Analyzing algorithms, 10 pts. Assume proc(I) is an algorithm that takes $\Theta(I)$ time and does not change $I$. What are the orders of the running times of the following two algorithms?

**Alg1(n)**

1. begin;
2. $I \leftarrow 1$;
3. While $I \leq n$ do:
4. begin;
5. proc(I)
6. $I++$
7. end;
8. end;

**Alg2(n)**

1. begin;
2. $I \leftarrow 1$;
3. While $I \leq n$ do:
4. begin;
5. proc(I)
6. $I \leftarrow 2 * I$
7. end;
8. end;

Order Notation, 5 pts. each Is $4^{\log n} \in O(n^2)$? Why or why not? (When unspecified, logs are base 2).

Is $\log(n!) \in \Theta(n \log n)$? Why or why not?

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

Is $n + (n - 1) + (n - 2) + ... + 1 \in O(n)$? Why or why not?
**Triangles (20 points)** Let $G$ be an undirected graph with nodes $v_1, \ldots, v_n$. The adjacency matrix representation for $G$ is the $n \times n$ matrix $M$ given by: $M_{i,j} = 1$ if there is an edge from $v_i$ to $v_j$, and $M_{i,j} = 0$ otherwise. A triangle is a set $\{v_i, v_j, v_k\}$ of 3 distinct vertices so that there is an edge from $v_i$ to $v_j$, another from $v_j$ to $v_k$ and a third from $v_k$ to $v_i$. Give and analyze an algorithm for deciding whether a graph has a triangle in it, where the input graph is given by its adjacency matrix representation. Analyze your algorithm’s worst-case performance first in terms of just the number of nodes $n$ of the graph, then in terms of $n$ and the number of edges $m$ of the graph. Your algorithm should be faster when $m << n^2$.

**Binary Conversion (10 points):** Consider the following algorithm to covert a decimal number to binary. More precisely, the input is a decimal representation of a number, given as an array of digits, $D[n-1], \ldots, D[0]$, representing $X = \sum_{i=0}^{n-1} D[i]10^i$. The output should be an array of bits $B[n'-1], \ldots, B[0]$ so that $X = \sum_{i=0}^{n'-1} B[i]2^i$.

The following algorithm uses a “long division by two” algorithm $LDIV$ that takes linear time ($O(n)$) to compute the decimal representation of $X \div 2$, given $X$ in decimal.

The binary conversion algorithm is: Convert($D[0..n-1]$: array of digits): array of bits

1. Initialize $B[0..4n]$ array of bits.
2. $I \leftarrow 0$ {a pointer to which bit we are computing}
3. While $I \leq 4n$ do:
4. begin;
5. $B[I] \leftarrow D[0] \mod 2$;
6. $D \leftarrow LDiv2[D]$;
7. $I++$;
8. end;
9. Return $B$ (possibly removing initial 0’s, if you want).

Prove that this algorithm is correct, and give a time analysis in terms of the number $n$ of digits.

**Summing triples (20 points)** Let $A[1,\ldots,n]$ 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 $A$. Try to be better than $O(n^3)$.

**Implementation (20 points)** Implement the algorithm you gave for the summing triples problem above. Try it on random arrays where each element
$A[i]$ is chosen in the range 1...$n$, for $n = 2^4, 2^6, 2^8, 2^{10}, 2^{12}, 2^{14}$, and $2^{16}$. Plot its performance on a $\log_2 n$ vs. $\log_2$ of the time scale. Then try the same experiment on random arrays where each element is chosen in the range 1...$n^4$. Do you see a difference? If so, can you explain it?