1. Relative Asymptotic Growth. Suppose we are comparing implementations of insertion sort and merge sort on the same machine. For inputs of size $n$, insertion sort runs in time $8n^2$ steps, while merge sort runs in $64n \lg n$ steps. For which values of $n$ does insertion sort beat merge sort? Show your work - giving just a number will not be adequate.

**Soln.** You need to find out the range of $n$ for which $8n^2 \leq 64n \lg n$. Dividing both sides by $8n$, we simplify to $n \leq 8 \lg n$. A small table of values of $n$ will give that a value $32 < n < 64$ will do the job ($32$ is OK, $64$ is not). You can be more precise, if you wish.

2. Recurrences. 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 $A[1..n-1]$

1. (5 pts) Write a recurrence for the running time of this recursive version of insertion sort. Explain your reasoning and the mechanics of the algorithm.

2. (5 pts) Guess and prove inductively a tight upper bound for the solution of this recurrence relation.

**Soln.** In the recursion, the size of the array decreases by 1. The worst-case cost of the insertion of a single element into a sorted array of $n-1$ elements is $O(n)$: up to $n-1$ comparisons (and copies), followed by one insertion. The Recurrence Relation is:

$$T(n) = T(n-1) + O(n).$$
The induction: let \( T(1) = c, T(n) = cn^2 \). Using the Recurrence:

\[
T(n) = T(n-1) + O(n)
\]
\[
= c(n-1)^2 + O(n)
\]
\[
\leq cn^2 - 2cn + c + kn \text{ for some } k > 0 \text{ and } n \text{ large}
\]
\[
= cn^2 - (2cn - c - kn)
\]
\[
< cn^2.
\]

Where you choose \( c > k \) to make \( 2cn - c - kn > 0 \)

3. **Asymptotic Notation.** Let \( f(n) \) and \( g(n) \) be asymptotically positive functions. Prove or disprove:

\[
f(n) + g(n) = \Theta(\min(f(n), g(n))).
\]

**Note:** you must give the definition of \( \Theta \) and either provide a proof that uses the definition or give a specific counterexample that contradicts it.

**Soln.** For the definition of \( \Theta \), see p. 44 and following. To disprove the claim, let \( f(n) = n \), and \( g(n) = 1/n \). Then \( \min(f(n), g(n)) = 1/n \) while \( f(n) + g(n) = n + 1/n \), for \( n \geq 1 \). The rest should be easy.

4. **Recurrences.**

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

\[
T(n) = 3T([n/2]) + n.
\]

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

**Soln.** This is rather hard. For the first part, we can simplify matters a little by dropping the “floor” operation: \( T(n) = 3T(n/2) + n \). This tells us that the number of levels in the tree is bounded by \( \lg n \). The first level of the tree contains \( n \) elements (i.e., costs \( n \) plus the cost of the recursion). The second level has a cost \( \frac{3}{2}n \), the third level has cost \( \left(\frac{3}{2}\right)^2n \), the next level has cost \( \left(\frac{3}{2}\right)^3n \), all the way to \( \left(\frac{3}{2}\right)^{\lg n}n \). Recall that \( 2^{\lg 3} = 3 \). The total cost is then

\[
\sum_{i=0}^{\lg n} \left(\frac{3}{2}\right)^i n = n \sum_{i=0}^{\lg n} \left(\frac{3}{2}\right)^i = n \frac{1 - \left(\frac{3}{2}\right)^{\lg n + 1}}{1 - \frac{3}{2}} = 2n \left(\frac{3}{2}\right)^{\lg n + 1} - 1
\]

\[
= 2n \left(\frac{3}{2}n^{\lg 3} - 1\right) = 3n^{1 + \lg \frac{3}{2}} - 2n = 3n^{\lg 3} - 2n.
\]

A “reasonable” upper bound would thus be \( cn^{\lg 3} \), which is consistent with Case 1 of the Master Theorem. For the second part, we notice that the induction goes from \( n/2 \) to \( n \), not increasing \( n \) by 1, but doubling.

\[
T(n) = 3T(n/2) + n \leq 3c(n/2)^{\lg 3} + n = 3c\frac{n^{\lg 3}}{2^{\lg 3}} + n = cn^{\lg 3} + n
\]

\[2\]
and we end up with a problem (why?). We retry with \( T(n) = 3n\log_3 - 2n \), for \( n \geq 1 \),
\[
T(n) = 3T(n/2) + n \leq 3(3(n/2)^{\log_3} - 2(n/2)) + n = 9n^{\log_3 \frac{1}{2}} - 3n + n
= 3n\log_3 - 2n.
\]

5. **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 \( T(n) = \Theta(f(n)) \).

1. (5 pts) Using the Master Theorem, show that the solution of the recurrence \( T(n) = 4T(n/5) + n \log n \) is \( T(n) = \Theta(n \log n) \).

2. (5 pts) What if \( 4T(n/5) \) is replaced by \( 5T(n/4) \)? Explain.

**Soln.** Since \( n^{\log_5 4 + \epsilon} < n^1 = n < n \log n \) for all large \( n \) and \( 0 < \epsilon \leq 1 - \log_5 4 \), the lower bound of Case 3 holds. For the inequality:

\[
4\left( \frac{n}{5} \log \frac{n}{5} \right) \leq \frac{4}{5} n \log n - \frac{4}{5} \log 5 \leq \frac{4}{5} n \log n.
\]

The change from base 2 to base \( e \) should be trivial.

For the second part, Case 1 holds, and \( T(n) \) is...

6. **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”: \( X_i = 1 \{ \text{customer } i \text{ gets his own hat back} \} \). 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...

**Soln.** \( X = X_1 + \cdots + X_n \) 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.

7. **Randomized Algorithms.** Prof. Kelp decides to write a procedure that produces at random any permutation besides the identity permutation. He proposes:

\[
\text{Permute-Without-Identity}(A) \\
1. \ n = A.length \\
2. \ \text{for} \ i = 1 \ \text{to} \ n - 1 \\
3. \ \text{swap} \ A[i] \ \text{with} \ A[\text{Random}(i + 1, n)]
\]

**Hints:** If the procedure were correct, how many different permutations would you expect to produce? How many do you? Try with the starting configuration \( A = [1, 2, 3] \) and see what happens.

**Soln.** Apply the algorithm to \([1, 2, 3]\) and observe that only 2 permutations are produced. Since the total number of permutations on 3 elements is \(3! = 6\), this algorithm falls far short of requirements.