CS-473: Algorithms I Spring 2018

Assignment 4: Sorting in Linear Time

Instructor: Mehmet Koyutürk

Due Date and Instructions

Please deliver hard-copies to Utku Norman in EA-527 by 17:00 on **Friday, March 30, 2018**.

- Hand-writing is accepted but the course personnel reserves the right to reject grading an assignment if the hand-writing is not legible.
- Assignments written in LaTeX will receive 5 bonus points.
- You can submit the assignment as a team of 2 students, however no pair of students are allowed to work together on more than 3 assignments.
- You are also allowed to talk to other students and the course personnel about the solutions, but you must write the answers yourself. Answers copied from other students or resources will be detected, and appropriate action will be taken.

Problem 1

[30 pts.] Given an array A of n integers in the range $[0, k)$, we would like to build an index, which we would like to use to answer any query of type "what is the number of integers in A that are in the range $[a, b]$?" For this purpose, write the following two procedures:

- Procedure PreProcess(A) should process A to build an index in $O(n + k)$ time. The size of your index should be $O(k)$.
- Procedure Query(A, a, b) should return the number of integers in A that are in the range $[a, b]$ in $O(1)$ time.

Problem 2

[50pts] Let A be an $m \times n$ matrix. The *transpose* of A is an $n \times m$ matrix A' such that for all $0 \leq i < n$ and $0 \leq j < m$, $A'(i, j) = A(j, i)$. For example, if

$$A = \begin{bmatrix} 0 & 9 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 3 & 8 & 0 & 0 \end{bmatrix} \tag{1}$$

then

$$A' = \begin{bmatrix} 0 & 0 & 3 \\ 9 & 0 & 8 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix} \tag{2}$$

Let k denote the number of non-zero entries in $m \times n$ matrix A (in the above example, $k = 5$). We say that A is *sparse* if $k = o(mn)$. Clearly, it is more efficient to store a sparse matrix using a special data structure, instead of storing it as a 2-dimensional array. One common data structure is known as the *compressed row storage (CRS)* (or alternatively, *compressed sparse row (CSR)*).

In compressed row storage (CRS) format, matrix A is stored using three arrays: R, C, and V. These arrays are respectively of length $m + 1$, k, and k. They are store Row pointers, Column indices, and Values, respectively. The array C stores the column indices of the non-zeros, such that for each $0 \leq i \leq m - 1$, the subarray $C[R[i], \ldots, R[i + 1] - 1]$ stores the column indices of the ith row of A, in increasing order (for convenience, the indexing of the arrays starts from 0). For each $C[j]$, $V[j]$ stores the value of the corresponding non-zero entry, i.e., if $R[i] \leq j < R[i + 1]$, then $V[j] = A(i, C[j])$. For example, the CRS of matrix A in Equation (1) is as follows:

$$\begin{array}{ll} R = & < 0, 2, 3, 5 > \\ C = & < 1, 3, 2, 0, 1 > \\ V = & < 9, 1, 1, 3, 8 > \end{array} \tag{3}$$

For this problem, you are asked to write the algorithm for transposing a matrix in CRS representation. In other words, write the pseudo-code of procedure SPARSE-TRANSPOSE(R, C, V, m, n, k) that will return arrays R', C', and V', representing the transpose of the matrix represented by R, C, and V in CRS format. For example, if your procedure is called with the R, C, and V in Equation (3), it should return:

$$\begin{array}{ll} R' = & < 0, 1, 3, 4, 5 > \\ C' = & < 2, 0, 2, 1, 0 > \\ V' = & < 3, 9, 8, 1, 1 > \end{array} \tag{4}$$

which is the row-major representation of A' in Equation (2). Your algorithm should work in $O(m + n + k)$ time. (*Hint:* The algorithm can be thought of as a generalization of COUNTING-SORT).

Problem 3

[20 pts.] Assume that you are given a procedure MYSTERY-SORT(A), which takes an array A of length n as input, sorts the numbers in A in non-decreasing order, and returns the sorted array. You do not know whether the algorithm implemented by MYSTERY-SORT is stable.

Dr. Bluecrane needs a procedure that takes an array of n integers and returns the sorted array in non-decreasing order, but she needs the procedure to be stable. To help Dr. Bluecrane, write the pseudo-code of a stable sorting procedure STABLE-SORT(A), which pre-processes and/or post-processes the elements in A in $O(n)$ time, makes only one call to MYSTERY-SORT, and returns the sorted array in non-decreasing order, such that the ordering of identical elements is preserved. (*Hint:* No comparison is necessary. Note also that the solution to this problem represents an implementation trick rather than an algorithmic insight, so it is mainly intended for fun).