**Design and Analysis of Algorithms (CS345A)**

# Practice-sheet : **Divide and Conquer**

1. **Counting inversions**

   Given an array $A$ storing $n$ distinct numbers. A pair $(i, j)$ where $0 \leq i < j \leq n-1$, is said to be an inversion if $A[i] > A[j]$. Design an $O(n \log n)$ time algorithm to count all inversions in $A$.

2. **Finding the missing element**

   Given an array that represents elements of arithmetic progression in order. One element is missing in the progression, find the missing number in $O(\log n)$ .

3. **Least parimeter 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 $P$.

4. **Binary search in 2 arrays**

   There are 2 sorted arrays A and B of size n each. Design an algorithm to find the median of the array obtained after merging the above 2 arrays (i.e. array of length $2n$). The time complexity of the algorithm should be $O(\log n)$

5. **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 $T$.

6. **Local minima in an array**

   There is an array $A$ storing $n$ distinct numbers. An index $i$ of the array is said to be local minima if the element stored at this inded is smaller than its left neighbour as well as right neighbour, if exist. Design an $\mathcal{O}(\log n)$ time algorithm to compute a local minima in $A$. What about extending this problem to "local minima in a matrix" ?

7. **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 $(x - z_j)$).

8. **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.

9. **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 $O(n^2)$.

# 1 Problems for fun only (not for exams)

1. **Binary Search without division operation** Design an algorithm that, given an array of $n$ distinct numbers in ascending order, determines whether a given integer is in the array. You may use only additions and subtractions. You are allowed to use $O(\log n)$ extra space. The running time of your algorithm should be $O(\log n)$ in the worst case. (One can avoid using extra $O(\log n)$ space)
   **Hint:** Try to get a better insight into the Binary search algorithm which you know. How crucial is the *division* operation there ? Knowledge of Fibonacci numbers might be helpful.