CSE 332 Data Abstractions, Winter 2011

Homework 4

Due Friday, Feb 11, 2011 at the **beginning** of lecture. Please be sure your work is readable (either written clearly or typed). This homework has four problems. Please write your **section** at the top of your homework.

Problem 1: Algorithm Analysis

The methods below implement recursive algorithms that return the first index in an unsorted array to hold 17, or -1 if no such index exists.

```
int first17_a(int[] array, int i) {
    if (i >= array.length) return -1;
    if (array[i]==17) return 0;
    if (first17_a(array,i+1) == -1)
        return -1;
    return 1 + first17_a(array,i+1);
    }
    int first17_b(array,i+1);
    if (x == -1) return -1;
    return x + 1;
}
```

(a) What kind of input produces the worst-case running time for first17_a(arr,0)?

(b) For first17_a, give a recurrence relation, including a base case, describing the worst-case running time, where n is the length of the array. You may use whatever constants you wish for constant-time work.

(c) Give a tight asymptotic ("big-Oh") upper bound for the running time of first17_a(arr,0) given your answer to the previous question.

(d) What kind of input produces the worst case running time for first17_b(arr,0)?

(e) For first17_b, give a recurrence relation, including a base case, describing the worst-case running time, where n is the length of the array. You may use whatever constants you wish for constant-time work.

(f) Give a tight asymptotic ("big-Oh") upper bound for the running time of first17_b(arr,0) given your answer to the previous question.

(g) Give a tight asymptotic ("big-Omega") worst-case lower bound for the problem of finding the first 17 in an array (not a specific algorithm). *Briefly justify your answer.*

Problem 2: Sorting Phone Numbers

The input to this problem consists of a sequence of 7-digit phone numbers written as simple integers (e.g. 5551212 represents the phone number 555-1212). The sequence is provided via an Iterator<Integer>. No number appears in the input more than once but there is no other limit on the size of the input. Write precise (preferable Java-like) pseudocode for a method that prints out the phone numbers (as integers) in the list in ascending order. Your solution must not use more than 2MB of memory. (Note: It cannot use any other storage – hard drive, network, etc.) Explain why your solution is under the 2MB limit.

(See back of this page for remaining problems)

Problem 3: OuickSort Variation

Consider this pseudocode for quicksort, which leaves pivot selection and partitioning to helper functions not shown:

```
// sort positions lo through hi-1 in array using quicksort (no cut-off)
quicksort(int[] array, int lo, int hi) {
      if (lo>=hi-1) return;
      pivot = pickPivot(array,lo,hi);
      pivotIndex = partition(array,lo,hi,pivot);
      quicksort(array,lo,pivotIndex);
      quicksort(array,pivotIndex+1,hi);
}
Modify this algorithm to take an additional integer argument enough:
```

// sort at least enough positions of lo through hi-1 in array using quicksort (no cut-off)

```
quicksort(int[] array, int lo, int hi, int enough) { ... }
```

We change the definition of correctness to require only that at least the first 'enough' entries (from leftto right) are sorted and contain the smallest enough values. (If enough >= hi-lo, then the whole range must be sorted as usual.) While one correct solution is to ignore the enough parameter, come up with a better solution that skips completely unnecessary recursive calls. Assume the initial call to quicksort specifies that 'lo' is 0 and 'hi' is the upper-bound of the array. Watch your off-by-one errors!

Problem 4: Decision Trees

}

Each of the sorting algorithms we discussed in class, with the exception of bucket/radix sort, can be translated into a specific decision tree. In this question, you'll give the decision tree for three sorting algorithms for a very small sized input. Assume you have an array with exactly three keys: [a, b, c]. Assume there are no duplicate keys. For each of the following sorting algorithms, give the full decision tree. Draw your decision tree in the style of slides 10 and 12 from the Beyond Comparison sorting lecture (with possible orderings in each node, and each branch species a comparison). Be sure to draw an edge for every comparison that the algorithm does, regardless of whether there are multiple orderings possible at that point or not. Feel free to annotate your tree with other information such as the current state of the array at that point in the algorithm if you find it helpful to do so.

(a) selection sort (use the code below, assume that "Find index of smallest entry" loops from left to right over the unsorted portion of the array):

```
void SelectionSort (Array a[0..n-1]) {
      for (i=0, i<n; ++i) {</pre>
        j = Find index of smallest entry in a[i..n-1];
        Swap(a[i],a[j]);
      }
```

- (b) insertion sort (use the code from p. 249 in Weiss)
- (c) merge sort (use the code on p. 260 and p. 261 in Weiss)
- (d) Considering only number of comparison operations, which of these three algorithms is the most efficient for a 3-element array? Explain why in terms of your decision trees.
- (e) Considering only number of comparison operations, which of these three algorithms is the least efficient for a 3-element array? Explain why in terms of your decision trees.