1. The goal is to design an algorithm to find the second largest element out of a set of \( n \) elements and to minimize the \textbf{exact} number of comparisons required for this task. But we will do it in stages: (i) Suppose we use merge-sort first, how many comparisons would be needed? Try to find out as close to an exact bound as possible rather than an asymptotic bound. (ii) Next we use the linear time selection algorithm for this task, again try to find out as close to an exact bound as possible rather than an asymptotic bound. (iii) Start from scratch and try to find a direct approach for this problem that is very efficient, something like \( n + \log n \) comparisons required at most. Explain this algorithm in detail and derive the exact number of comparisons. Compare the bounds you got in the three different subproblems.

2. What is the smallest number of comparisons required to sort 4 numbers and why? Give an algorithm that sorts 4 numbers in the smallest number of comparisons required.

3. Find a sorting algorithm from the book whose asymptotic average-case complexity is better than asymptotic its worst-case complexity. State the algorithm, its complexities and suggest one way in which the worst-case complexity could be made to match its average-case complexity.

\textbf{Note:} Remember the academic honesty policy for the course.