CS 577: Introduction to Algorithms — Homework 2

Out: 02/17/2022 — **Due: 03/03/2022**

Ground Rules

- Read the lecture notes and from the book: II Sorting and Statistics, and Hashing, Binary Search Trees.
- You should do ***all*** problems, but hand in only the graded problems.
- Write your name and student ID clearly on your submission.
- Clearly mark the beginning and end of your solution to each problem.

1. You are given $2n$ points on the plane lying on two parallel lines $y = 0$ and $y = 1$. n distinct points have x-coordinates $(p_1, p_2, \ldots, p_n)$ on the line $y = 0$ and another n distinct points have x-coordinates $(q_1, q_2, \ldots, q_n)$ on the line $y = 1$. Create a set of n line segments by connect each point $(p_i, 0)$ to the corresponding point $(q_i, 1)$. We would like to compute how many pairs of these line segments intersect in $O(n \log n)$ time.

   (a) Prove that, to solve this problem, we might as well assume $(p_1, p_2, \ldots, p_n)$ is sorted in increasing order. Explain why this is in fact without loss of generality. In fact, further prove that we could just as well assume $(p_1, p_2, \ldots, p_n)$ is the sequence $(1, 2, \ldots, n)$ on the real line (i.e., $y = 0$).

   (b) Describe and analyze a divide and conquer algorithm to determine how many pairs of these line segments intersect, in $O(n \log n)$ time. Your algorithm takes the $2n$ points as input, and return the number of intersections.

2. **Graded Problem (Page limit: 2 pages)**

   In this problem you are given an $n \times m$ table (a two-dimensional array), where every row is sorted in increasing order, and every column is also sorted in increasing order. We want to design and analyze an efficient algorithm that searches for a given value x in an $n \times m$ table (a two-dimensional array).

   (a) Use Divide-and-Conquer to design an algorithm, where the first step (some comparison) should cut the table into some pieces that you can discard a piece of A after the result of this comparison is known.

   (b) Now we assume $n = m$. Now suppose we want to sort this table, to produce a sorted array of n^2 elements. Suppose the only way to access the table elements is to make comparisons, i.e., we are in the decision-tree model, where every step is to make a comparison of two elements of the table. Clearly you can do so with $\Theta(n^2 \log(n^2)) = \Theta(n^2 \log n)$ steps. (Why?) Suppose someone claims that he can do so in $O(n)$ comparisons. Prove that he is wrong, and that is impossible in the worst case.

   (c) Suppose someone claims that he can do this sorting in $o(n^2 \log n)$ steps. Is that possible, in the worst case?

3. We define binomial-max-heaps $\{B_i\}_{i \geq 0}$ as follows: B_0 is a tree consisting of a single vertex r_0, also its root, which stores its unique key value v_0. If S and T are two binomial-max-heaps B_i with roots R and r, and suppose their values are $V \geq v$ respectively, then by adding a (directed) edge from r to R we form a binomial-max-heap B_{i+1} with root R.

   (a) Prove that a binomial-max-heap B_i has 2^i vertices, is in general not a binary tree but saisfies the Max-Heap property as defined in class.

   (b) Given $n = 2^k$ distinct integers, show that we can form a binomial-max-heap B_k containing these integers as keys, using $n - 1$ comparisons.

   (c) We are given $n = 2^k$ distinct integers, for some $k \geq 1$. Design a comparison algorithm that finds both the maximum and the second maximum element using $n + k - 2$ comparisons.

(d) Design a data structure using binomial-max-heaps to accomplish the following tasks: This data structure is expected to handle a sequence of $O(n)$ operations of either insertions and delete-max steps, on data items. Each data item has a key, and delete-max (on a nonempty set represented by the data structure) should delete the data item that has the maximum key value. We want the cost to build the data structure initially with n data items to be of order $O(n)$. Subsequently, every insertion and delete-max operation should take time $O(\log n)$.

Prove your design correct and prove your time bounds.

4. We are given an array $A[1 : n]$ of $n = 2^k$ distinct integers. Our task is to find both the maximum and the minimum integers of the array. A naive algorithm can find the maximum and the minimum separately in $n - 1$ comparisons each, resulting in $2n - 2$ comparisons.

   (a) Devise a Divide-and-Conquer algorithm that finds both the maximum and the minimum using at most $\frac{3}{2}n - 2$ comparisons.

   Prove that your algorithm is correct, and makes at most this many comparisons.

   (b) Prove that no comparison-based algorithm can find both the maximum and the minimum using fewer than $\frac{3}{2}n - 2$ comparisons in the worst case.

   (Hint: Let U be the potential candidates of the maximum of A *only*, and D be the potential candidates of the minimum of A *only*, and O be the elements of A that are no longer candidates for either. Analyze how the "potential function" $2|O|+|U|+|D|$ can change during the course of any comparison-based algorithm for this problem.)

5. **Graded Problem (Page limit: 2 pages)** An $n \times n$ grid is a graph whose node set is the set of all ordered pairs of natural numbers (i, j) with $1 \le i \le n$ and $1 \le j \le n$; the nodes (i, j) and (k, ℓ) are joined by an edge if and only if $|i - k| + |j - \ell| = 1$. In other words, it is the adjacency graph of an $n \times n$ chessboard.

   In this problem you are given an $n \times n$ grid with each node v labeled by a distinct number x_v. v is called a *local maximum* if x_v is greater than the labels of each of v's neighbors. Develop a Divide-and-Conquer algorithm for finding a local maximum in the grid that runs in $O(n)$ time. (Your algorithm is successful if it finds any local maximum. Recall that the number of nodes in the grid is n^2.)

   (a) A natural idea might be that, one starts at some vertex, say the upper left corner, and repeatedly do the following: If the current vertex v is a local maximum, then you are done. Otherwise go to the neighbor of v that has the maximum label among all neighbors of v. (Note that a vertex v might have 2 or 3 or 4 neighbors.)

   Prove that this algorithm does find a local maximum eventually. But give an example where this algorithm takes time $\Omega(n^2)$.

   (b) Devise your Divide-and-Conquer algorithm that runs in $O(n)$ time. Describe your algorithm succinctly but precisely.

   (c) Prove its correctness.

   (d) Analyze its running time. Prove that your algorithm runs in $O(n)$ time.