Section 5: Hashing & Sorting

0. Sorting Hat

Suppose we sort an array of numbers, but it turns out every element of the array is the same, e.g.,{17, 17, 17, ..., 17}. (So, in hindsight, the sorting is useless.)

a) What is the asymptotic running time of **insertion** sort in this case?

O(n) - This is the best case runtime of insertion sort as it only requires one pass through the data. Insertion sort will traverse the array but since each element is not less than the one before it, no extra computations are necessary.

b) What is the asymptotic running time of **selection** sort in this case?

O(n²) - Selection sort always has *n²* runtime, regardless of the nature of data

c) What is the asymptotic running time of **merge** sort in this case?

O(n∗log(n)) - Merge sort always has *n∗log(n)* runtime, regardless of the nature of data

d) What is the asymptotic running time of **quick** sort in this case?

O(n²) - This is the worst case runtime of quick sort. When partitioning, every element is going to fall to the same side of the pivot since they all have the same value which essentially only sorts 1 element per iteration of quicksort, leading to the *n²* runtime.

1. Another Sort of Sorting…

Given an array of integers as such: {11, 13, 55, 67, 79, 10, 8, 6, 4, 2}. Please answer the following questions (assume all sorts to be done in ascending order):

a. What is the asymptotic running time of insertion sort for this array?
   This takes the worst case runtime of O(n^2) because the array is not already in sorted order.

b. What is the asymptotic running time of **selection** sort in this case?

O(n²) - Selection sort always has *n²* runtime, regardless of the nature of data

c. What is the asymptotic running time of **merge** sort in this case?

O(n∗log(n)) - Merge sort always has *n∗log(n)* runtime, regardless of the nature of data

d. What is the asymptotic running time of **quick** sort in this case (assuming that we choose the leftmost element as the pivot each time)?

$O(n^2)$

Case 1: upon choosing a good pivot (one that partitions the array into roughly equal halves), one of the subarrays would be completely in reverse sorted order, so it's O(n) to sort one of the subarrays at each level, resulting in a $O(n^2)$ asymptotic running time.

Case 2: choosing a bad pivot (the smallest or largest element) each time would result in the worst case runtime of $O(n^2)$ as well.

2. Radix sort

Use Radix Sort to sort the following list of integers in ascending order:
[170, 45, 75, 90, 802, 24, 2, 66]

(a) Bucket Sort on 1's Digit

0	1	2	3	4	5	6	7	8	9
170, 90		802, 2		24	45, 75	66			

(b) Bucket Sort on 10's Digit

0	1	2	3	4	5	6	7	8	9
802, 2		24		45		66	170, 75		90

(c) Bucket Sort on 100's Digit

0	1	2	3	4	5	6	7	8	9
2,24, 45,66, 75,90	170							802	