Section 08: Sorting Out Sorting

Section Problems

1. Relaxation Zone

A timeless classic: https://www.youtube.com/watch?v=YvTW7341kpA

2. Mechanical 1: Sorting Algorithm Steps

Here is your input array: `32, 15, 2, 17, 19, 26, 41, 17, 17`.

Show the steps taken on this array for each sort as we have learned in lecture. Sort elements so that the result is ordered from smallest to largest. Also remark on whether each sort is **stable** or not.

(a) Insertion sort:

(b) Selection sort:

(c) **In-place** Heapsort (You may want to draw out the heap. Make sure the first step you do is the `buildHeap` step!):

(d) Merge sort:

(e) Quicksort (assume that we always choose the left-most element as the pivot):

3. Mechanical 2: More Sorting Algorithm Steps

Show the steps taken for each sort as we have learned in lecture. Sort elements so that the result is ordered from smallest to largest.

(a) Insertion sort on `0, 4, 2, 7, 6, 1, 3, 5`.

(b) Selection sort on `0, 4, 2, 7, 6, 1, 3, 5`.

(c) **In-place** Heapsort on `0, 6, 2, 7, 4`. (You may want to draw out the heap. Make sure the first step you do is the `buildHeap` step!)

(d) Merge sort on `0, 4, 2, 7, 6, 1, 3, 5`.

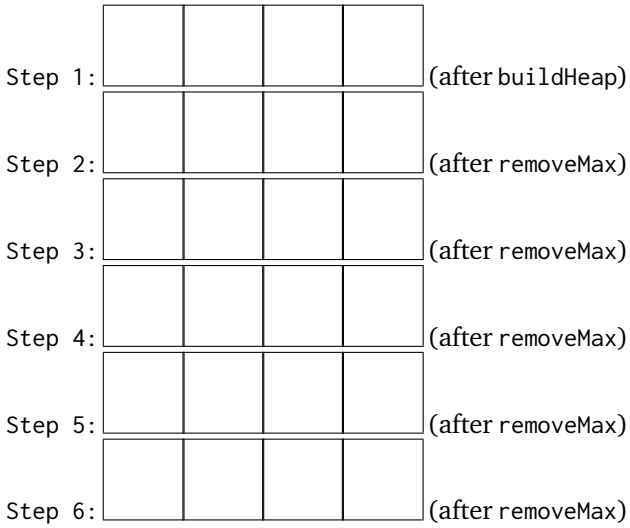
(e) Quicksort on `18, 7, 22, 34, 99, 18, 11, 4`. (Assume that we always choose the first element as the pivot. Show the steps taken at each partitioning step.)

4. Mechanical 3: More Heapsort

List out the steps of sorting the array `[5, 0, 1, 3]` into ascending order using **in-place** heap sort with a **max heap**.

The first step should be the array after the intial `buildHeap`, and each successive step should be the array after `removeMax`. You may not need every line provided to write your solution.

(Tip: Draw out the heap. Make sure your first step is `buildHeap`!)



5. Runtime 1: Insertion Sort Worst-Case Input

Give the worst possible order of input for insertion sort with the following integers: 1, 2, 3, 4, 5, 6, 7. Assume that the result of sorting should be in ascending order.

6. Runtime 2: All Worst-Case Inputs

When choosing an appropriate algorithm, there are often several trade-offs that we need to consider. Inspect the chart for the following sorting algorithms. Then, for each sort, provide an example of **both** a best-case and worst-case input.

	Runtime (best)	Runtime (worst)	Stable? (Y/N)	Notes
A: Selection Sort	$\Theta(N^2)$	$\Theta(N^2)$	No	
B: Insertion Sort	$\Theta(N)$	$\Theta(N^2)$	Yes	
C: Merge Sort	$\Theta(N \log N)$	$\Theta(N \log N)$	Yes	
D: Quicksort	$\Theta(N \log N)$	$\Theta(N^2)$	No	
E: Heapsort	$\Theta(N)$	$\Theta(N \log N)$	No	

7. Runtime 3: Quicksort Pivots

What are two techniques that can be used to reduce the probability of Quicksort taking the worst case running time?

