16/10/24 RECURSION – A process by which a func^n calls itself repeatedly

H.W (Execute head recursion, tail recursion)

gcd(15, 35) = gcd(15, 5) = 5

35 = 15·q + r q = 2, r = 5

15 = 5·3 + 0

Write code for euclidean algorithm

```
#include <stdio.h>
#include <stdlib.h>
int gcd (int m, int n) {
if (n==0) { return m; }
else { return gcd (n, m%n); }}
int main () {
int m=18, n=12;
int res = gcd (m, n);
printf ("%d", res); }
```

Debug in class 9-C – Ignore

17/10/24

Q: Difference b/w user defined and derived data type. → Later

COMPONENTS OF RECURSION:

(1) Base Case: Cond^n that STOPS RECURSION.

(2) Recursive Case: Cond^n where the f^n continues calling itself.

C.W – Ignore

65524

"

65522

65524

3

3

3

Overflow

TYPES OF RECURSION:

(1) Direct Recursion : A f^n directly calls itself.

(2) Head Recursion : Pos^n of its only recursive call is at the start of the f^n.

(3) Tail Recursion : Pos^n of recursive call is at the end of the f^n

(4) Tree Recursion : Multiple recursive calls present in the body of f^n.

(5) Indirect Recursion : A f^n calls another f^n, which in turn calls the original f^n.

→ Head Recursion:

```
void fun (int n) {
  if (n>0){
      fun(n-1);
      printf("%d", n); }
}
int main() {
   int x=3;
   fun(x);
   return 0;
}
```

OUTPUT : 1 2 3

fun(3) 3
|
fun(2) 2
|
fun(1) 1
|
fun(0)

→ Tail Recursion:

```
void fun (int n){
  if (n>0){
     printf ("%d",n); }
     fun(n-1);
}
int main() {
   int x=3;
   fun(x);
   return 0;
}
```

OUTPUT: 3 2 1

fun(3) 3
|
fun(2) 2
|
fun(1) 1
|
fun(0)

TC: O(n) SC: O(n)

ail Recursion

RECURSION	ITERATION
→ Elegant for dividing problems.	→ More efficient in terms of time & memory.
→ More readable for problems like tree traversals.	→ Safer, avoids stack overflow risks.

Iterative method is more efficient

Advantages of Recursion :

1) Simple to solve complex programs
2) Useful for certain data structures : trees & graphs
3) Cleaner code for problems like factorial, fibonacci series, GCD etc.

Disadvantages of Recursion :

1) Recursive calls add overhead to the call stack.
2) Risk of stack overflow
3) Can be slower due to additional f^n calls.

RECURSION IN ARRAYS

1 → Sum of array elements

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

int main(){
int i, n, sum = 0;
int a[] = {29, 27, 21, 36, 22};
n = size of(a) / size of(a[0]);
sum = sumarr(a, n);
printf("%d", sum);
return 0;}
```

2→ Reverse elements of an Array

```
void rev(int a[], int left, int right){
if (left < right){
  a[left] = a[left] + a[right];
  a[right] = a[left] - a[right];
  a[left] = a[left] - a[right];
  reverse(a, ++left, --right);}
}

int main(){
int i, n, sum = 0;
int a[] = {14, 46, 33, 46, 44, 48};
n = sizeof(a)/ size of(a[0]);

reverse(a, 0, n-1);}
```

3→ Linear Search (check whether element is in array or not)

```
int search(int a[], int start, int end, int k) {
 if (a[start]==k) return 1;
 if (start == end) return 0;
 return search(a, ++start, end, k);

int main(){
 int i, n=5; → int k=46;
 int a[] = {14, 46, 33, 46, 44};
 int ret = search(a, 1, n, k);
 if (ret==1){printf("found");}
 else{ printf("not found");}
}
```

→ Tree Recursion

Eg. Fibonacci

```
int fibo (int n){
   if (n==1){
      return 0; }
   else if (n==2){
      return 1; }
   else {
      return (fibo(n-1)+ fibo(n-2)); }
}

int main(){
   int res = fibo(5);
   printf("%d\n", res);
   for (int i=1; i<=5; i++){
      printf("%d ",fibo(i)); }
}
```

Iterative approach

```
void fibo (int n){
   int i=1;
   int first=1;
   int second=1;
   printf("%d %d ",first, second);
   while (i<=n-2){
      int f = first + second;
      first= second;
      second=f;
      i++;
      printf("%d",f);
   }
}
int main (){
   fibo (5);
}
```

Important Programs

① Sum of squares of integers [m,n]

```
int sum (int m, int n){
   int mid;
   if (m==n){ return m*m; }
   else {
    mid =(m+n)/2;
    return (sum (m,mid)+ sum (mid+1, n)); }
}
```

② Power function

```
int power(int base, int n) {
  if (n==0) { return 1; }
  return base * power(base, n-1);

int main() {
  int a=2, n=3;
  int ret = power(a,n);
  printf("%d", ret);
  return 0;
}
```

③ Permutations – Do later