Homework 6 (Cpt S 223)

Due Date: December 7, 2009

Total points: 46

---

1. (10 points) A university database contains records of all currently enrolled students in the following format: <student name, year of birth>. You are asked to group the set of students by birth year. There are two ways to solve this problem.

   - Solution A: Treat the problem as a problem of finding equivalence classes, defined by the relation: two students belong to the same class if they are born in the same year. Therefore, you can use the union-find data structure to compute equivalence classes.
   - Solution B: Treat the problem as a problem of sorting. Sort all the records by their year. You can use any sorting algorithm of your choice, whichever is best in this case. You are also allowed to make "reasonable" assumptions about the input but state it explicitly.

   Write a pseudocode for each of the above solutions and state which is better in terms of performance? Justify.

2. (6 points) As shown in Figure 7.2, the default implementation of insertion sort loops through an input array of size n. At any iteration i, it carries out these two steps:

   (i) Sequentially search for the proper place to insert array element $A[i]$ in the already sorted array prefix $A[1 \ldots i-1]$; and

   (ii) Insert $A[i]$ in that position after right-shifting all elements until $A[i-1]$ by one position.

   Suppose the sequential search in step (i) is replaced by a binary search. Is this modified implementation necessarily an improvement over the default implementation? Explain from the perspectives of both the worst-case complexity and on what you would expect in practice.

3. (12 points) If an input array of size n happens to be in reverse sorted order, then how much time will each of the following sorting algorithms take to sort it?

   i) Insertion sort (Figure 7.2)

   ii) Merge sort (Figures 7.11, 7.12)

iii) Quick sort with median-of-three pivot selection scheme (Figures 7.14, 7.15, 7.16)

Express the time using the Θ notation, and compare it with the corresponding algorithm's worst-case complexity.

4. (10 points) Many operations can be performed faster on sorted data than on unsorted data. For each of the following operations, state whether it could be performed faster or not, if the data values were sorted (do not take the cost of sorting into account). No justification is required.

   a) Searching for a word in a dictionary;

   b) Checking if a given word has an anagram in the dictionary (e.g., plum, lump)

   c) Computing the arithmetic mean of a set of integers i.e., $\frac{\Sigma_{i=1}^{n} A[i]}{n}$;

   d) Finding the median of a set of integers;

   e) Finding the mode of a set of integers i.e., the most frequently occurring element.

5. (8 points) Instead of the default implementation for merge sort (Figure 7.11, 7.12), we could think of a variant where: the array at every iteration is split into *three* roughly equal-sized partitions, recursed upon and then the three individually sorted partitions merged. Write down the run-time recurrence for this modified implementation and then solve it to get its run-time complexity. (You don't need to provide a pseudocode for this answer, as it is fairly obvious.)