COL351: Analysis and Design of Algorithms — Announced on: July 27, 2026 (Monday)

Tutorial Sheet 2

Problems marked with ($\star$) will not be asked in the tutorial quiz.

1. Consider the following algorithm for calculating a^b where a and b are positive integers.

```
ALGORITHM 1: FastPower(a, b)
Input: Positive integers a and b.
Output: a^b.
1 if b = 1 then
2     return a
3 c := a · a
4 d := FastPower(c, ⌊b/2⌋)
5 if b is odd then
6     return a · d
7 return d
```

   Suppose each multiplication and division operation can be performed in constant time. Determine the asymptotic running time of `FastPower` as a function of b.

2. Design an $\mathcal{O}(\log^2 n)$ algorithm that, given a positive integer n, determines whether n is of the form a^b for some positive integers a and $b > 1$. For the purpose of this problem, you may assume exponentiation to be $\mathcal{O}(1)$ time, i.e., computing p^q for two integers p and q takes constant time. Similarly, you can assume that the comparison of two integers (i.e., determining whether p equals q, $p > q$ or $p < q$) takes constant time.

3. Consider the following modification to the `MergeSort` algorithm: Divide the input array into *thirds* (rather than halves), recursively sort each third, and combine the results using a three-way `Merge` subroutine. What is the number of pairwise comparisons made by this algorithm as a function of the length n of the input array, ignoring constant factors and lower-order terms?

   (For the purpose of the tutorial quiz, you don't need to write the algorithm or its correctness analysis. Just highlight the key steps in your argument.)

4. You are given an n-by-n grid of distinct numbers. A number is a *local minimum* if it is smaller than all its neighbors. A *neighbor* of a number is one immediately above, below, to the left, or to the right. Most numbers have four neighbors; numbers on the side have three; the four corners have two.

(a) Prove that a local minimum always exists.

(b) Use the divide-and-conquer algorithm design paradigm to compute a local minimum with only $\mathcal{O}(n)$ comparisons between pairs of numbers. (Note: since there are n^2 numbers in the input, you cannot afford to look at all of them.)

5. Given an array of n objects, your task is to decide if there is an object that occurs strictly more than $\lfloor n/2 \rfloor$ times. The only operation by which you can access the objects is a function f that, given two indices i and j, outputs whether the objects at positions i and j in the array are identical or not. Design an $\mathcal{O}(n \log n)$-cost algorithm for this problem, assuming each call to f is counted as a unit-cost operation.

6. ($\star$) You are given a sequence of n numbers $a_1, a_2, \ldots, a_n$. Design an $\mathcal{O}(n)$ algorithm to find i, j with $i \leqslant j$ such that the sum $a_i + a_{i+1} + \cdots + a_j$ is maximum. Note that the numbers may not be positive.

7. ($\star$) You are given as input an unsorted array of n distinct numbers, where n is a power of 2. Give an algorithm that identifies the second-largest number in the array while using at most $n + \log_2 n - 2$ pairwise comparisons.

8. ($\star$) Consider a tournament with n teams in which each team plays every other team exactly once, and each game has one winner (i.e., no ties). Unfortunately, the tournament is being held behind closed doors, and you are unable to attend the games. You can, however, call the organizers and, on each call, ask for the result of exactly one game. Say you want to find out if there is an *undisputed* champion, which is a team that wins all its games. Show that you can determine the existence of such a team with at most $2n - \lfloor \log_2 n \rfloor - 2$ calls.