**Problem 1: Analysis of Recursive Algorithm (25 points)**
Consider the function Mystery defined below.
Mystery($n$)
if ($n = 0$) then write($x$) else begin
for $k := 1$ to $2^n$
Mystery($n - 1$)
end
If we call Mystery($n$), where $n$ is an integer $n \geq 1$, how many $x$'s (as a function of $n$) does call Mystery($n$) print? Justify your answer/show your work, ie give recurrence and solve by substitution.
Give exact solution, do not give $O()$.

**Answer:** We have to solve $T(n) = 2^n T(n-1)$, with $T(0) = 1$. Proceed to solve using substitution.

$$
\begin{aligned}
T(n) &= 2^n T(n-1) \\
&= 2^n 2^{n-1} T(n-2) \\
&= 2^n 2^{n-1} 2^{n-2} T(n-3) \\
&= 2^n 2^{n-1} 2^{n-3} 2^{n-3} T(n-4) \\
& \quad \text{guessing general term} \\
&= 2^n 2^{n-1} 2^{n-3} \ldots 2^{n-k+1} T(n-k) \\
& \quad \text{know that } T(0) = 1\text{, so stop when } n-k=0\text{, equivalently } k=n \\
&= 2^n 2^{n-1} 2^{n-3} \ldots 2^2 2^1 T(0) \\
&= 2^{\sum_{i=1}^{n} i} \\
&= 2^{\frac{n(n+1)}{2}}
\end{aligned}
$$

Sanity test: For $n = 0$ the solution $T(n) = 2^{\frac{n(n+1)}{2}}$ becomes $T(0) = 2^{\frac{0(0+1)}{2}} = 2^0 = 1$: Correct.
For $n = 1$ the solution $T(n) = 2^{\frac{n(n+1)}{2}}$ becomes $T(1) = 2^{\frac{1(1+1)}{2}} = 2^1 = 2$: Correct.

**Problem 2: Sorting Techniques (25 points)**
You are given a list of distinct numbers $a_1, \ldots, a_m$, where $m = n + \frac{n}{\log n}$.
The fist $n$ numbers are sorted in increasing order: $a_1 < a_2 < \ldots < a_n$.
The last $\frac{n}{\log n}$ numbers are not sorted.
Give an $O(n)$ comparison algorithm that sorts the entire list $a_1 < a_2 < \ldots < a_m$ in increasing order.
Justify correctness and running time.

**Answer:**
**First Way to Answer**
For any integer $N$, we can use mergesort and sort $N$ elements using $O(N \log N)$ comparisons.
We may thus sort the last $N = \frac{n}{\log n}$ elements of the array in time:
$O(N \log N) = O\left(\frac{n}{\log n} \log(\frac{n}{\log n})\right) < O\left(\frac{n}{\log n} \log n\right) = O(n)$.

We now have $n$ sorted elements and an additional $N = \frac{n}{\log n}$ sorted elements.

We can then use the procedure merge, to merge the above sorted lists and produce a completely sorted output $a_1 < a_2 < \ldots < a_m$. The number of comparisons for this step is $O(n+N) = O(n+\frac{n}{\log n}) < O(n+n) = O(n)$.

**Second Way to Answer**
The first part of the list is already sorted and has length $O(n)$.
Using binary search, in $O(\log n)$ comparisons, we find the correct position and insert each element of the second part of the list to the first part of the list.
This is repeated $\frac{n}{\log n}$ times, for a total running tims of $O(\log n \frac{n}{\log n}) = O(n)$.

**Problem 3: Longest Path in DAG, Dynamic Programming (25 points)**
You are given a directed acyclic graph in adjacency matrix representation, and so that edges are always directed from lower order to higher order vertices. In particular, where $n$ is the number of vertices, you are given an $n \times n$ matrix $A$, where:
(a) $a_{ij} = 0$ for all $i \geq j$
(b) $a_{ij} = 1$ for $i < j$, when is an edge from $i$ to $j$
(c) $a_{ij} = 0$ for $i < j$, when there is no edge from $i$ to $j$
Give an $O(n^2)$ algorithm that finds the length of the longest path from vertex 1 to vertex $n$, if such a path exists, and returns $-\infty$ if there is no path from vertex 1 to vertex $n$.
Justify correctness and running time.
Note: The length of a path is the total number of the edges of the path.

