CS 466/666
Assignment 3
Due: Noon Thursday October 23, 2008

1) [10 marks] Consider the problem of finding the second largest of n numbers. Give an algorithm using as few comparisons (n+o(n)) in the worst case to solve this problem, state this number of comparisons and justify your answer. (Hint: If you find the maximum, then the second largest is one that lost a comparison to this max value. So, the trick is to minimize the number of comparisons involving the maximum. Avoiding a bad idea: In the last assignment you looked at the "obvious" method for finding the maximum; if the max is the first, this method will do n -1 comparisons involving it… so this is clearly not a good approach.)

2) [10 marks] This question is to analyze the worst case number of comparisons in the linear median algorithm presented in class and present in the text (i.e. columns of length 5). We would like to analyze more carefully to determine the constant factor in front of n. Consider the algorithm outline with a pre-analysis:

Preprocessing: 7n/5 comparisons (sorting into columns of length 5; 5 values are sorted with 7 comparisons)

Procedure Median(n):
- Median(n/5)
- 2n/5 comparisons to insert the median of medians into the "half columns" after finding median of medians
- Time 6n/25 to reconstitute columns.
- Median(7/10)

Therefore the number of comparisons can be analyzed to 7n/5 + T(n) where

$T(n) = T(n/5) + T(7n/10) + 16n/25$

Argue about the base cases for this recurrence relation and determine the constant factor for this O(n) comparison algorithm (including the preprocessing).

3) [10 marks] How can you modify the "pivot selection" part of the quicksort so that the algorithm has O(n lg n) runtime in the worst case? Justify this claim. (Hint: this is not a method I would recommend in practice)

4) [10 marks] Let A[1..n] and B[1..n] be two array, each containing n numbers already in sorted order. Give an O(log n)-time algorithm to find the median of all 2n elements in arrays A and B.