HW 3 (due Tuesday, at noon, February 18, 2014)

CS 473: Fundamental Algorithms, Spring 2014

Make sure that you write the solutions for the problems on separate sheets of paper. Write your name and netid on each sheet.

Collaboration Policy: The homework can be worked in groups of up to 3 students each.

---

1. (35 PTS.) Selection.

You are given an array A with n distinct numbers in it, and another array B of ranks $i_1 < i_2 < \ldots < i_k$. An element x of A has rank u if there are exactly $u-1$ numbers in A smaller than it. Design an algorithm that outputs the k elements in A that have the ranks $i_1, i_2, \ldots, i_k$.

(A) (30 PTS.) Describe a $O(n \log k)$ recursive algorithm for this problem. Prove the bound on the running time of the algorithm. (As a warm up first obtain an $O(nk)$ time algorithm).

(B) (5 PTS.) Show, that if this problem can be solved in $T(n,k)$ time, then one can sort n numbers in $O(n + T(n,n))$ time (i.e., give a reduction). What is an informal implication of this reduction in terms of being able to solve the problem in time faster than $O(n \log k)$ time.

No proof of correctness necessary.

2. (35 PTS.) Inversions.

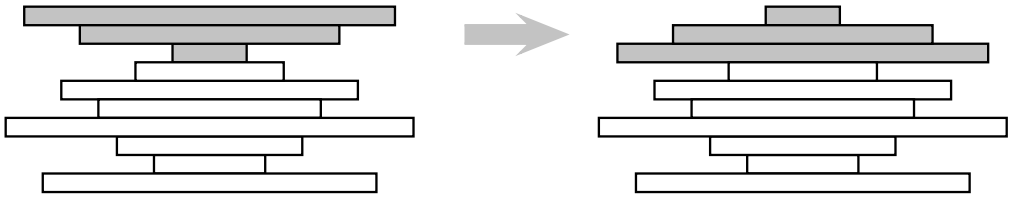
Given a sequence $a_1, a_2, \ldots, a_n$ of n distinct numbers, an *inversion* is a pair $i < j$ such that $a_i > a_j$. Note that a sequence has no inversions if and only if it is sorted in ascending order. The book by Kleinberg-Tardos describes (see Chapter 5) an $O(n \log n)$ algorithm to count the number of inversions in a given sequence. We consider two generalizations.

- (20 pts) Call a pair $i < j$ a *significant* inversion if $a_i > 2a_j$. Describe an $O(n \log n)$ time algorithm to count the number of significant inversions in a given sequence.
- (15 pts) Consider a further generalization. In addition to the sequence $a_1, \ldots, a_n$ we are given weights $w_1, \ldots, w_n$ where $w_i \geq 1$ for each i. Now call a pair $i < j$ a significant inversion if $a_i > w_j a_j$. Describe an $O(n \log^2 n)$ time algorithm to count the number of significant inversions given the sequences a and w. Extra credit (10 pts) if you can obtain an $O(n \log n)$ running time.

3. (30 PTS.) Saving the world, one pancake at a time.

Suppose we have a stack of n pancakes of different sizes. We want to sort the pancakes so that the smaller pancakes are on top of the larger pancakes. The only operation we can perform is a *flip* - insert a spatula under the top k pancakes, for some k between 1 and n, and flip them all over.

(A) (15 PTS.) Describe an algorithm to sort an arbitrary stack of n pancakes and give a bound on the number of flips that the algorithm makes. Assume that the pancake information is given to you in the form of an n element array A. $A[i]$ is a number between 1 and n and $A[i] = j$ means that the j'th smallest pancake is in position i from the bottom; in other words $A[1]$ is the size of the bottom most pancake (relative to the others) and $A[n]$ is the size of the top pancake. Assume you have the operation Flip(k) which will flip the top k pancakes. Note that you are only interested in minimizing the number of flips.



(B) (15 PTS.) Suppose one side of each pancake is burned. Describe an algorithm that sorts the pancakes with the additional condition that the burned side of each pancake is on the bottom. Again, give a bound on the number of flips. In addition to A, assume that you have an array B that gives information on which side of the pancakes are burned; $B[i] = 0$ means that the bottom side of the pancake at the i'th position is burned and $B[i] = 1$ means the top side is burned. For simplicity, assume that whenever Flip(k) is done on A, the array B is automatically updated to reflect the information on the current pancakes in A.

No proof of correctness necessary.