Problem 1. In-Order Input
As in life, computers often hate doing unnecessary work. A great example of this is with sorting
data that is already in-order (1, 2, 3, 4, 5, ...). For each of the following sorting algorithms,
describe how in-order input affects their performance. You should give both a big-O bound and a
brief description of what happens.

(a) Selection Sort
(b) Insertion Sort
(c) Heap Sort
(d) Merge Sort
(e) Quick Sort using the value at arr[lo] for the pivot
(f) Quick Sort using the median-of-3 rule to select the pivot

Problem 2. Duplicate Data
Another potential sticky widget with sorting algorithms is when the data has duplicate entries. The
worst is when the entire array is the same (13, 13, 13, 13, 13, ...). For each of the following sorting
algorithms, describe how they perform when the array is homogeneous. You should give both a
big-O bound and a brief description of what happens.

(a) Selection Sort
(b) Insertion Sort
(c) Heap Sort
(d) Merge Sort
(e) Quick Sort using the value at arr[lo] for the pivot
(f) Quick Sort using the median-of-3 rule to select the pivot

Problem 3. Diabolical Inputs
In algorithm analysis, one approach for determining worst-case behavior is to identify what are
called diabolical inputs that cause an algorithm to perform at its worst. For insertion sort and
selection sort, reverse-order input is diabolical. In this question, you will look at diabolical inputs
for quick sort.

(a) Assume that quick sort always selects the middle element (\(\lfloor \frac{lo+hi}{2} \rfloor\)) for the pivot. Give a
diabolical input (one that gives \(O(n^2)\) performance) for quick sort using the numbers 1–10.

(b) Assume that quick sort uses the median-of-3 rule for selecting the pivot. For simplicity, also
assume that the sort uses a CUTOFF of 3. Give a diabolical sequence using the numbers
1–20 that gives \(O(n^2)\) performance.