# CSCI E-124 Data Structures and Algorithms — Spring 2015

## Problem Set 6

Due: 11:59pm, Wednesday, April 1

See homework submission instructions at
`http://sites.fas.harvard.edu/~cs124/e124/problem_sets.html`

**Problem 5 is worth 40% of this problem set, and problems 1-4 constitute the remaining 60%.**

Upon completing this problem set, if you have roughly two minutes to spare then please fill out this survey: `http://goo.gl/forms/yq46D6jckP`

# 1 Problem 1

In class we analyzed QuickSelect using the notion of "good" recursive calls: a recursive call was good if it reduced the number of working elements by a factor of at least 3/4. Instead, we could adopt an analysis for QuickSelect similar to that for QuickSort. Suppose we call QuickSelect$(A, k)$, to find the $k$th smallest element in an array $A$ of size $n$. Assume the elements of $A$ are distinct. Note the running time of QuickSelect is proportional to the number of comparisons. Thus if we let $X_{i,j}$ be a random variable which is 1 if $i$th smallest item and $j$th smallest item are ever compared throughout the execution of QuickSelect$(A, k)$, then the running time is proportional to $\sum_{i<j} X_{i,j}$.

(a) For $i < j$, give an exact expression for $\mathbb{E}\, X_{i,j}$ in terms of $i, j, k, n$. You may need to employ case analysis.

(b) Using (a), show that $\mathbb{E} \sum_{i<j} X_{i,j} = O(n)$.

# 2 Problem 2

Consider the following implementation of a binary search tree (BST). When searching, we search based on key as in a normal BST. When inserting an element with key $k$, we assign the element a uniformly random ID in $[0, 1]$. We first insert the element into the BST based on its key as normal. We then *rotate* the node it lands in upward to preserve the invariant that, when looking at node IDs instead of node keys, the tree should be a min heap.

Suppose the keys in the BST at some point are $k_1 < k_2 < \ldots < k_n$. When searching for key $k_r$, note that we touch the node with key $k_i$ if and only if it is an ancestor of the node with $k_r$ in the BST. Using this fact, show that the expected time to perform a query is $O(\log n)$. Also show that the expected time to insert a new key into the data structure is also $O(\log n)$.

## 3 Problem 3

Suppose we use a hash table with chaining, and we hash $n$ distinct items in the set $\{1, \ldots, U\}$ into a table of size $m = n$. Furthermore suppose that the hash function is *totally random*. We saw in class that for any item, its expected query time is $O(1)$. Show that if the hash function is totally random, then in addition we have the stronger property that with probability 99%, no linked list in the entire table has size larger than $O(\log n / \log \log n)$. You can use the following two facts without proof:

(1) for all $1 \le k \le n$, $(n/k)^k \le \binom{n}{k} \le (en/k)^k$, and

(2) the "union bound": for any set of probabilistic events $\{\mathcal{E}_i\}_{i=1}^t$,

$$\mathbb{P}(\mathcal{E}_1 \text{ or } \mathcal{E}_2 \text{ or } \ldots \text{ or } \mathcal{E}_t) \le \sum_{i=1}^{t} \mathbb{P}(\mathcal{E}_i).$$

## 4 Problem 4

We will use the following scheme to hash $n \le m/2$ items into a hash table $A$ of size $m$. Each item $x$ with key $k$ is associated with a permutation $\pi_k$ chosen uniformly at random from all $m!$ permutations of the elements of $\{1, \ldots, m\}$. When we insert $x$, we first try putting it in $A[\pi_k(1)]$, then $A[\pi_k(2)]$ if that cell was already occupied, etc. A search is also done by probing locations in that order.

(a) Show that for any of the $n$ insertions, the probability the insertion makes more than $t$ probes is at most $1/2^t$.

(b) Let $X_i$ be the number of probes performed when inserting item $i$. Show that $\mathbb{E} \max_{1 \le i \le n} X_i = O(\log n)$. It may be helpful to use the general fact (which you can use without justification) that if $Z$ is a random variable taking non-negative integer values, then $\mathbb{E}\, Z = \sum_{z=0}^{\infty} \mathbb{P}(Z > z)$.

## 5 Problem 5

Solve "Problem A - Primes" on the programming server; see the "Problem Sets" part of the course web page for the link. **Hint:** You may find this list of primes helpful: `http://primes.utm.edu/lists/small/millions/`.