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 explanation for your answer.

1. \( n^2 \in O(2^n) \). Yes. Any polynomial is asymptotically smaller than any exponential function, and big \( O \) just means that the function is an upper bound, not necessarily tight.

2. \( n^2 \in \Theta(2^n) \). No. As we said above, a polynomial is asymptotically smaller than an exponential function, and to be \( \Theta \), the two functions need to have the same asymptotic order.

3. If \( f \) and \( g \) are functions from positive integers to positive integers, and \( f(n) \in O(g(n)) \), then \( f(n) + g(n) \in O(g(n)) \). Yes. We know \( f(n) \leq cg(n) \) for sufficiently large \( n \) and some \( c \). Then \( f(n) + g(n) \leq cg(n) + g(n) = (c+1)g(n) \). So letting the new constant \( c' \) be \( c+1 \), \( f(n) + g(n) \) meets the definition of \( O(g(n)) \).

4. If \( f \) and \( g \) are functions from positive integers to positive integers, and \( f(n) \in O(g(n)) \), then \( 2f(n) \in O(2g(n)) \). No. If \( f(n) = 2n \) and \( g(n) = n \), \( f(n) \in O(g(n)) \), but \( 2f(n) = 2^{2n} = 2^n 2^n = 2^n g(n) \).

Analyzing loops-20pts Say that an airline has a sorted array of arrival times for flights at an airport \( A[1...n_a] \) and a sorted array of departure times for flights at the same airport, \( D[1...n_d] \). The airline wants to give one hour for a passenger to make a connection. Here is an algorithm that finds for each arriving flight \( 1 \leq i \leq n_a \), the first departing flight \( 1 \leq j \leq n_d \) where a passenger arriving on the first has enough time to make their connection and depart on the second. If there are no such flights, we define the answer as 0.

FirstConnection(\( A[1...n_a], D[1...n_d] \): sorted lists of positive integers)

1. Initialize \( FC[1...n_a] \) as an array of integers.
2. \( I \leftarrow 1, J \leftarrow 1 \)
3. FOR I=1 to \( n_a \) do:
   4. While \( J \leq n_d \) and \( B[J] < A[I] + 1 \) do: \( J++ \)
   5. IF \( J > n_d \) do \( FC[I] \leftarrow 0 \) else \( FC[I] \leftarrow J \).
4. Return \( FC[1...n_a] \).

Give a time analysis, up to order, for this algorithm, in terms of the total input size \( n = n_a + n_d \). Be sure to explain your answer.

Although we have a while loop inside a for loop, and both loops are \( O(n) \) in the worst case, the time is better than \( O(n^2) \). We can observe that \( J \) is never decremented, and is incremented in every iteration of the inside while loop. So after \( n_d \) iterations of the inside while loop, we will have \( J > n_d \), and the while loop will be skipped after that. So the total cost of all iterations of the while loop is \( O(n_d) \) since each iteration is constant time. In addition, we will go through \( n_a \) iterations of the FOR loop. Not counting the while loop, which is already included above, the cost for each iteration is constant time. So that will contribute \( O(n_a) \) time total. The time to initialize the array is also \( O(n_a) \). Thus, the total time will be \( O(n_a + n_d) = O(n) \).

Correctness Proof: 20 points Prove the following loop invariant for the algorithm above: at all times, for every \( I \leq i \leq n_a, 1 \leq j \leq n_d \), if a passenger arriving on the i'th arriving flight can connect to flight \( j \), then \( j \geq J \).

At the beginning of the loop \( J = 1 \) and since each flight number \( j \geq 1 \), the claim is vacuously true.

Assume that before the iteration for \( I \), for every \( I \leq i \leq n_a \), and every if a passenger arriving on the i'th flight can connect to flight \( j \), then \( j \geq J \).
We want to guarantee that this stays true. Say that $i \geq I$ and a connection between flights $i$ and $j$ is possible. We know $j \geq J$ at the beginning of the iteration. Assume to get a contradiction that $j < J$ at the end. Then there must have been a time when $J = j$ and $J$ is incremented. But for that to happen when $J = j$, from line 4, we must have $B[j] = B[J] < A[I] + 1 \leq A[i]$ (since the array $A$ is sorted and $i \geq I$.) But then a passenger arriving at time $A[i]$ would not have the required hour to make a connection and leave at $B[i]$, contradicting the fact that there was a possible connection between $i$ and $j$. From this contradiction, it follows that $j \geq J$. Thus, if the claim is true before iteration $I$, it is true after iteration $I$. By induction on $I$, it is true at all times of the algorithm.