1 Searching an Array Randomly (25 points)

Let \( A \) be an unsorted array of length \( n \), where each entry of \( A \) is an integer. Suppose that we are looking for some integer \( x \) in \( A \), i.e., we want to find an index \( i \) such that \( A[i] = x \) if such an index exists. If no such index exists, we should return False. Consider the following randomized algorithm.

- Initially, all indices are unmarked.
- While not all indices are marked:
  - Pick an index \( i \in [n] \) uniformly at random.
  - If \( A[i] = x \) return \( i \).
  - Else mark index \( i \).
- Return false

Note that in each iteration we pick an index \( i \) uniformly at random from \([n]\), not from the set of unmarked indices. So we might examine a given index more than once.

(a) (12 points) Suppose that \( x \) appears in \( k \geq 1 \) places in \( A \), i.e., \( |\{i : A[i] = x\}| = k \). What is the expected running time of this algorithm, as a function of \( n \) and \( k \)?

(b) (13 points) Suppose that \( x \) does not appear in \( A \). What is the expected running time of the algorithm?

2 Median of Sorted Arrays (25 points)

Let \( A \) and \( B \) be sorted arrays of \( n \) elements each. We can easily find the median of \( A \) or the median of \( B \), since they are already sorted – it will be at index \( \lfloor \frac{n+1}{2} \rfloor \). But what if we want to find the median element of \( A \cup B \)? We could just concatenate them and use the \( O(n) \) time median algorithm, but is it possible to do better? In this problem you should give matching upper and lower bounds, i.e. you should find a function \( f(n) \) and do the following:

(a) (13 points) Design a deterministic algorithm whose running time (measured in terms of the number of comparisons) is \( O(f(n)) \), and

(b) (12 points) Give a lower bound showing that any comparison-based algorithm must make \( \Omega(f(n)) \) comparisons in the worst case.
3 More Lower Bounds (25 points)

Consider the following two-dimensional sorting problem: we are given an arbitrary array of \( n^2 \) numbers (unsorted), and have to output an \( n \times n \) matrix of the inputs in which all rows and columns are sorted.

As an example, suppose \( n = 3 \) so \( n^2 = 9 \). Suppose the 9 numbers are just the integers \{1, 2, \ldots , 9\}. Then possible outputs include (but are not limited to)

\[
\begin{array}{ccc}
1 & 4 & 7 \\
2 & 5 & 8 \\
3 & 6 & 9 \\
\end{array}
\quad
\begin{array}{ccc}
1 & 3 & 5 \\
2 & 4 & 6 \\
7 & 8 & 9 \\
\end{array}
\quad
\begin{array}{ccc}
1 & 2 & 6 \\
3 & 4 & 8 \\
5 & 7 & 9 \\
\end{array}
\]

It is obvious that we can solve this in \( O(n^2 \log n) \) time by sorting the numbers and then using the first \( n \) as the first row, the next \( n \) as the second row, etc. For this question, you should prove a matching lower bound of \( \Omega(n^2 \log n) \) in the comparison-based model. For simplicity, you can (as always) assume that \( n \) is a power of 2.

Hints: instead of reasoning directly about the decision tree, show that if we could solve this problem with \( o(n^2 \log n) \) comparisons we could break the sorting lower bound. Useful facts to keep in mind are that \( n! > (n/e)^n \) and that we can merge two sorted arrays of length \( n \) using \( 2n - 1 \) comparisons.

4 Linear Time Algorithms (25 points)

Let \( A \) be a list of \( n \) (not necessarily distinct) integers. Design a (deterministic) algorithm to test whether any item occurs more than \( \lceil n/2 \rceil \) times in \( A \) that runs in \( O(n) \) time (and prove correctness and running time).