1. **Analyzing an Elementary Algorithm.** We can express insertion sort as a recursive procedure as follows. In order to sort $A[1..n]$, we recursively sort $A[1..n-1]$ and then insert $A[n]$ into the sorted array $A[1..n-1]$. Write a recurrence relation for the running time of this recursive version of insertion sort. Explain your reasoning.

**Solution:** A recursion formula that captures the activity is

$$T(n) - T(n-1) + O(n).$$

Explanation: we sort a set of cardinality $n$ by first sorting a set of cardinality $n-1$ : this accounts for the $T(n) = T(n-1)$ part. Since we insert the remaining element into a sorted set of cardinality $n-1$, we will have at most $n-1$ comparisons and swaps, accounting for the $O(n)$ term (which could be replaced by an $O(n-1)$ without any loss).

2. **Asymptotic Notation and Properties.**
   a. (3 pts) Define $o(f(n))$ and $\Theta(f(n))$.
   b. (7 pts) Prove or disprove: $f(n) + o(f(n)) \in \Theta(f(n))$. 
Solution:

a. The Definitions. We assume \( f(n) \) and \( g(n) \) are positive for all \( n \geq 0 \). Recall \( o(f(n)) \) denotes \( \{g(n) \mid \lim_{n \to \infty} \frac{g(n)}{f(n)} = 0\} \) which is equivalent to the statement: for any \( c > 0 \) there exists \( n_c > 0 \) such that \( g(n) < cf(n) \ \forall \ n \geq n_0 \).

\( \Theta(f(n)) \) denotes \( \{g(n) \mid \exists \ c_1, c_2 > 0 \text{ and } n_0 > 0 \text{ such that } c_1f(n) \leq f(n) \leq c_2f(n), \ \forall \ n \geq n_0\} \).

b. The Proof. We observe that, for any \( c > 0 \ \exists \ n_c > 0 \), such that

\[
\begin{align*}
    f(n) + g(n) &\leq f(n) + cf(n) = (1 + c)f(n), \ \forall \ n \geq n_{c,g},
\end{align*}
\]

where \( n_{c,g} \) depends on \( g \) and \( c \). In particular, choose \( c = 1 \) to conclude, for appropriate \( n_{c,g} \), that \( f(n) + o(f(n)) \leq 2f(n) \in O(f(n)) \).

To show that \( f(n) + g(n) \in \Omega(f(n)) \) we simply observe that the assumption that \( g(n) \geq 0 \) implies that \( f(n) + g(n) \geq f(n) \) for all \( n \geq 0 \), so that we have

\[
\begin{align*}
    f(n) \leq f(n) + g(n) \leq 2f(n) \ \forall \ n \geq n_{c,g},
\end{align*}
\]

which gives the claimed result.

3. Recurrences.

1. (5 pts) Use a recursion tree to determine a good asymptotic upper bound on the recurrence

\[
T(n) = 2T(n-1) + 1.
\]

2. (5 pts) Use the substitution method to verify your answer.

Solution:

1. This gives rise to a full binary tree with \( n \) levels. Level \( i \), starting from the root, has \( 2^i \) nodes, each requiring a cost of 1 to combine the results of their left and right children. Total cost \( \sum_{i=0}^{n-1} 2^i = 2^n - 1 \).

2. To do the second part, assume \( T(1) = 1 \). Assume further that the formula holds for all \( n \) up to \( k \). Prove it true for \( k + 1 \).

\[
T(k+1) = T(k) + 1 = 2(2^{k+1} - 1) + 1 = 2^{k+2} - 1.
\]

4. The Master Theorem. You are given:
Theorem 4.1 (Master theorem)
Let \( a \geq 1 \) and \( b > 1 \) be constants, let \( f(n) \) be a function, and let \( T(n) \) be defined on the nonnegative integers by the recurrence
\[
T(n) = aT(n/b) + f(n),
\]
where we interpret \( n/b \) to mean either \( \lfloor n/b \rfloor \) or \( \lceil n/b \rceil \). Then \( T(n) \) has the following asymptotic bounds:

1. If \( f(n) = O(n^{\log_b a - \epsilon}) \) for some constant \( \epsilon > 0 \), then \( T(n) = \Theta(n^{\log_b a}) \).
2. If \( f(n) = \Theta(n^{\log_b a}) \), then \( T(n) = \Theta(n^{\log_b a} \log n) \).
3. If \( f(n) = \Omega(n^{\log_b a + \epsilon}) \) for some constant \( \epsilon > 0 \), and if \( af(n/b) \leq cf(n) \) for some constant \( c < 1 \) and all sufficiently large \( n \), then \( T(n) = \Theta(f(n)) \).

