Question 1. (20 = 8 + 12 points).
   (a) List the first four Catalan numbers.
   (b) Give all possible parenthesizations of a chain of five matrices.

Solution to Question 1. (a) For \( m \) from 1 through 4 we have the Catalan numbers
\[ 1, 2, 5, 14. \]
(b) There are 14 different parenthesizations, and they are
\[
(X(X((XX)X))), \quad (X(X((XX)X))), \\
(X((XX)(XX))), \quad (X(X((XX)X))), \\
(XX)((XX)XX)), \quad ((XX)(XX)(XX)), \\
((XX)(XX)XX)), \quad ((XX)(XX)(XX)), \\
(((XX)XX)XX), \quad (((XX)XX)(XX)).
\]

Question 2 (20 = 10 + 10 points). Consider a set of \( n \) intervals \([a_i, b_i]\) that cover the unit interval, that is, \([0, 1]\) contained in the union of the intervals.

(a) Describe an algorithm that computes a minimum subset of the intervals that also covers \([0, 1]\).

(b) Analyze the running time of your algorithm.

(For question (b) you will get credit both for the correctness of your analysis but also for the running time of your algorithm. In other words, a fast algorithm earns you more points than a slow algorithm.)

Solution to Question 2. Sort the intervals by left endpoint and re-index such that \( a_1 \leq a_2 \leq \ldots \leq a_n \). Let \( S \) be the set of selected intervals and let \( R \) be the right endpoint of the union of intervals in \( S \). Initially, \( S = \emptyset \) and \( R = 0 \). For the next step we select the interval that contains \( R \) and reaches farthest to the right. We add this interval to \( R \), update \( R \) and repeat until \( R \geq 1. \)

\[
S = \emptyset; \ R = 0; \ \text{max} = 1; \ i = 2; \\
\text{repeat} \\
\quad \text{while} \ a_i \leq R \ \text{do} \\
\quad \quad \text{if} \ b_i > \text{max} \ \text{then} \ \text{max} = i \ \text{endif} \\
\quad \quad i = i + 1 \\
\quad \text{endwhile} \\
\quad S = S \cup \{[a_{\text{max}}, b_{\text{max}}]\}; \ R = b_{\text{max}} \\
\text{until} \ R \geq 1.
\]

(b) We need time \( O(n \log n) \) to sort. After sorting each step takes only constant time and there are at most \( n \) steps. The total time is therefore \( O(n \log n) \).

Question 3. (20 = 8 + 12 points). Consider a minimum-height full binary tree with \( n \) nodes. Suppose you designed an algorithm that searches the tree by visiting the root and recursively visiting one child of the root and one child of the other child of the root. [Observe that the algorithm does not visit every node, for example it does not visit any node on the rightmost path, other than the root.]

(a) Draw a tree with \( n = 31 \) nodes and mark the nodes visited by your algorithm (assuming the recursion always takes it to the left child and the left child of the right child).

(b) Analyze the running time of the search algorithm.

Solution to Question 3. (a) Figure 62 answers the question by marking 12 of the 31 nodes as visited.

(b) The running time is the solution to the recurrence \( T(n) = 1 + T(n/2) + T(n/4) \). Setting \( n = 2^i \) and...
\[ t_i = T(2^i), \text{ we rewrite the recurrence as} \]
\[ t_i = 1 + t_{i-1} + t_{i-2}. \]

This is the same recurrence we used in class to compute the maximum height of an \( n \)-node AVL-tree. Using the annihilation method, we showed \( t_i = c \varphi^i \), where \( c \) is a positive constant and \( \varphi = (1 + \sqrt{5})/2 \) is the golden ratio. It follows that
\[ T(n) = O(\varphi^{\log_2 n}) = O(n^{0.694\ldots}). \]


(a) Is it true that \( A \) contains at least one local minimum? (Justify your answer.)

(b) We can obviously find all local minima in time \( O(n) \). Sketch an algorithm that finds one local minimum in time \( O(\log n) \).

**Solution to Question 4.** (a) Yes, for example the smallest item in \( A[2..n-1] \) is a local minimum.

(b) We can use binary search, maintaining an interval \( A[i..j] \). We also maintain that \( A[i] \geq A[i+1] \) and \( A[j-1] \leq A[j] \) as an invariant. If \( j - i < 3 \), we find the local minimum directly. Otherwise, we set \( m \) equal to the floor of \( i + j \) over 2. If \( A[m] \) is a local minimum we are done. Otherwise, either \( A[i..m] \) or \( A[m..j] \) satisfies the invariant and we continue the search in this half of the array.

**Question 5.** (20 = 5 + 5 + 10 points). Let \( A[1..n] \) be an array of \( n \) distinct numbers. If \( i < j \) and \( A[i] > A[j] \) then the pair \((i, j)\) is called an inversion of \( A \).

(a) List the inversions of the array \((2, 3, 8, 6, 1)\).

(b) How would you rearrange the five numbers to get the maximum number of inversions, and how many do you get?

(c) Give an algorithm that determines the number of inversions in any permutation of \( n \) items in \( O(n \log n) \) worst-case time. [Hint: modify merge sort.]

**Solution to Question 5.** (a) There are five inversions: \((3, 4), (1, 5), (2, 5), (3, 5), (4, 5)\).

(b) I would sort the items in a decreasing order as \((8, 6, 3, 2, 1)\). The number of inversion is \( \binom{5}{2} = 10 \).

(c) We run merge sort and for each item \( a \) we compute the number of other items that are smaller than \( a \) but appear after \( a \) in the fraction of the original sequence considered at this level of the recursion. We need to show that given two sorted lists of items with associated numbers, we can merge the lists and compute the updated numbers in linear time. For an item \( a \) at position \( i \) in the first list that ends up at position \( j \geq i \) in the merged list, the associated number increases by \( j - i \). For an item in the second list, the associated number stays the same. After finishing merge sort, the total number of inversions is the sum of all associated numbers.