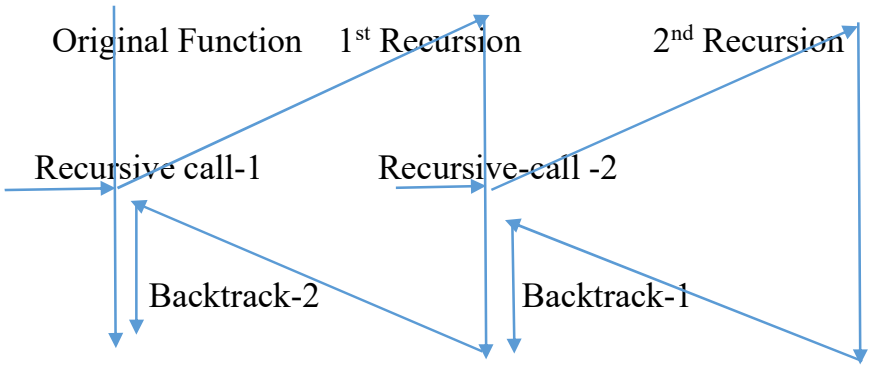
Recursion

Recursion is nothing but a function invoking itself. The notion of recursion is very similar to mathematical induction where in the inductive hypothesis you assume the problem to be true for a smaller sized sub problem without questioning. The idea of recursion is to break the original problem into smaller sized sub problem that you solve recursively and you assume the solution that you obtained are correct without questioning. Only thing that you have to ensure in a recursive program is to merge the solution of the smaller sized problem properly to obtain the correct solution of the original problem.

During recursive call the present environment of recursion is pushed into the stack of activation records and these are popped out of the stack in the backtracking phase of recursion. If there is no backtracking part, i.e., the recursive call is the last statement in the recursive function we call that recursion as tail recursion.



One important point to remember is that any recursive program has to have a terminating condition similar to the basis case of mathematical induction. Otherwise the program would recursively call itself indefinitely and would not terminate. This would overflow the stack of activation records and you will get run time error message.

We can easily convert a recursive program into an iterative program using stack data structure. In case of tail end recursion even this stack is also not required.

We will show the first recursive program for a problem named Tower of Hanoi. In this problem there are n disks and 3 towers namely a, b, c. You have to move these n disks that are initially on tower a to tower c using b as the intermediate tower. There are 2 conditions that has to be satisfied:

1. You can move only one disk at a time.
2. You cannot place a larger sized disk over a smaller sized disk.

To solve this problem using recursion we have to break the original problem into 3 smaller sized sub problems:

1. Move (*n*-1) disks from tower *a* to tower *b* using *c* as the intermediate tower.
2. Move the n^{th} or the largest disk from tower *a* to tower *c*.
3. Move (*n*-1) disks from tower *b* to tower *c* using *a* as the intermediate tower.

The code for this program is as follows:

```
#include<stdio.h>
int tower_of_hanoi(int n, int a, int b, int c)
{
    if(n>0)
        {
          tower_of_hanoi(n-1, a, c, b);
          printf("move disk %d from tower %d to tower
          %d\n", n, a, c);
          tower_of_hanoi(n-1, b, a, c);
        }
    else return(0);
}
```

```
int main()
{
    int n;
    printf("Enter the number of disks\n");
    scanf("%d", &n);
    tower_of_hanoi(n, 1, 2, 3);
    return(0);
}
```

Time Complexity Analysis:

Total number of recursive calls in this program is the total number of moves and can be estimated using the following recurrence:

$T(n) = 2T(n-1) + 1 = 2[2T(n-2) +1] + 1 = 2^2T(n-2) + 2 + 1 = 2[2[2T(n-3) + 1] + 1] + 1 = 2^3T(n-3) + 2^2+2+1 = 2^{(n-1)} +2^{(n-2)} + \ldots + 2+1 = 2^n -1$ [Since T(1) = 1]

Another recursive program for binary search that we have discussed earlier using iteration is described below:

```
#include<stdio.h>
#define MAX 1000
int binary_search(int key, int low, int high, int
                  A[])
{
    int mid;

    if(low<high){

        mid = (low+high)/2;
        if(A[mid]==key) return(mid);
```

```
        else if (A[mid] < key) binary_search(key,
                    mid+1, high, A);
                else binary_search(key, low, mid-1, A);
    }
    else return(-1);
}


int main()
{
    char c;
    int key, i, n, index,  A[MAX];

    printf("Enter the number of array elements\n");
    scanf("%d", &n);

    printf("Enter the array elements\n");

    for(i=0; i<n; i++)
     scanf("%d", &A[i]);

    printf("Enter the key value\n");
    scanf("%d", &key);

    index = binary_search(key, 0, n-1, A);

    if(index<0) printf("unsuccessful search\n");
```

```
    else printf("The key is present in index location
= %d\n", index);


    c=getchar();
    return(0);

}
```