Terminology:

**problem** - question we seek an answer for, e.g., "what is the largest item in a list/array?"

**parameters** - variables with unspecified values

**problem instance** - assignment of values to parameters, i.e., the specific input to the problem

```
myList:
    0 1 2 3 4 5 6
      5 10 2 15 20 1 11

largest: ?

n: 7
```

**algorithm** - step-by-step procedure for producing a solution

**basic operation** - fundamental operation in the algorithm (i.e., operation done the most) Generally, we want to derive a function for the number of times that the basic operation is performed related to the **problem size**.

**problem size** - input size. For algorithms involving lists/arrays, the problem size is the number of elements ("n").

```
import time

def main():
    aList = range(1,1000001)
    start = time.time()
    sum = sumList(aList)
    end = time.time()
    print "Time to sum the list was %.3f seconds" % (end-start)

def sumList(myList):
    """Returns the sum of all items in myList""
    total = 0
    for item in myList:
        total = total + item
    return total

main()
```

a) What is the basic operation?

b) What is the problem size?

c) What would determine how fast this algorithm would run?
**Big-oh Definition** - asymptotic upper bound
For a given complexity function \( f(n) \), \( O(f(n)) \) is the set of complexity functions \( g(n) \) for which there exists some positive real constant \( c \) and some nonnegative integer \( N \) such that for all \( n \geq N \),
\[
g(n) \leq c \times f(n).
\]

**Problem with big-oh:**
If \( T(n) \) is \( O(n) \), then it is also \( O(n^2) \), \( O(n^3) \), \( O(n^4) \), \( O(2^n) \), …. since these are also upper bounds.

**Omega Definition** - asymptotic lower bound
For a given complexity function \( f(n) \), \( \Omega(f(n)) \) is the set of complexity functions \( g(n) \) for which there exists some positive real constant \( c \) and some nonnegative integer \( N \) such that for all \( n \geq N \),
\[
g(n) \geq c \times f(n).
\]

"Proof": Pick \( c = 110 \) and \( N = 1 \), then \( 100 + 10 n \leq 110 n \) for all \( n \geq 1 \).
\[
\begin{align*}
100 & \leq 100 n \\
1 & \leq n
\end{align*}
\]

\( T(n) = c_1 + c_2 \cdot n = 100 + 10 n \) is \( O(n) \).
2. Let \( T(n) = c_1 + c_2 n = 100 + 10 n \). Show that \( 100 + 10 n \) is \( \Omega(n) \).

"Proof": We need to find a \( c \) and \( N \) so that the definition is satisfied, i.e., \( 100 + 10 n \geq c n \) for all \( n \geq N \).

What \( c \) and \( N \) will work?

\[ T(n) = c_1 + c_2 n = 100 + 10 n \text{ is } \Theta(n) \text{ since it is both } O(n) \text{ and } \Omega(n). \]

3. Suppose that you have an \( \Theta(n^2) \) algorithm that required 10 seconds to run on a problem size of 1000. How long would you expect the algorithm to run on a problem size of 10,000?
4. Analyze the below algorithm to determine its theta notation, $\theta()$.

```python
i = n
while i > 0:
    for j in xrange(n):
        # something of $O(1)$
    # end for
    i = i / 2
# end while
```

5.

```python
def linearSearch(target, aList):
    """Returns the index position of target in aList or -1 if target
    is not in aList""
    for position in xrange(len(aList)):
        if target == aList[position]:
            return position
    return -1
```

a) For sequential search, what is the best-case time complexity $B(n)$?

b) For sequential search, what is the worst-case time complexity $W(n)$?

c) If the probability of a successful sequential search is $p$, then what is the probability on an unsuccessful search?

d) If the probability of a successful sequential search is $p$, then what is the probability of finding the target value at a specific index in the array?

e) For each position in the list of size $n$, how many comparisons would be needed for a successful search?

```
0 1 2 3 . . . n-1
```

# compares:

probability:

e) Write a summation for the average number of comparisons.

f) What is the average time complexity, $A(n)$?