You miss 100% of the shots you don't take

Happy Birthday to Wayne Gretzky
Question 1 (10 marks):

(a) [5 marks] Suppose \( f(n) \) is in \( \Omega(n^2) \) and \( g(n) \) is in \( \Omega(n^3) \)

What is the \( \Omega \) class of \( f(n) + g(n) \)? Explain your answer.

(b) [5 marks] Suppose \( f(n) \) is in \( \Theta(n^2) \) and \( g(n) \) is in \( \Theta(n^3) \)

What is the \( \Theta \) class of \( f(n) + g(n) \)? Explain your answer.
Question 2 (10 marks):

Here is an algorithm that prints the base-2 representation of a base-10 non-negative integer.

convert(n):
    # n is a non-negative integer
    if n <= 1:
        print n
        return
    else:
        r = n % 2  # n % 2 is n mod 2
        convert(n/2)  # integer division rounds down
        print r

Rewrite the algorithm in a non-recursive form that uses a stack.

You may assume that a Stack class has been defined with object methods push(x), pop() and isEmpty().

Here is a start for your revised algorithm. Feel free to replace this if you wish.

convert(n):
    # n is a non-negative integer
    if n <= 1:
        print 1
    else:
        myStack = new Stack()
For Question 3 and Question 4:

Assume that we have a **Vertex** class and a **Tree** class.

**Vertex** objects have the following attributes:

- **value**, which we will assume is an integer
- **left**, which is either **None** or a link to another **Vertex**
- **right**, which is either **None** or a link to another **Vertex**

**Tree** objects have just one attribute:

- **root**, which is either **None** or a link to a **Vertex**
Question 3 (15 marks):

Create an algorithm which will take a binary search tree T and a value x and return the largest value in T which is \( \leq x \). If there is no such value, your algorithm should return “None”. For example, if your tree contains the values \{3,4,7,19,20\} and \( x = 18 \), the algorithm should return 7. If \( x = 2 \), the algorithm should return “None”.

Your solution should take advantage of the lexical ordering in the tree, and should be as efficient as you can make it. Your algorithm may be recursive or non-recursive. You should add comments to explain what your algorithm is doing.

Here is a header for your algorithm. You can substitute your own if you wish:

```python
find(T, x):  # T is a tree, x is a value
```
Question 4 (15 marks):

What is the Big-O (worst case) complexity of the total time required to build a binary search tree containing \( n \) values using the recursive insert algorithm given in class? The algorithm is repeated here for your convenience. Explain your answer.

```python
insert(T, x):
    T.root = rec_insert(T.root, x)

def rec_insert(current, x):
    if current == None:
        return new Vertex(x)
    else if current.value > x:
        current.right = rec_insert(current.right, x)
    else:
        current.left = rec_insert(current.left, x)
    return current
```

Hint: suppose the values to be inserted are in ascending order. What does the binary tree look like? How long does it take to insert each value?
BONUS TRIVIA QUESTION [0 marks]:

A Canadian named Dai Vernon, known to almost everyone as "The Professor", was widely considered to be the world's greatest expert in his field. What was his field?