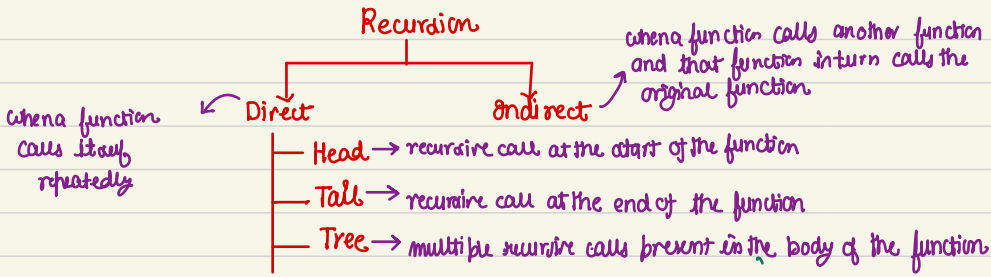
Recursion:

A process by which a function calls itself repeatedly

Components of recursion

Base condition: It is the condition that stops recursion

Recursive case: It is the condition where the function calls itself.



Example of head recursion

```
#include <stdio.h>
void fun(int n)
{
if(n>0) {
fun(n-1);
printf("%d", n);
}
}
```

output if n=3: 1 2 3

head recursion

Example of tail recursion

```
#include <stdio.h>
void fun(int n)
{
if(n>0) {
printf("%d", n);
fun(n-1);
}}
```

output if n=3: 3 2 1

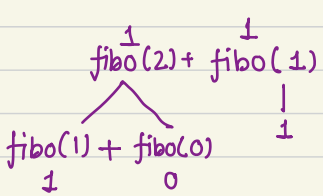
tail recursion

Example of tree recursion

```
#include <stdio.h>
int fibo(int n)
{
if(n<=1)
return n;
else
return fibo(n-1)+fibo(n-2);
}
```

tree recursion

Output if n=3



Output: 2

Time complexity: It is the amount of time taken by an algorithm to run as a function of the length of the input.

Space complexity: It is the amount of memory used by a program or algorithm to solve a problem as a function of length of input.

For tail recursion both are: O(n)

Recursion	Iteration
1) Elegant for dividing problems	1) More efficient in terms of time and memory
2) More readable for problems like tree traversals.	2) No risk of stack overflow.
3) It runs slower	3) It runs faster

sum of squares between m and n

```c
#include <stdio.h>
int sumsquares(int m,int n)
{
    if(m==n)
    return m*m;
    else
    {
    int mid=(m+n)/2;
    return sumsquares(m,mid)+sumsquares(mid+1,n);
    }
}
int main()
{
int ans=sumsquares(2,5);
printf("%d",ans);
return 0;
```

Program to find GCD

Euclid's algorithm

m = 21 n = 9

rem = m % n

m = n n = rem } while rem != 0

```c
#include <stdio.h>
int gcd(int m,int n)
{
    if(n!=0)
    {
        int rem=m%n;
    return gcd(n,rem);
    }
    else
    return m;
}
int main()
{
int ans=gcd(21,9);
printf("GCD= %d \n",ans);
return 0;
}
```

here m>n

Program to find permutation of all characters in a string

Applications of Recursion in C.

* Printing linked list
* Dynamic Programming
* Post fix to infix conversion
* Divide and conquer
* Tree- Graph algorithm
* Searching and sorting algorithm.

Advantages and disadvantages of recursion.

additional resources a computing system uses to manage and execute tasks.

Advantages	Disadvantages
Simplifies solving complex problems by dividing it into smaller problems	Recursive call adds overhead to the call stack
Useful for data structures like Trees and Graphs	May lead to stack overflow if the base case isn't handled properly. (stack cannot store more data)
Increases the readability of code.	It is slower than iterations and not memory efficient.

Recursion in arrays..

Reverse elements of an array using recursion.

swap elements without a third variable

Eg a = 7
b = 9

a = a + b = 16
b = a - b = 16 - 9 = 7
a = a - b = 9

```c
#include <stdio.h>
int sumsquares(int m,int n)
{
    if(m==n)
    return m*m;
    else
    {
    int mid=(m+n)/2;
    return sumsquares(m,mid)+sumsquares(mid+1,n);
    }
}
int main()
{
int ans=sumsquares(2,5);
printf("%d",ans);
return 0;
```

linear search using recursion

```c
#include <stdio.h>
int search(int arr[],int l,int k)
{
    if(l<0)
    return -1;
    if(arr[l]==k)
    return 1;
    else
    return search(arr,l-1,k);
}
int main()
{
int arr[]={1,2,3,45,6,7};
int ans=search(arr,5,7);
printf("%d \n",ans);
return 0;
}
```

```c
int search(int a[], int n, int k) {
   if (a[0] == k ) return 1;
   if ( n == 1 ) return 0;
   return search(&a[1], n-1, k);
}

void printVals(int a[], int n) {
   for (int i = 0; i < n; i++ )
      printf(" %d ", a[i]);
   printf("\n");
}

int main(int argc, char *argv[]) {
   int i, n = 10, sum = 0, k = (argc > 1) ? atoi(argv[1]) : 11;
   int a[] = {14, 46, 33, 41, 44, 48, 36, 42, 27, 43};
   printVals(a, n);
   int ret = search(a, n, k);
   if (ret == 1) printf("%d is found!\n", k);
   else printf("%d is NOT found!!\n", k);
   return 0;
}
```