CS3000: Algorithms & Data — Summer 2023 — Laney Strange

Homework 4 - Long
Due Tuesday June 6 @ 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 Tuesday June 6 @ 9pm Gradescope. You may submit up to 48 hours late for no penalty, but expect a delay in grading.
- 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. 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.
- 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.
- Finding solutions to homework problems on the web, or by asking students not enrolled in the class, is strictly forbidden.

Problem 1. *Heapsort* ($4 + 2 = 6$ *points)*

(a) Heapsort on a Min-Heap requires a HEAPIFY function, like would have happened in Part C of Recitation 3. Heapify takes two arguments: the array representing the min-heap, A, and the position of the current root, i. Heapify's job is to bubble-down the element currently in the root to its correct position while maintaining the heap properties (everything below a node is smaller, and a heap is a complete binary tree).

Recall that our array A has a *length* attribute but also a *heapsize* attribute. Give pseudocode for HEAPIFY. (Hint: A recursive algorithm is the simplest to implement!)

Solution:

(b) What is the run-time of your Heapify implementation?

Solution:

Problem 2. *Amortized Analysis* (1 + 1 + 1 + 1 + 2 = 6 *points)*

Let's say we build a binary counter. We store a bunch of bits in an array A, where each $A[i]$ is either 0 or 1. We start out with all the bits set to zero, and then start counting up. Every time we flip a bit, it costs 1.

$A[m]$	$A[m-1]$	...	$A[3]$	$A[2]$	$A[1]$	Cost
0	0	...	0	0	0	–
0	0	...	0	0	1	1
0	0	...	0	1	0	2
0	0	...	0	1	1	1
0	0	...	1	0	0	3
0	0	...	1	0	1	1
...						

(a) In a traditional worst-case analysis, what is the bound on how many bits flip for a single increment operation? The highest number we increment to is n.

Solution:

(b) In a traditional worst-case analysis, if we want to perform all the increments from 0 up to n, what is the bound on the overall run-time?

Solution:

(c) Let's switch to an amortized approach, using aggregate analysis, by focusing on specific individual bits. We know that the bit in $A[1]$ will be toggled with every increment. The bit in $A[2]$ will be toggled after every how many increments?

Solution:

(d) The bit in $A[3]$ will be toggled after every how many increments?

Solution:

(e) Give an expression for the total number of toggles over all the bits and state its bound. Is it an improvement over the original?

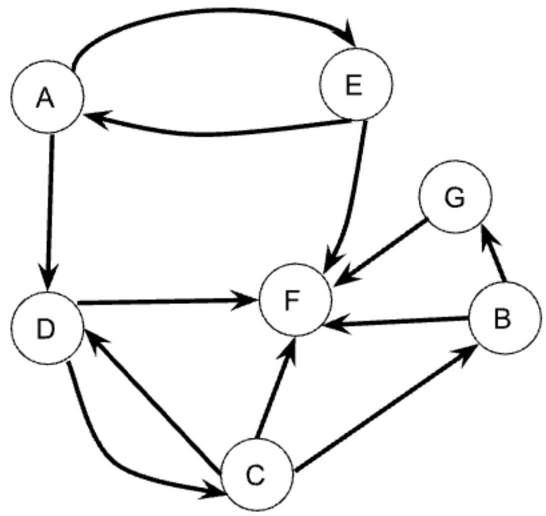
Solution:

Problem 3. *Amortized Analysis 2 (4 points)*

Suppose we perform a sequence of n operations on a data structure in which the ith operation costs i if i is a perfect square, and 1 otherwise. Use aggregate analysis to determine the amortized cost per operation.

Solution:

Problem 4. *BFS* ($2+4+4+2=12$ *points*)



(a) How would you represent the graph above using an adjacency matrix? (We've started the table for you.

vertex	A	B	C	D	E	F	G
A	0	0	0	1	1	0	0
B							
C							
D							
E							
F							
G							

Solution:

(b) In pseudocode, we use $G.A[v][u]$ to index into the adjacency matrix. For the example above, give a short pseudocode snippet that would print out all of E's neighbors.

Solution:

(c) You can find the Breadth-First Search pseudocode from class at https://course.ccs.neu.edu/cs3000/resources/BFS_Pseudocode_.pdf. The original version assumes that the graph is represented with an adjacency list. Copy the code below, and modify it to work with an adjacency matrix instead.

Solution:

(d) What is the run-time of BFS if done this way?

Solution: