UNIVERSITY OF MINNESOTA
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
4041
ALGORITHMS AND DATA STRUCTURES
FALL 2017

ASSIGNMENT 1 :
Assigned: 9/23/17 Due: 10/1/17 (submit via moodle)

Problem 1. (30 points)
Show the step-by-step process of the quicksort algorithm when the input sequence is [8, 9, 7, 2, 5, 4, 1, 3, 6, 4]. You should show the exchanges of elements, the pivots, and the modifications of the sequence. Pick the last element as the pivot always. (Note: use qucksort recursively. You do not need to show "merge" steps.)

Problem 2. (30 points)
Consider the "Sort.java" code. Find (1) the worst-case run-time of the code in big-O notation assuming the length of A could change. Then (2) prove the "correctness" of the code (i.e. prove that it will always sort the numbers).

Problem 3. (20 points)
Consider the pseudo-code below for both insertion sort and merge sort. Re-write the for-loop as a while loop in the merge sort pseudo-code. (You do not need to show the modified while-loop merge sort.) Then count how many lines need to execute to sort the array [3, 2, 1] for both algorithms. Show some work to receive full (and partial) credit. Only count lines the computer thinks on.

Note: for merge sort, do not count the function calls, splits or array creation. Just the number of lines **inside** the merge function. Also, assume no work needs to be done merging a size 1 and 0 array. (Example: sorting [2, 1] with insertion sort = 11 lines, merge = 14 lines.)

Merge sort

```
TopDownMerge(A[], iBegin, iMiddle, iEnd, B[])
{
    i = iBegin,
    j = iMiddle;
    // while there are elements in the left or right runs...
    for (k = iBegin; k < iEnd; k++) {
        // If left run head exists and is <= existing right run head.
        if (i < iMiddle && (j >= iEnd || A[i] <= A[j])) {
            B[k] = A[i];
            i = i + 1;
        } else {
            B[k] = A[j];
            j = j + 1;
        }
    }
}
```

Insertion sort

```
i ← 1
while i < length(A)
    x ← A[i]
    j ← i - 1
    while j >= 0 and A[j] > x
        A[j+1] ← A[j]
        j ← j - 1
    end while
    A[j+1] ← x[4]
    i ← i + 1
end while
```

Problem 4. (20 points)
Find the worst-case run-time of each of the situations below. Justify your answer.

(1) Merge sort, but splitting into three arrays instead of two (i.e. each array is N/3 instead of N/2 size). You may assume the original size of the array is a power of 3.

(2) Merge sort, except using insertion sort immediately instead of recursively calling merge sort. (Thus no recursion ever happens.)

(3) The algorithm in "Problem4Sort.java".

(4) Some algorithm with a recursive property: $T(n) = 10\, T(n/3) + O(n \lg n)$

Extra credit: Problem 5. (10 points)
This is a continuation of problem 3. Find an exact formula for the number of lines needed to be run in both the insertion sort and merge sort pseudo-code. This formula should be for arrays of size n that are in the reverse of sorted (e.g. [8, 7, 6, 5, 4, 3, 2, 1] with n = 8). Give the ranges of n where merge sort performs better than insertion sort. For full points, you cannot use a recursively defined function for merge sort.