Homework Nine, for Fri 12/4

CSE 101

Prepare a PDF file in which your solution to each of the following problems (1–10) begins on a fresh page. Upload the file to Gradescope, using your campus email address as login. The deadline is noon on Friday.

These problems cover the following skills and concepts:

- Notion of a randomized algorithm
- Reducing failure probability through repetition
- Using a balls-and-bins analogy for understanding random processes
- Randomized schemes for economically storing and accessing data

---

1. When we analyzed our randomized algorithm for median-finding (in Chapter 2), we used the following useful fact:

   > Suppose each time you pull the arm of a slot machine, you have a probability p of winning. Then the expected number of times you have to pull the arm in order to win is $1/p$.

   Using this fact, answer the following questions.

   (a) What is the expected number of times you need to roll a die in order to get a six?

   (b) A pond contains 50 catfish and 950 eels. You repeatedly pull a random fish out of the pond and then throw it back in. What is the expected number of times you need to do this until you get a catfish?

   (c) A computer repeatedly generates random integers in the range 0 to 99 until it gets a multiple of 10. What is the expected number of iterations?

2. Suppose that you wish to generate coin tosses that are *fair*—that is, heads (`H`) and tails (`T`) appear with probability 1/2 each—but the only coin available to you is biased: it has heads probability p for some value p that you don't know.

   Here is how you decide to generate a fair coin toss:

   ```
   Repeat:
      Toss the biased coin twice
      If you get HT: halt and output H
      If you get TH: halt and output T
   ```

   (a) Explain why this generates a fair coin toss.

   (b) What is the expected number of times that this procedure tosses the biased coin? Your answer should be in terms of p.

3. The most popular method for testing whether a number is prime is a randomized algorithm $\mathcal{A}$ with the following behavior:

   - If x is prime, $\mathcal{A}(x)$ always says "prime"
   - If x is not prime, $\mathcal{A}(x)$ says "not prime" with probability 1/2; otherwise it says "prime"

   You want to find out whether a certain x is prime or not. However, you want the probability of getting the wrong answer to be at most $1/2^{100}$. How would you do this?

4. Suppose you have access to two different algorithms for testing whether a number is prime. The first is a randomized algorithm $\mathcal{A}$ that runs in time $T(n)$ and has the following behavior:

- If x is prime, $\mathcal{A}(x)$ always says "prime"
- If x is not prime, $\mathcal{A}(x)$ says "not prime" with probability $1/2$; otherwise it says "prime"

Algorithm $\mathcal{B}$ is deterministic, runs in time $100T(n)$, and always returns the correct answer.

You wish to create an algorithm that is always correct and is as efficient as possible.

(a) Specify your algorithm.

(b) What is the expected running time of your algorithm when the input is prime?

(c) When the input is not prime?

5. One morning in San Diego, there are n cars that need to be repaired, and n repair shops, each of which works on one car at a time and takes an hour to do so. Suppose that each of the n car owners chooses a repair shop at random. How many hours will it be, roughly, before all the cars are fixed?

6. A country of n people introduces an innovative method for assigning each person a biometric identification number based on his or her DNA sequence:

   - Let U be the space of all DNA sequences.
   - A random hash function $h : U \to \{0,1\}^m$ is chosen.
   - A person x with DNA sequence $d(x)$ is assigned the m-bit ID number $h(d(x))$.

   We would like all n ID numbers to be different, with probability at least $1/2$. How large should m be?

7. *Hash tables with open addressing.* In this problem, we'll look at a way of designing hash tables that avoids any need for chaining (that is, linked lists). The items to be stored are, as usual, from some universe U.

   - A table $T[1 \cdots m]$ is created in which each position is either empty or holds one element of U.
   - A random hash function $h : U \times \{0, 1, 2, \ldots\} \to [m]$ is picked.
   - To insert a new item $x \in U$, we first try location $T[h(x, 0)]$. If it's already occupied, we try $T[h(x, 1)]$, and if that's full, then $T[h(x, 2)]$, and so on.
   - To look up an item $x \in U$, we try $T[h(x, 0)], T[h(x, 1)], T[h(x, 2)], \ldots$, until we either find x or we encounter an empty slot (in which case x is not present).

   In what follows, suppose we wish to store n items and we choose table size $m = 2n$.

   (a) Pick any of these items, x. Show that when inserting x, the probability that we need more than k probes (that is, we need to go beyond $T[h(x, k)]$) is at most $1/2^k$.

   (b) Show that with probability at least $1 - 1/n$, none of the insertions requires more than $2 \log n$ probes.

8. Suppose you have a linked list of n elements *in sorted order*. Here's the obvious way to look up an element x, given a pointer to the first node in the list:

```
Function lookup(L, x)
if L is NULL: return NULL
if value(L) = x:  return L
if value(L) > x:  return NULL
return lookup(next(L), x)
```

   Notation: given the pointer P to a node, value(P) and next(P) are the value in the node and the pointer to the next node.

   (a) Suppose an element of L is chosen at random. What is (roughly) the expected time (in terms of n) to look up this element? What is the worst-case lookup time?

(b) One way to get faster lookups is to add a second pointer jump(P) that points to the node k steps down the list from P (that is, following a "jump" pointer is like following k "next" pointers). The procedure above is then altered by adding the following line just before the final return statement:

`if value(jump(`L`)) ≤ `x`: return lookup(jump(`L`), `x`)`

With this addition, the data structure is called a *skip list*. Now suppose an element of L is chosen at random. What is, roughly, the expected time to look it up, as a function of n and k? What is the worst-case lookup time?

(c) What is (again, roughly) the best choice of k?

9. *Estimating the size of a set from its Bloom filter.* You have stored some items in a Bloom filter $T[1 \cdots m]$, using k hash functions. However, you have forgotten how many items you stored. Explain how you might estimate this number, given the values of m and k and the table $T[\cdot]$.

10. Barbara Smith is interviewing candidates to be her secretary, one at a time. After each interview she is able to determine the true competence level of the candidate (which can be thought of as some positive real number—and assume all candidates have distinct scores). However, she needs to make a spot decision whether or not to hire a candidate, before interviewing the remaining ones. If the candidates appear *in random order*, what is a good hiring strategy for Barbara?

    Suppose there are n candidates. Barbara decides to interview the first r candidates, noting down their scores but not hiring any of them. Let s be the largest score she records during this time. Then, she starts interviewing the remaining $n-r$ candidates, and hires the first one who scores more than s (if none of them do, then she simply picks the last candidate).

    (a) Show that the probability that Barbara ends up with the best secretary is at least $r(n-r)/n^2$.

    (b) What is a good setting for r, and what is the probability of success in this case?