CS 466/666, Spring 2011
Timothy Chan

Assignment 1 (due May 25 Wednesday 5pm)

Please read `http://www.student.cs.uwaterloo.ca/~cs466/policies.html` first for general instructions.

1. [*20 marks*] Given n lists $L_1, \ldots, L_n$ where each list L_k is a permutation of $\{1, \ldots, n\}$, we consider the problem of computing the following Boolean matrix:

   $$c_{ij} = \text{true} \iff i \text{ appears before } j \text{ in at least one list } L_k$$

   for all $1 \le i, j \le n$ $(i \ne j)$. (Informally, think of $\{1, \ldots, n\}$ as "candidates" and each L_i as a "preference list"; $c_{ij} =$ false means that candidate i appears after candidate j in all lists, i.e., candidate j is strictly "better" than candidate i.) Example: for $L_1 = \langle 3, 1, 2\rangle$, $L_2 = \langle 3, 2, 1\rangle$, and $L_3 = \langle 1, 3, 2\rangle$, we have $c_{23} =$ false but all other c_{ij}'s true.

   The problem can be solved easily by a brute-force algorithm in $O(n^3)$ time. In this question, you will explore a faster algorithm.

   (a) [*6 marks*] Given index ℓ, define

   $$d_{ij}[\ell] = \text{true} \iff \text{for some } k,\ i \text{ appears before position } \ell \text{ in } L_k \text{ and } j \text{ appears after position } \ell \text{ in } L_k.$$

   Show that for one given index ℓ, we can compute $d_{ij}[\ell]$ for all i, j $(i \ne j)$ in $O(n^{2.376})$ time. (You may use the fact that two $n \times n$ matrices can be multiplied in $O(n^{2.376})$ time.)

   (b) [*5 marks*] Given a pair of indices ℓ, ℓ' $(\ell < \ell')$, define

   $$e_{ij}[\ell, \ell'] = \text{true} \iff \text{for some } k, \text{ both } i \text{ and } j \text{ appear between position } \ell \text{ and } \ell' \text{ in } L_k \text{ and } i \text{ appears before } j \text{ in } L_k.$$

   Show that for one given pair of indices ℓ, ℓ', we can compute $e_{ij}[\ell, \ell']$ for all i, j $(i \ne j)$ in $O(n(\ell' - \ell)^2)$ time.

   (c) [*7 marks*] Using (a) and (b), give an algorithm that solves our problem in total time $O(n^{2.376}d + n^3/d)$ for any given value d.

   (d) [*2 marks*] What is the best choice of d (asymptotically) and the resulting time bound?

2. [*22 marks*] Given n numbers $x_1, \ldots, x_n$, we want to cover them with unit-length intervals while minimizing the number of intervals used. Let h denote the minimum number of intervals. We *do not* assume that the input numbers $x_1, \ldots, x_n$ are given in sorted order.

   The following greedy approach is known to produce an optimal solution: find the smallest x_i, pick the interval $[x_i, x_i+1]$, remove all points covered by this interval, and repeat. (You do not need to prove the correctness of this greedy approach.) Example: for the numbers 1.2, 1.4, 2.1, 2.3, 2.5, 3.9, 4.2, 4.8, 6.1, we would pick the intervals $[1.2, 2.2]$, $[2.3, 3.3]$, $[3.9, 4.8]$, $[6.1, 7.1]$.

(a) [*2 marks*] Show that this greedy approach can be implemented in $O(n \log n)$ time.

(b) [*2 marks*] Show that the approach can be implemented in $O(nh)$ time.

(c) [*12 marks*] Give an $O(n \log h)$-time algorithm. (Hint: adapt a technique from class—namely, divide input into groups of a certain size, sort each group, ...)

(d) [*6 marks*] Prove that in terms of n (and not h), the problem requires a lower bound of $\Omega(n \log n)$ worst-case time in a comparison model of computation. You may use the known fact that the following CLOSE-PAIR problem has an $\Omega(n \log n)$ lower bound:

> CLOSE-PAIR: given n numbers $a_1, \ldots, a_n$ (not necessarily sorted) and a value $\varepsilon > 0$, decide whether there exist numbers a_i, a_j $(i \neq j)$ such that $|a_i - a_j| \leq \varepsilon$.

3. [*18 marks*] Given an ordered list L of n elements and a fixed parameter k, we want to partition L into $O(n/k)$ *blocks* so that each block contains at most $O(k)$ elements. This problem is simple if L is static, but we are interested in the setting where L undergoes insertions and deletions of elements and we want to maintain such a partition dynamically.

   We assume that during an insertion/deletion of an element x, we are given a pointer to its location in the list L, and we know which block B the element x is in. The sequence of blocks is kept in a linked list, along with the sizes of the blocks. The following method maintains the invariant $(*)$ that each block B has size between $k/3$ and $2k$ (for simplicity, assume k is divisible by 3):

```
insert(B, x):
 1.  add x to the block B
 2.  if |B| = 2k then
 3.      split B into two blocks B' and B'' of size k
delete(B, x):
 4.  delete x from block B
 5.  if |B| = k/3 then {
 6.      let B' be a block adjacent to B
 7.      merge B' and B into a block B''
 8.      if |B''| ≥ 2k then
 9.          split B into two blocks of size |B''|/2
     }
```

   (a) [*2 marks*] Show that assuming the invariant $(*)$, the number of blocks is indeed bounded by $O(n/k)$ at all times.

   (b) [*5 marks*] Define the following potential function: $\Phi = \sum_B ||B| - k|$, where the sum is over all blocks B currently in the data structure.

   Show that splitting a block of size at least $2k$ into two blocks of equal size decreases Φ by at least $\Theta(k)$.

(c) [*6 marks*] Show that merging a block of size $k/3$ with another block also decreases Φ by at least $\Theta(k)$. (Hint: $|x| - |y| \leq |x - y|$.)

(d) [*5 marks*] Assume that lines 3,7,9 (splitting a block into two, or merging two blocks) take $O(k)$ time, and lines 1,4,6 take $O(1)$ time. Show that the amortized cost for insert() and delete() is $O(1)$, using (b) and (c) with the above potential function Φ. (You may assume that initially the data structure contains just one block of size k.)