



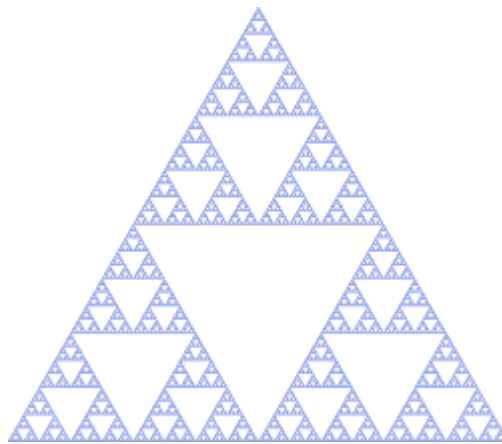
Islamic University of Gaza
Faculty of Engineering
Computer Engineering
Department



ECOM 3001

Data Structures and Algorithms Lab [Fall 2024]

Lab "04" Topic:
Recursion



Prepared by:
Eng. Waseem Alfarram

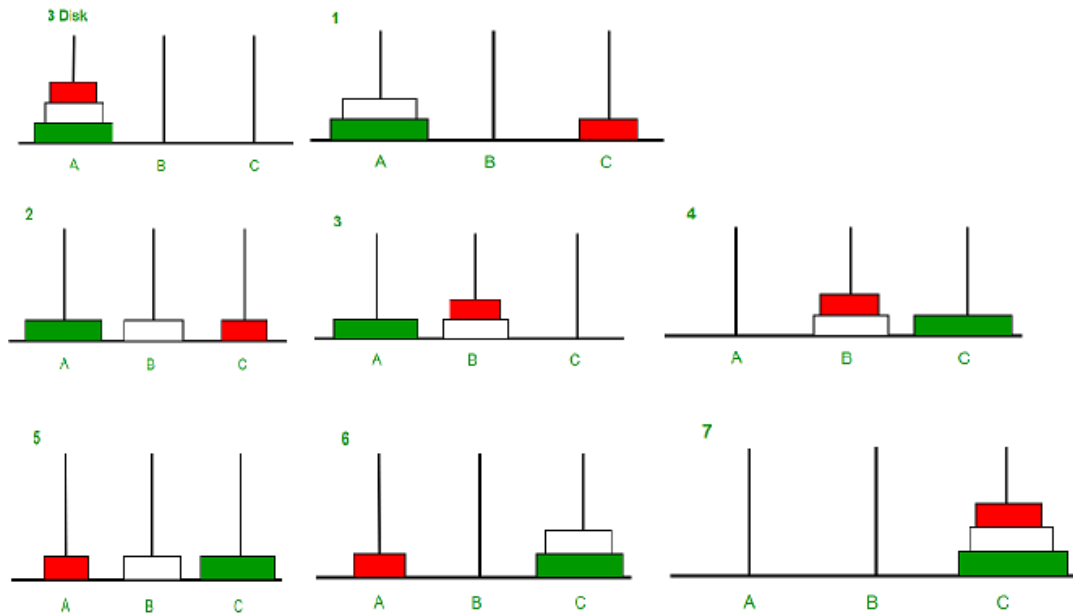
Supervised by:
Dr. Ahmad Mahdi

Lab Objectives:

- To define **Recursion** and how it makes things and solutions easier to solve.
- To learn the different types of **Direct** and **Indirect** recursion and their subtypes with example about each one.
- To recognize the Big-Oh for each type and how it works step by step.
- To convert each recursion type from a recursive approach to an iterative one and comparing the complexities for each one.
- To solve some tasks about recursion types as a practice for what you learned in this lab.

What is Recursion?

It is the process in which a function calls itself directly or indirectly and the corresponding function is called a recursive function. Using recursive algorithm, certain problems can be solved quite easily. Examples of such problems are [Towers of Hanoi \(TOH\)](#) .



Types of Recursions:

Recursion is mainly of two types depending on whether **a function calls itself from within itself** or **more than one function call one another mutually**. The first one is called direct recursion and another one is called indirect recursion. Thus, the two types of recursions are:

1. Direct Recursion: It is when a function calls itself from within itself. This can be further categorized into four types:

- Tail Recursion
- Head Recursion
- Tree Recursion
- Nested Recursion

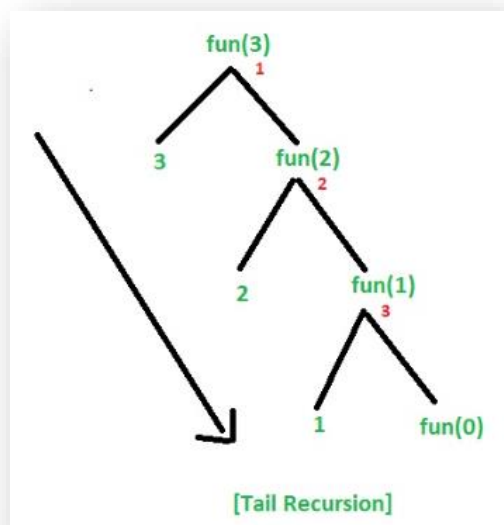
2. Indirect Recursion: It is when a function calls more than one function call one another mutually in a circular manner.

Direct Recursion:

❖ Tail Recursion:

If a recursive function calling itself and that recursive call is the last statement in the function then it's known as Tail Recursion. After that call, the recursive function performs nothing. The function has to process or perform any operation at the time of calling and it does nothing at returning time.

➤ Recursive approach:



```
// Recursion function
static void fun(int n) {
    if (n > 0) {
        System.out.print(n + " ");
        // Last statement in the function
        fun(n - 1);
    }
}

// Driver Code
public static void main(String[] args) {
    int x = 3;
    fun(x);
}
```

Output: 3 2 1

Time Complexity: $O(n)$

Space Complexity: $O(n)$

➤ Iterative approach:

```
// Recursion function
static void fun(int y) {
    while (y > 0) {
        System.out.print(" "+ y);
        y--;
    }
}

// Driver Code
public static void main(String[] args) {
    int x = 3;
    fun(x);
}
```

Output: **3 2 1**

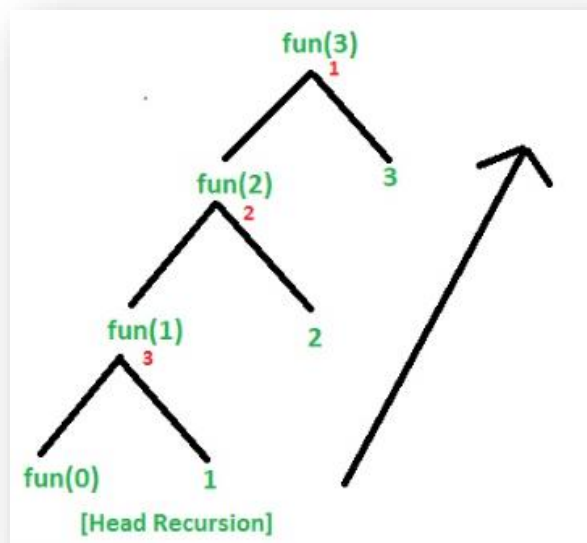
Time Complexity: **O(n)**

Space Complexity: **O(1)**

❖ **Head Recursion:**

If a recursive function calling itself and that recursive call is the first statement in the function then it's known as Head Recursion. There's no statement, no operation before the call. The function doesn't have to process or perform any operation at the time of calling and all operations are done at returning time.

➤ Recursive approach:



```
// Recursion function
static void fun(int n) {
    if (n > 0) {
        // First statement in the function
        fun(n - 1);
        System.out.print(" "+ n);
    }
}

// Driver Code
public static void main(String[] args) {
    int x = 3;
    fun(x);
}
```

Output: 1 2 3

Time Complexity: $O(n)$

Space Complexity: $O(n)$

➤ Iterative approach:

```
// Recursion function
static void fun(int n) {
    int i = 1;
    while (i <= n) {
        System.out.print(" "+ i);
        i++;
    }
}

// Driver Code
public static void main(String[] args) {
    int x = 3;
    fun(x);
}
```

Output: 1 2 3

Time Complexity: $O(n)$

Space Complexity: $O(1)$

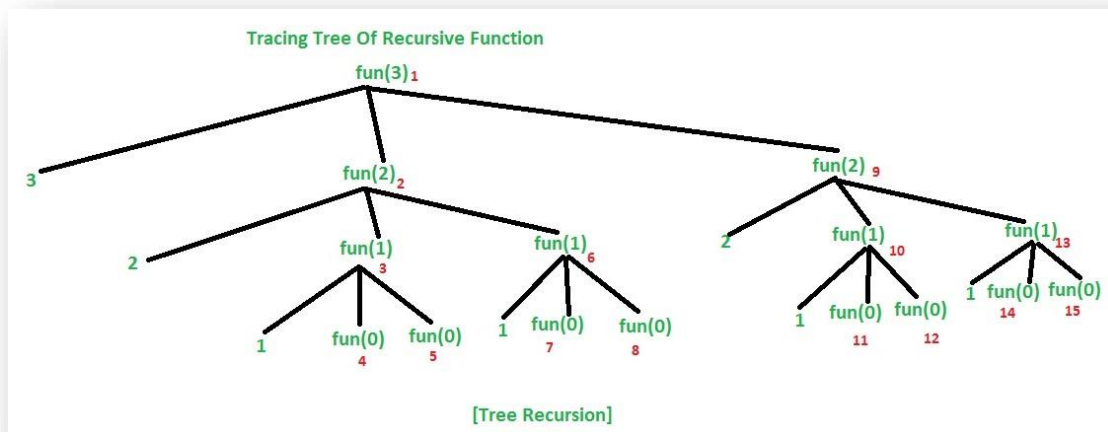
❖ Tree Recursion:

If a recursive function calling itself for one time, then it's known as **Linear Recursion**. Otherwise, if a recursive function calling itself for more than one time then it's known as **Tree Recursion**.

➤ Pseudo Code for linear function:

```
fun(n) {  
    // some code  
    if (n > 0) {  
        fun(n-1); // Calling itself only once  
    }  
    // some code  
}
```

➤ Program for tree recursion:



```
// Recursive function  
static void fun(int n) {  
    if (n > 0) {  
        System.out.print(" " + n);  
        // Calling once  
        fun(n - 1);  
        // Calling twice  
        fun(n - 1);  
    }  
}  
  
// Driver code  
public static void main(String[] args) {  
    fun(3);  
}
```

Output: 3 2 1 1 2 1 1

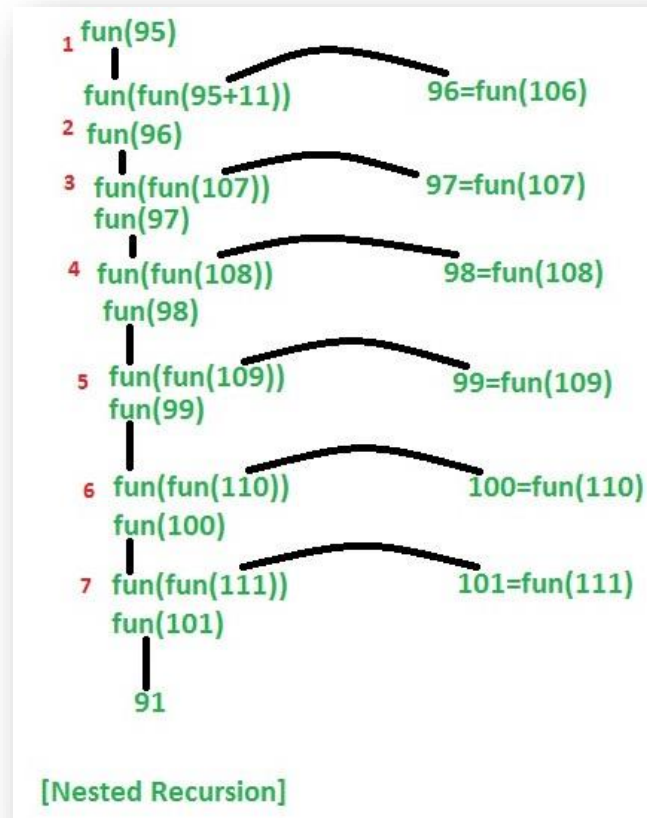
Time Complexity: $O(2^n)$

Space Complexity: $O(n)$

❖ Nested Recursion:

In this recursion, a recursive function will pass the parameter as a recursive call. That means “**recursion inside recursion**”. Let see the example to understand this recursion.

