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. **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 $O(\log n)$

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

5. **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).

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

6. **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.

7. **Share market**

   There is an array $A[1..n]$ storing the prices of a share on n consecutive days. If we buy a share on ith day and sell it on jth day, we incur a profit of $A[j] - A[i]$. Note that it may be loss as well if $A[j] < A[i]$. Our aim is to compute the maximum possible profit we can obtain by buying a share on some day and selling it on some other (same/later) day. Note that we are allowed to buy on exactly one day and sell on exactly one day. The following is the pseudo-code of a **divide and conquer** algorithm to compute this maximum profit for a period $[i, j]$ where $1 \leq i \leq j \leq n$. Fill in the blanks suitably. You may use <u>min</u> or <u>max</u> operator according to your convenience. But you may not add any extra statement.

```
if i = j then
    return (A[i],A[j],0)
else
    k ← (i + j)/2;
    (low_L, high_L, profit_L) ← HighestProfit(A,....,....);

    (low_R, high_R, profit_R) ← HighestProfit(A,....,....);

    low ← ...................;

    high ← ....................;

    profit ← ..................................................;
    return (low, high, profit);
end
```

Algorithm 1: $HighestProfit(A, i, j)$

The time complexity of this algorithm is ....

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.