- In this lecture we will:

  1. Define a function inductively (recursive definition containing a base case and a recurrence relation)

  2. Construct a recursive implementation (a Java method) of the function

  3. Construct an iterative implementation (a Java method) of the function

- Suppose we wish to find the sum of a list of numbers.

- Intuitively if you were to do this in your head you would start at the start of the list, keeping account of the total sum in your head and adding each number to this sum until you get to the end of the list.

- This process which you have followed is an iterative process.

- An alternative way to find the total sum of the list of say n numbers is to say that the sum of the n numbers is the sum of the first $n-1$ numbers plus the nth number.

- This is a recursive approach to finding the sum of the list of n numbers since it involves finding the sum of $n-1$ numbers first.

- A recursive approach is based on a recurrence relation and a base case. These are defined mathematically in the recursive definition and so just need to be converted into Java code to write the recursive implementation.

- Now, suppose you want to put some order on the list of numbers.
  - To do this in Java code you might store the list of number in an array.
  - To do this in mathematics you could store the list of numbers as a function.

- Consider the list: 5,7,2,6,11

- This could be stored as an array in Java as follows. Note that the array index of the first number (5) is 0 and of the last number (11) is 4.

```
int numberslist[]={5,7,2,6,11};
```

- This list can be stored as a mathematical function as follows. Note that the first element of each ordered pair is the array index(from above) of each number.
  $f = \{\langle 0,5\rangle, \langle 1,7\rangle, \langle 2,2\rangle, \langle 3,6\rangle, \langle 4,11\rangle\}$

- The sum of the five numbers on the previous slide can be gotten by getting the sum of the first four numbers (5,7,2,6) and then adding the fifth number (11) to this result.

- Then in order to get the sum of the first four numbers you get the sum of the first three numbers (5,7,2) and add the fourth (6) to this.

- Keep repeating this until either the list has just one element left or else no elements. This is the base case.

- If you know the list will always have at least one number then the base case can deal with this situation. However, if it is possible that the list may be empty then the base case should reflect this. Our base case will handle this latter situation.

- Now we can give the recursive definition. Our function Sum should sum the second element of the ordered pairs of f. (This is the same as saying that Sum should sum the original list of numbers.)

- Recursive Definition:
  - Base Case: $Sum(f, 0) = 0$
  - Recurrence Relation:
    $Sum(f, n) = Sum(f, n-1) + f(n-1)$

- The base case says that the sum of 0 numbers from the list denoted by f is 0.

- The recurrence relation says that the sum of the first n numbers from f equals the sum of the first $n-1$ numbers plus the n^{th} number.

- The n^{th} number is denoted by $f(n-1)$ since (like in arrays in Java) the first number is denoted by $f(0)$ and so on.

- Now we will demonstrate how the recursive definition can be used to find the sum of the list of numbers stored in f.

- $f = \{\langle 0, 5\rangle, \langle 1, 7\rangle, \langle 2, 2\rangle, \langle 3, 6\rangle, \langle 4, 11\rangle\}$

- The first call to this function is $Sum(f, 5)$ since there are 5 numbers in the list. Applying the recurrence relation here gives:
  $Sum(f, 5) = Sum(f, 4) + f(4)$
  $Sum(f, 4)$ needs to be calculated to evaluate $Sum(f, 5)$. Apply recurrence relation again and keep doing this until you reach the base case.

- $Sum(f, 4) = Sum(f, 3) + f(3)$
  $Sum(f, 3) = Sum(f, 2) + f(2)$
  $Sum(f, 2) = Sum(f, 1) + f(1)$
  $Sum(f, 1) = Sum(f, 0) + f(0)$

- $Sum(f, 0)$ is specified by the base case so we can substitute in the value 0 for $Sum(f, 0)$.

- From this we can work back to calculate $Sum(f, 5)$.

  $Sum(f, 1) = Sum(f, 0) + f(0) = 0 + 5 = 5$
  $Sum(f, 2) = Sum(f, 1) + f(1) = 5 + 7 = 12$
  $Sum(f, 3) = Sum(f, 2) + f(2) = 12 + 2 = 14$
  $Sum(f, 4) = Sum(f, 3) + f(3) = 14 + 6 = 20$
  $Sum(f, 5) = Sum(f, 4) + f(4) = 20 + 11 = 31$

- A recursive implementation for Sum looks very like the recursive definition.

- A recursive implementation needs an if-else statement.

- The if branch implements the base case and the else branch implements the recurrence relation.

```
int SumMethod(int f[], int n)
{
    if (n==0)
       return 0;
    else
       return (SumMethod(f,n-1) + f[n-1]);
}
```

- In the next lecture we will:
  - Implement (i.e. write a Java method) the recursive definition of Sum iteratively (using a while loop).
  - Draw a flowchart for this iterative implementation.
  - Annotate the flowchart with assertions.
  - Prove by induction that the iterative implementation of the recursive definition is correct.