Recursion:

defining something in terms of itself

e.g.

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

a method that call itself

e.g.  

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

a sequence $t(n)$ 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 taken to execute a recursive algorithm, as a function of the size $n$ of the input.
Motivation

What is the $O(\ )$ time complexity of recursive algorithms?

Today:

• Reversing a list
• Sorting a list
• Tower of Hanoi
• Binary search
Example 1: Reversing a list

reverse( list ){
    if list.size == 1
        return list
    else{
        firstElement = list.removeFirst()
        list = reverse( list )
        return list.addLast( firstElement )
    }
}
Solving a recurrence using back substitution

\[
t(n) = c + t(n - 1) \\
= c + c + t(n - 2) \\
= c + c + c + t(n - 3) \\
= \ldots \\
= cn + t(0).
\]

which is \( O(n) \).
Sorting a list

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

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

\[ t(n) = cn + t(n - 1) \]
\[ t(n) = c \, n + t(n - 1) \]
\[ = c \, n + c \cdot (n - 1) + t(n - 2) \]
\[ = \ldots \]
\[ = c \left\{ n + (n - 1) + (n - 2) + \cdots + (n - k) \right\} + t(n - k - 1) \]
\[ = c \left\{ n + (n - 1) + (n - 2) + \cdots + 2 + 1 \right\} + t(0) \]
\[ = \frac{cn(n + 1)}{2} + t(0) \]

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)
    }
}

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

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

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 notes 2, page 5)
Example 4: Binary Search

```plaintext
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 \( n = 2^k \).

\[
\begin{align*}
t(n) &= c + t(n/2) \\
     &= c + c + t(n/4) \\
     &= c + c + \cdots + t(n/2^k) \\
     &= c + c + \cdots + c + t(n/n) \\
     &= c \log_2 n + t(1)
\end{align*}
\]

which is \( O(\log_2 n) \).
Summary of today’s recurrences

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

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

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

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

• Midterm exam solutions are posted.

• Assignment 2 is posted.