Recursion:

defining something in terms of itself

e.g.

class SNode<E> {  
    SNode<E> next;  
    E element;  
}
Recursive method:

a method that call itself

e.g.  

reverse( list ){
    if list.size == 1
        return list
    else{
        firstElement = list.removeFirst()
        list = reverse( list )
        return list.addLast( firstElement )
    }
}
Recurrence Relation

A recurrence relation is a sequence of numbers \( t(n) \) where the \( n \)-th term depends on previous terms.

- e.g. Fibonacci \( F(n) = F(n-1) + F(n-2) \)

We will consider \( t(n) \) be the time or number of instructions to execute a recursive algorithm, as a function of the input size \( n \).
Example 1: Reversing a list

```java
reverse(list) {
    if list.size == 1
        return list
    else{
        firstElement = list.removeFirst()
        list = reverse(list)
        return list.addLast(firstElement)
    }
}
```

\[ t(n) = \boxed{c} + t(n - 1) \]
Solving a recurrence using back substitution

\[ t(n) = c + t(n - 1) \]
Solving a recurrence using back substitution

\[ t(n) = c + t(n-1) \]
\[ = c + c + t(n-2) \]
Solving a recurrence using back substitution

\[ t(n) = c + t(n-1) \]
\[ = c + c + t(n-2) \]
\[ = c + c + c + t(n-3) \]
Solving a recurrence using back substitution

\[
\begin{align*}
t(n) & = c + t(n - 1) \\
& = c + c + t(n - 2) \\
& = c + c + c + t(n - 3) \\
& = \ldots \\
& = cn + t(0).
\end{align*}
\]

which is \( \mathcal{O}(n) \).
Sorting a list

\[
\text{sort}( \text{list} ) \{ \\
\quad \text{if list.size} == 1 \\
\qquad \text{return list} \\
\quad \text{else} \{ \\
\qquad \quad \text{minElement} = \text{list.removeMin()} \\
\qquad \quad \text{list} = \text{sort}(\text{list}) \\
\qquad \quad \text{return list.addFirst( minElement )} \\
\quad \} \\
\}
\]

\[
t(n) = c_1 + c_2 n + t(n - 1)
\]
Let’s look at the simpler recurrence.

\[ t(n) = c \cdot n + t(n - 1) \]
\[ t(n) = cn + t(n-1) \]

\[ = cn + c \cdot (n-1) + t(n-2) \]
\[ t(n) = c n + t(n - 1) \]

\[ = c n + c \cdot (n - 1) + t(n - 2) \]

\[ = \ldots \]

\[ = c \left\{ n + (n - 1) + (n - 2) + \cdots + (n - k) \right\} + t(n - k - 1) \]
$$t(n) = cn + t(n-1)$$

$$= cn + c \cdot (n-1) + t(n-2)$$

$$= \ldots$$

$$= c \left\{ n + (n-1) + (n-2) + \cdots + (n-k) \right\} + t(n-k-1)$$

$$= c \left\{ n + (n-1) + (n-2) + \cdots + 2 + 1 \right\} + t(0)$$
\[ t(n) = c \cdot n + t(n - 1) \]

\[ = c \cdot n + c \cdot (n - 1) + t(n - 2) \]

\[ = \ldots \]

\[ = c \left\{ n + (n - 1) + (n - 2) + \cdots + (n - k) \right\} + t(n - k - 1) \]

\[ = c \left\{ n + (n - 1) + (n - 2) + \cdots + 2 + 1 \right\} + t(0) \]

\[ = \frac{cn(n + 1)}{2} + t(0) \quad \text{which is } O(n^2). \]
Example 3: Tower of Hanoi

tower(n, start, finish, other) {
    if n > 0 {
        tower(n-1, start, other, finish)
        move from start to finish
        tower(n-1, other, finish, start)
    }
}

\[ t(n) = c + 2t(n-1) \]
Tower of Hanoi recurrence

\[ t(n) = c + 2t(n-1) \]
\[ = c + 2(c + 2t(n-2)) \]
Tower of Hanoi recurrence

\[ t(n) = c + 2t(n - 1) \]
\[ = c + 2(c + 2t(n - 2)) \]
\[ = c(1 + 2) + 4t(n - 2) \]
Tower of Hanoi recurrence

\[
\begin{align*}
t(n) &= c + 2t(n - 1) \\
     &= c + 2(c + 2t(n - 2)) \\
     &= c(1 + 2) + 4t(n - 2) \\
     &= c(1 + 2) + 4(c + 2t(n - 3))
\end{align*}
\]
Tower of Hanoi recurrence