**Answer:**
**First Way to Answer**
RECURSIVE FORM OF THE SOLUTION:
Let $L(i)$ be the length of the longest path from $i$ to $n$, $1 \leq i \leq n$.
We know that $L(n) = 0$ and we want $L(1)$.
Also, initially, all we know about $L(i)$, $1 \leq i \leq n-1$ is $L(i) = -\infty$.
Assuming that we have correctly computed $L(n), L(n-1), L(n-2), \ldots, L(k+1)$, then $L(k) = \max_{k' > k : a_{kk'} = 1} \max\{1 + L(k')\}$.

We may now use dynamic programming and write:
INITIALIZATION: $L(n) := 0$
for $k := 1$ to $n$ set $L(k) := -\infty$
MEMOIZATION: for $k := (n-1)$ to 1
for $k' := (k+1)$ to $n$
if $(a_{kk'} = 1)$ then $L(k) := \max\{L(k), 1 + L(k')\}$
OUTPUT: return($L(1)$)

COMPLEXITY ANALYSIS: Initialization is a single loop, so $O(n)$.
Memoization is a double loop, so $O(n^2)$.
Total running time is $O(n^2)$.

**Second Way to Answer**
RECURSIVE FORM OF THE SOLUTION:
Let $L(i)$ be the length of the longest path from 1 to $i$, $1 \leq i \leq n$.
We know that $L(1) = 0$ and we want $L(n)$.
Also, initially, all we know about $L(i)$, $2 \leq i \leq n$ is $L(i) = -\infty$.
Assuming that we have correctly computed $L(2), L(3), L(n-2), \ldots, L(k-1)$, then $L(k) = \max_{k' < k : a_{k'k} = 1} \max\{1 + L(k')\}$.

We may now use dynamic programming and write:
INITIALIZATION: $L(1) := 0$
for $k := 2$ to $n$ set $L(k) := -\infty$
MEMOIZATION: for $k := 2$ to $n$
for $k' := 1$ to $(k-1)$
if $(a_{k'k} = 1)$ then $L(k) := \max\{L(k), 1 + L(k')\}$
OUTPUT: return($L(n)$)

**Problem 4: Algorithm Design, Max and 2nd-Max (25 points)**
Let $n$ be a power of 2, $n \geq 2$, and let $a_1, \ldots, a_n$ be an array of $n$ usorted distinct positive integers. Give an algorithm that finds the maximum (largest) element of the above array and the 2nd-maximum (2nd-largest) element of the above array, using at most $(n-1) + (\log_2 n - 1)$ comparisons.
Justify correctness and number of comparisons.

**Answer:** We will find the max with $(n-1)$ comparisons and the 2nd-max with $(\log_2 n - 1)$ comparisons.
In particular, to find the max, we build a binary tree with $n$ leaves $a_1, \ldots, a_n$.
Such a binary tree has $(n-1)$ internal nodes and $\log_2 n$ levels. Each internal node represents a comparison.
At the bottom level we compare $a_i$ with $a_{i+1}$, $i \in \{1, 3, 5, \ldots, n-1\}$ and promote the winner one level higher.
We keep comparing the elements of every level in pairs, promoting the maximum one level higher,
until we are left with a single element, which is the maximum.
This is $(n-1)$ comparisons.
The 2nd-max is one of the elements that was immediately compared with the max over the $\log_2 n$ levels.
We can thus form a list of $\log_2 n$ elements, namely the ones that were immediately compared with the maximum at the binary tree. We know that the 2nd-max of the entire list is the max of this secondary list.
Since the secondary list has $\log_2 n$ elements, we can find the max of this list in $(\log_2 n - 1)$ comparisons.
Thus the total number of comparisons is $(n-1) + (\log_2 n - 1)$.