CS240E W25	**Tutorial 3**	Jan. 24

Overview

- Expected Run Time with Recursion
- Average-case Run Time
- Probability and Expected Value
- Partition
- Quick-select

Problems

Q1. Expected Runtime Analysis.

Give the best-case, worst-case, and expected running time for the infamous Bogosort algorithm in terms of the size n of the array. You can assume that the shuffle function takes $O(n)$ time and produces each permutation equally likely. You can also assume that the array contains no duplicates.

```
bogosort(A):
        shuffle(A);
        if (A is sorted) {
                return A;
        } else {
                bogosort(A);
        }
```

Q2. Average-case Analysis.

Let A be an array containing each of the numbers $\{1, \ldots, n\}$ exactly once, in some order. Analyze this algorithm to determine a tight bound on the average number of ?s that are printed. You may assume n is divisible by 2.

```
mystery(A, n):
        count = 1;
        for (i from 1 to n-1) {
                if (A[i] is divisible by A[0]) { count++; }
        }
        for (i = 1 to count) {
                print('?');
        }
```

Q3. Probabilistic Counting.

With a normal b-bit counter, we can only count up to $2^b - 1$. But with *probabilistic counting* we can count to larger values at the cost of precision.

We let a counter reading of i represent a count of v_i, for $0 \leq i \leq 2^b - 1$. Initially the counter reads 0, indicating the count of $v_0 = 0$.

The operation *increment* works on a probabilistic counter with reading i in a randomized way:

1. If $i < 2^b - 1$, increase counter reading with probability
$$\frac{1}{v_{i+1} - v_i},$$
and leave the counter unchanged otherwise.

2. If $i = 2^b - 1$, report overflow.

Note that if we select $v_i = 1$, then the counter is an ordinary deterministic counter. More interesting situations arise if $v_i = 100i$, $v_i = 2^i$, or $v_i = i$-th Fibonacci number. For instance, if $v_i = 2^i$ then a reading of $i = 101_2 = 5_{10}$ represents a count of "approximately $2^5 = 32$".

Assuming that an overflow does not occur, show that the expected value represented by the counter after n *increment* operations is n.

Q4. Partition and quick-select.

Consider the following variation of the `partition` routine from class, which we'll call `partition++`:

1. Take the pivot-index $n - 1$ (so the pivot-value is $v = A[n-1]$).

2. Rearrange A as explained in class (so that everything less than v is to its left and everything greater than v is to its right) and compute the pivot index i.

3. If $i \neq 0$, then further rearrange the left part of A such that $A[i-1]$ is the predecessor of v (i.e., $A[i-1]$ and $A[i]$ would be consecutive in sorted order).

(a) Explain how `partition++` can be implemented with $\Theta(n)$ key comparisons.

(b) For parts (b)-(d), consider running `quick-select`$(A, 0)$ using `partition++`. If $i = n - 1$ in the first round, what is the run-time in terms of n?

(c) If $i > 0$, what is the run time in terms of i and/or n?

(d) Show that the average-case run time (considering all possibilities for the initial pivot index i) is in $\Theta(n^2)$.