Homework 4

1. Recall that a nearly complete binary tree is one that is full in every level except possibly the last level, which is full from the left until some point.
   (a) (3 points) What is the height of a nearly complete binary tree containing n elements?
   (b) (3 points) Give lower and upper bounds for the number of nodes at height h in a nearly complete binary tree.
   (c) (3 points) Where in a max-heap might the smallest element reside, assuming that all elements are distinct? Justify your answer.
   (d) (1 point) Is an array that is sorted in ascending order a min-heap?

2. For all the following questions, give pseudocode and justify correctness and running time briefly. You may assume that the input heap is valid, i.e. that it satisfies the max-heap property.
   (a) (2 points) Give a $O(1)$ algorithm for an operation `Find-2nd-Max(A)` that returns the value of the second largest element in a heap `A`.
   (b) (3 points) Give a $O(\log n)$ algorithm for an operation `Extract-2nd-Max(A)` that returns the value of the second largest element in a heap `A` and then *removes* it from the heap. Do this while only using operations on a heap defined in class: `Max-Heapify`, `Heap-Maximum`, `Heap-Extract-Max`, `Heap-Increase-Key`, and `Max-Heap-Insert`.
   (c) (5 points) Now give an implementation of `Extract-2nd-Max` that runs in time "close to" $\log n$ (notice the lack of the $O(\cdot)$). This time you can use any operations, and modify the array representation of the heap directly.
   (d) (5 points) (**Bonus**[1]) Give a high level description of an $O(\log n)$ algorithm that extracts the i-th largest element from a heap, and prove its correctness and running time. You may assume that $i < \frac{1}{2} \log \log n$.

3. (10 points) Prove formally that the heap sort algorithm presented in class is correct, and show that in the worst case it does $\Omega(n \log n)$ comparisons.

4. Given a max-heap A of size n, let $pos = pos(A, i, n)$ denote the number of positive elements in the sub-heap of A rooted at i. Consider the following recursive procedure that computes pos:

```
1 Positive-Count(A, i, n):
2     if i > n:
3         return 0
4     if A[i] <= 0:
5         return 0
6     pos = 1 + Positive-Count(A, Left(i), n)
7     pos = pos + Positive-Count(A, Right(i), n)
8     return pos
```

   (a) (4 points) Prove correctness of the above algorithm. Make sure to explain the meaning of each line 2-5. Then argue that the algorithm above runs in time $O(pos)$.
   (b) (6 points) Assume now that we do not really care about the exact value of pos when $pos > k$; i.e., if the heap contains more than k positive elements, for some parameter k. More formally, you wish

[1] Bonus questions are not mandatory, and not attempting them will not harm you in any way. They are often more difficult (so that you don't complain the homework isn't challenging enough) and/or teach extra material.

to write a procedure `Positive-Count(A,i,n,k)` which returns the value $\min(pos, k)$, where $pos =$ `Positive-Count(A,i,n)`.

Of course, you can implement `Positive-Count(A,i,n,k)` by calling `Positive-Count(A,i,n)` first, but this will take time $O(pos)$, which could be high if $pos \gg k$. Show how to (slightly) tweak the pseudocode above to *directly* implement `Positive-Count(A,i,n,k)` (instead of `Positive-Count(A,i,n)`) so that the running time of your procedure is $O(k)$, irrespective of pos. Make sure you explicitly write the pseudocode of your new recursive algorithm (which should be similar to the one given above), prove its correctness, and argue the $O(k)$ run time.

Hint: There is a reason we did not consolidate lines 6 and 7 of our pseudocode to the following single line:

```
pos = 1 + Positive-Count(A, Left(i), n) + Positive-Count(A, Right(i), n)
```