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
```

(number of elements)

**largest:** ?

**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.

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

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 \( n \geq N \),
\[
g(n) \leq c \times f(n).
\]

T(n) = \( c_1 + c_2 \ n = 100 + 10 \ n \) is \( O(n) \).

"Proof": Pick \( c = 110 \) and \( N = 1 \), then \( 100 + 10 \ n \leq 110 \ n \) for all \( n \geq 1 \).

100 + 10 \ n \leq 110 \ n
100 \leq 100 \ n
1 \leq n

**Problem with big-oh:**
If T(n) is \( O(n) \), then it is also \( O(n^2) \), \( O(n^3) \), \( O(n^3) \), \( O(2^n) \), ... 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 \( n \geq N \),
\[
g(n) \geq c \times f(n).
\]
T(n) = c_1 + c_2 n = 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?

**Theta Definition** - asymptotic upper and lower bound, i.e., a "tight" bound or "best" big-oh
For a given complexity function $f(n)$, $\Theta(f(n))$ is the set of complexity functions $g(n)$ for which there exists some positive real constants $c$ and $d$ and some nonnegative integer $N$ such that for all $n \geq N$,
$$c \times f(n) \leq g(n) \leq d \times f(n).$$

![Diagram of execution time versus problem size](image)

T(n) = c_1 + c_2 n = 100 + 10 n is $\Theta(n)$.

1) 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?
2) Analyze the below algorithm to determine its theta notation, $\Theta()$.

# Selection sort that sorts a list into ascending order
def selectionSort(myList):
    for lastUnsortedIndex in range(len(myList)-1, 0, -1):
        maxIndex = 0
        for testIndex in range(1, lastUnsortedIndex+1):
            if myList[testIndex] > myList[maxIndex]:
                maxIndex = testIndex
        temp = myList[maxIndex]
        myList[maxIndex] = myList[lastUnsortedIndex]
        myList[lastUnsortedIndex] = temp

3) Analyze the below algorithm to determine its theta notation, $\Theta()$.

i = n
while i > 0 do
    for j = xrange(n) do
        k = 1
        while k < i do
            # something of $O(1)$
            k = k * 2
        # end while
    # end for
    i = i / 2
# end while
def sequentialSearch(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

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

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

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

7) 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?

Write a summation for the average number of comparisons.

8) What is the average time complexity, $A(n)$?
9) For binary search, what is the best-case time complexity $B(n)$?

10) What is the basic operation for binary search?

11) “Trace” binary search to determine the total number of worst-case basic operations?

<table>
<thead>
<tr>
<th>level</th>
<th># of basic operations</th>
<th># of elements</th>
<th>worst-case operations</th>
<th>target</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>$n$</td>
<td>0 1 2 . . . mid n-1</td>
<td>151</td>
</tr>
</tbody>
</table>

10 200