\[ t(n) = c + 2t(n - 1) \]
\[ = c + 2(c + 2t(n - 2)) \]
\[ = c(1 + 2) + 4t(n - 2) \]
\[ = c(1 + 2) + 4(c + 2t(n - 3)) \]
\[ = c(1 + 2 + 4) + 8t(n - 3) \]
Tower of Hanoi recurrence

\[ t(n) = c + 2t(n - 1) \]
\[ = c + 2(c + 2t(n - 2)) \]
\[ = c(1 + 2) + 4t(n - 2) \]
\[ = c(1 + 2) + 4(c + 2t(n - 3)) \]
\[ = c(1 + 2 + 4) + 8t(n - 3) \]
\[ = \ldots \]
\[ = c(1 + 2 + 4 + 8 + \cdots + 2^{k-1}) + 2^k t(n - k) \]
\[ = c(1 + 2 + 4 + 8 + \cdots + 2^{n-1}) + 2^n t(0) \]
Tower of Hanoi recurrence

\[
t(n) = c + 2t(n - 1)
\]
\[
= c + 2(c + 2t(n - 2))
\]
\[
= c(1 + 2) + 4t(n - 2)
\]
\[
= c(1 + 2 + 4) + 8t(n - 3)
\]
\[
= \ldots
\]
\[
= c(1 + 2 + 4 + 8 + \cdots + 2^{k-1}) + 2^k t(n - k)
\]
\[
= c(1 + 2 + 4 + 8 + \cdots + 2^{n-1}) + 2^n t(0)
\]
\[
= c(2^n - 1) + 2^n t(0)
\]

which is \( O(2^n) \).
You should know ....

\[ 1 + 2 + 3 + \ldots + k = ? \]

\[ 1 + 2 + 4 + 8 + \ldots + 2^k = ? \]

\[ 1 + x + x^2 + x^3 + \ldots + x^k = ? \]

(See Lecture 2 p. 4 and Exercises 6 Q1)
Example 4: Binary Search

binarySearch( list, value, low, high ){
    if low <= high {
        mid = low + (high - low) / 2
        if value == list[mid]
            return value
        else if value < list[mid]
            return binarySearch(list, value, low, mid - 1)
        else
            return binarySearch(list, value, mid + 1, high)
    }
    else
        return -1
}

\[ t(n) = c + t\left(\frac{n}{2}\right) \]
Suppose $n = 2^k$. 

$$t(n) = c + t(n/2)$$
Suppose \( n = 2^k \).

\[
\begin{align*}
t(n) & = c + t(n/2) \\
& = c + c + t(n/4)
\end{align*}
\]
Suppose \( n = 2^k \).

\[
t(n) = c + t(n/2) = c + c + t(n/4) = c + c + \cdots + t(n/2^k)
\]
Suppose $n = 2^k$. 

\[
t(n) = c + t(n/2) \\
= c + c + t(n/4) \\
= c + c + \cdots + t(n/2^k) \\
= c + c + \cdots + c + t(n/n)
\]
Suppose $n = 2^k$.

\[
t(n) = c + t(n/2) \\
    = c + c + t(n/4) \\
    = c + c + \cdots + t(n/2^k) \\
    = c + c + \cdots + c + t(n/n) \\
    = c \log_2 n + t(1)
\]

which is $O(\log_2 n)$. 

Today’s Recurrences

\[ t(n) = c + t(n - 1) \]

\[ t(n) = cn + t(n - 1) \]

\[ t(n) = c + 2t(n - 1) \]

\[ t(n) = c + t\left(\frac{n}{2}\right) \]
Logarithms
(review)

**Definition:** for any $b > 0$, the function $x = \log_b(y)$ is the inverse of the exponential function $y = b^x$.

\[
\log_b b^x \equiv x
\]

\[
b^{\log_b y} \equiv y
\]
Some Useful Properties you should know
(assume $a, b, c > 0$)

$$a^{b+c} = a^b \cdot a^c$$

$$(a^b)^c = a^{bc}$$

$$\log_b (a^c) = c \log_b a$$

$$\log_b (ac) = \log_b a + \log_b c$$
Some Less Obvious but Still Useful Properties

(I would give these to you on an exam, if needed.)

\[ \log_b a = (\log_b c)(\log_c a) \]

\[ a^{\log_b c} \equiv c^{\log_b a} \]

Exercise: Try to prove these relationships using the definition on the previous slide. (See lecture notes for proofs.)