Comprehensive Exam: Algorithms and Concrete Mathematics Autumn 2003

1. **[10 pts]** Prove a tight asymptotic bound on the behavior of $T(n) = T(n/3) + T(n/9) + n^3$, where $T(n) \leq b$ for $n < n_0$, where n_0 and b are some given constants. For simplicity, solve for n being a power of 9.

   Simplest approach is to guess $T(n) = \Theta(n^3)$ and prove by induction. First notice that $T(n) = \Omega(n^3)$ just by examining the recurrence. Now assume $T(n) \leq cn^3$ for some constant c. This constant should be large enough to satisfy initial conditions. Now the inductive step is:

   $$T(n) = T(n/3) + T(n/9) + n^3 \leq c(n/3)^3 + c(n/9)^3 + n^3 = cn^3(1/3^3 + 1/9^3 + 1/c)$$

   The claim follows since the expression in the parenthesis is smaller than 1 for large enough c.

2. **[10 pts]** Given integers $a_1, a_2, \ldots, a_n$, give a randomized algorithm that outputs all pairs (i, j) such that $a_i = a_j$. Your algorithm should have expected running time $O(n+K)$ where K is the number of pairs output. Prove the upper bound on the running time.

   Use hashing with chaining, with size of the hash table constant factor larger than n. Moreover, construct each chain as a chain of linked lists, each list holding equal elements. In other words, if a_j hashes into position q, look through the linked list and find the list that corresponds to the value of a_j. Then, for each element a_k of this list, output (a_j, a_k). The claim follows since the expected number of lists attached to a single hash cell is constant.

3. **[10 pts]** Assume that you are given a "black box" implementation of a comparisons-based data structure that supports "extract minimum" and "insert element". You are also told that such data structure can be constructed on n elements in time $\Theta(n \log\log n)$. Let $g(n)$ be the amortized time it takes to execute "extract min", i.e. extract the smallest element from the above data structure and delete it from the data structure. Is it possible that $g(n) = \log\log n$? What is the (asymptotically) fastest possible $g(n)$?

   By using this black-box, one can sort by building the structure, and using extract-min n times. Since total time for sorting in comparisons-only model is $\Omega(n \log n)$, the extraction should take at least $\Omega(\log n)$ amortized per extracted element.

4. **[15 pts]** You are given a graph $G = (V, E)$ with non-negative weights on edges, i.e. $w : E \to R^+$. You are also told that all weights are distinct, i.e. $\forall e_1, e_2 \in E, e_1 \neq e_2 : w(e_1) \neq w(e_2)$.

   (a) **[10 pts]** Prove that the Minimum Spanning Tree in G is unique.

   (b) **[5 pts]** Say you succeeded in proving (a). Now assume that you were given tree T that is the minimum spanning tree for G with weight w. Explain how to use T in order to compute T', which is a minimum spanning tree for G with weights $w'(e) = [w(e)]^2, \forall e \in E$.

a. Let T be some MST. Consider Kruskal's MST algorithm executing on the given data to construct T'. We will show that T and T' are identical. Assume not. Now let e be the "first difference". More precisely, the first edge where Kruskal's decision was different from T. Observe that it is not possible that the first difference is of the form "Kruskal's algorithm rejected e, but e is in T". This is due to the fact that up until e, all edges taken by Kruskal's algorithm are also in T. Thus, the only reason Kruskal's algorithm rejected e was because it closes a cycle, which means that e can not belong to T either.

So the only conclusion is that e is not in T but it is in T'. Add e to T and consider the created cycle. Note that at least one of the edges on this cycle (call it e') was considered *after* e, since when e was considered, Kruskal's algorithm acceped it, which means it did not close a cycle at that moment. Thus, by adding e and deleting e', we improved the weight of T (recall that all weights are different), leading to a contradiction.

b. Since MST is unique, we can assume that the given tree T was constructed by Kruskal's algorithm running on the original weights. Observe that squaring of the weights does not change the sorted order of the edges, since all weights are positive. Hence, Kruskal's algorithm will still produce T, even after the edge weights are squared.

5. **[15 pts]** You are given n villages situated along a highway. Let $x_1, x_2, \ldots, x_n$ represent the positions of these villages on the highway (the highway is a straight line). We need to build hospitals in k of these villages. Let d_i denote the distance from the i-th village to the nearest hospital, and let $D = d_1 + d_2 + \ldots + d_n$. Our goal is to minimize D.

   (a) **[5 pts]** Give an efficient algorithm to solve the problem for $k = 1$.

   (b) **[5 pts]** Consider an optimum placement for some k. Prove that this optimum solution can be viewed as consisting of two optimum solutions for smaller problems, where one problem is optimum placement of $k - 1$ hospitals to cover some villages and the other problem is placing a single hospital to cover the rest of the villages. State the "smaller problems" completely.

   (c) **[5 pts]** Give a polynomial time algorithm to find the smallest possible value of D for a given k.

a. Median-index village is the answer to this question. To see this, consider a hospital with q_1 villages to the left of the hospital and q_2 villages to the right, with $n = q_1 + 1 + q_2$. Now move the hospital one village left, say x miles. All villages on the left improve distance by x, while all villages on the right (including the one where the hospital was up until now) increase their distance by the same x. Total change in D is $-q_1 x + x + q_2 x$, which is a gain as long as $q_1 > q_2$.

b. First of all, assume that the villages are sorted with x_1 being the leftmost and x_n being the rightmost. If they are not sorted, then sort and rename.

Since villages are assigned to closest hospital when computing D, the set of villages assigned to a specific hospital "has no holes", i.e. if x_i and x_j are assigned to a specific hospital, then all villages between x_i and x_j are assigned to the same hospital as well.

In particular, consider optimum solution, and consider all the villages assigned to the rightmost hospital. Per our discussion above, these are villages starting from some index

$q+1$ and up to n. The rest of hospitals ($k-1$ of them) are covering villages x_1 through x_q. We claim that these $k-1$ hospitals are the best solution to cover villages x_1 through x_q with $k-1$ hospitals. If not, then take the better solution, add the k-th hospital with its villages, and you will get a better solution than the original optimum one, which is a contradiction.

c. Let $D(q,p)$ be the best way to cover villages x_1 through x_q using p hospitals. Previous discussion implies that in order to compute this value, we should compare all values $D(i,p-1)+Q(i+1,q)$, where $Q(s,t)$ is the best cost of using a single hospital (in the best possible way as per (a)) to cover villages x_s through x_t.

Observe that we are looking for $D(n,k)$, which is the answer to the posed question. This value can be computed using dynamic programming. First, one can compute $D(i,1)$ for all i using median as in (a). Then one can compute $D(i,2)$ for all i, $D(i,3)$, etc. There are nk elements in the dynamic programming table, with each taking $O(n)$ to compute, leading to $O(n^2k)$ algorithm.