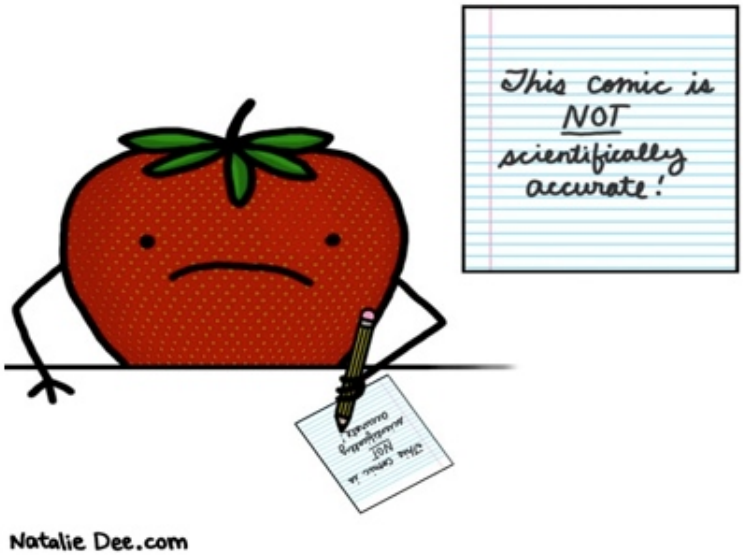
CS302 - Assignment 8

Due: Tuesday, Mar. 12 at the beginning of class
Hand-in method: paper



For this assignment you must use latex to generate your work.

1. **[5 points]** In class, we looked at two different methods for building a heap from an array (`BuildHeap1` and `BuildHeap2`. Do they always create the same heap when run on the same input array? Prove that they do or provide a counterexample.

2. **[7 points]** Your friend (who hasn't taken algorithms) needs your help. She thinks that she's 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, her 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).

3. **[20 points]** Rather than creating binary heaps, we can create a d-ary heap, where each node has d children, rather than 2.

(a) **[3 points]** How would you represent a d-ary heap in an array? Be precise, specifically give pseudo-code for parent and children functions.

(b) **[8 points]** Write pseudo-code for the `Heapify` function for the d-ary heap. You may make any reasonable assumption about referencing the children as long as it is clear. What is the running time of `Heapify` in terms of n **and** d.

(c) **[6 points]** For each of the heap functions below, state what the running time would be in a d-ary heap in terms of n **and** d and give a *very* brief justification:

- `ExtractMax`
- `IncreaseElement`
- `Insert`

(d) **[3 points]** If you wanted to write a `BuildHeap` method that uses `Heapify` and worked from the bottom up and you wanted to minimize the number of calls to `Heapify` where in the array could you start calling `Heapify` (e.g. for binary heaps, we started at $\lfloor n/2 \rfloor$).