CS 161: Practice Midterm (Fall 2016) — Solutions

Question 1: True or False (10 points)

For each statements, indicate whether it is true or false. No justification is needed.

1. For all positive functions $f(n)$, we have $f(n) + O(f(n)) = \Theta(f(n))$.
   True: Upper bound: $f(n) + O(f(n)) \leq f(n) + cf(n) = (1 + c)f(n) = O(f(n))$ for some constant c.
   Lower bound: $f(n) + O(f(n)) \geq f(n) = \Omega(f(n))$.

2. $n^{\log n} = O(2^n)$.
   True: $n^{\log n} = (2^{\log n})^{\log n} = 2^{\log^2 n} = O(2^n)$.

3. Depth-first search can be used to detect cycles in both directed and undirected graphs.
   True

4. A list of n English letters can be sorted in $O(n)$ time.
   True: There are a constant number of English letters, so each one has only constant bit length. Hence radix sort, counting sort, and bucket sort will all sort this in linear time.

5. Given a sorted list of n distinct numbers, we can construct a binary search tree on the numbers in $O(n)$ time.
   True: Set the median as the root and recurse on both sides.

6. Suppose a hash table draws its hash function from a universal hash family. Then there will be no collisions if the number of buckets (size of the underlying array) exceeds the number of items in the hash table.
   False: The hash function we pick can easily map two distinct elements to the same bucket.

7. Suppose a hash table has more buckets than the size of the universe of keys. Then there exists a hash function such that there will be no collisions.
   True: Your hash function can simply map each key to a unique hash value.

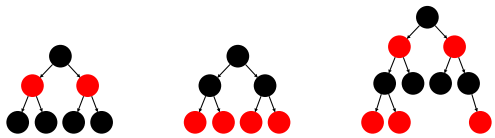
8. A binary search tree with n nodes has depth $O(\log n)$.
   False: Consider a binary search tree where the entries are inserted in sorted order. Then the tree will be a linked list (depth $\Theta(n)$).

9. $4^n = \Theta(2^n)$.
   False: $\lim_{n\to\infty} \frac{2^n}{4^n} = \lim_{n\to\infty} \left(\frac{1}{2}\right)^n = 0$. So, $4^n = o(2^n)$, and thus $4^n \notin \Theta(2^n)$.

10. The expected running time of quicksort is $O(n \log n)$ even if the inputs are not random, but picked by an adversary.
    True: This is precisely what we mean by the "expected" running time as shown in lecture—it is the expected performance over our choice of pivots, regardless of the input.

Question 2 (8 points)

The following trees are valid red-black trees.



Question 3 (20 points)

1. $\mathsf{CountReversals}(L[1:n]) = \mathsf{CountReversals}(L[1:\frac{n}{2}]) + \mathsf{CountReversals}(L[\frac{n}{2}+1:n]) + X$, where X is the number of reversals that pass through the center (ie, $i \leq \frac{n}{2}$ and $j > \frac{n}{2}$). Notice that calculating X can be done $O(n)$ time if $L[1:\frac{n}{2}]$ and $L[\frac{n}{2}:n]$ are sorted, even though there are $O(n^2)$ possible reversals that pass through the center.

2. Modify $\mathsf{MergeSort}(A)$ to return the number of reversals in A while sorting A; call this new procedure $\mathsf{CountReversals}$. Then, the number of reversals that cross the center is not difficult to calculate. $\mathsf{CountReversals}(A) = \mathsf{CountReversals}(A[1:\frac{n}{2}]) + \mathsf{CountReversals}(A[\frac{n}{2}+1:n]) + \mathsf{Merge}(A[1:\frac{n}{2}], A[\frac{n}{2}+1:n])$.

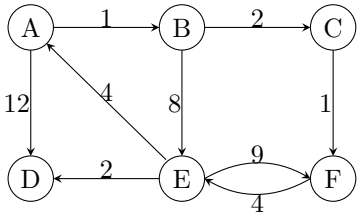
   Our new merge step would initialize two pointers i and j at the beginning of $L[1:\frac{n}{2}]$ and $L[\frac{n}{2}+1:n]$, a counter c at 0, and an empty array B which we sort the entries of A into.

   Case 1. If $L[i] \leq L[j]$, then there is no reversal between indices i and j, so we copy element $L[i]$ to B and increment i.

