Midterm
Introduction to Algorithms
601.433/633

Tuesday, 22 October 2019, 12-1:15pm

Name:

Ethics Statement

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

Signature: Date:

1 Problem 1 (25 points): Short Answer

Answer the following (circling your choice for the multiple choice questions). You *do not* need to give a proof, nor do you need to give counterexamples.

(a) For the Union-Find problem, if we use union by rank and path compression, then the rank stored in each node is always the height of this node.
true **false**

(b) Take the following list of functions and arrange them in ascending order of growth rate. That is, if function $g(n)$ immediately follows $f(n)$ in your list, then it should be the case that $f(n)$ is $O(g(n))$.

$$\sqrt{n} \qquad \log^* n \qquad n^{\log n} \qquad 2^{\sqrt{\log n}}$$

$\log^* n, 2^{\sqrt{\log n}}, \sqrt{n}, n^{\log n}$

(c) Let $T(n) = 4T(n/7) + 8n$. Then $T(n)$ is asymptotically equal to: $\Theta(n)$

(d) The worst-case expected running time of Randomized Quicksort is: $\Theta(n \log n)$

(e) Every leaf of a red-black tree has the same depth.
true **false**

(f) The time to do a lookup in a B-tree is independent of the parameter t used for the tree.
true false

(g) There is a heap data structure which has $O(1)$ amortized Insert and $O(1)$ amortized Extract-Min
true **false**

(h) Let H be a universal family of hash functions from a universe U onto a table of size M. Then $\Pr_{h\sim H}[h(e) = i] = 1/M$ for all elements $e \in U$ and table positions $i \in \{1, 2, \dots, M\}$.
true **false**

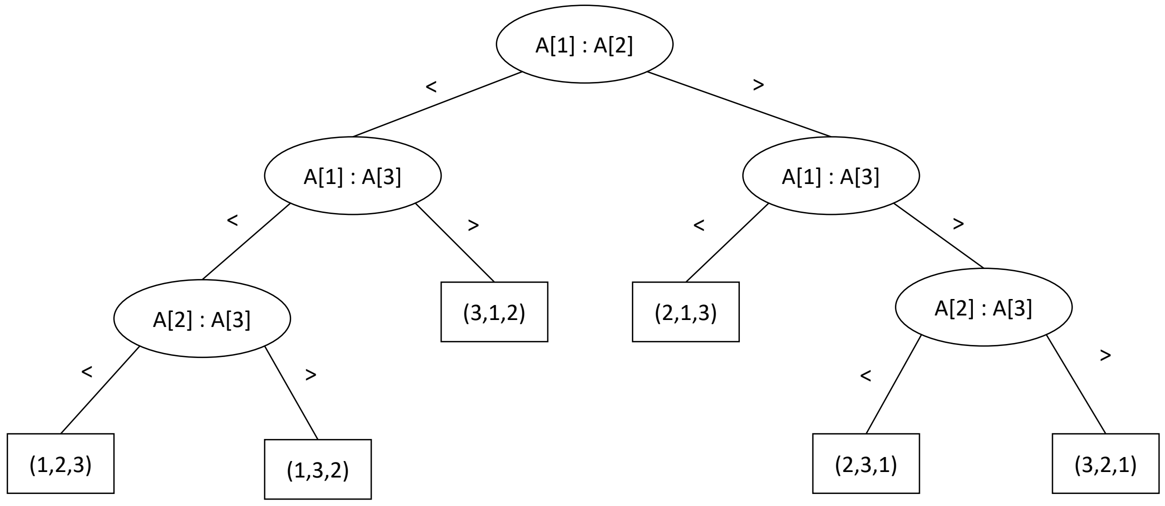
(i) Least-significant first radix sort is correct no matter which sorting algorithm we use on each digit.
true **false**

(j) There is an algorithm which can find the $n^{1/10}$-smallest number in an unsorted array of n numbers in $O(n)$ time.
true false

2 Problem 2 (25 points): Searching/Sorting

(a) (12 points) We talked in class about the correspondence between sorting algorithms and decision trees. Consider deterministic quicksort where the pivot is always the first element, done an array A of size 3 with entries $[A[1], A[2], A[3]]$. Draw the decision tree corresponding to this algorithm.

Solution.



(b) (13 points) Suppose we modify the median-of-medians algorithm for selection (i.e., the BPFRT algorithm) to use $n/3$ groups of size 3 rather than $n/5$ groups of size 5. What is the running time of the resulting algorithm? Justify your answer, but a formal proof is not necessary.

Solution. $\Theta(n \log n)$. To see this, note that $|L|, |G|$ are both at most $2n/3$ under the new algorithm. This is because $n/6$ groups contribute at least two elements to L, and $n/6$ groups contribute at least 2 elements of G. Thus $|L|, |G| \geq 2(n/6) = n/3$, and hence $|L|, |G| \leq 2n/3$. Thus the running time is $T(n) = T(n/3) + T(2n/3) + cn = \Theta(n \log n)$.

