Say that, on an input of size \( n \geq 1 \), an algorithm calls itself recursively on an input of size \( n-1 \) and otherwise takes \( O(n) \) time in addition. On an input of size 0, it terminates in some fixed time \( T(0) \). Write down a recurrence relation for the total time \( T(n) \) the algorithm takes on inputs of size \( n \), and use that to find \( T(n) \) up to order. (5 points correct recurrence with explanation, 5 points solving recurrence correctly.)

Order Notation, 5 pts. each = 2 points correct answer + 3 points explanation

Is \( 2^n \in O(n!) \)? Why or why not?

Is \( 2^{[\log n]} \in O(n) \)? Why or why not? (When unspecified, logs are base 2).

Is \( 4^n \in O(2^n) \)? Why or why not?

If \( f \) and \( g \) are functions from positive integers to positive integers, is \( f(n) + g(n) \in \Theta(\max(f(n), g(n))) \)?

Analyzing algorithms Here is an algorithm that, given two sorted lists \( A[1..n] \) and \( B[1..n] \), decides whether there is a pair of indices \( i, j \) with \( 1 \leq i \leq n \) and \( 1 \leq j \leq n \) so that \( A[i] = B[j] \).

Intersect \( A[1..n], B[1..n] \): sorted list of integers

1. \( I \leftarrow 1, J \leftarrow 1, Found \leftarrow False \).
2. While \( I \leq n \) and \( J \leq n \) and \( Found = False \) do:
   3. \hspace{1em} IF \( A[I] = B[J] \) THEN \( Found \leftarrow True \).
   4. \hspace{1em} IF \( A[I] > B[J] \) THEN \( J++ \)
   5. \hspace{1em} IF \( A[I] < B[J] \) THEN \( I++ \)
3. Return \( Found \).

First, prove that this algorithm is correct, i.e., that it returns true if and only if such a pair \( i, j \) exists (15 points). Second, give a time analysis, up to order, for this algorithm. Be sure to explain your answer. (15 points)

Summing triples (20 points) Let \( A[1..n] \) be an array of positive integers.

A summing triple in \( A \) is 3 distinct indices \( 1 \leq i, j, k \leq n \) so that \( A[i] + A[j] = A[k] \). Give and analyze an algorithm that, given \( A \), determines whether there is any summing triple in \( A \). Your algorithm must take \( o(n^3) \) time, i.e., some time function that is asymptotically strictly faster than \( O(n^3) \).
Implementation (20 points) Implement the algorithm you gave for the summing triples problem above. For $n$ as many different powers of two as possible, and for many random arrays where each of the $n$ elements $A[i]$ has a random value between 1 and $n$, try your algorithm and plot the average time your algorithm took on a log-log scale, i.e., plot $\log(n)$ on the $x$-axis and $\log(Time)$ on the $y$-axis. Then plot the same information but where each $A[i]$ has a random value between 1 and $n^3$. Do you see a difference? If so, can you explain it?