**CS 477/677 Analysis of Algorithms**

**Homework 3**

**Due September 29, 2014**

**For the programming problems below, include in your hardcopy submission a printout of your algorithm and of the output. Also send your source code as an attachment to cs477677@cse.unr.edu. The subject line of the email should be HW3 - <Your section (CS477 or CS677> - <Your Name>.**

**1. (U & G-required) [40 points]**

**Implement** a `Max` algorithm (in C/C++) that uses a divide-and-conquer strategy to find the maximum among N items stored in an array A[0],…, A[n-1]. To illustrate the sequence of function calls made by your algorithm, each call to the algorithm should print the following tuple <`Return_Value`, "`Max(left_idx`, `right_idx`)">, where `Return_Value` is the maximum value returned by that call and `left_idx` and `right_idx` are the indices of the array which that particular recursive call is made. Submit the following: i) your code for the algorithm, ii) the analysis of the running time for your algorithm, and iii) the output of your algorithm for the following input A = [T I N Y E X A M P L E].

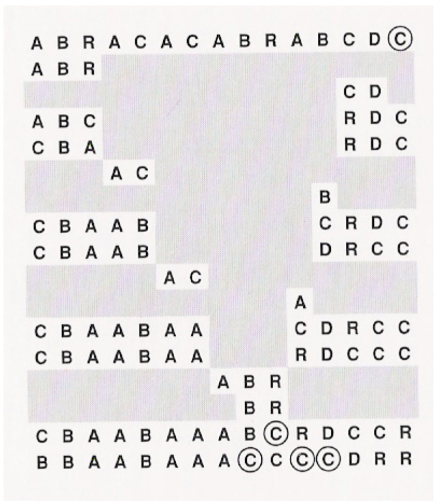
**2. (U & G-required) [40 points]**

**Implement** a bottom-up Mergesort algorithm (in C/C++) that works by making a sequence of passes over the entire array doing `m`-by-`m` merges, doubling `m` on each pass. For example, the algorithm first scans through the input performing 1-by-1 merges to produce ordered sub-arrays of size 2; then, it scans through the input again performing 2-by-2 merges to produce ordered subarrays of size 4, and so on until the entire array is sorted. At each call to `Merge`, print out the input array that is being processed by that function call. Note: the final `Merge` may be an `m`-by-`x` merge, for some `x` less than or equal to `m` (if the array size is not a multiple of `m`). Submit the following: i) your code for the algorithm and ii) the output of your algorithm for the following input A = [A S O R T I N G E X A M P L E].

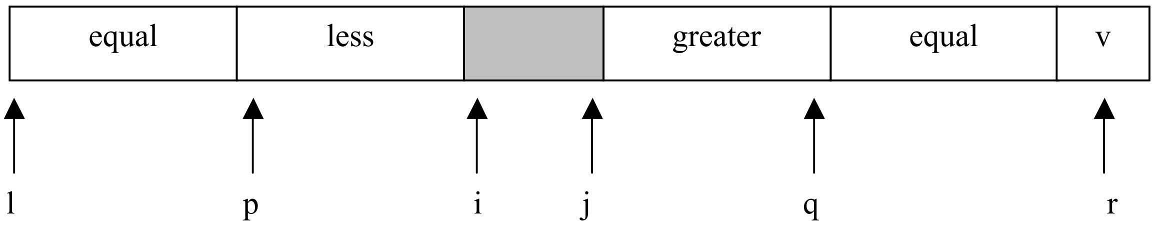
**3. (U & G-required) [20 points]**

Does the Merge procedure produce proper output if and only if the two input sub-arrays are in sorted order? Prove your answer, or provide a counterexample.

**4. (G-required) [20 points]**



For Quicksort, in situations where there are large numbers of duplicate keys in the input file, there is potential for significant improvement of the algorithm. One solution is to partition the file into **three** parts, one each for keys *smaller than*, *equal to* and *larger than* the partitioning element. In this approach the keys equal to the partitioning element that are encountered in the left partition are kept at the partition's left end and the keys equal to the partitioning element that are encountered in the right partition are kept at the partition's right end. **Implement** in C/C++ this partitioning strategy as follows: choose the pivot to be the last element of the array. Scan the file from the left to find an element that is not smaller than the partitioning element and from the right to find an element that is not larger than the partitioning element, then exchange them. If the element on the left (after the exchange) is equal to the partitioning element, exchange it with the one at the left end of the partition (similarly on the right). When the pointers cross, put the partitioning element between the two partitions, then exchange all the keys equal to it into position on either side of it (the figure on the right illustrates this process). During partitioning the algorithm maintains the following situation:



Illustrate the behavior of your algorithm on the input in the above figure by printing, after each iteration (left & right scan and eventual exchanges), the elements in the partitions as in the example above.

**Extra credit**

**5. [20 points]** Problem 7-2 (page 186). Answer only questions (a), (c) and (d). For (c) and (d) assume that you have the procedure at point (b) available.