CS380: Algorithm Design and Analysis
Review for Exam 2

Friday, March 17th, 2017
Material Covered: Lectures 7-12a

Brief Summary of Material:

**Lecture 7: Heapsort**
- Binary Trees: You should know the basic definitions associated with binary trees, such as the depth of a node, height of a node, height of the tree, perfect binary tree, and complete binary tree.
- Perfect Binary Trees: You should be able to justify how many nodes there are at a depth $d$ in a perfect binary tree, and how many nodes in total there are in a perfect tree of height $h$.
- Heaps: Know a heap is a complete binary tree that also satisfies either the min heap or max heap property (know what these are as well).
- Storage of Heaps: Know heaps are typically stored as arrays, and know how to store a heap in an array and how to get the root and parent and left or right child of a node.
- MAX-HEAPIFY: This is the most important algorithm regarding heaps that we discussed because all the other algorithms use it. Make sure that you can max heapify a heap at a particular node (and know that this algorithm is recursive). See example 6.2 from the notes/text. Also know that MAX-HEAPIFY has a recurrence relation of $T(n) = T \left( \frac{2}{3} n \right) + 1$ and this recurrence can be solved to give a runtime of $O(\lg n)$ (you don’t need to solve this recurrence or derive it on this exam).
- BUILD-MAX-HEAP: Understand why it is sufficient to start the loop in this algorithm at $A.length/2$. Be able to justify why this algorithm is $O(n \ \lg n)$, briefly (although you can actually show that it is $O(n)$ with a careful analysis).
- HEAPSORT: Understand the basics of heapsort in that builds a max heap, then recursively swaps the last element with the root and then MAX-HEAPIFYs the subtree that remains. Know that HEAPSORT is a $\Theta(n \ \lg n)$ algorithm.
- The last example is a good one to understand (it is already a max heap, so Build_Max_Heap doesn’t change the tree). Could you show the first few steps of HEAPSORT on this array?

**Lecture 8: Priority Queues**
- HEAP-EXTRACT-MAX: Be able to show you understand this algorithm on a simple array (not that the algorithm assumes that the heap is already a Max heap).
- HEAP-INCREASE_KEY: Similarly, be able illustrate you understand this algorithm by increasing a key within a heap.
- MAX-HEAP-INSERT: Similar to above. Why is the $A[A.heap-size]$ key set to $-\infty$ in this algorithm?
Lecture 9: Linear sorting

- What does it mean when we say that a sorting method is a Comparison Sort? Why is a comparison sort perhaps not always ideal? Said another way, how many comparisons must a comparison sort necessarily make? Why? Generically, what are the best comparison-based sorting algorithms because of this?
- Counting sort: Understand the restrictions under which counting sort applies, and be able to sort a short array using counting sort. What does it mean when we say counting sort is stable?
- Counting sort analysis: Know that if you are sorting n numbers chosen from the digits \{0,1,...,k\}, that counting sort is a \(O(k + n)\) sorting algorithm and that it is generally only useful if \(k = O(n)\). In particular, if we’re sorting a few numbers chosen from a large interval this would be a poor algorithm to use.
- Radix Sort: Know that radix-sort is used to sort n numbers with d digits each chosen from k possible values, and that given these parameters the runtime is \(\theta(d(k + n))\) provided counting sort is used for each of the d digits of the numbers. Why is it important that a stable sort is used? Be able to do a counting sort on a simple array
- Bucket Sort: Understand the conditions under which Bucket Sort is used, and in particular why it is important that the data is approximately uniformly distributed on the interval (this is an assumption of the algorithm). If you were to use Bucket Sort on data that was very much skewed to one side of the interval, the \(\theta(n)\) complexity would no longer apply.

Lecture 10: Median Order Statistics

- Know the definition of the ith order statistic for n elements and that the naïve algorithm for finding any ith order statistic is \(O(n \lg(n))\) (either Heapsort or Mergesort).
- Understand and be able to use the RANDOMIZED-SELECT algorithm on a simply array (assuming random numbers are provided for you). The example we did in class should be sufficient.
- Know that the issue with RANDOMIZED-SELECT is that, although the average runtime is \(\theta(n)\) (that is, for the “average” collection of data you might encounter with a “random” choice of pivot), if your “random” pivot happens to be a particularly poor choice each time, this algorithm can run with complexity \(\theta(n^2)\). Thus the desire to have a deterministic algorithm SELECT (A,n,i) that does not rely on a random process over which we have no control. Could you, briefly, describe the basics of how SELECT (A,n,i) works?

Lecture 11: STL Vectors: Not on the exam

Lecture 12: Red Black Trees

- Binary Trees: You should understand the Binary Tree Insert algorithm as a matter of course.
- Know the 5 conditions that a binary tree must satisfy to be a red-black tree.
- Have a basic understand how you justify that the maximum height of a red-black tree with n nodes is at most floor\((2\lg(n+1))\).
- Have a basic sense of how to color a tree (similar to example in notes)
- Understand both the Left-Rotate and Right-Rotate algorithms.
- Understand RB-Insert (essentially binary tree insert) and RB-Insert-Fixup(T,z). In particular, make sure you know how to get out of the three different cases that may arise.