Recursion
CS 201
Introduction

● Recursion is an extremely powerful problem-solving technique
  ○ It breaks a problem into smaller identical problems and uses the same function to solve these smaller problems
  ○ It is an alternative to iterative solutions, which use loops

● Facts about recursive solutions
  ○ A recursive function calls itself
  ○ Each recursive call solves an identical but a smaller problem
  ○ Base case must be defined (it enables to stop the recursive calls)
  ○ Eventually, one of the smaller problems must be the base case
**Simple example:** Write a global function that displays a given C-style string backward

Recursive solution:
- Each recursive call diminishes the string length by 1
- Base case: displaying the empty string backward

```c
void displayBackward( char* str ) {
    if ( str[0] == '\0' )
        return;

    displayBackward( str + 1 );
    cout << str[0];
}
```
Recursion and efficiency: Fibonacci function

Recurrence relation:
\[ F(n) = F(n - 1) + F(n - 2) \]

Base cases:
\[ F(1) = 1 \]
\[ F(2) = 1 \]

```c
int recursiveFib( int n ) {
    if ( n <= 2 )
        return 1;
    return recursiveFib( n - 1 ) + recursiveFib( n - 2 );
}

int iterativeFib( int n ) {
    int previous = 1;
    int current = 1;
    int next = 1;       // result when n is 1 or 2

    // compute next Fibonacci values when n >= 3
    for ( int i = 3; i <= n; i++ ) {
        next = current + previous;
        previous = current;
        current = next;
    }
    return next;
}
```
Recursion and efficiency

● Some recursive solutions are so inefficient that they should not be used

● Factors contributing to this inefficiency
  ○ Inherent inefficiency of some recursive algorithms (such as the `recursiveFib` function)
  ○ Overhead associated with function calls

● Do not use a recursive solution if it is inefficient and there is a clear and efficient iterative solution
More examples: Write a recursive function for the binary search algorithm

A high-level pseudocode for binary search

```java
if ( anArray is of size 1 )
    determine if anArray’s item is equal to the searched value
else {
    find the midpoint of anArray
    determine which half of anArray contains the searched value
    if ( the value is in the first half of anArray )
        binarySearch( first half of anArray, value )
    else
        binarySearch( second half of anArray, value )
}
```

Implementation issues

- How to pass “half of anArray” to the function?
- How to determine the base case(s)?
- How to return the result?
More examples: Write a recursive function for the binary search algorithm

```c
int binarySearch( int* arr, int low, int high, int key ) {
    if ( low > high )
        return -1;

    int mid = (low + high) / 2;

    if ( arr[mid] == key )
        return mid;

    if ( arr[mid] > key )
        return binarySearch( arr, low, mid - 1, key );

    return binarySearch( arr, mid + 1, high, key );
}
```
More examples: Write a recursive function that finds the connected components of a given black-and-white image

Application 1: Suppose that we want to locate cell nuclei in a gray-level image whose pixel intensities are in between 0 and 255. To find the nucleus locations, one may first obtain a black-and-white image, whose intensities are either 0 or 1, using some image processing techniques (e.g., thresholding). Then, s/he may identify each connected component of the 1-pixels as a cell nucleus.
More examples: Write a recursive function that finds the connected components of a given black-and-white image.

Application 2: Similarly, in the image below, we want to identify individual buildings. Connected component analysis can be used after obtaining a black-and-white image of buildings.
More examples: Write a recursive function that finds the connected components of a given black-and-white image

```c
int** findConnectedComponents( int** arr, int row, int column ) {

    int** labels, i, j, currLabel;

    labels = new int* [ row ];
    for ( i = 0; i < row; i++ ) {
        labels[ i ] = new int [ column ];
        for ( j = 0; j < column; j++ )
            labels[ i ][ j ] = 0;
    }

    currLabel = 1;
    for ( i = 0; i < row; i++ )
        for ( j = 0; j < column; j++ )
            if ( arr[ i ][ j ] && !labels[i][j] )
                fourConnectivity( arr, labels, row, column, i, j, currLabel++ );

    return labels;
}
```
More examples: Write a recursive function that finds the connected components of a given black-and-white image

```c
void fourConnectivity( int** arr, int** labels, int row, int column,
                      int i, int j, int currLabel ) {

    if ( arr[i][j] == 0 )
        return;
    if ( labels[i][j] > 0 )
        return;

    labels[i][j] = currLabel;

    if ( i - 1 >= 0 )
        fourConnectivity( arr, labels, row, column, i - 1, j, currLabel );
    if ( i + 1 < row )
        fourConnectivity( arr, labels, row, column, i + 1, j, currLabel );
    if ( j - 1 >= 0 )
        fourConnectivity( arr, labels, row, column, i, j - 1, currLabel );
    if ( j + 1 < column )
        fourConnectivity( arr, labels, row, column, i, j + 1, currLabel );
}
```