CS3000: Algorithms & Data — Summer 2023 — Laney Strange

Homework 1 - Short
Due Thursday May 11 @ 9pm Gradescope

Name:
Collaborators:

- Put your name on the first page. If you are using the LaTeX template we provided, then you can make sure it appears by filling in the `yourname` command.
- This assignment is due Thursday May 11 @ 9pm Gradescope. You may submit up to 48 hours late for no penalty, but expect a delay in grading.
- Show ALL your work, even if the problem doesn't specify it.
- You will have an opportunity to resubmit one short homework and one long homework for new grades, at the end of the semester.
- Solutions must be typeset, preferably in LaTeX. I recommend using the source file for this assignment to get started. If you need to draw any diagrams, you may draw them by hand as long as they are embedded in the PDF.
- I encourage you to work with your classmates on the homework problems. *If you do collaborate, you must write all solutions by yourself, in your own words.* Do not submit anything you cannot explain. Please list all your collaborators in your solution for each problem by filling in the `yourcollaborators` command.
- If you get stuck on a homework problem, come by office hours or post on Piazza! We recommend you spend about 30 minutes trying to figure out a problem, and then ask for help. We'll be happy to clarify material from class and algorithm concepts, but we will not give out solutions or confirm your answers are correct.
- Finding solutions to homework problems on the web, or by asking students not enrolled in the class, is strictly forbidden.

Problem 1. *Insertion Sort* ($4 + 1 + 1 + 1 = 7$ *points)*

In class, we covered Selection Sort, which repeatedly finds the smallest element and puts it "next" in the output. Now, we'll try its cousin, Insertion Sort.

Like Selection Sort, Insertion Sort always has a subarray that's unsorted and a subarray that's sorted. At the very beginning of the algorithm, the unsorted subarray is everything and the sorted subarray is empty.

Insertion Sort moves left to right through the array. It repeatedly picks the "next" unsorted element and places it in the sorted subarray by figuring out its correct position and shifting the existing, sorted elements accordingly.

Insertion sort maintains the following loop invariant: At the beginning of iteration j of the outer loop, the subarray $A[1...j-1]$ contains the elements originally found in $A[1...j-1]$, but in sorted order.

(a) Complete the pseudocode below for Insertion Sort.

```
INSERTIONSORT(A)
1  for $j = 2$ to $A.length$
2      $item = A[j]$
```

Solution:

(b) Unlike its cousin Selection Sort, Insertion Sort has different best-case and worst-case runtimes. Give an example of an array that would produce a worst-case scenario for Insertion Sort.

Solution:

(c) An algorithm is *in place* if it requires that the only memory allocation be for constant space (i.e., it does not require an auxiliary data structure). Based on your solution above, is Insertion Sort in-place?

Solution:

(d) A sorting algorithm is considered "stable" if it respects the original order of elements with the same value. If there are two 3s, for example, then they appear in the same order before sorting as they do after sorting. Based on your solution above, is Insertion Sort stable?

Solution:

Problem 2. *Bubble Sort* ($1 + 4 + 1 + 4 = 10$ *points)*

Another well-known sort algorithm is Bubblesort. The pseudocode is below.

```
BUBBLESORT(A)
1  for $i = 1$ to $A.length - 1$
2      for $j = A.length$ downto $i + 1$
3          if $A[j] < A[j-1]$
4              swap $A[j], A[j-1]$
```

(a) Suppose your starting array is $< 8, 2, 5, 4, 12, 3 >$. What does this array look like after the inside loop (lines 2-4) completes its first iteration?

Solution:

(b) Prove that the following loop invariant holds, for the inner loop on lines 2-4: At the start of each iteration of the for loop, the subarray $A[j..n]$ consists of the elements originally in $A[j..n]$ before entering the loop, but possibly in a different order, and the first element $A[j]$ is the smallest among them.

Solution:

(c) In the table below, which shows the cost and number of times for each line of the Bubblesort pseudocode, t_i represents the number of times the loop test at line 2 will execute when $i = 1, 2, ..., A.length\text{-}1$.

line	cost	times
1	1	n
2	1	$\sum_{i=1}^{n-1} t_i$
3	1	$\sum_{i=1}^{n-1} (t_i - 1)$
4	1	$\sum_{i=1}^{n-1} (t_i - 1)$

In general, what is t_i in the best case? The worst case?

Solution:

(d) Formally show the run-time $T(n)$ of bubblesort by summing the terms in the table above, and state its bound.

Solution:

Problem 3. *Run-Time Comparisons (4 points)*

Sort all the functions below in increasing order of asymptotic (big-O) growth. (As usual, lg means logarithm base 2.) You do not need to justify your answer.

1. 5^{5^n}
2. $5n$
3. 5^{5n}
4. $4 \lg n$
5. $2 \lg \lg n$
6. n^6
7. $5(n \lg n)$
8. 5^n
9. 5^{n^5}
10. $(n/6)^6$

Solution: