

CS 152
Computer Programming
Fundamentals
Recursion

Brooke Chenoweth

University of New Mexico

Fall 2024

What is Recursion?

- Defining something in terms of itself.
- A *recursive definition* is one that uses the concept or thing that is being defined as part of the definition.
 - For example: An “ancestor” is either a parent or an ancestor of a parent.
- A *recursive method* is a method that calls itself, either directly or indirectly.

Factorial

$$0! = 1$$

$$1! = 1$$

$$2! = 2 \cdot 1$$

$$5! = 5 \cdot 4 \cdot 3 \cdot 2 \cdot 1$$

$$n! = n \cdot (n - 1) \cdot (n - 2) \cdots 3 \cdot 2 \cdot 1$$

Factorial – Recursive definition

$$0! = 1$$

$$n! = n \cdot (n - 1)! \text{ when } n > 0$$

Recursive Implementation

```
public static long factorial(long n) {  
    if(n < 1) return 1;  
    else return n * factorial(n - 1);  
}
```

The code corresponds to the mathematics.

Loop implementation

```
public static long factIter(long n) {  
    long result = 1;  
    for(long i = n; i > 0; i--) {  
        result *= i;  
    }  
    return result;  
}
```

Iterative implementation has more explicit bookkeeping.

Accumulator implementation

```
public static long factIter2(long n) {  
    return factAcc(n, 1);  
}  
  
public static long factAcc(long n, long result) {  
    if(n < 1) return result;  
    else return factAcc(n-1, result*n);  
}
```

Fibonacci Sequence

$$F(0) = 1$$

$$F(1) = 1$$

$$F(n) = F(n - 1) + F(n - 2)$$

```
public static long fib(int n) {  
    if(n <= 1) return 1;  
    else return fib(n-1) + fib(n-2);  
}
```


MergeSort: mergeSort

```
public static int[] mergeSort(int[] nums) {
    if(nums.length <= 1) {
        return nums;
    } else {
        int mid = nums.length/2;
        int[] left = new int[mid];
        int[] right = new int[nums.length - mid];
        for(int i = 0; i < mid; ++i) {
            left[i] = nums[i];
        }
        for(int i = mid; i < nums.length; ++i) {
            right[i-mid] = nums[i];
        }
        int[] leftSorted = mergeSort(left);
        int[] rightSorted = mergeSort(right);
        return merge(leftSorted, rightSorted);
    }
}
```

Mergesort: merge

```
public static int[] merge(int[] a, int[] b) {
    int[] c = new int[a.length + b.length];
    int aIndex = 0; int bIndex = 0; int cIndex = 0;
    while(aIndex < a.length && bIndex < b.length) {
        if(a[aIndex] < b[bIndex]) {
            c[cIndex] = a[aIndex]; aIndex++;
        } else {
            c[cIndex] = b[bIndex]; bIndex++;
        }
        cIndex++;
    }
    while(aIndex < a.length) {
        c[cIndex] = a[aIndex]; aIndex++; cIndex++;
    }
    while(bIndex < b.length) {
        c[cIndex] = b[bIndex]; bIndex++; cIndex++;
    }
    return c;
}
```

Mergesort: testing

```
public static void main(String[] args) {
    int[] arr = new int[]{3, 5, 2, 7, 1, 43, 17, 20};
    printArray(arr);
    int[] sorted = mergeSort(arr);
    printArray(sorted);
}

public static void printArray(int[] array) {
    System.out.print("[");
    String delim = "";
    for(int i = 0; i < array.length; ++i) {
        System.out.print(delim + array[i]);
        delim = ", ";
    }
    System.out.println("]");
}
```