Types of DS

primitive

- Integers
- Float
- character
- Boolean

Non-primitive

- linear
  - Static
    - array
  - Dynamic
    - linkedlist
    - stack
    - queues
- Non-linear
  - tree
  - graph

Types of Algorithms

- recursive Algo
- Divide and conquer
  - merge sort
  - quick sort
- Dynamic programming (Find best among available sol^n)
- greedy algorith
- Brute force
- randomized algo

Recursion

A way of solving problem having a Function calling itself.

- same opeation with smaller inputs
- make problem smaller each time
- give a base condition to avoid infinite loop

why we need Recursion ?

- helps in breaking big problem into smaller sub problem.

when to choose ?

- Design an algorith to compute nth
- write code to list the nth ...
- implement method to compute all
- practice
- tree, graph, greedy, DP etc DS

logic Behind Recussion

- calling itself
- exit from infinite loop
- uses stack (time and space overhead ?)

```
static void recussion (string s){
    if (base condition)              ← Base
        return some-value;
    else
        recursion (modified string)  ← Recussion calling
                                        itself.
}
```

let's see an example !

* compute power of two of given number.

```
static void powerTwo (int n){
    if (n==0)                } base condition
       return 1;

else                         } recursion
int ans = 2* powerTwo(n-1)
    return ans;

}
```

working: n = 5

```
ans = 2* (4) = 32 ← It will be final and returned
          ↓
    ans = 2* (3) = 16
              ↓
        ans = 2* (2) = 8
                  ↓
            ans = 2*(1) = 4
                     ↓
                ans = 2*(0) = 2
                         ↓
                         1
```

$2^5 = 32$ Ans

working of Recursion (iteration vs recursion)

points	Recursion	iteration	Reason
space efficient	NO	yes	Rec. uses stack memory space (overhead)
time efficient	NO	yes	Rec. has to push/pop previous func result.
easy to code	yes	NO	

when to use Recursion ??

to use

- Same problem can be divided into similar sub problem
- Trees, Graphs, DP, memoization
- No issue with memory space and time
- less time to code

Avoid
- very high memory/time consuming
- not in apps

steps to write Recursion

Step1 : Recursive case : the flow

$n! = n*(n-1)*(n-2)* \ldots *2*1$

$\boxed{n! = n*(n-1)!}$

Step 2 : Base case : the stopping criteria

Step 3 : unintentional case - the constraints

* write code to find factorial of number

```
static int fac(int n){
    if(n==1 || n==0)            step 2 : the base
        return 1;

    if(n<0)                     step 3 : the constraints.
        return -1;

    else
        return n* fac(n-1);     step1 : the flow

}
```

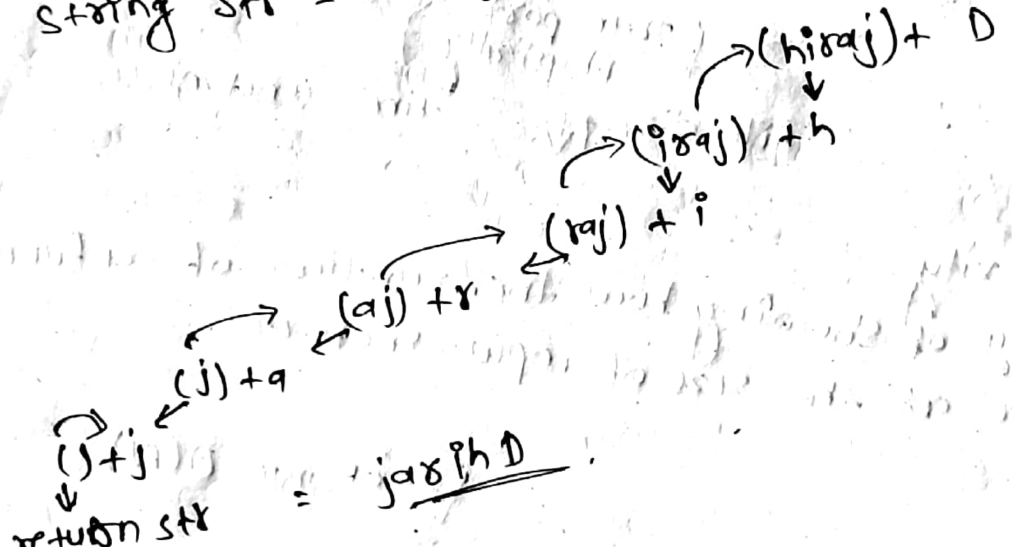
working: 5! = 120

```
5 * (4)  ← = 120
    ↓
   4x(3)  ← = 24
      ↓
     3x(2)  ← = 6
        ↓
       2x(1)  ← = 2
          ↓
         1x(0)  = 1
            ↓
            1
```

Reverse string

```
static string reverse (string str) {
        if (str. isEmpty)
              return str;
    else
     return  reverse (str.substring (1) ~~str.length~~ + str.charAt (0) ;
}
```

working   string str = Dhiraj.



(hiraj) + D

(iraj) + h

(raj) + i

(aj) + r

(j) + a

() + j

return str = jarihD

palindrome

```
    if (str.length() == 0 || str.length() == 1)    } the base
          return true;
    if (str.charAt(0) == str.charAt (str.length() -1)
        return isPalindrome (str.substr (1, str.length() -1)) ;
                                  ↑
                                 the flow
        return False;
```

=|| many other programs are written on pc | Github.