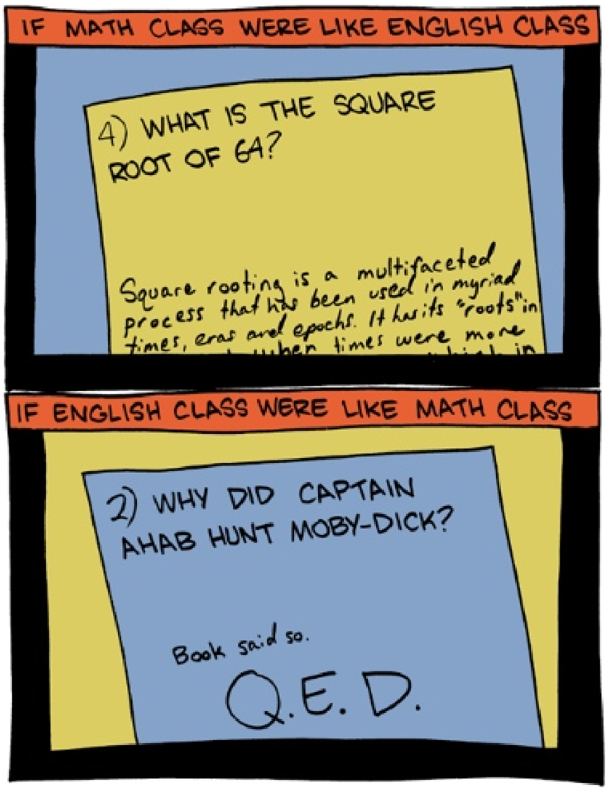
CS140 - Assignment 4

Due: Sunday, February 18 @ 10pm


IF MATH CLASS WERE LIKE ENGLISH CLASS
4) WHAT IS THE SQUARE ROOT OF 64?
Square rooting is a multifaceted process that has been used in myriad times, eras and epochs. It has its "roots" in
IF ENGLISH CLASS WERE LIKE MATH CLASS
2) WHY DID CAPTAIN AHAB HUNT MOBY-DICK?
Book said so.
Q.E.D.

http://www.smbc-comics.com/index.php?db=comics&id=1872

You may (and are encouraged) to work with a partner on this assignment. If you're looking for a partner, post on slack (or email me). Please do this sooner than later!

0. Optional: We're about a third of the way through the course and I wanted to checkin and see how things are going. I've gotten some feedback the group assignments, but I'd also love to get individual feedback on things. If you want (it's anonymous), take 5 minutes and let me know how things are going:

   `https://forms.gle/tfjwU7igYkNxy1gR8`

1. **[12 points]** In a binary search tree, we might also keep track of the total number of nodes in that subtree (including the node itself).

   (a) **[5 points]** Assuming we store this value (e.g. $x.size$) write pseudocode for a function BSTKeyLessThan(T, k) that takes a tree T and a number k and returns the number of values in the tree T that are less than k. For example, if the tree had the number 1 through 9 in it, then BSTKeyLessThan$(T, 5)$ should return 4. You will be graded on how efficient your algorithm is and how effectively you utilize the tree structure. For example, traversing the entire tree and counting all of the values would not get very many points.

(b) [**2 points**] What is the best-case and worst-case running time of your algorithm? State your run-time with respect to either n the number of nodes or h the height of the tree, whichever is more precise.

(c) [**5 points**] Describe an algorithm Median(T) that finds the median element in a binary search tree. You don't have to write psedocode, but if you don't, make sure that you state your algorithm precisely. Hint: you likely will need some sort of helper function. State your run-time with respect to either n the number of nodes or h the height of the tree, whichever is more precise.

2. [**26 points**] Stacks and queues

Assume you're given an implementation of a stack that supports `push` and `pop` in $O(1)$ time. Now you'd like to implement a queue using these stacks.

(a) [10 points] Explain how you can efficiently implement a queue using two of these stacks. ("Efficiently" means in a way that allows you to do the next part of the problem.)

(b) [8 points] Prove that the amortized cost of each `enqueue` and `dequeue` operation is $O(1)$ for your stack-based queue by using the the aggregate amortized analysis technique.

(c) [8 points] Prove the same amortized constant time using the accounting method for amortized analysis.

3. [**10 points**] Your friend (who hasn't taken algorithms) needs your help. They think that they have found another method for sorting data that is $O(n \log n)$ but needs your help proving it. Given an array of n numbers, A, the idea is as follows:

- Call `BuildHeap` (the $O(n)$ version that creates a max-heap) on the array to get a heap.
- Then, you swap the element at the root with the element at location n and decrement the value of n
- You then call `BuildHeap` but with a heap size reduced by one (so that it will ignore everything after the most recently copied element).
- You repeat this process n times.

(a) Is the algorithm correct (i.e. will is always sort an array)? Clearly and succinctly explain your answer.

(b) What is the run-time of this algorithm?

(c) Is there a way we could modify `BuildHeap` to create a new function `HeapFix` and call this in the third step instead of a full call to `BuildHeap` that would result in a better run-time? If no, explain why not. If yes, then briefly explain the algorithm (no need for pseudocode).

4. [**6 points**] Binomial heaps

(a) Insert the following numbers into a min-ordered (i.e., smaller values have higher priority, like in the notes) binomial heap: `7, 1, 3, 8, 2, 4, 10, 9`. Insert the values in that order. Show what the heap looks like after inserting the 2 as well as the final tree after inserting 9. You can visualize the heaps however you want as long as we can understand it.

(b) A binomial heap is a collection of heap-ordered binomial trees. If we count "empty" trees, the number of binomial trees that makes up the heap only increases occasionally. For example, if you have a binomial heap with one item and you insert a second item, you go from one binomial tree to two binomial trees (one empty and one with two things in it). If you insert elements into a binomial heap one at a time, for what values of n will the number of binomial trees in the heap increase (including "empty" trees)?