CS 477/677 Analysis of Algorithms

Fall 2017

Homework 3

Due date: October 3, 2017

For the programming problems below, include in your hardcopy submission a printout of your algorithm and of the output. In addition, please follow the submission instructions on the course website for the electronic submission of your code.

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

(a) [30 points] Implement in C/C++ a version of bubble sort that alternates left-to-right and right to left passes through the data. For example, if sorting the array [6, 5, 2, 8, 3, 1], the first left-to-right pass will swap elements that are out of order and get the result: [5, 2, 6, 3, 1, 8]. The right-to-left pass begins at element 1 (on position 5) and goes all the way to the beginning of the array. At the end of this pass we have [1, 5, 2, 6, 3, 8]. The next phase works only on the segment [5, 2, 6, 3] as elements 1 and 8 have already been placed at their final location.

Show how your algorithm sorts the following array: “EASYQUESTION”. Print the status of the array at the end of each left-to-right and right-to-left pass.

(b) [10 points] How many comparisons does this modified version of bubble sort make?

2. (U&G-required) [30 points] Answer the following questions (justify your answers)**:**

(a) [15 points] During the running of the procedure RANDOMIZED-QUICKSORT, how many calls are made to the random-number generator RANDOM in the worst case?

(b) [15 points] How about the best case?

3. (U&G-required) [15 points]
Problem 2.3-6 (page 39).

4. (U & G-required) [15 points] Show how MERGE-SORT sorts the following array:

14 40 31 28 3 15 17 51

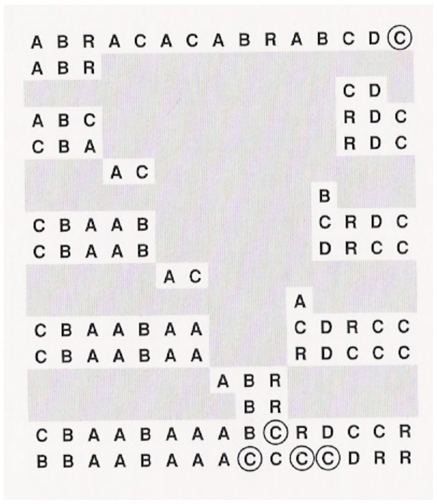
5. (U&G-required) [20 points] The Mergesort algorithm we discussed in class is a recursive, divide-and-conquer algorithm in which the order of merges is determined by its recursive structure. However, the subarrays are processed independently and merges can be done in different sequences. Implement in C/C++ a non-recursive version of Mergesort, which performs a sequence of passes over the entire array doing **m**-by-**m** merges, doubling **m** at each pass. (Note that the final merge will be an **m**-by-**x** merge, if the size of the array is not a multiple of **m**. Show how your algorithm sorts the sequence "ASORTINGEXAMPLE". At the end of each **merge** step print the values in the resulting subarray.

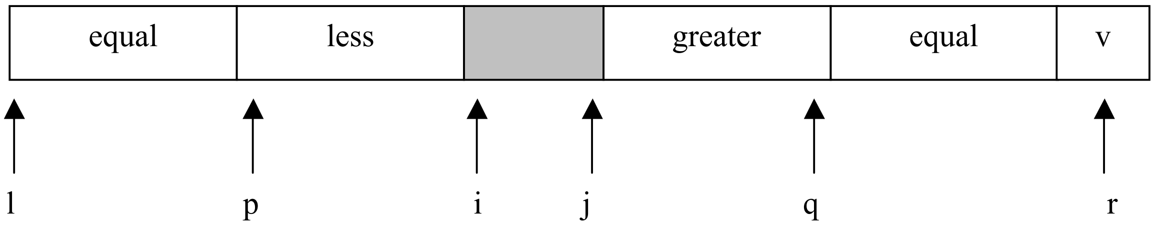
For example, if sorting [3, 2, 5, 1]:

- At first pass (1-by-1 merges): 3 and 2 are merged → [2, 3]
  5 and 1 are merged → [1, 5]
- At next pass (2-by-2 merges): [2, 3] and [1, 5] are merged → [1, 2, 3, 5]

Extra credit:
6. [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.