Problem Set #6

Back to Week 6

5/5 points earned
(100%)

Quiz passed!

1 / 1
points

1.
Suppose we use a hash function h to hash n distinct keys into an array T of length m. Assuming simple uniform hashing --- that is, with each key mapped independently and uniformly to a random bucket --- what is the expected number of keys that get mapped to the first bucket? More precisely, what is the expected cardinality of the set $\{k : h(k) = 1\}$.

- $m/(2n)$
- $1/n$
- m/n
- n/m

Correct Response

Use linearity of expectation, with one indicator variable for each key. The probability that one key hashes to the first bucket is $1/m$, and by linearity of expectation the total expected number of keys that hash to the first bucket is just n/m.

- $n/(2m)$
- $1/m$

---

1 / 1 points

2.

You are given a binary tree (via a pointer to its root) with n nodes, which may or may not be a binary search tree. How much time is necessary and sufficient to check whether or not the tree satisfies the search tree property?

- $\Theta(n \log n)$
- $\Theta(n)$

Correct Response

For the lower bound, if there is a violation of the search tree property, you might need to examine all of the nodes to find it (in the worst case).

- $\Theta(height)$
- $\Theta(\log n)$

---

1 / 1 points

3.

You are given a binary tree (via a pointer to its root) with n nodes. As in lecture, let size(x) denote the number of nodes in the subtree rooted at the node x. How much time is necessary and sufficient to compute size(x) for every node x of the tree?

- $\Theta(height)$

$\Theta(n^2)$

$\Theta(n \log n)$

$\Theta(n)$

Correct Response
For the lower bound, note that a linear number of quantities need to be computed. For the upper bound, recursively compute the sizes of the left and right subtrees, and use the formula size(x) = 1 + size(y) + size(z) from lecture.

---

1 / 1
points

4.
Which of the following is *not* a property that you expect a well-designed hash function to have?

The hash function should be easy to compute (constant time or close to it).

The hash function should "spread out" every data set (across the buckets/slots of the hash table).

Correct Response
As discussed in lecture, unfortunately, there is no such hash function.

The hash function should "spread out" most (i.e., "non-pathological") data sets (across the buckets/slots of the hash table).

The hash function should be easy to store (constant space or close to it).

---

1 / 1
points

5.

Suppose we relax the third invariant of red-black trees to the property that there are no *three* reds in a row. That is, if a node and its parent are both red, then both of its children must be black. Call these *relaxed* red-black trees. Which of the following statements is *not* true?

- Every red-black tree is also a relaxed red-black tree.
- Every binary search tree can be turned into a relaxed red-black tree (via some coloring of the nodes as black or red).

Correct Response
A chain with four nodes is a counterexample.

- There is a relaxed red-black tree that is not also a red-black tree.
- The height of every relaxed red-black tree with n nodes is $O(\log n)$.