**Data Strictures:**

1. **What is DS?**

DS is a way to organize data in a way that enables it to be processed in an efficient time.

1. **Algorithm?**

Alog is a set of rules to be followed to solve a problem

**Types of DS**

1. Primitive DS
   1. Int
   2. Float
   3. Char
   4. Boolean
2. Non Primitive DS
   1. Physical : They are implemented standalone, physically present in RAM
      1. Array
      2. Linked list
   2. Logical: These are logically nature, depends on physical ds.
      1. Stack
      2. Queue
      3. Tree
      4. Graph

**I. Recursion**:

* Same operation performs multiple times with diff. input
* Try to make problem smaller in every step
* Must have a base condition, where we need to stop recursion

**Pros of Recursion:**

* Makes code easy to write, whenever a given problem broken down to small problems.
* It always used in DS, like Tree, Graphs
* Always used in problem solving techniques: “Divide and Conquer”, “Greedy” and “Dynamic Programming”

**Format of Recursive Function:**

* Recursive Case: Case where the function recr
* Base Case: Case where the function doesn’t recr

**How Recursive method store in stack memory**:

* **Main()**

Bar()

System.out.println(“”)

* **Bar()**

DoWork()

System.out.println(“”)

* **DoWork()**

DoMore ()

System.out.println(“”)

* **DoMore ()**

System.out.println(“”)

**Stack Memory**: System maintained memory for method invocation.

Push: If anything store in stack

PoP: If anything move out

Stack follows “LIFO (Last In First Out)”

**Recursive Problem Solving:**

1. **Factorial:**

* Is the product of all +ve integers from 1 to n
* Denoted by n!
* Factorial of a non –ve integer n

Factorial (n):

If n equals 0

Return 1

Return (n\*factorial (n-1))

1. **Fibonacci Series:**

* A series of numbers in which each number is the sum of the two preceding numbers.
* Fst 2 numbers by definition are 0 and 1

Fib(n)

If n is lessthan 1

Return error message

Else if n is equals to 1 or 2

Return n-1

Else

Return fib(n-1) +fib(n-2);

**Recursion Vs Iteration:**

Recursion Iteration

Space Efficient: No Yes

Time Efficient No Yes

Ease of code: Yes No

**When to use/avoid Recursion:**

Use**:**

* When we can easily breakdown a problem into smaller problems
* When we are ok with extra time space
* When we need a quick working solution instead of efficient one

**Practical use of ‘Recursion’**

Stack

Tree-Traversal/searching/insertion/Deletion

Sorting-Quick sort, merge sort

Divide and Conquer

Dynamic programming

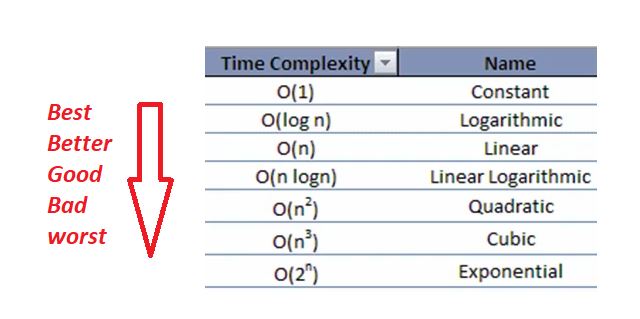
**Algorithm Run Time Analysis:**

Algorithm’s running time by identifying its behavior as the input size for the algorithm increases.

**Notations:**

* **Omega**: This notation gives the tighter lower bound of a given algorithm
* **Big**-O : This gives tighter upper bound of a give algorithm.
* **Theta** () notation decides whether upper bound and lower bound of a given algorithm same or not.

**Algorithm run time complexities:**



**Back Substitution:**

))::

**II. Array**

Array is a data structure consisting of a collection of elements, each identified by array index. An array is stored such that the position of each element can be computed from its index cell by mathematical formula.

**Properties:**

* Array can store data of specified data type
* It has contiguous memory location
* Every cell of an array has an unique index
* Index start with 0
* Size of array needs to be specified mandatorily and cannot be modified

**Types of Arrays:**

**1) One Dimensional Array**: In it each element is represented by a single subscript. The elements are stored in consecutive memory location. EX : Arr[7], Arr[col]

**2) Multi-Dimensional Array:**

a) Two Dimensional Array:

Array[row][col]:: EX: Arr[2][3]

b) Three Dimensional Array

Java having 255 dimensions

**Array Representation in Memory:**

Array stores in memory like one dimensional array only, even if its two, three… dimensional arrays also it’s store as single dimensional array only.

EX:

One Dimensional: [0][1][1][1]

Two Dimensional: [row0][col0][row1][col1]…

Three Dimensional: [Depth0][Row0][col0][Depth0][Row1][Col1]……

**Common Operations on Array:**

* Creation Array: Declaring/Initializing/Instantiating an Array
* Insert a Value
* Traversing given cell
* Accessing given cell
* Searching a given value
* Deleting a given value

**Declaring**: Creates a reference variable to Array [it won’t allocate any memory]

dataType []arr

Ex: int[] arr

**Instantiation**: Creates an Array [It’s allocate memory in RAM]

Array refVar = new dataType[Size]

Ex: array a = new int[5]

**Initializing:** Assigns values to cells in Array [BaseAdress+Index] ex: X102+0, X102+1….

a[0] = 10;

a[1]=20;

**In one Step**: int a[] = {12.20};

**Insert a Value:**

a[0] = 10;

a[1]=20;

**Traversing given cell:**

TraveseArray(array)

Loop: I =0 to array:length

Print arr[i]

**Accessing given cell:**

AccessingCell(arr,cellnumber):

If(cellNumber>sizeof(arr))

Return exception cell number cannot be bigger than size of Array

Else

Retrun arr[cellNumber]

**Searching a given value:**

**Algo:**

SearchinAnArry(arr, valueToSearch);

Loop: I =0 to arr:length

If(arr[i] equals valuetoSearch)

Return i

else

Return error //value not found

**Deleting a given value:** Blank cells have a value as Minimum Integer Value (-2 the power of 31)

**Algo:**

DeletingVlaueFromArray(arry, location);

If(arr[location]is occupied)

Arr[location]= integer.MinValue

Else

Return //location is already blank