CS61B Lab #10

This is more playing around with data structures related to sorting and hashing.

## Bitsort

Bitsort is radix sorting where the "characters" are bits of the keys (which in our case will be integers). First, fill in the sortOnBit function in Bitsort.java to make the program work (it's self-checking). The tricky part is figuring out how to modify A in place (i.e., without moving its contents to a new array) and stably.

Next run the program on a variety of inputs (I suggest starting at 'java Bitsort 1000000' and working up. Compare the results to quicksort, which you can run with, e.g.,

```
java Quicksort 1000000 1
```

The last number represents the minimum size array that quicksort sorts; after sorting into arrays of at most that size, it does a final pass of insertion sort. Try fiddling with that last parameter to Quicksort to get best results. Report results (parameters and times for the programs) in lab10.txt. Do the trends in the results for Bitsort look like what you'd expect?

## Nitty-Gritty Data Structures

In actual, high-performance hash-set implementations, one generally wants to squeeze out space and execution time. One approach is to use data structures at a lower level, closer to the hardware. For example, we can implement linked lists using arrays of items, with integer indices into the array serving as pointers. One thereby avoids the overheads that accompany every Java object and thus make linked list cells in Java rather bulky. This particular implementation maintains a free list of array items that are not in use (have not been filled yet or have been removed), which allows it to manage its own memory.

The file HashSet.java contains most of an implementation of a (very simplified) HashSet class. Read and understand the data structure, and then fill in the missing .remove method. The HashSetTest.java program will test the result: when invoked with java HashSetTest N S, it will produce N random integers, remove about half, and make sure the resulting set has the right elements remaining. S is a seed for the random-number generator; you get identical results with identical seeds.