the leftmost pivot and uses in-place Hoare partitioning.

   Give the **worst-case runtime** of both sorting algorithms in terms of N and W, where W is the number of digits in each key. For simplicity, assume all keys have the same number of digits. Don't worry about the alphabet size R. Also state whether or not they always return a correct sort and explain.

   LSD radix sort needs a stable subsorting algorithm. When sorting items on the ten's place, for example, we don't want to mix-up the sorting already completed for the one's place. Since merge sort is stable, LSDRadixMergeSort is correct. Since quicksort is unstable, LSDRadixQuicksort is not correct.

   LSD radix sort involves running W iterations of the subsorting algorithm. Imagine a for-loop across all W digit places, where each loop calls either merge sort or quicksort.

(a) LSDRadixMergeSort Worst-case: $\Theta(WN \log N)$

Correct? ● Yes ○ No

Explanation: Merge sort is stable.

(b) LSDRadixQuicksort Worst-case: $\Theta(WN^2)$

Correct? ● Yes ○ No

Explanation: Quicksort is unstable.