8. The Legacy of Robbie: A Sorting Mystery

Consider the following sorting algorithm in pseudocode. Note that, in this case, the upper bound of each for loop is *inclusive*, so they run up to and including $i = A.\text{length} - 1$ and $j = i - 1$.

```
1: function MysterySort(A)
2:     for i = 1 to A.length − 1 do
3:         for j = 0 to i − 1 do
4:             if A[j] ≥ A[i] then
5:                 x = A[i]
6:                 shift every item from j to i − 1 right by one
7:                 A[j] = x
8:                 break
```

(a) Is MysterySort most similar to insertion sort, merge sort, quick sort, or selection sort?

(b) Is MysterySort a stable sorting algorithm? Why or why not?

(c) What is the best-case runtime (as a tight big-$\mathcal{O}$ bound) for MysterySort? Why is this the best case?

Hint: What happens when MysterySort is given an array that is already sorted?

9. Sorting Design Decisions 1

For each scenario below, fill in all squares with the letter corresponding to the sorting algorithm that would perform the best. Some of the questions may have multiple answers, in which case you should fill in multiple squares.

A: Insertion Sort, B: In-place Heapsort, C: Merge Sort, D: Selection Sort, E: Quicksort, F: None of the Above

Note: You should definitely have the **"cheat sheet"** from the **Sorting Study Guide** open as you work through these problems!

(a) Java integer array with a length of 5. A ☐ B ☐ C ☐ D ☐ E ☐ F ☐

(b) An array of comparable UW classes (objects), that are already sorted by number and need to be additionally sorted by department. For example, the input `[CSE 142, CHEM 142, CSE 143, CSE 373]` would result in the output `[CHEM 142, CSE 142, CSE 143, CSE 373]`.

A ☐ B ☐ C ☐ D ☐ E ☐ F ☐

(c) We need to sort integers in $\Theta(\log N)$ time for the best case. A ☐ B ☐ C ☐ D ☐ E ☐ F ☐

(d) Sorting doubles with a $\Omega(N \log N)$ best-case lower bound. A ☐ B ☐ C ☐ D ☐ E ☐ F ☐

10. Sorting Design Decisions 2

For each of the following scenarios, say which sorting algorithm you think you would use and why. There may be more than one right answer.

Note: You should definitely have the **"cheat sheet"** from the **Sorting Study Guide** open as you work through these problems!

(a) Suppose we have an array where we expect the majority of elements to be sorted "almost in order". What would be a good sorting algorithm to use?

(b) You are writing code to run on the next Mars rover to sort the data gathered each night. (Think about sorting with limited memory and computational power.)

(c) You're writing the backend for the website `SortMyNumbers.com`, which sorts numbers given by users.

(d) Your artist friend says for a piece she wants to make a computer sort every possible ordering of the numbers $1, 2, \ldots, 15$. Your friend says something special will happen after the last ordering is sorted, and you'd like to see that ASAP.

11. A Sorting Puzzle

For each scenario given below, identify which sort is being used out of the following possibilities:

In-place Heapsort, Selection Sort, Insertion Sort, Merge Sort

(a) Suppose we're executing our unknown sorting algorithm on an array of 8 integers. The following steps represent the order of elements at different points in time.

Initial Array:

```
0, 4, 2, 7, 6, 1, 3, 5
```

During Execution:

```
0, 2, 4, 7, 6, 1, 3, 5
```

Sorted Result:

```
0, 1, 2, 3, 4, 5, 6, 7
```

(b) Suppose we're executing our unknown sorting algorithm on an array of 8 integers. The following steps again represent the order of elements at different points in time.

Initial Array:

```
0, 4, 2, 7, 6, 1, 3, 5
```

During Execution:

```
0, 1, 2, 3, 6, 4, 7, 5
```

Sorted Result:

```
0, 1, 2, 3, 4, 5, 6, 7
```

(c) Below you will find **intermediate** steps in performing our unknown sorting algorithm on the given input array. The steps do not necessarily represent consecutive steps in the algorithm (that is, many steps are missing), but they are in the correct sequence.

Initial Array:

```
1429, 3291, 7683, 192, 1337, 594, 4242, 9001, 4392, 129, 1000
```

Intermediate Steps:

```
1429, 3291, 192, 1337, 7683, 594, 4242, 9001, 129, 1000, 4392
```

```
192, 1337, 1429, 3291, 7683, 129, 594, 1000, 4242, 4392, 9001
```