

Name:

EID:

Exam #2 Practice Questions

Instructions. No calculators, laptops, or other devices are allowed. This exam is **closed book**, but you are allowed to use a **one-page** cheat sheet. You must submit your cheat sheet with the exam. Write your answers on the test pages. If you need scratch paper, use the back of the test pages, but indicate where your answers are. Write down your process for solving questions and intermediate answers that **may** earn you partial credit.

If you are unsure of the meaning of a specific test question, write down your assumptions and proceed to answer the question on that basis. **Questions about the meaning of an exam question will not be answered during the test.**

You have **75 minutes** to complete the exam. The maximum possible score is 100.

Some useful information:

Logarithms and Factorial:

$$\log(n!) = \Theta(n \log n)$$

Arithmetic Series:

$$\sum_{k=1}^n k = \frac{1}{2}n(n+1)$$

Sum of Squares:

$$\sum_{k=0}^n k^2 = \frac{n(n+1)(2n+1)}{6}$$

Sum of Cubes:

$$\sum_{k=0}^n k^3 = \frac{n^2(n+1)^2}{4}$$

Geometric Series:

$$\sum_{k=0}^n x^k = \frac{x^{n+1} - 1}{x - 1}$$

Infinite Geometric Series:

$$\sum_{k=0}^{\infty} x^k = \frac{1}{1-x}$$

Practice Problem 1: Minimum Spanning Trees

Suppose you are given a connected graph $G = (V, E)$, with edge costs that you may assume are all distinct. Given a particular edge $e \in E$, give an algorithm with $O(|V| + |E|)$ running time to decide whether e is contained in a minimum spanning tree of G .

Practice Problem 2: Binary Search Trees

Consider the problem of building a binary search tree T . In this tree, the root is identified as $T.root$ and each node x has a key value $x.key$, a right child $x.right$, and a left child $x.left$. Nodes with equal keys pose a problem for the implementation of binary search trees. Consider the following algorithm that adds a new node z to an existing binary search tree T :

TREE-INSERT(T, z)

```

1   $y \leftarrow \text{NIL}$ 
2   $x \leftarrow T.root$ 
3  while  $x \neq \text{NIL}$ 
4      do  $y \leftarrow x$ 
5          if  $z.key < x.key$ 
6              then  $x \leftarrow x.left$ 
7              else  $x \leftarrow x.right$ 
8   $z.p \leftarrow y$ 
9  if  $y = \text{NIL}$ 
10     then  $T.root \leftarrow z$ 
11     else if  $z.key < y.key$ 
12         then  $y.left \leftarrow z$ 
13         else  $y.right \leftarrow z$ 
```

- a) What is the asymptotic performance of TREE-INSERT when used to insert n items with identical keys into an initially empty binary search tree?
- b) We propose to improve TREE-INSERT by testing before line 5 whether or not $z.key = x.key$ and by testing before line 11 whether or not $z.key = x.key$. If equality holds, we implement one of the three following strategies. For each strategy, find the asymptotic performance of inserting n items with identical keys into an initially empty binary search tree. (The pseudocode for TREE-INSERT is given above for reference; the strategies are described for line 5, in which we compare the keys for z and x ; substitute y for x to arrive at the strategies for line 11.)

Strategy 1: Keep a boolean flag $x.b$ at node x and set x to either $x.left$ or $x.right$ based on the value of $x.b$, which alternates between FALSE and TRUE each time x is visited during insertion of a node with the same key as x .

Strategy 2: Keep a list of nodes with equal keys at x , and insert z into the list.

Strategy 3: Randomly set x to either $x.left$ or $x.right$. (For this strategy, give the worst-case performance and informally derive the average-case performance.)

Practice Problem 3: Breadth-First Search and Depth-First Search

We have a connected graph $G = (V, E)$, and a specific vertex $u \in V$. Suppose we compute a depth-first search tree rooted at u and obtain a tree T that includes all nodes of G . Suppose we then compute a breadth-first search tree rooted at u and obtain the same tree T . Prove that $G = T$. (In other words, if T is both a depth-first search tree and a breadth-first search tree rooted at u , then G cannot contain any edges that do not belong to T .)

Practice Problem 4: Topological Sort

The following algorithm relies on the fact that the *first* node in the order of a topological sort must have no incoming edges. This algorithm outputs nodes in a topologically sorted order.

TOPOLOGICAL-SORT(G)

- 1 select $v \in G.V$ with no incoming edges
- 2 output v
- 3 $G.V = G.V - v$
- 4 TOPOLOGICAL-SORT(G)

In general, TOPOLOGICAL-SORT assumes that G is a directed acyclic graph (a DAG). Suppose we were given an arbitrary directed graph that may or may not be a DAG. Extend the topological sort algorithm so that, given an input directed graph G , it outputs one of two things: (a) a topological ordering, thus establishing that G is a DAG; or (b) a cycle in G , thus establishing that G is not a DAG. The running time of your algorithm should be $O(|V| + |E|)$ for a directed graph $G = (V, E)$. (Hint: If you're stuck, try running a couple of instances of this topological sort algorithm to see how it's working.)

Scratch Page