York University
EECS 2011E Fall 2015 – Problem Set 4
Instructor: James Elder

This problem set will not be graded, but will help you consolidate the material from the second half of the course and prepare for the final exam. You are free to work together on these if you prefer. I will post the solutions by Dec 11.

1. **Hashing**
   Given input keys $\{71, 23, 73, 99, 44, 19, 49\}$ and hash function $h(k) = k \bmod 10$, show the resulting hash table based upon:

   (a) Separate chaining

0	1	2	3	4	5	6	7	8	9

   (b) Linear probing

0	1	2	3	4	5	6	7	8	9

   (c) Double hashing with secondary hash function $h'(k) = 7 - k \bmod 7$

0	1	2	3	4	5	6	7	8	9

2. Nearest node in BST
   Given a binary search tree T and a number k, the algorithm $Nearest(k, v)$ should return the key in the subtree of T rooted at vertex v that is closest in value to k. For example, if $k = 7$ and the subtree contains keys 17, 5, 13, and 11, then the algorithm should return 5. In the event of a tie, either key can be returned.

   (a) Design a recursive algorithm for $Nearest(k, v)$. Describe your algorithm using concise pseudocode. You can assume access to an object T that refers to the tree and supports the following operations

   - T.isExternal(v): Return true if v is an external node, false otherwise.
   - T.left(v): return the left child vertex of v
   - T.right(v): return the right child vertex of v
   - key(v): return the key of vertex v

   You can assume that v is a valid vertex of T. You may also assume access to a constant k_∞ that is larger than the difference between any two keys.

   (b) What is the asymptotic running time of your algorithm for a tree with n nodes?

   (c) What is the asymptotic running time of your algorithm for a tree with n nodes, if the tree is an AVL tree?

3. Highest Points
   You are implementing an algorithm that draws part of the landscape of a 3D terrain, and you are faced with the following problem: You are given the heights of n points of the terrain's grid, and you need to find the $\lfloor \sqrt{n} \rfloor$ highest of them, returning them in descending order. Note that these heights are real numbers, not integers.

(a) Design an algorithm that does this in $O(n)$ time. Assume that the points are given as an array $A[1 \ldots n]$.

Briefly describe your algorithm using concise pseudocode. Any of the algorithms we have covered in class may be used as subroutines. Refer to them by name (do not re-describe them).

(b) Briefly state the asymptotic running time taken by each step of your algorithm, and thus show that it is has a total running time of $O(n)$.

(c) Provide a Θ bound on the time complexity of this problem. Briefly justify your answer.

4. Deleting Edges in Graphs
Let $G = (V, E)$ be an *undirected* graph with vertex set $\{0, 1, \ldots, |V| - 1\}$ and with $|E|$ edges. We denote with (i, j) the edge connecting vertex i and vertex j, and with $d(i)$ the degree of vertex i.

We define the following two operations:

deleteEdge(i, j)**:** delete from G a given edge (i, j).

deleteIncidentEdges(i)**:** delete from G all the $d(i)$ edges incident on a given vertex i.

Provide a precise analysis of the time complexity of operations
deleteEdge(i) and deleteIncidentEdges(i,j) for the following two representations of graph G:

(a) Graph G is represented by a $|V| \times |V|$ boolean matrix A such that $A[i, j]$ is true if and only if G contains edge (i, j).

(b) Graph G is represented by $|V|$ sequences $S_1, \ldots, S_{|V|}$, where sequence $S_i = \{S_i(1), \ldots, S_i(d(i))\}$ contains the $d(i)$ vertices adjacent to vertex i and is realized by means of a doubly-linked list.

5. kth-Largest Element
You are to design an efficient recursive algorithm for finding and returning the k^{th} largest element in an array of n elements. Here k can be any number from 1 to n. For simplicity, you can assume that the elements are all unique.

Your recursive algorithm will have the interface kthLargestElement(A, p, q, k), where A is the array and p and q are the upper and lower index bounds of the current search region within A. Your algorithm will make use of the in-place partition method used in QuickSort. In particular, calling r = partition(A, p, q) will reorganize the contents of A between indices p and q so that $A[i] \leq A[r]$ if $p \leq i < r$ and $A[i] > A[r]$ if $r < i \leq q$, where $p \leq r \leq q$. Note that the partition function returns the pivot index r. Your algorithm would be invoked as kthLargestElement(A, 0, n-1, k), where n is the size of the array.

Important: 1) Do not sort the entire array. **2)** You do not have to implement the partition function.

(a) (12 marks) Describe your algorithm in concise pseudo-code or in English.

(b) (3 marks) What is the worst-case asymptotic running time of your algorithm? Briefly justify your answer.

(c) (1 mark) What is the expected asymptotic running time of your algorithm, assuming random input? No justification required.

(d) (1 mark) For comparison, suppose you implemented an alternative algorithm that simply sorts the input using QuickSort and then finds the k^{th} largest element in the sorted array. What is the expected asymptotic running time of this algorithm?