Midterm
Introduction to Algorithms
601.433/633

Tuesday, October 22rd, 9am-10:15am

Name:

Ethics Statement

I agree to complete this exam without unauthorized assistance from any person, materials, or device.

Signature: Date:

1 Problem 1: Short Problems (25 points)

For the following, you *do not* need to give proofs or counterexamples. If the problem is multiple-choice, please circle your answer(s).

(a) Let $T(n) = 3T(n-2)$, with $T(1) = T(2) = 1$. Circle whichever of the following are true (there may be more than one).
$T(n) = \Theta(3^{n/2})$ $\quad T(n) = \Theta(2^{n/3})$ $\quad T(n) = \Theta(3^n)$ $\quad T(n) = \Theta(2^n)$

(b) $\log \log n = O(\log^* n)$
true false

(c) On *every* input, the expected running time of randomized quickselect is $O(\log n)$
true false

(d) If we changed the median-of-median algorithm (BPFRT) to use groups of size 7 rather than 5, then its running time is still $O(n)$.
true false

(e) The sorting lower bound implies that no sorting algorithm can sort n integers between 1 and n^{10} in $o(n \log n)$ time (in the worst case).
true false

(f) If we do the exact same number of Extract-Mins as Inserts, then the total running time of a binary heap and a binomial heap are the same asymptotically (they are $\Theta(\cdot)$ of each other)
true false

(g) If we use trees with union-by-rank and path compression, the amortized cost of both Union and Find is $O(\log^* n)$.
true false

(h) Let H be a universal family of hash functions from U to $[M]$. Then for any sequence of L insert, delete, and lookup operation, the expectation of the total running time (when h is drawn uniformly from H) is $O(L)$.
true false

(i) Let H be a universal family of hash functions. Then for every $h \in H$, if we choose two elements $x, y \in U$ uniformly at random, the probability that $h(x) = h(y)$ is at most $1/M$
true false

(j) If we build a B-tree with $t = \Theta(\log n)$, then the running time of Insert is $O(\log n)$
true false

2 Problem 2 (25 points): Data Structures

(a) (8 points) Draw all possible 2-3-4 trees on the keys $\{1, 2, 3, 4, 5\}$.

(b) (8 points) For each of the trees in the previous part, draw a corresponding red-black tree (use a dashed line for red and a solid line for black).

(c) (9 points) How many different binomial heaps are there on the keys $\{1, 2, 3, 4, 5\}$? Justify your answer. Hint: think about the structure of a binomial heap. How many ways are there to fill in that structure legally?

3 Problem 3 (25 points): Amortized Analysis

Suppose that we want to implement a stack as an array, for example if we want to also have read-access to elements in the middle rather than just being able to pop out the element on top. We saw in class that if we double the size of the array when it is full, then the amortized cost of an insert is still only $O(1)$. In other words, n inserts only take time $O(n)$ even though a single insert might take $\Omega(n)$ time (to create a new array of twice the size and copy over all of the elements).

What if we add in the ability to pop from the stack, though? Pops can be implemented quickly, but we might end up in the undesirable situation of having an array that is much, much bigger than the size of the stack. For example, if we do n pushes and $n-1$ pops, then the size of the array will be $\Theta(n)$ even though there is only one element in the stack!

To fix this, consider the following algorithm. Let D be the current array, and suppose that α positions of the array have a stack element. So if $\alpha = |D|$ then the array is full, and if $\alpha = 0$ then the array is empty. As before, if the array is full then when we push a new element we double its size. Let's say that this takes time exactly $|D| + 1 = \alpha + 1$, since it takes $|D| = \alpha$ time to copy the elements and 1 to push the new element. On the other hand, if after a pop the array is less than $1/3$ full then we *contract* the array by making a new array of $2/3$ the size and copying all of the elements. This also takes time $\alpha + 1$ (to pop a single element and then copy the rest), but note that in this case $\alpha = |D|/3$ rather than $|D|$.

Consider the potential function $\Phi = |2\alpha - |D||$.

(a) (9 points) Prove that on any push, the amortized cost (when using the above potential function) is at most $O(1)$.

(b) (9 points) Prove that on any pop, the amortized cost (when using the above potential function) is at most $O(1)$.

(c) (7 points) Using the previous parts, prove that for every sequence of n operations (pushes and pops) the total running time is $O(n)$. You may assume initially $|D| = 2$ and there is one element in the array.

4 Problem 4 (25 points): Dynamic Programming

Let $G = (V, E)$ be a directed graph with vertices $v_1, v_2, \ldots, v_n$. Recall that $|E| = m$. We say that G is an *ordered graph* if it has the following two properties:

1. Every edge goes from a vertex with a lower index to a vertex with a higher index. That is, every edge has the form (v_i, v_j) with $i < j$.

2. Every vertex other than v_n has at least one outgoing edge.

The length of a path is the number of edges on it. In this problem you will design an efficient algorithm to find the length of the *longest* path which starts at v_1 and ends at v_n. Note that by the two properties of an ordered graph, every node has at least one path to v_n.

(a) Let $S(i)$ denote the length of the longest path that starts at v_i and ends at v_n. Write a (recursive) formula for $S(i)$, which in the next part will form the basis of a dynamic programming algorithm. You do not need a formal proof of correctness, but should justify your answer.

(b) Design an $O(m)$-time algorithm to find the longest path which starts at v_1 and ends at v_n (using your solution from part (a)). Argue correctness and running time (again, you do not need a fully formal proof, but should provide justification).