Given the family of recurrences \( T(n) = aT(n/b) + n^k \log n \), where \( k \) is a non-negative number,

a. Find the relative ranges of \( a, b \) and \( k \) for which The Master Theorem provides solution bounds (don’t forget checking the regularity condition, if that case seems to apply).

b. Give a specific example for each case.

Solution: under what conditions on \( a, b \) and \( k \) does \( f(n) = n^k \log n \) fall into the cases of the Master Theorem? Case 2 is easy: never, since \( n^k \log n \not\in \Theta(n^r) \) for any value of \( r \) real: recall that \( \log n \) grows more slowly than any positive power of \( n \). For Case 1, we notice that we need \( \log_a b > k \), with Case 3 being covered by \( \log_a b < k \). For the specific examples, pick appropriate values of \( a, b \) and \( k \).

5. Probabilistic Analysis. Use indicator random variables to solve the following problem (known as the hat check problem). Each of \( n \) customers gives a hat to a hat-check person at a restaurant. The hat-check person gives the hats back to the customers in a random order. What is the expected number of customers who get back their own hat?

Hints: for each \( i \), let \( X_i \) be the indicator variable for the event “customer \( i \) receives his hat back”: \( X_i = I \{ \text{customer } i \text{ gets his own hat back} \} \). Then \( X = X_1 + \cdots + X_n \) is what? What does \( E[X] = E[\sum_{i=1}^{n} X_i] \) compute? What is \( E[X_i] \), for \( i = 1 \ldots n \)? Remember each customer receives a random hat...

Solution: \( X = X_1 + \cdots + X_n \) is the number of customers receiving their hats back.
\( E[X] = E[\sum_{i=1}^{n} X_i] \) is the expected number of customers receiving their hats back.
\( E[X_i] \), for \( i = 1 \ldots n \) is the expectation that customer \( i \) receives his hat back. Since the hat-check person operates randomly and uniformly, \( E[X_i] = \frac{1}{n} \), for \( i = 1 \ldots n \)
By linearity of expectation \( E[X] = E[\sum_{i=1}^{n} X_i] = \sum_{i=1}^{n} E[X_i] = \sum_{i=1}^{n} \frac{1}{n} = 1 \).
The number of people expected to receive their hats back is 1.

6. Randomized Algorithms. Recall that
Randomize-In-Place(A)

1. \( n = A.length \)
2. for \( i = 1 \) to \( n \)
3. swap \( A[i] \) with \( A[Random(i, n)] \)

provides a random permutation of \( n \) elements.

Professor Marceau objects to the loop invariant that starts before the first iteration of the loop with an empty subarray \( A[1..0] \) of \( A[1..n] \) which cannot possibly contain a 0-permutation (or anything else, for that matter). How would you solve the problem so that no empty arrays are needed? Rewrite the code fragment and explain.

Hint: how can you make sure that your loop starts with a non-empty subarray containing a 1-permutation?

Solution: Instead of starting \( i \) at 1, swap \( A[1] \) with \( A[Random(1, n)] \) and run the loop from \( i = 2 \). You then have a random 1-permutation in position 1 (instead of a 0-permutation in position 0) before the loop starts and the same invariant holds as the loop repeats. Same termination, too.

7. Asymptotic Behavior

Indicate, for each pair of expressions \((A, B)\) in the table below, whether \( A \) is \( O \), \( o \), \( \Omega \), \( \omega \), or \( \Theta \) of \( B \). Assume that \( k \geq 1, \epsilon > 0 \) and \( c > 1 \) are constants. Write yes or no in each box.

<table>
<thead>
<tr>
<th>( A )</th>
<th>( B )</th>
<th>( O )</th>
<th>( o )</th>
<th>( \Omega )</th>
<th>( \omega )</th>
<th>( \Theta )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \lg^* n )</td>
<td>( n! )</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( n^k )</td>
<td>( c^n )</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( \sqrt{n} )</td>
<td>( n^\cos n )</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( 2^n )</td>
<td>( 2^{n/2} )</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( n^{\lg c} )</td>
<td>( c^{\lg n} )</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( \lg(n!) )</td>
<td>( \lg(n^n) )</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( \lg^* n \lg(n) )</td>
<td>( \lg^*(n^n) )</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Solution: keep trying. Also, try to figure out the last line, in both the \( \lg^*(n \lg(n)) \) and \( \lg^*(n) \cdot \lg(n) \) cases.