15-451 Algorithms, Spring 2017
Recitation #1 Worksheet

-IV. Asymptotic analysis. [1] For each pair $\langle f, g \rangle$ of functions below, list which of the following are true: $f(n) = o(g(n))$, $f(n) = \Theta(g(n))$, or $g(n) = o(f(n))$.

1. $f(n) = \ln(n), g(n) = \log_{10}(n)$. **Solution:** $f(n) = \Theta(g(n))$

2. $f(n) = n^{1.5}, g(n) = n \log^4(n)$. **Solution:** $g(n) = o(f(n))$

3. $f(n) = 2^{2n}, g(n) = 2^n$. **Solution:** $g(n) = o(f(n))$

4. $f(n) = n^{0.001}, g(n) = (\log_2 n)^{100}$. **Solution:** $g(n) = o(f(n))$

-III. Solving recurrences by unrolling. Solve the following recurrences in Θ notation by unrolling (assume all base cases have $T(1) = 0$):

1. $T(n) = 1 + 2T(n-1)$. **Solution:** $T(n) = \Theta(2^n)$

2. $T(n) = n + T(n/2)$. **Solution:** $T(n) = \Theta(n)$

3. $T(n) = n + 2T(n/2)$. **Solution:** $T(n) = \Theta(n \log n)$

[1] The Romans did not have negative numbers, so we're playing fast and loose with notation here.

-II. Solving by guess and inductive proof. Consider the recurrence $T(n) = 2T(n/2) + \log_2(n)$ with base case $T(1) = 0$. This is sometimes called the "heapsort recurrence" and can be viewed as the sum of the heights of all nodes in a completely balanced binary tree of n leaves, where leaves are at height 0, their parents are at height 1, etc.

(a) Show that $T(n) = O(n \log n)$ and $T(n) = \Omega(n)$.

Solution: To prove that $T(n) = O(n \log n)$, upper bound each of the $2n - 1$ nodes by $O(\log n)$. To prove that $T(n) = \Omega(n)$, lower bound each of the $n - 1$ non-leaf node by 1.

But what's the truth? One way to solve it is to try small cases ($n = 1, 2, 4, 8, 16, 32$) and notice that this seems to fit the pattern $T(n) = 2n - \lg(n) - 2$.

(b) Prove by induction that $T(n) = 2n - \lg(n) - 2$ indeed is the solution.

Solution: $T(1) = 2 \cdot 1 - \lg(1) - 2 = 0$.
$T(n) = 2 \cdot T(n/2) + \lg(n) = 2(2 \cdot n/2 - \lg(n/2) - 2) + \lg n = 2n - \lg n - 2$.

(c) Can you think of other ways to prove $T(n) = \Theta(n)$? (E.g., you could unroll the recursion, or give a "combinatorial proof".)

Solution: We could sum up the number of nodes with with height ≥ 1, nodes with height ≥ 2, nodes with height ≥ 3, etc. The sum is $(n-1) + (n/2-1) + (n/4-1) + \ldots = \Theta(n)$

-I. Using the master formula. Solve the following recurrences (usual base case) using the master formula.

1. $T(n) = 3T(n/2) + n^2$. **Solution:** $T(n) = \Theta(n^2)$

2. $T(n) = 4T(n/2) + n^2$. **Solution:** $T(n) = \Theta(n^2 \log n)$

3. $T(n) = 5T(n/2) + n^2$. **Solution:** $T(n) = \Theta(n^{\log_2 5})$

I. Probability Facts. Let's follow the convention of using uppercase letters X, Y, Z to denote random variables (which we often abbreviate to *r.v.*s).

Independence: Random variables X, Y are independent if for any values a, b, we have

$$\mathbf{Pr}[X = a, Y = b] = \mathbf{Pr}[X = a] \cdot \mathbf{Pr}[Y = b].$$

1. Consider flipping two fair coins. Let $X \in \{0, 1\}$ be the outcome of the first coin (where we think of heads as 1 and tails as 0) and let $Y \in \{0, 1\}$ be the outcome of the second coin. Let $Z = X + Y \bmod 2$. Are X and Y independent? What about X and Z?

   Solution: X and Y are independent by definition. X and Z are also independent, despite us defining Z in terms of X. To prove this, just calculate $\mathbf{Pr}[X = a, Z = c]$ for each setting of a, c and show that it equals $\mathbf{Pr}[X = a] \cdot \mathbf{Pr}[Z = c]$.

Linearity of expectation: Given any two r.v.s X, Y,

$$\mathbf{E}[X + Y] = \mathbf{E}[X] + \mathbf{E}[Y].$$

2. Suppose we take n books numbered 1 to n, and place them on a shelf in a (uniformly) *random* order. What is the expected number of books that end up in their correct location? ("correct" = location it would be in if they were sorted).

   Solution: Let X_i be the "indicator" r.v. which is 1 when the i^{th} book is in the correct location, and 0 otherwise. Note that $\mathbf{E}[X_i] = \mathbf{Pr}[X_i = 1] =$ the probability that the i^{th} book is in the correct location, which is $1/n$. Now $X = \sum_{i=1}^{n} X_i$ is the number of books in the correct location, so $\mathbf{E}[X] = \mathbf{E}[\sum_i X_i] = \sum_i \mathbf{E}[X_i] = \sum_i (1/n) = n \cdot 1/n = 1$.

Markov's inequality: given a *non-negative random variable* X with mean/expectation $\mu = \mathbf{E}[X]$,

$$\mathbf{Pr}[X > c\,\mu] \leq \frac{1}{c}.$$

(Exercise: give a proof!)

