COMPSCI330 Design and Analysis of Algorithms
Assignment 7

Due Date: Wednesday, April 15, 2020

Guidelines

- **Describing Algorithms** If you are asked to provide an algorithm, you should clearly define each step of the procedure, establish its correctness, and then analyze its overall running time. There is no need to write pseudo-code; an unambiguous description of your algorithm in plain text will suffice. If the running time of your algorithm is worse than the suggested running time, you might receive partial credits.
- **Typesetting and Submission** Please submit the problems to GradeScope. You will be asked to **label your solution for individual problems**. Failing to label your solution can cost you 5% of the total points (3 points out of 60 for this homework).
- LaTeX is preferred, but answers typed with other software and converted to pdf is also accepted. Please make sure you submit to the correct problem, and your file can be opened by standard pdf reader. **Handwritten answers or pdf files that cannot be opened will not be graded. (Exceptions apply to students who don't have regular access to computers.)**
- **Timing** Please start early. The problems are difficult and they can take hours to solve. The time you spend on finding the proof can be much longer than the time to write. If you need more time for your homework please use this form and submit a STINF.
- **Collaboration Policy** Please check this page for the collaboration policy. You are **not** allowed to discuss homework problems in groups of more than 3 students. **Failure to adhere to these guidelines will be promptly reported to the relevant authority without exception.**

Problem 1 (Basic Probabilities)**.** (15 points)
(a) (5 points) Suppose you toss a fair coin until you get heads, what is the probability that you toss the coin at least t times?
(b) (5 points) In the setting of (a), what is the expected number of coin tosses you will make?
(c) (5 points) Now every time you will throw three fair dice (with 6 faces each), and you will stop only if all three dice roll 6. What is the expected number of times you need to try?

Problem 2 (Randomized Binary Search)**.** (20 points) Consider the following version of binary search:

```
Input: Sorted array a[1..n] in increasing order, number x
Output: True if x is in the array a[], False otherwise.
Random_Search(a[], x)
   n = length(a[])
   IF n == 0 THEN return False
   IF n == 1 THEN return (a[1] == x)
   Let k be a uniform random number between 1..n
   IF x == a[k] THEN return True
   ELSE IF x < a[k] THEN return Random_Search(a[1..k-1],x)
   ELSE return Random_Search(a[k+1..n], x)
```

Let $T(n)$ be the worst-case expected running time of this algorithm, prove by induction that $T(n) \le C \log_2 n$ for some suitable constant C (your constant does not need to be optimal, as long as it is a constant it is OK). The base case is $T(1) = T(0) = 0$.

(Hint: The worst-case happens when you always recurse on the larger side.)

Problem 3 (Multi-Hashing)**.** (25 points) Another way to avoid conflicts in Hashing is to use multiple Hash tables. Suppose there are k hash tables of size m each (we will use $a_1, a_2, ..., a_k$ to denote these tables), and every hash table has an independent hash function $f_i (i = 1, 2, ..., k)$, each f_i is chosen from a uniform hashing family.

The way this Hashing algorithm works is that: initially all hash tables are set to 0. To insert number x, increase $a_1[f_1(x)], a_2[f_2(x)], \cdots, a_k[f_k(x)]$ by 1. To check whether number x is already in the Hash table, check all values of $a_1[f_1(x)], a_2[f_2(x)], \cdots, a_k[f_k(x)]$, return *False* if any one of these k numbers is equal to 0, otherwise return *True*.

Suppose there are currently n numbers in the hash table, and we query a number x that is not in the table.
(a) (10 points) Show that the probability that for an fixed i, the probability that $a_i[f_i(x)] > 0$ is at most n/m.
(b) (5 points) Prove that the probability that the algorithm makes a mistake (reporting x as in the array) is at most $(n/m)^k$.
(c) (10 points) Suppose there are a total of T queries and the number of elements in the Hash table is never larger than n, show that when $m \ge 2n$ and $k \ge \log_2(T/\epsilon)$, the probability that the algorithm makes at least one mistake is at most ϵ.