CS515: Algorithms and Data Structures
Midterm Exam

The exam is TWO pages.

Problem 1. An inversion in an array $A[1..n]$ is a pair of indices (i, j) such that $i < j$ and $A[i] > A[j]$. The number of inversions in an n-element array is between 0 (if the array is sorted in increasing order) and $\binom{n}{2}$ (if the array is sorted in decreasing order).

Describe and analyze an algorithm to compute the number of inversions of an n-element array in $O(n \log n)$ time. Suppose, to simplify the explanation, that the elements of the array are distinct.

Solutions 1. Here is a reduction to the problem of computing intersecting segments in the homework. We build an instance of intersecting segments as follows. For each $1 \leq i \leq n$, let $p[i] = (i, 1)$, and let $q[i] = (A[i], 0)$.

We claim that (i, j) is an inversion if and only if $(p[i], q[i])$ intersects $(p[j], q[j])$. To verify that, suppose (i, j) is an inversion, and $i < j$. Then, $A[i] > A[j]$, so $(p[i], q[i])$ intersects $(p[j], q[j])$. On the other hand, if $(p[i], q[i])$ and $(p[j], q[j])$ intersect, then the order of $p[i]$ and $p[j]$ are different from the order of $q[i]$ and $q[j]$. Consequently, either $i < j$ and $A[i] > A[j]$, or $i > j$ and $A[i] < A[j]$. In both cases, there is an inversion

Problem 2. Suppose you are given a tree T with root r and positive weights on its edges.

(a) Describe and analyze an algorithm to find a simple path in T with maximum total weight that starts at r.

(b) Describe and analyze an algorithm to find a simple path in T with maximum total weight.

For both parts, $O(n)$ time algorithms receive full credit.

Solution.

(a) For any vertex x of the T, let $T[x]$ denote the subtree rooted at x. In particular, $T = T[r]$. Let $\ell(x)$ denote the longest path in $T[x]$ starting at x. We have:

$$\ell(x) = \max_{c \in C(x)} \ell(c) + w(c, x),$$

where $C(x)$ denotes the set of all children of x.

Having the value of $\ell(\cdot)$ for all children of x we can compute $\ell(x)$ in $O(|C(x)|)$ time. Using dynamic programming we can compute the value of $\ell(\cdot)$ for all vertices of T in $\sum_{x \in T} |C(x)|$ time. But, in that summation each edge of the graph is counted exactly once, so it is $O(n)$.

(b) Let $L(x)$ denote the length of the longest path in T that contains x, but not necessarily start at x. The value of $L(x)$ is the maximum of the following two values.

$$\max_{c_1,c_2 \in C(x)} \ell(c_1) + \ell(c_2) + w(c_1, x) + w(c_2, x), \tag{1}$$

$$\max_{c \in C(x)} \ell(c) + w(c, x). \tag{2}$$

In equation (1) we consider all paths that contain x as an intermediate vertex. In equation (2) we consider all paths that contain x as an end vertex.

Again, having $\ell(\cdot)$ for all children of x we can compute $L(x)$ in $O(|C(x)|)$ time. Therefore, we can compute all values of $L(\cdot)$ in linear time. Finally, we compute the maximum of L in linear time and return it.

Problem 3. Recall the job scheduling problem. The input is composed of the starting and finishing times of n jobs. We would like to find the maximum set of pairwise disjoint jobs.

Consider the following alternative greedy algorithms for the job scheduling problem. For each algorithm, either prove or disprove (by presenting a counter example) that it always constructs an optimal schedule.

(a) Choose the job that ends last, discard all conflicting jobs, and recurse.

(b) Choose the job that starts first, discard all conflicting jobs, and recurse.

(c) Choose the job that starts last, discard all conflicting jobs, and recurse.

(d) Choose the job with shortest duration, discard all conflicting jobs, and recurse.

Solution.

(a), (b)

(c) This is the exact same algorithm presented in the class if we look at the jobs from right to left. Therefore, the symmetry of the problem implies the correctness of this algorithm.

(d)

Problem 4. Suppose you are given n bolts and n nuts. The bolts have distinct sizes. Also, the nuts have distinct sizes. Each bolt match exactly one nut.

Consider the following randomized algorithm for choosing the **largest** bolt. Draw a bolt uniformly at random from the set of n bolts, and draw a nut uniformly at random from the set of n nuts. If the bolt is smaller than the nut, discard the bolt, draw a new bolt uniformly at random from the unchosen bolts, and repeat. Otherwise, discard the nut, draw a new nut uniformly at random from the unchosen nuts, and repeat. Stop either when every nut has been discarded, or every bolt except the one in your hand has been discarded.

(a) What is the probability that the algorithm discards NO bolts?

(b) What is the exact expected number of discarded bolts when the algorithm terminates?

(c) What is the probability that the algorithm discards exactly one nut?

(d) What is the exact expected number of discarded nuts when the algorithm terminates?

(e) What is the exact expected number of nut-bolt tests performed by this algorithm?

Solution.

(a) The algorithm discards no bolts if it picks the largest bolt first. This happens with probability $1/n$

(b) The algorithm discards i bolts if and only if the largest bolt is chosen as the $i+1$th bolt. This happens with probability $1/n$. So, the expected number of discarded bolts is:

$$\sum_{i=0}^{n-1} i/n = (n-1)(n-2)/2n$$

(c) What is the probability that the algorithm discards exactly one nut?

(d) What is the exact expected number of discarded nuts when the algorithm terminates?

(e) What is the exact expected number of nut-bolt tests performed by this algorithm?

Problem 5. [Extra Credit] Suppose you are given an $n \times n$ bitmap, represented by an array $M[1 \ldots n, 1 \ldots n]$ of 0s and 1s. A solid block in M is a subarray of the form $M[i \ldots i', j \ldots j']$ containing only 1-bits. Describe and analyze an algorithm to find the maximum area of a solid block.

The amount of extra credit you earn highly depends on the running time of your algorithm. To earn any extra credit the running time of your algorithm must be $O(n^4)$.