3. Show that if X is allowed to take negative values then the above inequality is no longer true.

   Solution: Suppose $X = 1$ with probability 0.51 and -1 with probability 0.49. Then $\mu = \mathbf{E}[X] = 0.02$, but $\mathbf{Pr}[X > 5\mu] = \mathbf{Pr}[X > 0.1] = 1/2$ which is more than $1/5$.

II. Sorting by Swaps. Imagine a sorting algorithm that somehow picks two elements that are out of order with respect to each other (not necessarily adjacent, as in `insertion-sort`) and swaps them. *Question: can we argue that such a procedure (no matter how stupidly*

it picks the elements to swap so long as they were out of order) has to terminate (i.e., the number of swaps it will perform is bounded)?

To do this, one good way is to find a potential function: a finite, non-negative quantity that is strictly reduced with each swap. Any ideas? One quantity that works is the total number of pairs of elements that are out of order w.r.t. each other (this is called the number of *inversions*). When a swap is performed, clearly the inversion of those two is removed, but notice that new inversions may be created. (e.g., in $[5, 1, 2]$, swapping 5 and 2 creates a new inversion between 1 and 2).

4. Show the total number of inversions goes down with each swap.

   Solution: Suppose we swap a with b, where a was in a position before b (so $a > b$). When we do so note that we have removed the inversion between a and b. We may have also created or removed inversions between a or b and some other element c. This would only happen if c is between a and b. In other words, we changed the order from a, c, b to b, c, a. If $a > b > c$, then we have removed an inversion between a and c, but we have created an inversion between c and b. If $c > a > b$, then we have created an inversion between a and b, but we have removed an inversion between c and b. Finally, if $a > c > b$, then we have removed an inversion both between a and c and also between c and b. In all cases, we see that the number of inversions never increases. Since we always remove the inversion between a and b, the total number of inversions goes down with each swap.

III. Breaking Eggs. Say I choose a number between 1 and N. You want to guess it in as few questions as possible (each time you make an incorrect guess, I'll tell you if it is too high or too low). As we know, the strategy that minimizes the worst-case number of guesses is to do binary search: it takes $\lceil \log_2 N \rceil$ guesses.

But, what if you are only allowed **one** guess that is too high? Can you still solve the problem in $o(N)$ guesses? [If you were not allowed **any** guesses that are too high, the only option you would have would be to guess 1,2,3,... in order].

Q: Any ideas? [To restate the problem: you want to figure out how many centimeters high you can drop an egg without it breaking, and you only have two eggs.] (For answer, see footnote.[2])

Solution: See footnote.

Q: Can improve the constant to below 2?

[2]Here's one strategy: guess $1, \sqrt{N}, 2\sqrt{N}, \cdots$ until you get to some $(i+1) \cdot \sqrt{N}$ that's too high. Now the number is between $i\sqrt{N}$ and $(i+1)\sqrt{N}$ so we can finish it off in $\sqrt{N}$ more guesses. #Guesses $\leq 2\sqrt{N}$.

Solution: Yes. Slightly modify the previous strategy. Guess $\sqrt{2}\sqrt{N}$, then $\sqrt{2}\sqrt{N} + (\sqrt{2}\sqrt{N}-1)$, then $\sqrt{2}\sqrt{N}+(\sqrt{2}\sqrt{N}-1)+(\sqrt{2}\sqrt{N}-2)\cdots$ until you get a number that is too high. Now we can finish it off with guesses in between. The constant is improved to $\sqrt{2}$.

Q: Can you show an $\Omega(\sqrt{N})$ lower bound for any deterministic algorithm? (Hint.[3])

Solution: If the algorithm makes a guess g_i at least $\sqrt{N}$ larger than any previous guess, then g_i might be too large and we will have to make at least $\sqrt{N}$ guesses in between.
If the algorithm never makes such a guess, then it will have to make over $\sqrt{N}$ guesses if the answer is N.

Q: Show upper/lower bounds of $\Theta(n^{1/3})$ if you're allowed *two* guesses that are too high?

Solution: Upper bound: Guess $N^{2/3}$, then $2N^{2/3}$, then $3N^{2/3}$, $\cdots$, until you get some $(i+1)N^{2/3}$ that's too high. Then guess $iN^{2/3}+N^{1/3}$, then $iN^{2/3}+2N^{1/3}$, $iN^{2/3}+3N^{1/3}$, $\cdots$, until we get some $iN^{2/3}+(j+1)N^{1/3}$. Finally we can finish it off with $\sqrt{N}$ consecutive guesses This gives a constant of 3.
You can improve this constant by using the technique presented in the second part.

Lower bound: Consider the first guess. If the algorithm never makes the guess at least $N^{2/3}$ larger than any previous guess, then it will have to make over $N^{1/3}$ guesses if the answer is N.
If the algorithm does makes a guess g_i, then g_i might be too large and we have to guess $N^{2/3}$ numbers in between:
Now consider the second guess. If the algorithm never makes the guess at least $N^{1/3}$ larger than any previous guess, then it may have to make over $N^{1/3}$ guesses.
If the algorithm does make a guess g_i, then g_i might be too large and we have to use our third guess and make up to $N^{1/3}$ guesses.

[3]Hint: What if the algorithm makes a guess g_i that is at least $\sqrt{N}$ larger than any previous guess? And what if the algorithm *never* makes such a guess?