4

1. [6 points] T/F - State whether the statements below are true or false. No need for justification for your answer, though you can include one if you think it would be helpful.

(a) F If the worst case running time of an operation is $\Theta(n^2)$ the amortized worst case running time of that operation cannot be $O(n)$.

Answer is still False: I turned out to be right for this question.

(b) F In a binary heap, if we swap the *values* of the left and right children of a heap, it is still a valid heap.

(c) T You can solve the following recurrence using the master method: $T(n) = 2T(n/4) + \sqrt{n}$.

(d) T Randomized-Quicksort is $\Theta(n \log n)$ in the worst case.

→ Basically just ignoring the case where it randomizes the array in sorted order ($\Theta(n^2)$) ↳ with the small caveat that even in the case where Randomized-Quicksort randomizes our list in sorted order, I'm just ignoring it since the probability of that happening is very very small.

(e) F In a red-black tree if $n > 1$ is is not possible for all of the nodes to be black. ↳ We could have all nodes being ~~red~~black and it would still be a valid RB-tree.

* (f) T The best case running time for unioning two binomial heaps is $O(1)$.

(d) correction:

Answer => F

Explanation => Even though randomizing the data usually mitigates the issue of encountering the worst-case O(n^2) scenario, which is common with sorted and reverse-sorted arrays, it doesn't alter the underlying theoretical worst-case complexity. That is, despite the practical improvements with randomization, the possibility of a sorted or reverse-sorted array means we cannot definitively state that Randomized Quicksort operates in Theta(nlogn) in the worst case. The theoretical worst case complexity remains O(n^2).

5

2. **[5 points]** For each pair of functions below, state whether they are in the same big-O category or, if they are different, state which function is in a larger big-O category (*circle one of the three options*). For example, if $f(n) = n$ and $g(n) = n^2$, you would say that $g(n)$ is larger.

a. (circle one) $f(n)$ is $O(g(n))$ $g(n)$ is $O(f(n))$ (same big-O)	$f(n) = \log_5 3^n$	$g(n) = 10n + 2$
b. (circle one) $f(n)$ is $O(g(n))$ ($g(n)$ is $O(f(n))$) same big-O	$f(n) = 5n^2 + n^2 \log n$	$g(n) = 6n^2$
c. (circle one) $f(n)$ is $O(g(n))$ $g(n)$ is $O(f(n))$ (same big-O)	$f(n) = \log_3(n!)$	$g(n) = 10n \log_2 n$
d. (circle one) ($f(n)$ is $O(g(n))$) $g(n)$ is $O(f(n))$ same big-O	$f(n) = \log n$	$g(n) = \sqrt{n}$
e. (circle one) $f(n)$ is $O(g(n))$ $g(n)$ is $O(f(n))$ (same big-O)	$f(n) = 3^{\log_4 n}$	$g(n) = n^{\log_4 3}$

*f(n) is in a larger big-O category, which is, $O(n^2 \log n)$.

*g(n) is in a larger big-O category, which is $O(n^{1/2})$. $n^{1/2}$ dominates $\log n$ since according to the big-O rules of thumb, any polynomial beats any logarithmic function.

5

3. [5 points] A complete binary tree is a tree where every internal node has two children and all leaves are at the same level. Prove *using induction* that in a complete binary tree the number of internal nodes (non-leaf nodes) is $(n-1)/2$, where n is the total number of nodes in the tree.

- We want to show that the number of internal nodes in a complete binary tree is $\frac{(n-1)}{2}$
- for the base case, the simplest binary tree (complete) one can have is just the single node, where n=1.
  If n=1, #internal nodes $= \frac{1-1}{2} = 0$ ✓ (which is true since such a tree doesn't have any internal nodes)
- for the inductive case (using structural induction), we assume same event is true and prove the next event is true
  → Inductive hypothesis: Assume that the statement holds for the left and right subtrees, each with K nodes where $K < n$, thus both have $\frac{K-1}{2}$ internal nodes.
  → Inductive step to prove: We want to show/prove that the number of internal nodes in a complete binary tree is ~~$\frac{(2K+1)-1}{2}+1$~~ for the root node.
  $\frac{(2K+1)-1}{2} = \frac{2K}{2} = K$ * internal nodes ✓

→ We know that: #of internal nodes (left subtree) $= \frac{K-1}{2}$

∴ #of internal nodes (right subtree) $= \frac{K-1}{2}$

Therefore, the total number of internal nodes is

$\frac{K-1}{2} + \frac{K-1}{2} + 1 = \frac{K-1}{2} + \frac{K-1}{2} + \frac{2}{2} = \frac{2K-2+2}{2} = \frac{2K}{2} = K$ ✓ ok

↳ total internal nodes.

