CPSC 320 Midterm 2
Wednesday November 13th, 2013

[9] 1. State whether each of the following statements is true or false. Justify each of your answers briefly.

[3] a. If a recurrence relation can be solved using the Master theorem, then it can also be solved using recursion trees.

Solution : This is true: the Master theorem was proved using recursion trees.

[3] b. When we create a new node in a skip list, we choose the level of the new node uniformly and at random between 1 and the maximum level used for the skip list (that is, it has equal probabilities of having height 1, 2, 3, etc).

Solution : This is false: a node has a probability p^i of having more than i levels, so the probabilities decrease exponentially with i. In other words, a node is much more likely to only have one level than two, much more likely to have two levels than three, etc.

[3] c. A binomial heap with 21 elements always consists of exactly three separate binomial trees.

Solution : This is true: a binomial heap with 21 elements (010101 in binary) will always have a tree of order 4, a tree of order 2 and a tree of order 0.

[5] 2. Randomized Caching

[2] a. How does the randomized caching algorithm discussed in class choose an item to evict when a cache miss occurs?

Solution : It chooses an element uniformly and at random out of the *unmarked* elements current held in the cache.

[3] b. Why does the probability of a cache miss for the i^{th} request for a recurring unmarked item increase with i?

Solution : As i increases, the number of unmarked recurring items remaining in the case decreases. So it becomes more and more likely that the element we are requesting was evicted (there may also be more fresh items in the cache).

[7] 3. Describe an efficient divide and conquer algorithm that takes as input a sorted array A of real numbers, and returns a binary search tree of minimum height that contains the elements of the array. If you want, you can add parameters other than the array A to your function.

Solution : The idea is to place the median of the array at the root, and the smaller (larger) elements in its left (right) subtree. In pseudo-code:

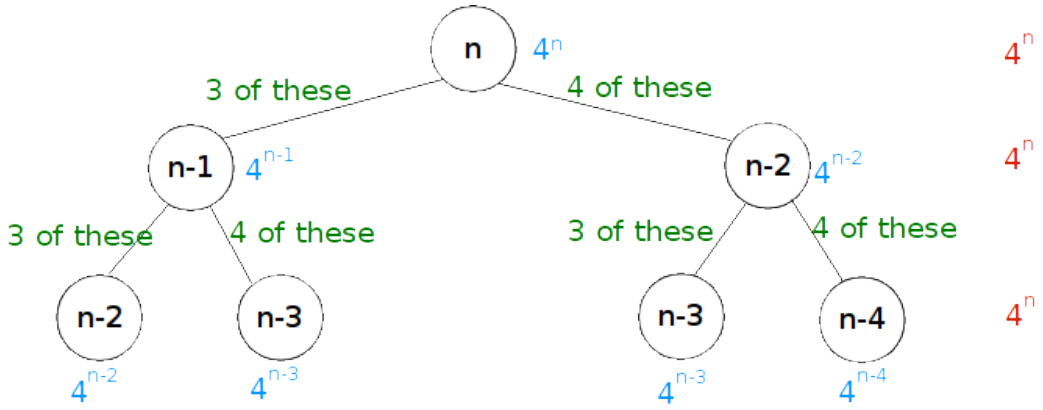
```
Function makebalancedtree(A, first, last)
  if first ≤ last then
    mid ← ⌊ (first + last) / 2 ⌋
    N2 ← new node with key A[mid]
    left[N2] ← makebalancedtree(A, first, mid - 1)
    right[N2] ← makebalancedtree(A, mid + 1, last)
    return N2
  endif
  return null
```

[8] 4. Using recursion trees, prove tight upper and lower bounds on the function $T(n)$ defined by

$$T(n) = \begin{cases} 3T(n-1) + 4T(n-2) + 4^n & \text{if } n \geq 2 \\ 1 & \text{if } n \leq 1 \end{cases}$$

Your grade will depend on the quality of the bounds you provide (that is, showing that $T(n) \in \Omega(1)$ and $T(n) \in O(100^n)$, while true, will not give you many marks).

Solution : Here are the first three levels of the recursion tree (some of the repeated nodes have been removed, and replaced by an indication that there are "3 of these" since the tree would not have fit on the page otherwise).



As we can see, the children of a node do the same amount of work, together, as their parent. For instance, on level 2, the total amount of work done is $3 \cdot 4^{n-1} + 4 \cdot 4^{n-2} = 3 \cdot 4^{n-1} + 4^{n-1} = 4 \cdot 4^{n-1} = 4^n$. Hence the work done on every level of the tree is exactly 4^n, at least up to the level containing the leaf closest to the root (after which the amount of work done starts decreasing). The tree contains n levels, since the path along which the size of the subproblems decreases slowest is that where the size goes down by 1 when we go from one level to the next. Hence the total amount of work done is at most $n4^n$.

For the lower bound, notice that the first leaf occurs at level $n/2$, and that every level up to that one does exactly 4^n work, and so the total amount of work done is at least $n4^n/2$. Therefore $T(n) \in \Theta(n4^n)$.

[11] 5. You are asked to design a data structure that supports the following two operations on a set S:

- `Insert(`S`,` x`)`: adds the element x to S.
- `Delete-Larger-Half(`S`)`: deletes the largest $\lceil |S|/2 \rceil$ elements of S.

After pondering the problem for a few hours, you decide to store the elements of S in an `ArrayList`, since it supports $\Theta(1)$ insert operations.

Note that parts (a) and (b) are independent, and you can do them in either order.

[4] a. Explain how to implement `Delete-Larger-Half(`S`)` in $\Theta(|S|)$ worst-case time. Hint: your first step should be to use an algorithm from one of the assignments.

Solution : Note that the question did *not* state that the elements are guaranteed to be distinct, and hence we need to be careful when deleting elements. Let S_{orig} denote the set S before we delete half its elements. The algorithm proceeds in three steps:

i. Find the median m of S_{orig} using `DeterministicSelect`.
ii. Delete all of the elements of S_{orig} that are larger than m.
iii. Delete enough elements equal to m so the cardinality of S becomes $\lfloor |S_{orig}|/2 \rfloor$

[7] b. Prove that the worst-case running time of a sequence of n operations, starting with an empty set, is in $O(n)$. Hint: use $\Phi(S_i) = u|S_i|$ for some constant u (that you can choose later on).

Solution : Following the hint, let $\Phi(S_i) = u|S_i|$. Clearly $\Phi(S_0) = 0$ since S is initially empty, and $\Phi(S_i) \geq 0$ since $|S_i| \geq 0$. Now we need to compute the amortized costs of the two operations.

- `Insert`: $cost_{real}$(*Insert*) = 1 and the potential difference $\Phi(S_{i+1}) - \Phi(S_i)$ is u. Hence $cost_{am}$(*Insert*) $= 1 + u \in \Theta(1)$.
- `Delete-Larger-Half`: $cost_{real}$(*Delete-Larger-Half*) is in $\Theta(|S|) \leq c|S|$. The potential difference $\Phi(S_{i+1}) - \Phi(S_i) = -u\lceil |S|/2 \rceil \leq -(u/2)|S|$. Therefore, as long as $u \geq 2c$, $cost_{am}$(*Delete-Larger-Half*) $\leq c|S| - (u/2)|S| \leq c|S| - c|S| = 0$.

Thus a sequence of n operations takes time at most $\sum_{i=1}^{n} cost_{am}(op_i) \leq \sum_{i=1}^{n} \Theta(1) \in \Theta(n)$.