CSE332 Data Abstractions, Spring 2012
Homework 4

Due: **Friday, May 4, 2012** at the beginning of class. Your work should be readable as well as correct.

This assignment has **four** problems.

Problem 1. Algorithm Analysis

The methods below implement recursive algorithms that return the first index in an 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(int[] array, int i) {
  if(i >= array.length)
     return -1;
  if(array[i]==17)
     return 0;
  int x = 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(a,0)`?

(b) For `first17_a(a,i)`, 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(a,0)` given your answer to the previous question.

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

(e) For `first17_b(a,i)`, 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 for the running time of `first17_b(a,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. *Briefly justify your answer.*

Problem 2. Deletion with Open Addressing Hash Tables

We claimed in class that a hash table using open addressing must use "lazy deletion."

(a) Give a short-as-possible example that demonstrates that using "full deletion" can lead to the hash table returning the wrong result for an operation. Make your example complete:

- Explain your hash table features, including the table size, probing strategy, and hash function.
- Give the sequence of operations and the state of the hash table after each operation.
- Demonstrate how lazy deletion leads to the correct result.
- Argue that no shorter sequence of operations can lead to the wrong result.

(b) When rehashing to a larger table, do lazily-deleted items need to be included? Briefly explain.

Problem 3. Partial Quicksort

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 left-to-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 length of `array`. Watch your off-by-one errors!

Problem 4. 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 (preferably 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.