(1 ↓ root node)

* Brief explanation for how I got $\frac{(2K+1)-1}{2}$;

- for the complete binary tree, the total number of nodes is $K+K+1 = 2K+1$, where the 1 represents the root node, this results in us needing to show that such a tree has to have K internal nodes

* We've shown the statement to hold true via the inductive hypothesis (K-total internal nodes, which is where we wanted to arrive).
* Since we've proved the statement is true for n=1 and for n'=K+1, assuming it's true for n=K', we've shown the statement is true for $\forall n \geq 1$, thus the total number of internal nodes in a complete binary tree is $\frac{(n-1)}{2}$

Master method general form: $T(n) = aT(\frac{n}{b}) + f(n)$

4. [5 points] Solve the following recurrences. If you use the master method, give a Θ bound, otherwise, big-O is fine. You do not need to *prove* that it is correct.

(a) $T(n) = 5T(n/2) + n^2$

- Using the master method: $a=5$, $b=2$ and $f(n)=n^2$
- $\log_b a = \log_2 5$ (which is a figure between 2 and 3 but closer to 2).
- We now ask ourselves these 3 questions?

a) Is $n^2 = O(n^{\log_2 5 - \varepsilon})$ for $\varepsilon > 0$? Well, if we choose ε to be that number that brings $\log_2 5$ down to 2 then $O(n^{\log_2 5 - \varepsilon}) = O(n^2)$, which is indeed true since $n^2 = O(n^2)$

b) Is $n^2 = \Theta(n^{\log_2 5})$? No, n^2 is not lower-bounded by $\Omega(n^{\log_2 5})$ since $n^{\log_2 5} > n^2$.

c) Is $n^2 = \Omega(n^{\log_2 5 + \varepsilon})$? No, even worse $g(n) = n^{\log_2 5 + \varepsilon}$ is an even bigger function than n^2 thus n^2 cannot possibly be lower-bounded by $\Omega(n^{\log_2 5 + \varepsilon})$.

→ By the master method, since $n^2 = O(n^{\log_2 5 - \varepsilon})$ for ε, our recurrence relation is $T(n) = \Theta(n^{\log_2 5})$ ✓

(b) $T(n) = T(n-4) + \sqrt{n}$ (I choose the substitution method)

Rationale for guess: $T(n)$ does $\sqrt{n}$ amount of work initially, the next time the input decrease by 4 and the work done becomes $\sqrt{n-4}$. The next iteration, the work done is $\sqrt{n-8}$ and so on and so forth. Eventually n will reduce to 1 and the work done is $\sqrt{1} = 1$. I make an educated guess that a possible upper bound is $O(n\sqrt{n})$ since at each level the work done will be at most $\sqrt{n}$ and the number of levels (till the base case) will never exceed n so a reasonable upper-bound big-O is $O(n\sqrt{n})$.

- To prove this guess, we use induction: We begin by assuming $T(k) = O(k\sqrt{k})$ for all $k < n$.
- Thus, using the definition of big-O, we know that $T(n-4) \leq c(n-4)\sqrt{n-4}$
- We now need to show that $T(n) = O(n\sqrt{n})$.

i) $T(n) = T(n-4) + \sqrt{n}$

ii) $T(n) \leq c(n-4)\sqrt{n-4} + \sqrt{n}$ (by inductive hypothesis)

iii) $T(n) \leq cn\sqrt{n-4} - 4c\sqrt{n-4} + \sqrt{n}$

iv) $T(n) \leq cn\sqrt{n} - 4c\sqrt{n-4} + \sqrt{n}$ (If $T(n) \leq cn\sqrt{n-4} + (\ldots)$, then it is definitely smaller than $cn\sqrt{n} + (\ldots)$, that's why I replaced $cn\sqrt{n-4}$ with $cn\sqrt{n}$.)

- Our residual is $\sqrt{n} - 4c\sqrt{n-4}$, if $T(n)$ is to be $\leq cn\sqrt{n} + \text{residual}$, then the residual must be ≤ 0.
- The question now becomes, does some constant c' exist such that $T(n) \leq c' n\sqrt{n}$.

(Continued on the next page → → →

- Showing that the residual is ≤ 0.

$$\sqrt{n} - 4c\sqrt{n-4} \leq 0$$

$$\sqrt{n} \leq 4c\sqrt{n-4}$$

$$4c\sqrt{n-4} \geq \sqrt{n}$$

$$c \geq \frac{\sqrt{n}}{4\sqrt{n-4}}$$

- We need to find a 'c' and an 'n', such that this holds, if $n=100$, $\sqrt{n} = 10$ and $\sqrt{n-4} = \sqrt{96}$ = something between 9 and 10 but closer to 10. (Note: I don't know exactly what $\sqrt{96}$ is but I'm going to say $\sqrt{96} \approx 9.8$ since I don't have a calculator)

$$c \geq \frac{10}{4 \times 9.8}$$

$$c \geq \frac{10}{39.2}$$

- Therefore, a 'c' and an 'n' exist which means $T(n) = T(n-4) + \sqrt{n} = O(n\sqrt{n})$.