Problem Set 1

This problem set is due **at 10:00pm** on **Wednesday, February 12, 2020**.

Please make note of the following instructions:

- This assignment, like later assignments, consists of *exercises* and *problems*. **Hand in solutions to the problems only.** However, we strongly advise that you work out the exercises for yourself, since they will help you learn the course material. You are responsible for the material they cover.
- Remember that the problem set must be submitted on Gradescope. If you haven't done so already, please signup for 6.046 Spring 2020 on Gradescope, with the entry code MNEBKP, to submit this assignment.
- We require that the solution to the problems is submitted as a PDF file, **typeset on LaTeX**, using the template available in the course materials. Each submitted solution should start with your name, the course number, the problem number, your recitation section, the date, and the names of any students with whom you collaborated.
- You will often be called upon to "give an algorithm" to solve a certain problem. Your write-up should take the form of a short essay. A topic paragraph should summarize the problem you are solving and what your results are. The body of your essay should provide the following:
  1. A description of the algorithm in English and, if helpful, pseudocode.
  2. A proof (or indication) of the correctness of the algorithm.
  3. An analysis of the asymptotic running time behavior of the algorithm.
  4. Optionally, you may find it useful to include a worked example or diagram to show more precisely how your algorithm works.

EXERCISES (NOT TO BE TURNED IN)

Asymptotic Analysis, Recursion, and Master Theorem

- Do Exercise 4.3-7 in CLRS on page 87.
- Do Exercise 4.3-9 in CLRS on page 88.

Randomized Algorithms

- Do Exercise 7.4-4 in CLRS on page 184.
- Do Exercise 9.2-4 in CLRS on page 220.

Problem 1-1. Recurrences and Asymptotics [30 points]

Let $T(n)$ be the time complexity of an algorithm to solve a problem of size n. Assume $T(n)$ is $O(1)$ for any n less than 3. Solve the following recurrence relations for $T(n)$.

(a) [3 points] $T(n) = 8T\left(\frac{n}{2}\right) + n^2$

(b) [3 points] $T(n) = 8T(\sqrt[4]{n}) + \log^4 n$.

(c) [5 points] $T(n) = T(n-1) + 3n$.

(d) [5 points] $T(n) = T(n/7) + 2T(n/5) + O(n)$.

(e) [14 points] Consider the following functions. Within each group, sort the functions in asymptotically increasing order, showing strict orderings as necessary. For example, we may sort $n^3, n, 2n$ as $2n = O(n) = o(n^3)$.

1. [4 points] $\log n$, $\sqrt[5]{n}$, $\log_9 n$, $\log n^5$, $\log\log n^{20}$, $\log^9 n$.
2. [5 points] n^7, $n\log n^2$, $n^2 \log\log n$, 3^n, $\log(n!)$.
3. [5 points] $(\log n)^n$, $(\log n)^{n-1}$, 5^n, $n^{\log n}$.

Problem 1-2. Probability practice: Dice Rolling [20 points]

A pair of dice is rolled until a sum of either 5 or 7 appears. Find the probability that a 5 occurs first.

(a) [2 points] Let E_n denote the event that a 5 occurs on the nth roll and no 5 or 7 occurs on any of the first $(n-1)$ rolls. Compute $P(E_n)$.

(b) [8 points] Compute $\sum_{n=1}^{\infty} P(E_n)$ and argue that it is the desired probability. **HINT:** $\sum_{n=0}^{\infty} r^n = \frac{1}{1-r}$ for $|r| < 1$.

(c) [10 points] Now, suppose that we roll a standard fair die 100 times. Let X be the sum of the numbers that appear over the 100 rolls. Find an upper bound for the following probability $P[|X - 350| \geq 50]$.

1. [2 points] Let X_i be the number on the face of the die for roll i. Compute the expected value of X_i.

2. [2 points] Compute $\mathbf{E}[X]$ where X is the sum of the dice rolls.
3. [5 points] Compute the variance of X where X is again the sum of the dice rolls.
   Hint: $\sum\limits_{j=1}^{n} j^2 = \frac{n(n+1)(2n+1)}{6}$
4. [1 points] Find an upper bound for $P[|X - 350| \geq 50]$.

Problem 1-3. Mostly Sorting [50 points] In this problem we consider the task of determining whether an array of numbers is sorted or not, without examining all elements in the array. In order to do this, your algorithm is not required to give the correct answer if the array is *mostly sorted* but not completely sorted.

Precisely, given a array A of n elements, we say that S is *mostly sorted* if it contains a sorted subsequence of length at least $\frac{9}{10}n$. The sorted subsequence is not necessarily contiguous. That is, you can remove some $k \leq \frac{1}{10}n$ elements from A such that the remaining elements are in sorted order.

If the input array is completely sorted, your algorithm must return "SORTED". If the input array is mostly sorted, but not completely sorted, your algorithm may output anything. Otherwise, if the input array is not mostly sorted, your algorithm must return "UNSORTED".

Assume that binary comparisons on any two elements can be done in constant time.

(a) [10 points] Prove that any deterministic algorithm satisfying the above must have a runtime that is $\Omega(n)$.

(b) [10 points] Now let's add randomness. One property of a sorted array is that any consecutive three elements in the array are also sorted. As a first try, our algorithm will repeatedly sample consecutive triples of elements and check whether the triple is sorted. The algorithm will output 'SORTED' if and only if all sampled triples are sorted.

Let the number of sampled triples be k. Show that we must have k be $\Omega(n)$ for the probability of success of this algorithm to exceed $\frac{1}{2}$.

Hint: It suffices to demonstrate a single "adversarial" input.

Hint: $(1 - \frac{1}{n})^n \approx e^{-1}$. You may also find the inequality $1 - x \leq e^{-x}$ useful.

(c) [10 points] Another property of a sorted array is that binary search can be used to find values in the array. Before we can use this property as the basis of an algorithm, we should investigate the behavior of binary search on an unsorted array.

For simplicity, assume that input arrays will not contain any repeated numbers.

Fixing an input array A of length n, we define binary search as:

```
BINARY-SEARCH($A, x$, left, right)
    if right − left == 1
        return left
        else median ← $\lfloor$(left + right)/2$\rfloor$
        if $x < A$[median]
            return BINARY-SEARCH($A, x$, left, median)
        else
            return BINARY-SEARCH($A, x$, median, right)
```

Prove that for any two distinct elements x_1 and x_2 in A, if
BINARY-SEARCH$(A, x_1, 0, n) <$ BINARY-SEARCH$(A, x_2, 0, n)$ then $x_1 < x_2$, *even if A is unsorted*.

Hint: Consider the point at which the execution of BINARY-SEARCH$(A, x_1, 0, n)$ diverges from that of BINARY-SEARCH$(A, x_2, 0, n)$.

(d) [20 points] Design a randomized algorithm for our problem that succeeds with probability at least $\frac{3}{4}$. The runtime of your algorithm should be $O(\log n)$.

Hint: Use BINARY-SEARCH as a subroutine.