   Case 2. If $L[i] > L[j]$, then there are $\frac{n}{2} - i + 1$ reversals in total between indices $\{i, i+1, \ldots, \frac{n}{2}\}$ and j, so we add $\frac{n}{2} - i + 1$ to c, copy $L[j]$ to B, and increment j.

   Continue doing this, and handle edge cases at the end properly. Finally, replace the values in A with the values in B, and return c.

Question 4 (15 points)



1. BFS: A, B, D, C, E, F
2. DFS: A, B, C, F, E, D
3. The order in which Dijkstra's finds the correct distances is A, B, C, F, E, D. The paths generated by Dijkstra's, and the distances to each node are:

   A: A (0)

   B: A $\rightarrow$ B (1)

   C: A $\rightarrow$ B $\rightarrow$ C (3)

   F: A $\rightarrow$ B $\rightarrow$ C $\rightarrow$ F (4)

   E: A $\rightarrow$ B $\rightarrow$ C $\rightarrow$ F $\rightarrow$ E (8)

   D: A $\rightarrow$ B $\rightarrow$ C $\rightarrow$ F $\rightarrow$ E $\rightarrow$ D (10)

Question 5: Quick Questions (22 points)

Merging Binary Search Trees (6 points)

Assume T_1 and T_2 are binary search trees, not necessarily balanced, with m and n nodes, respectively. Design an algorithm that merges T_1 and T_2 into a balanced binary search tree in $O(m+n)$ time.

Solution: Do the inorder traversal on each tree to get two sorted lists. Merge them to get one sorted list. Make its median the root of the tree and recursive on both sides to get a balanced search tree.

2-SUM (6 points)

You are given two unsorted arrays A, B of n integers, and an integer z. Find if there are two elements $A[i]$ and $B[j]$ such that $A[i] + B[j] = z$ in expected running time $O(n)$.

Solution: Create a hash table T. Insert every element of A into T. Now, take each integer $b \in B$ and check if $z - b$ is in T. If it is, report YES. If no such match is found for any $b \in B$, report NO.

Recurrences (10 points)

(a) $T(n) = 4T(n/2) + n^2\sqrt{n}$

Solution: $T(n) = \Theta(n^2\sqrt{n})$. This falls into case 3 of the master theorem: $f(n) = n^2\sqrt{n} = \Omega(n^{\log_2 4+\frac{1}{2}}) = \Omega(n^{\log_b a+\epsilon})$, and $af\left(\frac{n}{b}\right) = 4f\left(\frac{n}{2}\right) = 4\left(\frac{n}{2}\right)^2\sqrt{\frac{n}{2}} = \frac{\sqrt{2}}{2}n^2\sqrt{n} \leq cf(n)$ for some $c < 1$ (namely, $\frac{\sqrt{2}}{2}$) and all sufficiently large n.

(b) $T(n) = 3T(n/2) + n\log n$

Solution: $T(n) = \Theta(n^{\log_2 3})$. This falls into case 1 of the master theorem: $f(n) = n\log n = O(n^{\log_2 3-\epsilon}) = O(n^{\log_b a-\epsilon})$ where $\epsilon < \log_2 3 - 1 = \log_2 \frac{3}{2}$.

(c) $T(n) = T(n/2) + T(n/4) + T(n/8) + n$

Solution: $T(n) = \Theta(n)$. The proof uses the substitution method.

(d) $T(n) = T(n-1) + 1/n$

Solution: This solution corresponds to the harmonic series, where $T(n) = 1/1 + 1/2 + 1/3 + ... + 1/n = \sum_{i=1}^{n} \frac{1}{i}$. In homework 1, this was proved to be $\Theta(\log n)$. Therefore, $T(n) = \Theta(\log n)$.

(e) $T(n) = T(\sqrt{n}) + 1$

Solution: Let $m = \log n$. $n = 2^m$. Also, let $S(m) = T(2^m)$. We have $T(2^m) = T(2^{\frac{m}{2}}) + 1$, so $S(m) = S\left(\frac{m}{2}\right) + 1$. This falls into the case 2 of the master theorem where $a = 1, b = 2, d = 0$. $S(m) = \Theta(\log m)$. Therefore, $T(n) = \Theta(\log\log n)$.