

**CSE 830: Homework #2**  
**Due Sept. 29<sup>th</sup> 2011, 10:20am**

1. Use the partitioning step of quicksort to give an algorithm that finds the *median* element of an unsorted array of  $n$  integers in expected  $O(n)$  time.
2. Outline an efficient method of solving each of the following problems. Give a tight bound on the worst-case complexity of your methods. You may describe your algorithm in a common programming language, in pseudocode, or in English, but please be sure to be thorough.
  - a. Given an unsorted list of  $m$  stocks that you are investing in, and a sorted list of all  $n$  stocks available saying if their value has gone up or down today. Determine how many of your stocks have dropped.
  - b. You are given a list of  $N$  packages that UPS delivered in a given month. For each package, you are given an unsorted list of the names of the  $M$  trucks it was transported on to reach its destination. Find the number of packages that were transported on each specific truck. (Note: this one can be a bit tricky if you consider that there may be many more packages than trucks.)
  - c. You have a huge stack of papers that you are carrying from one end of the company to another when a co-worker slams a door causing the first 34 pages to fly off the top. The helpful co-worker aides you in gathering them up, but to your horror you realize he is placing the ones he collected into multiple points in the middle of the once-sorted pile. How can you restore the papers to a sorted state?
3. You have a stack of  $n$  elements to sort (with the smallest value on top). You are allowed to use only the following operations to manipulate the data:
  - Pop-Push( $s_1, s_2$ ) - pop the top item from stack  $s_1$  and push it onto stack  $s_2$ .
  - Compare( $s_1, s_2$ ) - test whether the top element of stack  $s_1$  is less than the top element of stack  $s_2$ .
  - IsEmpty( $s_1$ ) - determine if stack 1 has no elements left in it.
  - a. Give an initial stack that cannot be sorted just using these operations and two stacks.
  - b. Give a worst-case  $\Theta(n^2)$  time sorting algorithm using just these operations and three stacks. Provide an intuitive explanation for how your algorithm works.
  - c. Give a  $\Theta(n^2)$  time sorting algorithm that will still work on three stacks if the third stack has a fixed (constant) cap on the number of elements it can hold. (hint: use insertion sort) Provide an intuitive explanation for how your algorithm works.
4. Devise an algorithm for finding the  $k$  smallest elements of an unsorted set of  $n$  integers in  $O(n + k \lg n)$  time. For partial credit, find an algorithm that takes  $O(n + n \lg k)$  time instead.
5. Exercise 6-2 (Analysis of d-ary heaps)
6. Exercise 8-4 (Water jugs)
7. (*extra credit*) Design a data structure that allows one to search, insert, and delete an integer  $X$  in  $O(1)$  time (i.e., independent of the total number of integers stored). Assume that  $0 < X \leq n$ , and that  $m + n$  units of memory are available for the symbol table, where  $m$  is the maximum number of integers that can be in the table at any one time. (*Hint*: Use two arrays  $A[1..n]$  and  $B[1..m]$ .) You are not allowed to initialize either  $A$  or  $B$ , as this would take  $O(n+m)$  operations. This means that the arrays are full of random garbage to begin with, so you must be careful.