1. Prove each by applying directly the definitions given in class of $O$, $\Omega$, and $\Theta$:

A. \[83n is \Theta(n)\]
   \[a = 1 \text{ and } b = 83 \text{ suffice for the } O \text{ definition, since for all integers } n \geq 0, 83n \leq 83n = bn.\]
   \[c=1 \text{ suffices for the } \Omega \text{ definition, since for infinitely many } n \text{ (all } n \geq 0), 83n \geq n = cn.\]
   (In fact, any any $c < 83$ suffices.)
   
   Since $83n$ is both $O(n)$ and $\Omega(n)$, it is $\Theta(n)$.

B. \[n^2 is \Omega(22n)\]
   \[c=1 \text{ suffices for the } \Omega \text{ definition, since for infinitely many } n \text{ (all } n > 22), n^2 \geq 22n.\]
   (In fact, any $c>0$ suffices since for all integers $n \geq 22c, n^2 = (n/22)(22n) \geq c22n.$)

C. \[n^2/1000 is \Omega(1000n)\]
   \[c=1 \text{ suffices for the } \Omega \text{ definition, since for infinitely many } n \text{ (all } n \geq 1,000,000): n^2/1000 \geq 1000n.\]
   (In fact, any $c>0$ suffices since for all integers $n \geq 1,000,000c, n^2/1000 \geq c1000n.$)

D. \[2n^4 - 3n^2 + 32n\sqrt{n} - 5n + 60 \text{ is } \Theta(n^4)\]
   \[a=0 \text{ and } b=94 \text{ suffice for the } O \text{ definition, since for all } n \geq 0: 2n^4 - 3n^2 + 32n\sqrt{n} - 5n + 60 \leq 2n^4 + 32n^4 + 60n^4 = 94n^4 = bn^4\]
   (In fact, any $b>2$ will work by using a larger value of $a$).
   
   \[c=1 \text{ suffices for the } \Omega \text{ definition, since for infinitely many } n \text{ (all } n \geq 3): 2n^4 - 3n^2 + 32n\sqrt{n} - 5n + 60 \geq 2n^4 - 3n^2 - 5n = n^4 + (n^4 - 3n^2 - 5n) \geq n^4\]
   (In fact, any $c<2$ will work.)
   
   Since the expression is both $O(n)$ and $\Omega(n)$, it is $\Theta(n)$.

E. \[2n^2\sqrt{n} \text{ is not } \Omega(n^3)\]
   \[2n^2\sqrt{n} \text{ cannot be } \Omega(n^3) \text{ since no matter how small } c>0 \text{ is chosen, for all } n > 4/(c^2): 2n^2\sqrt{n} = c\left(\sqrt{4 / c^2}\right)n^2\sqrt{n} < c\sqrt{n} n^2\sqrt{n} = cn^3\]
2. Using the definitions given in class of \( O \) and \( \Omega \), prove that:

A. \( f(n) \) is \( O(g(n)) \) and \( g(n) \) is \( O(h(n)) \) implies \( f(n) \) is \( O(h(n)) \)

By definition, there exists constants \( a_1 \) and \( b_1 \) such that for all \( n \geq a_1, f(n) \leq b_1 g(n) \).

By definition, there exists constants \( a_2 \) and \( b_2 \) such that for all \( n \geq a_2, g(n) \leq b_2 h(n) \).

Let:

\[
a_3 = \max\{a_1, a_2\} \\
b_3 = b_1 b_2
\]

Then for all \( n \geq a_3, f(n) \leq b_3 h(n) \), and hence by definition, \( f(n) \) is \( O(h(n)) \).

B. \( f(n) \) is \( \Omega(g(n)) \) and \( g(n) \) is \( \Omega(h(n)) \) does not imply \( f(n) \) is \( \Omega(h(n)) \)

For example, let:

\[
f(n) = 1 \\
g(n) = 1 \text{ if } n \text{ is even and } n \text{ if } n \text{ is odd} \\
h(n) = n
\]

Then \( f(n) \geq g(n) \) infinitely often and \( g(n) \geq h(n) \) infinitely often but \( f(n) \) is not \( \Omega(h(n)) \).

Note: This is sort of an odd circumstance; if we used a stronger version of the \( \Omega \) definition that was symmetric with the big \( O \) definition (the condition must be true for all but a finite set of values), then we would have transivity for \( \Omega \). However, in practice, if \( f(n) \) denotes the running time of a program and \( g(n) \) is a smooth monotonic function like \( n \log(n) \) and \( n^2 \), the infinitely often condition of the \( \Omega \) definition is all we really care about (i.e., if the program's running time is as bad as \( g(n) \) infinitely often, that is enough to say it is that bad in the worst case).
3A. For \( n \geq 1 \), prove that \( \lceil \log_2(n) \rceil + 1 \) is the number of bits required to represent \( n \) in binary.

By definition, \( 2^k \) in binary is a 1 followed by \( k \) 0's, and so the expression is correct when \( n \) is a power of 2.

In addition, a number \( 2^k \leq 2^k + i < 2^{k+1} \) uses the same number of bits as \( 2^k \), since it is just the low order \( k \) bits that differ.

Since rounding the logarithm down gives \( k \) for any value of \( i \) in this range, the expression remains correct for all \( i \) in this range.

Hence the expression is correct for all \( n \).

3B. For \( n \geq 1 \), describe in English and give pseudo-code to compute \( \lceil \log_2(\lceil \log_2(n) \rceil) \rceil \).

It is simple to make a function \( \text{LOG}(n) \) that determines how many times you can divide by 2 to get 1 (or it could determine how many times to multiply by 2 to get \( n \)). Then we can do \( \text{LOG}(\text{LOG}(n)) \):

Define the function \( \text{LOG} \):

\[
\text{function LOG}(n) \\
\quad x = 0 \\
\quad \text{while } n > 1 \text{ do begin} \\
\quad \quad n = n/2 \\
\quad \quad x = x+1 \\
\quad \text{end} \\
\quad \text{end} \\
\quad \text{return } x
\]

Then do:

\( \text{LOG}(\text{LOG}(n)) \)

\text{Note:} Equivalently, we could have defined \( \text{LOG}(n) \) by starting with 1 and counting the number of times we had to multiply by 2 to get \( n \).
4. Binary search worked by dividing the problem in half. Given an array $A[1] \ldots A[n]$ of distinct integers that is sorted (i.e., $A[i] < A[i+1]$) and an integer $x$, describe in English and give pseudo-code for ternary search that works as a generalization of binary search algorithm presented in class to find the position of $x$ in $A$ (or determines that $x$ is not in $A$) by dividing the problem into three parts (that is, it works by making at most two comparisons and then narrows the range to size $\frac{n}{3}$).

The question pretty much describes the idea. As with the standard algorithm, we must be careful when the range narrows to 1, not to get stuck in an infinite loop. For simplicity, we terminate the loop when $b - a = 1$, rather than when $b = a$ as in the standard algorithm:

while $(a+1)<b$ do begin
    
    $m1 := \lfloor a + \frac{b-a}{3} \rfloor$
    $m2 := \lfloor a + \frac{2(b-a)}{3} \rfloor$
    
    if $x \leq A[m1]$ then $b = m1$
    else if $x \leq A[m2]$ then begin
        $a = m1 + 1$
        $b = m2$
    end
    else $a = m2 + 1$

    end

if $x=A[a]$ then \{ $x$ is at position $a$ \}
else if $x=A[b]$ then \{ $x$ is at position $b$ \}
else \{ $x$ is not in $A$ \}