Problem 1

(a) Identify the flaw in the following proof.

If \( T(1) = 1 \) and \( T(n) = 4T(n/2) + n \), then \( T(n) = O(n) \).

**Proof.** We use a proof by strong induction. Let \( p(n) \) be the predicate that \( T(n) \) as defined above, is \( O(n) \).

**Base case:** For \( n = 1 \), \( p(1) \) is true because \( T(1) = 1 \) is \( O(1) \).

**Inductive step:** Assume \( p(1) \land p(2) \land \cdots \land p(n-1) \land p(n) \) are true. We need to show that \( p(n+1) \) is true. Plugging \( n+1 \) into the recurrence relation gives

\[
T(n+1) = 4T((n+1)/2) + n + 1.
\]

By our assumption \( p((n+1)/2) \) is true, so \( T((n+1)/2) \) is \( O(n) \), as is \( 4T((n+1)/2) \). Also, \( n + 1 \) is \( O(n) \), so \( T(n+1) \) is \( O(n) \).

Therefore, by induction on \( n \), \( T(n) \) is \( O(n) \).

(b) Express \( T(n) \) in big-oh notation (correctly).

Problem 2

Consider a variant of mergesort, called tri-mergesort. During the recursive step of tri-mergesort, instead of breaking up the length-\( n \) array into two subarrays, we will break it up into three subarrays of length \( n/3 \), and recursively sort these three subarrays. Once these three subarrays are sorted, we will merge them into the length \( n \) array. In the following analysis, you may assume that the length of the array is a power of 3 (i.e. \( n = 3^k \) for some integer \( k \)).

(a) How many comparisons are needed to merge three subarrays of length 1?
(b) Roughly how many operations are needed to merge three subarrays of length $n/3$? A handful of operations (constant)? Some factor of the length of the array? $n^2$?

(c) Define a divide-and-conquer recurrence for this algorithm. Let $T(n)$ be the number of operations used to sort a list of $n$ items.

(d) Use the Tree Method (or Alpha-Bits theorem) to express the number of operations in tri-mergesort in big-oh notation.