Answer all questions. Give informal (at least) proofs for all answers. Grading will be on completeness and logical correctness, and if applicable, efficiency, as well as correctness. Out of 80 points.

Order questions- 10 points each For each, answer True or False, and give a short (1–2 sentence) explanation for your answer.

1. \( n + (n - 1) + (n - 2) \in O(n) \)
   True. The sum of any fixed number of terms have the same order as the largest term, which in this case is \( n \).

2. \( n + (n - 1) + (n - 2) + (n - 3) + \ldots + 2 + 1 \in O(n) \).
   False. Since there are \( n \) terms, rather than a fixed number, we need to estimate the entire sum. The sum given is an arithmetic progression, and the sum is equal to \( (n)(n + 1)/2 = \Omega(n^2) \), which is larger than any \( O(n) \) function.

3. If \( f \) and \( g \) are functions from positive integers to positive integers, and \( f(n) \in O(g(n)) \), then \( f(n) \cdot g(n) \in O((g(n))^2) \). (If not always true, give an example of functions \( f \) and \( g \) for which it is false.)
   True. \( f(n) \in O(g(n)) \) so for sufficiently large \( n \), and some \( c > 0 \), \( f(n) < cg(n) \). Then \( f(n) \cdot g(n) < cg(n) \cdot g(n) = cg(n)^2 \), so \( f(n)g(n) \in O(g(n)^2) \).

4. If \( f \) is functions from positive integers to positive integers, then \( f(2n) \in O(f(n)) \). (If not always true, give an example of a function \( f \) for which it is false.)
   False. For \( f(n) = 2^n \), \( f(2n)/f(n) = 2^{2n}/2^n = 2^n \), which goes to infinity as \( n \) gets large. Therefore \( f(2n) \notin O(f(n)) \).

Analyzing loops-20pts Here is an algorithm that, given an array of integers \( A[1]...A[n] \), outputs an array \( B[1..n] \), where for each \( I \), \( B[I] \) is either the first \( J \) greater than \( I \) with \( A[J] > A[I] \) or \( n + 1 \) if no such \( J \) exists. The algorithm uses a stack of pairs (position, value), Stack, with constant time operations Top (returning the first element or a special message if the stack is empty) Push and Pop. Infinity is a value bigger than any array position. Give a time analysis for the algorithm, in \( O \) notation.

FirstLargest(\( A[1..n] \): sorted list of integers)

1. Initialize an empty stack Stack of pairs with two fields, position and value
2. \( \text{Push}(n+1, \text{Infinity}) \)
3. FOR \( I = n \) down to 1 do:
4. \( \text{While } A[I] \geq \text{Top.value} \) do Pop
5. \( B[I] \leftarrow \text{Top.position} \)
6. Push \( (I, A[I]) \)
7. Return \( B \).

There are two nested loops, the FOR and the WHILE loop. Since both could take up to \( n \) time in the worst case, it is tempting to say that the algorithm is \( O(n^2) \) and leave it at that. However, this is not tight. Note that we perform the Push operation \( n + 1 \) times, once at the start, and once in each iteration of the FOR loop. Throughout the life of the algorithm, then, we can Pop at most this number of times. Since we Pop each iteration of the While loop, the total number of iterations (through the entire algorithm) is at most \( n + 1 \). So the total time for all of the executions of the While loop is \( O(n) \). Since the rest of the FOR loop is constant time, the entire algorithm is thus \( O(n) \) time.

Correctness Proofs: 20 points Prove the following loop invariant about the algorithm above, that is useful in showing the algorithm correct.

Lemma: After each iteration, both the positions and values of the elements in the stack are increasing from top to bottom.
Proof: We show this as a loop invariant. For the base case, before the while loop begins, the stack contains a single pair. Since any list of one element is sorted, both position and value are in sorted order.

Assume the stack is sorted at the start of an iteration of the FOR loop. Deleting the first element of the stack does not change the order of the other elements, so the stack remains sorted (in both fields) through the WHILE loop. We then insert \((I, A(I))\) at the top of the stack. Since all positions currently on the stack were inserted in previous iterations, with a larger \(I\), the position of the element we are inserting is the smallest one on the stack, so the positions are still sorted from smallest to largest. Since we only exit the While loop when \(A[I] < Top.value\), and by the invariant, \(Top.value\) is the smallest value on the stack, the new value we insert is the smallest value on the stack, so the values remain sorted from smallest to largest.

Thus, since the invariant is true at the start of the loop, and since if true at the start of each iteration, it is still true at the end of the iteration, by induction on the number of iterations, it remains true throughout the algorithm.