Recursion

The process in which a function calls itself directly or indirecty is called recursion. and the corresponding function is called a recursive function.

direct recursion.

```
f()
{
   f()
}
```

indirect recursion

```
f()
{
   f1()
}
```

```
f1()
{
   f()
}
```

In case of a recursive function, all the recursive functions will be pushed onto stack., if there is no base case or base condition, the stack becomes overflow, so, to avoid this, every recursive function should have a base case.

A recursive function to find the factorial of a given number

$$fact(n)\begin{cases} 1 & : \text{if } n=1 \\ n * fact(n-1) & : \text{otherwise.} \end{cases}$$

```
int fact (int n)
{
    if (n == 1)
        return 1;
    else
        return n* fact(n-1);
}
```

Write recursive function to find sum of n elements.

```
int sum (int n, int a[10])
{
    if (n==1)
        return a[0];
    else
        return a[n-1] + sum (n-1, a);
}
```

To get biggest element in the array.

```
int max (int max, int n, int a [10]
{
    if (n==1)
        if (max < a[0])
            return a[0];
        return max;
    return max (
```

```
int big (int n, int a[10])
{
    if (n==1)
        return a[0];
    return max (big (n-1, a), a [n-1])
}
```

```
int max (int a, int b)
{
    if (a>b)
        return a;
    return b;
}
```

GCD of 2 numbers

```
int gcd (int x, int y)
{
    if (x == y)
        return x;
    if (x > y)
        return gcd(x-y, y);
    if (x < y)
        return gcd(y, x);
}
```

Write recursive function to multiply 2 numbers

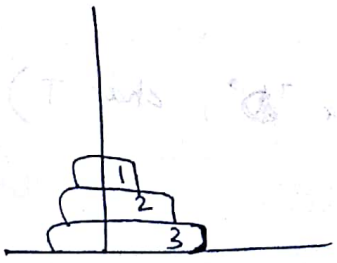
```
int mul (int x, int y)
{
    if (y == 1)
        return x;
    else {
        return [mul(x, y-1) + m];
}
```

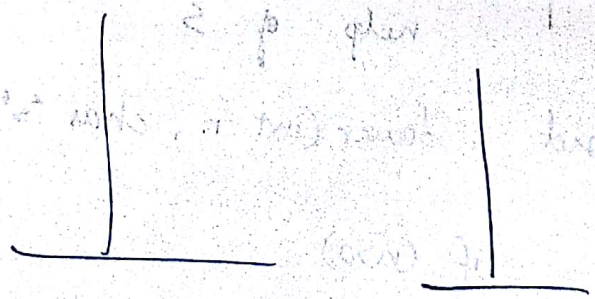
Write a recursive func to perform binary search

```
int bs(int a[30], int low, int high, int key)
{
    if (low > high)
        return -1;
    int mid = (low+high)/2;
    if (key = a[mid])
        return mid+1;
    if (key < a[mid])
        bs(a, low, mid-1, key);
    if (key > a[mid])
        bs(a, mid+1, high, key);
}
```

Tower of Hanoi

In the tower of Hanoi prob, we have 3 poles denoted with source (S), destination(D) and temp(T). On the source (S), n discs are stacked one above the other such that the disc with the larger diameter will be at the bottom.





The objective here is to move all the discs from the source to destination such that the destination should have the same property of source. To acheive this, the limitations are

i) only one disc can be moved across the poles.
ii) at any time, the disc with the larger dia should always be at the bottom.

Let n denotes the number of discs

Case 1: if n=1,

- we need only one move ie move disc from S to D.

if n=2, move are as follows.

- 1st (S to T)
- 2nd (S to D)
- 1 (T to D).

In general, the TOH problem can be solved recursively in the following manner

step 1; recursively move (n-1) discs from S to T. with the help of D.

step 2: Move nth disc from ~~source~~ S to D.

step 3: recursively (n-1) discs from T to D with the help of S.

```
void tower(int n, char 'S', char 'D', char T)
{
    if (n>0)
    {
        Tower(n-1, S, T, D);
        pf("\n move %d disc from %c to %c", n, S, D);
        Tower (n-1, T, D, S);
    }
```