Design and Analysis of Algorithms

Practice-sheet : **Divide and Conquer**

Miscellaneous

1. **Counting Double-Inversions**

   You are given an array A storing n numbers. A pair (i, j) with $0 \leq i < j \leq n-1$ is said to be a double-inversion if $A[i] > 2A[j]$. Design and analyze an $O(n \log n)$ time algorithm based on divide and conquer paradigm to compute the total number of double-inversions in A.

2. **Local minima in an array**

   Let $A[1..n]$ be an array storing n distinct numbers. An index $1 \leq i \leq n$ is said to be a local minima if $A[i]$ is strictly smaller than its neighbors, if exists, that is, $A[i-1]$ and $A[i+1]$. Design an $O(\log n)$ time algorithm to find one local minima in A.

3. **Local minima in a complete binary tree**

   Consider an n-node complete binary tree T, where $n = 2^d - 1$ for some d. Each node v of T is labeled with a real number x_v. You may assume that the real numbers labeling the nodes are all distinct. A node v of T is a local minimum if the label of x_v is less than the label x_w for all nodes w that are joined to v by an edge.

   You are given such a complete binary tree T, but the labeling is only specified in the following implicit way: For each node v,you can determine the value x_v by probing the node v. Show how to find a local minimum of T using only $O(\log n)$ probes to the nodes of T.

4. **Real life problem of divide and conquer**

   Suppose you are consulting for a bank that is concerned about fraud detection, and they come to you with the following problem. They have a collection of n bank cards that they have confiscated, suspecting them of being used in fraud. Each bank card is a small plastic object, containing a magnetic strip with some encrypted data, and it corresponds to a unique account number in the bank. Each account can have many bank cards corresponding to it, and we'll say that two cards are equivalent if they correspond to the same account.

   It is difficult to read the account number off a bank card directly, but the bank has a high tech "equivalence tester" that takes two bank cards and, after performing some computations, determines whether they are equivalent.

   The question is the following: among the collection of n cards, is there a set of more than $n/2$ of them that are equivalent to one another ? Assume that the only feasible operations you can do with the cards are to pick two of them and plug them into equivalence tester. Design a divide and conquer based algorithm to decide the answer of their question with only $O(n \log n)$ invocations of the equivalence tester.

Geometric Problems

1. **Closest Pair Distance**
   The following problems are based on the divide and conquer algorithm for computing closest pair distance discussed in the class.

   (a) Write a neat and complete pseudocode of the algorithm we discussed in the lecture (without referring to the slides).

   (b) The algorithm we discussed in the lecture assumed that the median line passes through only one point. What if the median line passes through multiple points ? In particular, how will you ensure that the left half as well as right half has $\lceil n/2 \rceil$ points given that the median line passes through multiple points ?

2. **Non-dominated points**

   In the lectures, we discuss a divide and conquer based algorithm to compute non-dominated points. However, there is an alternate and equally simple algorithm for this problem that runs in $O(n \log n)$ time. Provide complete description and analysis of this algorithm.

   Hint: The algorithm makes use of sorting as the first step.

3. **Least perimeter triangle***
   Let P be a set of n points in a plane. Design an $O(n \log n)$ algorithm to find the least perimeter triangle out of all possible triangles defined by P. Assume, without loss of generality, that there are no three collinear points in P.

   Hint: If you fully internalized the algorithm for closest pair problem, then you just have to pursue the same direction and make minor changes.

Problems involving polynomial multiplication

1. **Finding Polynomial given all its zero's**

   Given a list of values $z_0, \cdots, z_{n-1}$ (possibly with repetitions), show how to find the coefficients of the polynomial $P(x)$ of degree less than n that has zeros only at $z_0 \ldots, z_{n-1}$ (possibly with repetitions). Your procedure should run in time $O(n \log^2 n)$. *Hint:* The polynomial $P(x)$ has a zero at z_j if and only if $P(x)$ is a multiple of $(x - z_j)$.

2. Consider two sets A and B, each having n integers in the range from 0 to $10n$. We wish to compute the **Cartesian** sum of A and B, defined by

   $$C = \{x + y \ : \ x \in A \ \text{ and } \ y \in B\}.$$

   Note that the integers in C are in the range from 0 to $20n$. We want to find the elements of C and the number of times each element of C is realized as a sum of elements in A and B. Design an $O(n \log n)$ time algorithm to achieve this objective.

3. **Toeplitz Matrix**

   A Toeplitz matrix is an $n \times n$ matrix $A = (a_{i,j})$ such that $a_{i,j} = a_{i-1,j-1}$ for $i = 2, 3, \ldots, n$ and $j = 2, 3, \ldots, n$.

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

   (b) Describe how to represent a Toeplitz matrix so that two $n \times n$ matrices can be added in $O(n)$ time.

   (c) Give an $O(n \log n)$ time algorithm for multiplying an $n \times n$ Toeplitz matrix by a vector of length n. Use your representation from part (b).

   Hint: Using part (b), express this problem as a multiplication of two *suitably defined* polynomials.

4. Give an efficient algorithm for multiplying two $n \times n$ Toeplitz matrices. Analyze its running time.

5. **Cartesian sum**

   Consider two sets A and B, each having n integers in the range from 0 to $10n$. We wish to compute the **Cartesian** sum of A and B, defined by

   $$C = \{x + y \ : \ x \in A \ \text{ and } \ y \in B\}.$$

   Note that the integers in C are in the range from 0 to $20n$. We want to find the elements of C and the number of times each element of C is realized as a sum of elements in A and B. Design an $O(n \log n)$ time algorithm to achieve this objective.

   Hint: Express this problem as a multiplication of two *suitably defined* polynomials.

6. **A computational physics problem**

   You have been working with some physicists who need to study, as part of their experimental design, the interactions among large numbers of very small charged

particles. Basically, their setup works as follows. They have an inert lattice structure, and they use this for placing charged particles at regular spacing along a straight line. Thus we can model their structure as consisting of the points $\{1, 2, 3, ...n\}$ on the real line; and at each of these points j, they have a particle with charge q_j. (Each charge can be either positive or negative.)

They want to study the total force on each particle, by measuring it and then comparing it to a computational prediction. This computational part is where they need your help. The total net force on particle j, by Coulomb's Law, is equal to

$$F_j = \sum_{i<j} \frac{Cq_iq_j}{(j-i)^2} - \sum_{i>j} \frac{Cq_iq_j}{(j-i)^2}$$

They have written the following simple program to compute F_j for all j.

```
For j = 1,2,...,n
    Initialize F_j to 0
    For i = 1, 2, ..., n
        If i < j then
            Add Cq_iq_j/(j-i)^2 to F_j
        Else  if i > j  then
            Add -Cq_iq_j/(j-i)^2 to F_j
        Endif
    Endfor
    Output F_j
Endfor
```

It is not hard to observe that the running time of the above algorithm is $\Theta(n^2)$. The trouble is, for the large values of n they are working with, the program takes several minuted to run. On the other hand, their experimental setup is optimized so that they can throw down n particles, perform the measurements, and be ready to handle n more particles within a few seconds. So they would really like it if there were a way to compute all the forces F_j much more quickly, so as to keep up with the rate of the experiment. Help them out by designing an algorithm which is much faster than $O(n^2)$. *Hint:* Express this problem as a multiplication of a few *suitably defined* polynomials.