➤ Program for nested recursion:



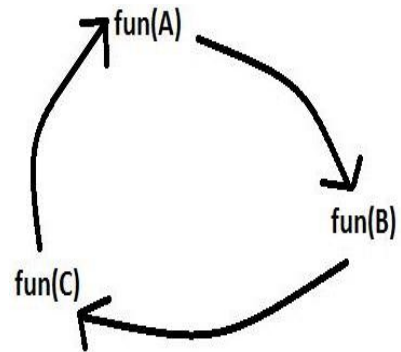
```
static int fun(int n) {  
    if (n > 100)  
        return n - 10;  
  
    // A recursive function passing parameter  
    // as a recursive call or recursion  
    // inside the recursion  
    return fun(fun(n + 11));  
}  
  
// Driver code  
public static void main(String args[]) {  
    int r;  
    r = fun(95);  
    System.out.print(" "+ r);  
}
```

Output: 91

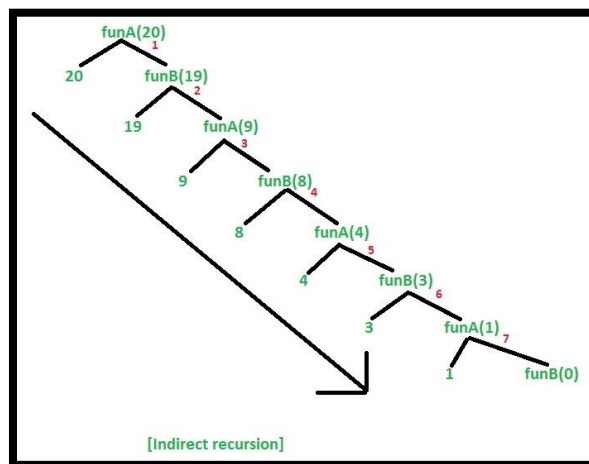
Indirect Recursion:

In this recursion, there may be more than one functions and they are calling one another in a circular manner.

From the shown diagram, fun(A) is calling for fun(B), fun(B) is calling for fun(C) and fun(C) is calling for fun(A) and thus it makes a cycle.



➤ Program for indirect recursion:



```
static void funA(int n) {  
    if (n > 0) {  
        System.out.print(" " + n);  
        // Fun(A) is calling fun(B)  
        funB(n - 1);  
    }  
}  
  
static void funB(int n) {  
    if (n > 1) {  
        System.out.print(" " + n);  
        // Fun(B) is calling fun(A)  
        funA(n / 2);  
    }  
}  
  
// Driver code  
public static void main (String[] args) {  
    funA(20);  
}
```

Output: 20 19 9 8 4 3 1

Exercises:

➤ Exercise 1: (Power of Three)

Leetcode Question Link: <https://leetcode.com/problems/power-of-three/>

Given an integer n , return true if it is a power of three. Otherwise, return false. An integer n is a power of three, if there exists an integer x such that $n == 3^x$.

(Try solving the question using **Direct Recursion**)

```
public boolean isPowerOfThree(int n)
```

Example 1:

Input: $n = 27$

Output: true

Explanation: $27 = 3^3$

Example 2:

Input: $n = 0$

Output: false

Explanation: There is no x where $3^x = 0$

Example 3:

Input: $n = -1$

Output: false

Explanation: There is no x where $3^x = (-1)$

Constraints:

- $-2^{31} \leq n \leq 2^{31} - 1$

➤ Exercise 2: (Pow(x, n))

Leetcode Question Link: <https://leetcode.com/problems/powx-n/>

Implement $\text{pow}(x, n)$, which calculates x raised to the power n (i.e., x^n).
(Try solving the question using **Indirect Recursion**)

```
public double myPow(double x, int n)
```

Example 1:

Input: $x = 2.00000$, $n = 10$

Output: 1024.00000

Example 2:

Input: $x = 2.10000$, $n = 3$

Output: 9.26100

Example 3:

Input: $x = 2.00000$, $n = -2$

Output: 0.25000

Explanation: $2^{-2} = 1/2^2 = 1/4 = 0.25$

Constraints:

- $-100.0 < x < 100.0$
- $-2^{31} \leq n \leq 2^{31}-1$
- n is an integer
- Either x is not zero or $n > 0$
- $-10^4 \leq x^n \leq 10^4$

End of Lab 😊