Recursion (Review) – programming technique where solution to the problem depends on solutions to sub-problems.

2 Parts:

- Base Case: solved without recursion
- Recursive Step: makes progress toward base case

1) Summing values from 1 to n.

Problem: summing from 1 to n.

Sub-problem: “” to values < n.

\[ \sum_{i=1}^{n} i = 1 + 2 + 3 + 4 + \cdots + n \]

= \( n + n-1 + n-2 + \cdots + 1 \)

Write in terms of smaller versions of the same problem (summing)?

\[ \sum_{i=1}^{n} i = n + \sum_{i=1}^{n-1} i \text{ recursion} = \]

\[ = n + n-1 + \sum_{i=1}^{n-2} i \text{ recursion} \]

We can stop expressing as a smaller problem when we get to a point where we can solve the problem directly. Which value of \( n \)? \( n=1 \) → Base case.
Base Case: For \( n=1 \): \[ \sum_{i=1}^{n} i = 1 \] no recursion!

Write as a recursive function:

```c
int sum(int n)

if (n == 1)
    return 1
else
    return (n + sum(n-1));
```

ex: \( \text{sum}(5) = 15 \)

\[
\begin{align*}
5 + \text{sum}(4) &= 5 + 10 = 15 \\
4 + \text{sum}(3) &= 4 + 6 = 10 \\
3 + \text{sum}(2) &= 3 + 3 = 6 \\
2 + \text{sum}(1) &= 2 + 1 = 3 \\
1 &
\end{align*}
\]

ex2: Factorial: \( n! = n \times (n-1) \times (n-2) \times \ldots \times 2 \times 1 \)

How to write \( n! \) in terms of smaller factorials?

\[ n! = n \times (n-1)! \]
\[ = n \times (n-1) \times (n-2)! \]
\[ = n \times (n-1) \times (n-2) \times (n-3)! \]

Base Case? \( n = 1 \) (or \( n = 0 \)). \( \Rightarrow \) then \( n! = 1 \)
factorial(n) =
    if (n = 1)
        return 1
    else
        return n * factorial(n - 1)

ex: factorial(4) = 24
    4 * factorial(3) = 4 * 6 = 24
    3 * factorial(2) = 3 * 2 = 6
    2 * factorial(1) = 2 * 1 = 2

3 Sorted list A, of n numbers. Return true if x is in list; o/w return false.
    check middle

Time to scan? O(n)
Better: Binary Search:

\[
\begin{align*}
    \text{BinSearch}(A) &= \text{BS}(A[0...\frac{n}{2}]) \text{ or } \text{BS}(A[\frac{n}{2}+1...n-1]) \\
    &= \text{BS}(A[0...\frac{n}{4}]) \text{ or } \text{BS}(A[\frac{n}{4}+1...\frac{n}{2}]) \\
\end{align*}
\]

Base Case? When just one element in list.

Time: \( \frac{n}{2} \rightarrow \frac{n}{4} \rightarrow \cdots \) Start with list of size n. Continuously halve
    the list until (worst-case) list is
    of size 1 \( \Rightarrow O(\log n) \).
```plaintext
BinSearch(A, start, end, x)
// if just one element in list (Base Case)
if (start == end)
    if (A[start] == x)
        return true
    else
        return false

k = \frac{start + end}{2}
if (A[mid] == x)
    return true
else if (A[mid] < x) // search right
    return BinSearch(A, mid+1, end, x)
else // search left
    return BinSearch(A, start, mid-1, x).
```
BS(A, 0, 6, 19)

\[ \downarrow \]

mid = 3 \quad A[3] = 12 < 19 \implies \text{right}

BS(A, 4, 6, 19) \quad \text{mid} = 5 \quad A[5] = 20 > 19 \implies \text{left}

BS(A, 4, 4, 19) \quad \text{start} = \text{end} = 4

A[4] = 19 \quad \text{return true}

0x: \ x = 18 \ ?

BS(A, 0, 6, 18)

\[ \downarrow \]

mid = 3 \quad A[3] = 12 < 18 \implies \text{right}

BS(A, 4, 6, 18) \quad \text{mid} = 5 \quad A[5] = 20 > 19 \implies \text{left}

BS(A, 4, 4, 18) \quad \text{start} = \text{end} = 4

A[4] \neq 18 \quad \text{return false}
Recall: Operation contains(x) takes O(n) for ArrayLists, LinkedLists, arrays since there is no ordering among the elements.

Binary Search Trees (BST) - data structure that has more ordering among elements.

Definitions:
- **tree**: an ordered collection of nodes
  - **root** - (a)
  - **edge**
  - **subtree** - tree contained in a larger tree
  - **node**
    - b, c, d, e, f, g, h, i
  - **leaves**
  - **edge** - connects 2 nodes

- **child**: b is child of a if b is directly below a.
- **parent**: a "parent" b if b is above a.
- **leaf**: node with no children
- **root**: "" parent
- **siblings**: nodes with same parent
- **grandparents**: parents of parent
- **grandchildren**: children of child


path: from node \( n_i \) to \( n_k \) - sequence of nodes \\
\( n_1 \ldots n_k \) such that \( n_i \) is parent of \( n_{i+1} \)

example: path from \( a \) to \( k \) \( a \rightarrow c \rightarrow h \rightarrow k \)

path length: \# of edges on the path 

example: \( a \rightarrow c \rightarrow h \rightarrow k \Rightarrow 3 \)

Notice for every node, there is a unique path from root to that node.

depth of a node \( n_i \) - length of path from root to \( n_i \) 

example: depth of \( b \) \( = 1 \) depth of \( k \) \( = 3 \)

height of a node \( n_i \) - length of longest path from \( n_i \) to a leaf 

example: height of \( b \) \( = 2 \), \( a \) \( = 3 \)
Application for Trees

```
<table>
<thead>
<tr>
<th>My Comp</th>
<th>Project</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>Docs</td>
<td>Pics</td>
<td></td>
<td></td>
</tr>
<tr>
<td>- CS201</td>
<td>CS202</td>
<td>Baby</td>
<td>BDay</td>
</tr>
<tr>
<td>- HW1</td>
<td>- Pic1</td>
<td>Pic2</td>
<td></td>
</tr>
<tr>
<td>- Lab1</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

Want to print as:

```
MyComp:
  Project
    C:
      Docs
        - CS201
          HW1
          Lab1
          CS202
        Pics
          Baby
          Pic1
          Pic2
        BDay
      D:
```

"on slide"