Recursion
Recursion

- Sometimes, certain statements in an algorithm are repeated on different sizes of an input instance.
- Repetition can be achieved in two different ways.
  - **Iteration**: uses for and while loops
  - **Recursion**: function calls itself

**Example -1:**
- Factorial Function
- Factorial function of any integer $n$ is defined as

\[
 n! = \begin{cases} 
 1 & \text{if } n = 0 \\
 n(n-1)! & \text{if } n \geq 1 
\end{cases}
\]

- This is recursive definition. It consists of two parts:
  i. **Base case**
  ii. **Recursive case**
Recursion

Example -1 (Continue) :

- It can be written as:

<table>
<thead>
<tr>
<th>n</th>
<th>fact(n)</th>
<th>Recursion Case</th>
</tr>
</thead>
<tbody>
<tr>
<td>n = 0</td>
<td>1 if n = 0</td>
<td>Base Case</td>
</tr>
<tr>
<td>n ≥ 1</td>
<td>nfact(n − 1)!</td>
<td></td>
</tr>
</tbody>
</table>

where fact(n) is the function that calculates $n!$. 

Recursion

Example -1 (Continue) :

− Implementation

Recursive:

```java
public static int recursiveFact(int n) {
    if(n==0) return 1;
    else
        return n*recursiveFact(n-1);
}
```

Iterative:

```java
public static int iterativeFact(int n) {
    int fact = 1;
    for(i = 1; i <= n; i++)
        fact=fact*I;
    return fact;
}
```
Recursive Trace

- A graphical representation of recursive calls.
- It is used to analyze the algorithm.

**Example -2:**

Recursive trace for `recursiveFact(4)`

```
public static int recursiveFact(int n) {
    if(n==0) {
        return 1;
    } else {
        return n*recursiveFact(n-1);
    }
}
```
Main Types of Recursion

- Linear Recursion
- Binary Recursion
Linear Recursion

In this case a recursive method makes at most one recursive call each time it is invoked.

Example – 3:

- **Problem:** Given an array A of n integers, find the sum of first n integers.
- **Observation:** Sum can be defined recursively as follows:

\[
\text{Sum}(n) = \begin{cases} 
A[0] & \text{if } n = 0 \\
\text{Sum}(n-1) + A[n-1] & \text{if } n \geq 1
\end{cases}
\]

← Base Case

← Recursive Case
Linear Recursion

Example – 3 (Continued):

Algorithm

Sum(A, n)

Input: An integer array A and an integer \( n \geq 1 \), such that A has at least \( n \) elements

Output: The sum of the first \( n \) integers in A.

Processing:

if \( n = 1 \);

\[ \text{return } A[0]; \rightarrow \text{base case} \]

else

\[ \text{return } \text{Sum}(A, n-1) + A[n-1]; \rightarrow \text{recursive case}. \]

Note:

• Base case should be defined so that every possible chain of recursive calls eventually reach a base case.
• Algorithm must start by testing a set of base cases.
• After testing for base cases perform a single recursive call.
Linear Recursion

Example – 3 (Continued):

- Recursive trace for $\text{sum}(A, n)$, where $A = \{4, 3, 6, 2, 5\}$, $n = 5$

Note:

- For an array of size $n$, $\text{Sum}(A, n)$ makes $n$ calls.
- Each spends a constant amount of time.
- So time complexity is $O(n)$. 
Binary Recursion

In this case, a recursive algorithm makes two recursive calls.

Example – 4

- **Problem:** Find the sum of $n$ elements of an integer array $A$.
- **Algorithm:**
  - Recursively find the sum of elements in the *first half* of $A$.
  - Recursively find the sum of elements in the *second half* of $A$.
  - Add these two values

---

**BinarySum($A, i, n$)**

- **Input:** An integer array $A$ and an integer $n \geq 1$, such that $A$ has at least $n$ elements
- **Output:** The sum of the first $n$ integers in $A$.
- **Processing:**
  - if $n = 1$
    - return $A[i]$;
  - Else
    - return $\text{BinarySum}(A, i, \lceil n/2 \rceil) + \text{BinarySum}(A, i+\lfloor n/2 \rfloor, \lfloor n/2 \rfloor)$;
Example – 4 (Continued):

- Recursive trace

```plaintext
BinarySum (A, 0, 5) + BinarySum (A, 3, 2)

BinarySum (A, 0, 3) + BinarySum (A, 3, 2)

BinarySum (A, 0, 2) + BinarySum (A, 2, 1)

BinarySum (A, 0, 1) + BinarySum (A, 1, 1)


6 + 5 + 2 + 8 = 11 + 3 + 11 = 24
```
Binary Recursion

Example – 5

- The Fibonacci Number
  1, 1, 2, 3, 5, 8, 13, 21, 34, 55, . . . . .
- Each number after the second number is the sum of the two preceding numbers.
- These numbers can naturally be defined recursively:

\[
F(n) = \begin{cases} 
1 & \text{if } n = 0 \\
1 & \text{if } n = 1 \\
F(n-1) + F(n-2) & \text{if } n > 1 
\end{cases}
\]

← Base Case-1
← Base Case-2
← Recursive Case
Binary Recursion

Example – 5 (Continued)

- Recursive Implementation of Fibonacci Function

```java
public static int fib(int n) {
    if (n < 2) {
        return 1; // base cases
    } else {
        return fib(n-1)+fib(n-2); // recursive part
    }
}
```
Binary Recursion

Example – 5 (Continued)

- Recursive Trace of Fibonacci Function: fib(5)
Linear Recursion

Example – 6: Binary Search

Problem: Given $S=\{s_0, s_1, \ldots, s_{n-1}\}$ is a sorted sequence of $n$ integers, and an integer $x$. Search whether $x$ is in $S$.

Binary Search Algorithm:

- If the sequence is empty, return -1.
- Let $s_i$ be the middle element of the sequence.
  - If $s_i = x$, return its index $i$.
  - If $s_i < x$, apply the algorithm on the subsequence that lies above $s_i$.
  - Otherwise, apply the algorithm on the subsequence of $S$ that lies below $s_i$. 

Linear Recursion

Example – 6 (continued): Binary Search

  Implementation:

```java
public static int search(int a[], int lo, int hi, int x) {
    if (lo > hi) return -1; // Basis
    else { // Recursive part
        int i = (lo+hi)/2;
        if (a[i] == x) return i;
        else if (a[i] < x)
            return search(a, i+1, hi, x);
        else
            return search(a, lo, i-1, x);
    }
}
```