CS 211 Summer 2007, Review problem set #1

1. Consider a priority queue `p`, implemented as a balanced heap and stored in an array. What are the contents of the array after the following sequence of operations?
   `p.enQueue(12); p.enQueue(5); p.enQueue(3); p.enQueue(48); p.deQueue(); p.enQueue(0); p.enQueue(10); p.enQueue(7); p.enQueue(101); p.deQueue(); p.deQueue(); p.deQueue();`
   Solution shown in class.

2. A dictionary d is implemented as a hash table with chaining and table-doubling. The initial table size is 2, λ_0 is 0.55, and the hash function for strings counts the number of characters in the string. What does the hash table look like after the following sequence of operations?
   `d.insert(‘‘maine’’); d.insert(‘‘wisconsin’’); d.insert(‘‘missouri’’); d.insert(‘‘south carolina’’); d.insert(‘‘pa’’); d.insert(‘‘texas’’); d.insert(‘‘minnesota’’); d.insert(‘‘ny’’); d.insert(‘‘oregon’’);`

   Solution shown in class.
   What is the average and worst-case running time for each `insert()` operation? What is the worst-case running time for `find()`? What is the average-case running time?
   `insert` takes $O(n)$ in the worst case (a table doubling operation occurs). `find` takes $O(n)$ in the worst case (a bad hash function, and all the keys are in the same bin of the hash table). In the average case, assuming a good hash function, `insert` and `find` take $O(1)$ time.

3. How would you implement each of the following? Answer in words; Java code is not required.
   (a) Compute the median of an unsorted set of n numbers in $O(n \log n)$ time.
   Sort using merge sort in $O(n \log n)$ time. Then return the middle element of the array in $O(1)$ time.
   (b) Compute the mode of an unsorted set of n numbers in $O(n)$ time.
   Use a hash table to store (number, count) pairs. Loop through the numbers, doing `hashtable[number]++`, which takes a total of $O(n)$ time. Then loop through again, looking at the count values in the hash table to find the maximum, in $O(n)$ time.
   (c) Compute the mean of the smallest k numbers of an unsorted set of n numbers in $O(n \log n + k)$ time. Also give an algorithm that takes $O(nk)$ time.
   Sort in $O(n \log n)$ time, then scan through the smallest k entries in $O(k)$ time, for a total of $O(n \log n + k)$ time. For the $O(nk)$ solution, loop through the entire array k times: during the first iteration look for the smallest entry, then the second smallest, then the third smallest, etc.
   (d) Compute the union of two sets of unsorted numbers in $O(n)$ time.
   Add both sets into a `hashset`, which automatically removes duplicates. `hashset` is implemented using a hash table so operations take $O(1)$ time on average. We do $O(n)$ operations, so the total time is $O(n)$.
   (e) Sort a set of n integers in $O(n + p)$ time, if the integers are in the range $[0, p]$.
   Use the priority queue designed for the special case when priorities are integral and bounded (see lecture notes).

4. Answer the following questions about ADT and algorithm implementations.

   (a) Which priority queue implementation would you use to find the 10 smallest of a set of 10^{12} random numbers?
   No clear answer. A heap would take $O((n + k) \log n)$ time (n insert operations, k removeMin operations, each taking $O(\log n)$ time), where $n = 10^{12}$ and $k = 10$. An unsorted array would take $O(n + kn)$ time ($O(n)$ for the inserts and $O(kn)$ for the removeMins), which might be just as fast since n is so much larger than k.

   (b) Which dictionary implementation would you use if you insert and delete infrequently, but want to answer range queries very efficiently? (A range query is a request like "find all keys less than k" or "find all keys in the range $[k, l]$.")
   Use a data structure that keeps the keys sorted, like a binary tree or a sorted array.

   (c) Which sorting algorithm would you choose if heap space is limited?
   Probably quicksort, because it's fast but doesn't require a temporary array (like Mergesort does).

   (d) Which sorting algorithm would you choose if the array to be sorted is too large to fit into memory?
   Mergesort.