**Algorithms (CSCI 323 & 700)**

**Spring 2016 - Homework #4**

**(Due at the beginning of class on 3/2/2016)\***

1. A potential variation of Insertion Sort is to use Binary Search – rather than Linear Search – to find the point in which to insert the next element from the unsorted portion into the sorted portion. How many operations are required to find this point for the next unsorted element? How many operations are required to find these points over the entire algorithm, keeping in mind that the size of the sorted portion grows with each iteration?
2. The “Selection Problem” is to find the ith smallest element among elements. (Note: when i = n/2, this is the problem of finding the median.) Describe how a *complete* sort can be used as a (naïve) “brute force” approach to solve the Selection Problem. What is its time complexity?

1. Describe how a *partial* sortcan be used to solve the Selection Problem. Which of Bubble Sort, Insertion Sort, and Selection Sort is best suited for this task? What is its time complexity? What might you differently depending on whether *i* is closer to 1 or n?
2. Describe how a variation of Quicksort can be used to solve the Selection Problem. (Hint consider which of the two parts to the partition contains the element you want.) What are the average and worst case time complexities?
3. Imagine a variation of Merge Sort in which at each call the array is divided into *three* lists, instead of two, that are individually sorted and then merged into one big sorted array. Explain why this variant’s worst case time complexity is described by the recurrence T(1) = 0, T(n) = 3T(n/3) + 2n – 3. Then solve the recurrence using the methods of domain and range transformations.
4. Consider recurrences of the form T(n) = aT(n/b) + O(nd) where n is the size of the original problem, n/b is the size of each sub-problem, a is the number of sub-problems (often, but not necessarily, the same as b), and O(nd) is the time for dividing and combining in “Divide and Conquer” problems. The “Master Theorem” (aka “Master Method”) states that there are three cases:

Case 1: d < logb a: T(n) = O(nlog\_b a)

Case 2: d = logb a: T(n) = O(nd log n)

Case 3: d > logb a: T(n) = O(nd)

(More on the Master Theorem can be found at https://en.wikipedia.org/wiki/Master\_theorem)

Use the Master Theorem to determine the order of magnitude of the recurrence in question (5).

1. Use the Master Theorem to determine the order of magnitude of the recurrence for Binary Search discussed in class on 2/17.

\* Only if you will not be able to attend class on the due date, submit your solutions - *before 6:00 p.m. on the due date* - to the instructor at LT.CS320@yahoo.com