University of Washington
Department of Computer Science and Engineering
CSEP 521, Winter 2021

January 9, 2021

Homework 1, Due Thursday, January 14, 2020

Turnin instructions: Electronic submission on gradescope using the CSEP 521 gradescope site. Submit the assignment as a PDF, with separate pages for different numbered problems. Problems consisting of multiple parts (e.g., 2a, 2b) can be submitted on the same page.
You should provide justifications for your answers and explain the key ideas in reaching your conclusions.

Problem 1 (10 points):

Suppose you are given a coin, which has probability of heads of p and probability of tails of q, with $q = 1 - p$ and $p \neq q$. How can you use this coin to generate an unbiased coin flip, where the probability of heads is $\frac{1}{2}$ and the probability of tails is $\frac{1}{2}$. What is the expected number of coin flips that you need to generate an unbiased coin flip?

Problem 2 (10 points):

Consider the following algorithm for generating permutations (which is fairly similar to the one presented in class). Show that this algorithm does *not* generate perfectly uniform random permutations. In other words, for some n, there is a permutation on n items that is generated with probability p, where $p \neq \frac{1}{n!}$. (Note that the main difference from the lecture algorithm is in the range of indices for swapping the elements.)

```
public static int[] Permutation(int n, Random rand) {
    int[] arr = IdentityPermutation(n);

    for (int i = 1; i < n; i++) {
        int j = rand.Next(0, n);    // Random number in range 0..n-1
        int temp = arr[i];          // Swap arr[i], arr[j]
        arr[i] = arr[j];
        arr[j] = temp;
    }

    return arr;
}
```

Problem 3 (10 points):

Suppose you have a source of uniform random bits and you want to generate random integer in the range $1, \ldots, n$, and you want to use as few random bits as possible. If $n = 2^k$ you can generate a random number by using k random bits. What if n is not a power of two? Hint: for $n = 3$, you can start with two random bits. On 00, 01 and 10, you can return 1, 2, and 3 respectively, and on 11, you start over. Let E be the expected number of bits to generate an element from $\{1, 2, 3\}$. Observe that $E = 2 + \frac{1}{4}E$, so $\frac{3}{4}E = 2$, and $E = \frac{8}{3}$.

a) Give a scheme to generate random numbers in the range $1, \ldots, 6$ that is as efficient as possible (in terms of the random bits). What is the expected number of bits you use?

b) Give a scheme to generate random numbers in the range $1, \ldots, 9$ that is as efficient as possible (in terms of the random bits). What is the expected number of bits you use?

(Note: this problem may be harder than it seems - so you should come up with as good a solution as you can, and justify the expected number of random bits.)

Programming Problem 4 (20 points):

In comparing selection algorithms, it is natural to count the number of comparisons done between elements of the input. For deterministic algorithms it is known that $2n$ comparisons are required to find the median, and there is an algorithm that can find the median with $3n$ comparisons[1].

Implement the randomized selection algorithm from class, where the pivot is chosen at random from the array. What is the expected number of comparisons that you do between data elements to find the median?

Intuitively, it seems like a good idea to pick a pivot close to the median, so it is often suggested to use a median-of-three or median-of-five heuristic where three (or five) random elements are picked, and the median of those elements are used as the pivot.

Determine the expected number of comparisons for the medium-of-three and median of five heuristics. (You can count the comparisons for finding the medians of the samples separately).

Experimentally determine the expected number of comparisons to compute the median. You should do this for a range of fairly large values of n, possibly between $n = 1,000,000$ and $n = 10,000,000$, and a number of samples for each value on n.

Submit your results in a table, with a brief summary of what you found. You should also submit your source code, but we are not going to look at it very carefully, and you are not going to be graded on the quality of your code.

You can test your algorithm using the identity permutation, since with randomization, the algorithm performs that same on all inputs.

[1]Actually, the best known algorithms is something like $2.96n$ comparisons.