CS3000: Algorithms & Data — Summer 2025 — Laney Strange

Homework 2
Due Thursday May 22 @ 9pm via 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 22 @ 9pm via 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'll have an opportunity to resubmit one homework at the end of the semester.
- Solutions must be typeset, preferably in LaTeX. If you need to draw any diagrams, you may draw them by hand as long as they are embedded in the PDF. I recommend using the source file for this assignment to get started.
- Any solutions that include pseudocode must be in the CLRS style.
- 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 online, or by speaking with students not enrolled in the class, is strictly forbidden.

Problem 1. *Mergesort (5 points)*

We can analyze an algorithm's space complexity in the same way we analyze its run-time. In the case of mergesort, every time we call merge we create two new subarrays.

(a) Give a $S(n)$ expression representing the amount of space created in Mergesort for an array of length n. You can ignore all constant-space requirements and restrict your expression to just the subarrays created.

Solution:

(b) Give a tight bound on the $S(n)$ space complexity.

Solution:

Problem 2. *Recurrence (10 points)*

(a) Use the iteration method to give a closed form run-time for Heapsort's MAX-HEAPIFY, given by the recurrence $T(n) = c_1 + T(2n/3)$, where c_1 is a constant, with base case $T(1) = c_2$.

Solution:

(b) Give a bound on the recurrence.

Solution:

(c) Solve the same recurrence using the Master Method, and explain your steps.

Solution:

Problem 3. *Mystery Algorithm Proof (10 points)*

You encounter the following pseudocode:

```
MYSTERY(a, n)
if n == 1
    return (1, a)
elseif n == 2
    return (a, a · a)
elseif n is odd
    (u, v) = MYSTERY(a, ⌊(n+1)/2⌋)
    return (u · u, u · v)
elseif n is even
    (u, v) = MYSTERY(a, ⌊(n+1)/2⌋)
    return (u · v, v · v)
```

(a) What would this function return in the following examples, assuming a is any integer?

- MYSTERY$(a, 1)$?

  Solution:

- MYSTERY$(a, 2)$?

  Solution:

- MYSTERY$(a, 3)$?

  Solution:

- MYSTERY$(a, 4)$?

  Solution:

(b) What does MYSTERY do in general, when given any integer a and an integer $n > 0$? Prove your assertion by induction on n.

Solution:

Problem 4. *Sum of Elements*

(a) Give pseudocode for an algorithm that takes in an array of integers A, length n, and an integer k. Return a boolean indicating whether there exist two positions i, j such that $k = A[i] + A[j]$. (Hint: Your solution can call any other procedure we've defined in lecture.)

Solution:

(b) Give a step-by-step explanation of how your algorithm would work on inputs $A = < 3, 15, 5, 16, 7, 1 >$, $n = 6, k = 12$ (it should return TRUE).

Solution:

(c) Give a tight bound on the run-time of your algorithm.

Solution:

Problem 5. *Quicksort V2 (10 points)*

The version of Quicksort we've covered assumes you choose one pivot q and place everything smaller (or equal) to its left and everything larger to its right.

The Java version[1] of Quicksort picks two pivots q_1 and q_2 where $q_1 \leq q_2$. We partition the array into three parts: (1) all the elements less than or equal to q_1, (2) all the elements greater than q_1 and less than or equal to q_2, and (3) all the elements greater than q_2.

(a) Give the pseudocode for this new version of QUICKSORT – just the Quicksort procedure. You can assume the correct PARTITION procedure exists and returns the locations of the two pivots.

Solution:

(b) In the original version, we chose the pivot $q = A[r]$ and partition returns its final location. In this version, we'll choose $q_1 = A[p], q_2 = A[r]$. What would the following array look like after the first call to PARTITION? $< 7, 6, 12, 8, 10 >$

(c) What would PARTITION return after being called that first time?

Solution:

(d) Give a recurrence for the best-case run-time of this new version of Quicksort and state its bound.

Solution:

[1]Implemented in JDK 7+ for sorting an array of primitives of length 47 or more.