CS350 — Spring 2013
Homework 3

Due Tuesday 30th April, on paper, at the start of class. This assignment will ***not*** be graded. You will get credit for turning it in. I will post an answer key, so you will be able to grade your own work. Keep a copy!

It's OK to work on this with other members of your workgroup, but each student should turn in his or her own answers to get credit. The exception is question 4: if you like pair-programming, then it's acceptable to work on the implementation with a partner, and turn in one piece of code for the pair of you.

1. **Source-removal algorithm for Topological Sort**

   (a) Prove that a non-empty DAG must have at least one source. **Hint:** Try a proof by contradiction.

   (b) In class, we considered the problem of finding a source in a DAG with v vertices and e edges. When the graph is represented by an adjacency *matrix*, the time complexity of this operation is $O(v^2)$. How would you find a source if the graph is represented by an adjacency *list*? Write down your algorithm, and find its time efficiency. Suppose that your answer is T_{al}.

   (c) Develop a decrease-by-one algorithm for topological sorting of a DAG represented by an adjacency list. It should remove a source, topologically sort the remaining nodes, and then add the source back as the least element in the topologically sorted list.

   (d) If you did part 1c in the obvious way, your algorithm will have time complexity $(v-1)\,T_{al}$, because you will have to remove a source $v-1$ times. Modify your algorithm so that its overall time complexity is linear in the number of vertices and edges, that is, in $O(v+e)$. **Hint:** modify your decrease-by-one algorithm into a decrease-by-variable-amount algorithm.

2. **Binary Search.** Consider the following sorted array A of keys:

0	1	2	3	4	5	6	7	8	9	10	11	12
5	13	19	23	28	34	46	49	57	89	92	94	97

   (a) True or false: The binary search algorithm from Levitin §4.4 will make the first key comparison against $A[6]$.

   (b) What is the largest number of comparisons necessary to find any of the keys in A?

   (c) *Which* keys will require the largest number of comparisons?

(d) Suppose you are searching for key 65. How many comparisons will be made before you know that it is not present?

3. **Binary Search Speedup.** Suppose that you have an array of 50000 elements. How much faster, on average, will it be to search it using binary search, compared to sequential search. (Show your working.)

4. **Generating Permutations.** Implement the *Johnson-Trotter* Algorithm, or the *LexicographicPermute* Algorithm, for generating permutations, as described in Levitin §4.3. You can use any programming language, but you should write and run tests that check

   (a) that no permutation is generated more than once,

   (b) that exactly $n!$ permutations are generated, and

   (c) if you choose LexicographicPermute, that the permutations are indeed in lexicographic order.

   Test your algorithm by generating permutations of numbers up to 10. Turn in your code, and the results of running your tests (that is, the pass/fail/error numbers from the tests, not all of the permutations!)