3 Problem 3 (25 points): Amortized Analysis

Assume we are given an implementation of a stack in which the Push and Pop operations each take constant time. For concreteness, let's suppose that Push and Pop each take exactly 1 unit of time. We now implement a queue using two stacks A and B as follows:

```
ENQUEUE($x$):
    1. Push $x$ onto $A$
DEQUEUE():
    1. If $B$ is nonempty, return B.Pop()
    2. Otherwise, pop each element from $A$ and push it into $B$, except return the final element
       of $A$ (the last one popped) rather than push it onto $B$.
```

(a) What is the worst-case running time of ENQUEUE(x) and DEQUEUE() (asymptotically, as a function of the number of elements n)? Justify your answer.

Solution. Enqueue has worst case running time of $1 = O(1)$, since it is just a single push. Dequeue has worst-case running time of $\Theta(n)$, since if we first do n Enqueues and then a single Dequeue, that Dequeue will have to pop all n elements off of A and then push them all onto B (except the last element), for a total time of $2n = \Theta(n)$.

(b) What is the *amortized* running time of ENQUEUE(x) and DEQUEUE()? Justify your answer. Hint: consider the potential function $\Phi = 2k$, where k is the number of elements in A.

Solution. Under this potential function, the amortized cost of Enqueue is $1 + \Delta\Phi = 1 + 2 = 3 = O(1)$.

For a Dequeue, we break into two cases. If B is nonempty, then the algorithm just pops from it (which doesn't change the potential), for a total amortized cost of $1 + \Delta\Phi = 1 + 0 = 1 = O(1)$. On the other hand, if B is empty, then the true cost is $2k - 1$ (since $k - 1$ elements are popped and then pushed, and one element is just popped) and the change in potential is $-2k$. Thus the amortized cost is $2k - 1 - 2k = -1 = O(1)$. Thus in either case, the amortized cost is $O(1)$.

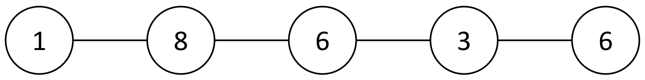
4 Problem 4 (25 points): Dynamic Programming

Let $P = (V, E)$ be the path graph: $V = \{v_1, v_2, v_3, \ldots, v_n\}$ and $E = \{\{v_i, v_{i+1}\} : 1 \le i \le n-1\}$. Let $w : V \to \mathbb{R}_{\ge 0}$ be a nonnegative weighting of the vertices.

We say that a set $U \subseteq V$ is *independent* if $\{x, y\} \notin E$ for all $x, y \in U$. In other words, a set of vertices is independent if no two are adjacent. Our goal will be to find the value of the maximum weight independent set in the path graph, i.e., the value

$$\max_{U \subseteq V : U \text{ independent}} \left(\sum_{u \in U} w(u) \right).$$

For example, consider the five node path graph with the following weights:



In this example, the maximum weight independent set consists of $\{v_2, v_5\}$ and has weight $8+6 = 14$.

(a) Consider the following greedy algorithm. Initially $S = \emptyset$. While P is nonempty, add the maximum weight vertex v in P to S, and delete v and its neighbors from P. Return S.

This algorithm works in the above example. Give an instance of the path graph with weights in which it *does not* return the right solution, and explain why (what it will return and what the correct answer is).

Solution. Consider the path of length 3 with weights 2, 3, 2 respectively. Then the algorithm will pick only the middle node for a total weight of 3, while the optimal solution is to pick the outside nodes for a total weight of 4.

(b) Let $OPT(i)$ denote the total weight of the maximum weight independent set of the prefix ending at i, i.e., of the subgraph $\{v_1, v_2, \ldots, v_i\}$. Write a formula for $OPT(i)$ in terms of the smaller subproblems. Justify your answer (formal proof not necessary).

Solution. Consider $OPT(i)$. If v_i is part of this optimal solution, then v_{i-1} cannot be but we can choose the optimal solution for the prefix ending at $i - 2$. So in this case, $OPT(i) = w(i) + OPT(i - 2)$. On the other hand, if v_i is not part of this optimal solution, then the optimal solution is the same as for the prefix ending at $i-1$, so $OPT(i) = OPT(i-1)$. Thus we get that

$$OPT(i) = \begin{cases} 0 & \text{if } i = 0 \text{ or } i = -1 \\ \min(w(i) + OPT(i-2), OPT(i-1)) & \text{otherwise} \end{cases}$$

(c) Now consider the obvious dynamic programming algorithm (top-down or bottom-up) based on your answer to part (b). What is its running time? Justify your answer (formal proof not necessary).

Solution. Each table entry takes $O(1)$ time to fill in (two table lookups, a sum, and a min), and there are $O(n)$ table entries. Thus the running time is $O(n)$.