CSE 373 -- Autumn 1999 -- Assignment #6

(*due Friday, December 3*)

1. **Suppose you have an array containing your friends' names in sorted order, and that you have *n* friends. You go on a spectacular cruise to avoid the Seattle winter and in doing so meet *k* new friends (where $k = O(1)$). You resize your original array, add the *k* new names into the top slots (numbers $n..n+k-1$), and then re-sort the array to get everyone back into alphabetical order.**

   (a) What would be the worst-case running time if you used Insertion Sort to fix things up? Why?

   (b) What would be the worst-case running time if you were to use Mergesort on the entire $n+k$ element array? Why?

   (c) What if you were to use mergesort on the final *k* elements and then a merge operation to combine the original list and the *k* new elements?

2. **Give an example of a 9-element array that would form a bad input for a Shellsort algorithm whose increment sequence is 6, 3, 1. Explain why this input is bad.**

3. **A *stable* sorting algorithm is one that preserves the ordering of elements whose values are equal. (You might ask: why would anyone care about this? As an example, suppose I have an array of students which I've sorted by last name. If I were to then sort this list by GPA using a stable algorithm, students who have the same GPA would remain in alphabetical order by last name. In contrast, an unstable algorithm could potentially mix these students up into an arbitrary order. For example, if AJ, Sun Liang, and I all had a 3.2 GPA, the following might be returned by stable and unstable algorithms):**

```
given                     stable algorithm          unstable algorithm

Aue:          ???
Brush:        3.2         ...                       ...
Buys:         ???         Brush:        3.2         Chamberlain: 3.2
Chamberlain: 3.2          Chamberlain: 3.2          Sun:          3.2
Gross:        ???         Sun:          3.2         Brush:        3.2
Sun:          3.2         ...                       ...
Ware:         ???
```

   (a) What needs to be done when writing the Mergesort algorithm to ensure that it is stable?

   (b) Quicksort (as we've discussed it) is unstable due to its partition step. Illustrate why this is by giving an input to a single Quicksort partition step and showing how it loses its stability. Assume that we use the middle element as the pivot. Please label identical values in some manner that keeps them distinct (*e.g.*, 10_A, 10_B, *etc.*). *Note: you should assume that the pivot value is unique -- having multiple values equal to the pivot can create instability, but also raises more fundamental questions that we haven't addressed.*

4. **Assume that we select the pivot for every Quicksort partitioning step by selecting the median of the first, middle, and last elements (assume that the middle value for an array with an even number of values is the one in the first half of the list --** ***e.g.*****, the middle of "10, 21, 13, 6, 17, 8" would be "13").**

   (a) Show a 7-element input to Quicksort with no duplicate values that results in a worst-case runtime for this choice of pivot. For simplicity, you may do the partitioning conceptually (as I did in my "Quicksort Example" slide), rather than using the precise in-place algorithm (which complicates the design of a worst-case input by moving the pivot around in the array).

   (b) Illustrate that this is a worst-case run by showing how each step of the Quicksort would be partitioned.

5. **Write a linear-time function that takes in an array of GPAs with one decimal point accuracy (*e.g.,* 0.0, 0.1, 0.2, ..., 4.0) and prints them out in sorted order:**

```
void PrintSortedGPAs(float GPA[], int numGPAs);
```