IITM-CS2800-2025 : Design and Analysis of Algorithms

Practice Problem Sheet #4 Released On : Feb 21, 10:30 PM

1. A *Toeplitz matrix* is an $n \times n$ matrix $A = (a_{ij})$ such that $a_{ij} = a_{i-1,j-1}$ for every $i, j \in \{2, 3, \ldots n\}$. Here is an example:

$$A = \begin{pmatrix} a & b & c & d \\ e & a & b & c \\ f & e & a & b \\ g & f & e & a \end{pmatrix}$$

   (a) Is the sum of two Toeplitz matrices necessarily Toeplitz? What about the product?

   (b) Describe how to represent a Toeplitz matrix such that two Toeplitzes can be added in linear time. (Compare this with $\Omega(n^2)$ lower bound that we discussed in class for adding two arbitrary $n \times n$ matrices).

   (c) Given an $O(n \log n)$ algorithm for multiplying a Toeplitz matrix with a vector T of length n (assume $n = 2^k$ for simplicity). (Hint: Use your representation from the previous part of this question). Hint: Decompose A as a matrix of blocks of size 2^{k-1}, decompose T accordingly:

$$A = \begin{pmatrix} E & F \\ G & E \end{pmatrix} \text{ and } T = \begin{pmatrix} X \\ Y \end{pmatrix}$$

   and consider the three matrices $U = (G + E)X$, $V = E(Y - X)$, and $W = (E + F)Y$.

   (d) Give an efficient algorithm for multiplying two Toeplitz matrices using part (c).

2. Using the method that we discussed in class, find the unique polynomial of degree 4 that takes on values $p(1) = 2$, $p(2) = 1$, $p(3) = 0$, $p(4) = 4$, and $p(5) = 0$.

3. In justifying our matrix multiplication algorithm, we claimed the following block-wise property: if X and Y are $n \times n$ matrices, and

$$X = \begin{pmatrix} A & B \\ C & D \end{pmatrix} \text{ and } Y = \begin{pmatrix} E & F \\ G & H \end{pmatrix}$$

   where A, B, C, D, E, F and H are $\frac{n}{2} \times \frac{n}{2}$ submatrices, then the product XY can be expressed in terms of these blocks:

$$XY = \begin{pmatrix} A & B \\ C & D \end{pmatrix} \begin{pmatrix} E & F \\ G & H \end{pmatrix} = \begin{pmatrix} AE + BG & AF + BH \\ CE + DG & CF + DH \end{pmatrix}$$

   Prove this property formally.

4. (*k-way* **merging**) Consider the following sorting algorithm (see Algorithm 1) which takes a set of n integer and outputs non-decreasing order of the same. (Assume that k divides n). The

Algorithm 1: : (k-way Merge-Sort)

1: Divide the given n integers into k groups of each size $\frac{n}{k}$.
2: Sort elements of each group by insertion sort.
3: Merge the elements of sorted k groups by k-way merge operation.

k-way merge operation is similar to 2-way merge in merge-sort, however, it takes as input k sorted lists and outputs their merged list.

(a) Design an algorithm for the k-way merge operation that runs in $O(nk)$ time.

(b) Design an algorithm for the k-way merge operation that runs in $O(n \log k)$ time

(c) Find the time complexity of the sorting algorithm and also compute the value of k where the above sorting algorithm faster than the insertion sort.

5. You are given an infinite array A in which the first n cells contain integers in sorted order and the rest of the cells are filled with ∞. You are not given the value of n. Describe an algorithm that takes an integer x as input and finds a position in the array containing x, if such a position exists, in $O(\log n)$ time. (If you are disturbed by the fact that the array A has infinite length, assume instead that it is of length n, but that you do not know this length, and that the implementation of the array data type in your programming language returns the error message ∞ whenever elements $A[i]$ with $i > n$ are accessed.)

6. Given an array A of size n, the task is to find the length of the Longest Increasing Contiguous Subsequence (LICS) i.e., the longest possible contiguous subsequence in which the elements of the subsequence are sorted in increasing order. For example, for the input array $(3, 10, 2, 1, 20)$, the output is 2 since the sequence $(3, 10)$ is sorted in the increasing order and the sequence $(3, 10, 20)$ is the not contiguous although it is a sorted subsequence.

   Design an algorithm that uses divide and conquer approach to achieve this in $O(nlogn)$ time.

7. Given a matrix A of size $n \times n$, the task is to find the determinant of A.

   (a) Write down a recursive algorithm for solving this problem. *Hint: simply use the high school method of expanding the determinant and implement it recursively.*

   (b) Write down the recurrence relation for the running time.

   (c) Solve the recurrence relation and conclude that the running time is $2^{O(n \log n)}$.

   There is an $O(n^3)$ algorithm which we will discuss later in the course.