ABES

ABES Engineering College

Estd. 2000

College Code-032

Data Structure Training

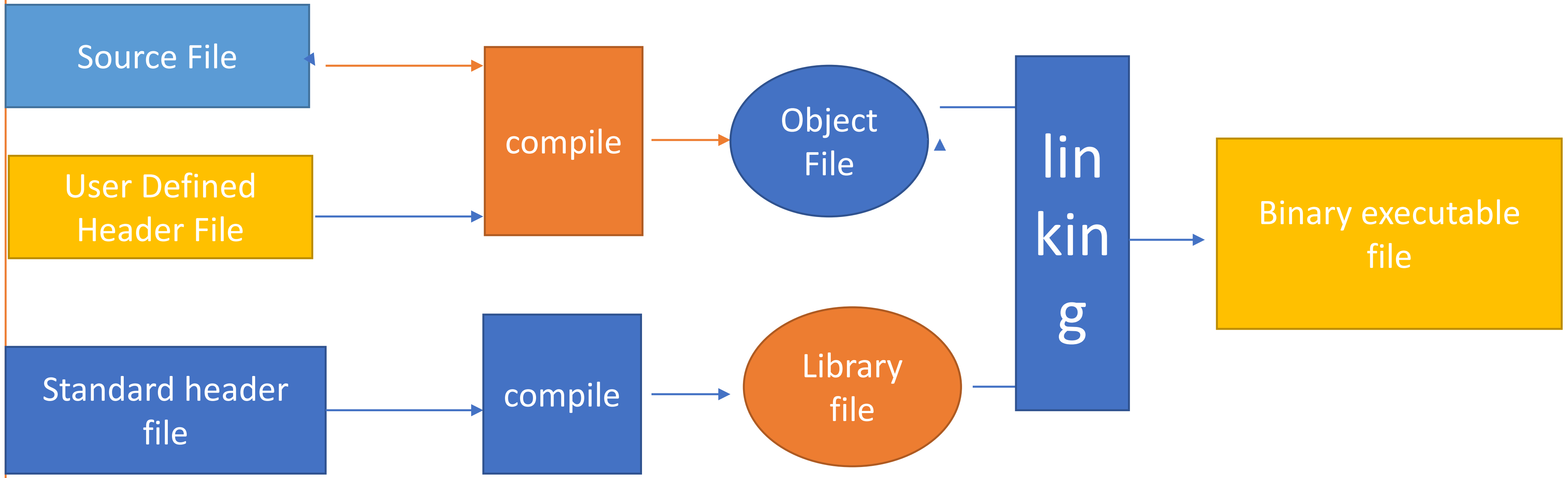
Recursion

Department of CSE, CE, IT, CEIT, CS

ABES Engineering College, Ghaziabad

Data Structure Training(Recursion)

Life Cycle Of Program



1- After compiling and linking the binary executable of program gets generated(.exe on windows).When this file actually executing(running) then it is called as process.

2-When a process is executed , first it is loaded into RAM. Area of memory where process is loaded is called process address space

Process Address Space

#include<Stdio.h> //Library
#include<math.h> //Library

#include"stack.h" //user defined header file
#define Pi 22/7

Program.c
Lines 10
Execution 1785-10 =1775
1- Expansion → Preprocessing (Macro)
2- Linking
    10
    library header file → 1000
    user defined file → 500
    10+1000+500 → 1510

Object file:
High Level Language ==> Low Level Language

Exe File
Binary File
Compilation → Linking (own object code+ library file object code) → Exe

Exe → RAM load
Loading

1. Expansion
2. Compilation → Object file
3. Linking → user object file + Library object file → Binary Executable code
4. Binary Executable Code → loading RAM

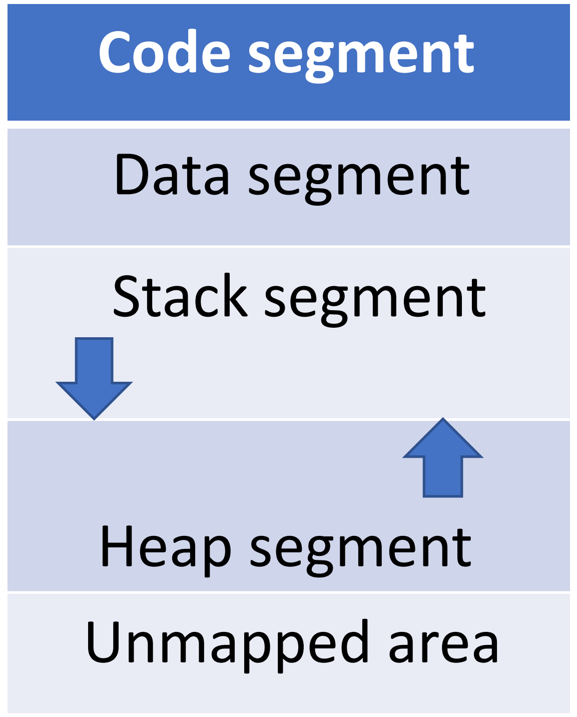
Program.c → Source File

Program.o → Object File

Program.exe → Executable File

Data Structure Training (Recursion)

Process Address Space



Process address space has following segments

1-Code Segment

2-Data Segment

3-Stack

4-Heap

Data Structure Training(Recursion)

Code Segment-

1- This segment contains machine code of compiled program.
2- It is read only and cant be changed when the program is executing.
3-size of code segment is fixed during load time.

Data Segment-

1-All global and static variables are allocated in this segment during load time(before main called).
2-All load time variables are initialized at the load time.
3-Size of data segment is fixed at load time and does not change when program is executing.

Data Structure Training(Recursion)

Stack Segment

Stack segment contains activation record called stack frames of all active functions.
An active function is a function which is currently under the call.

```
#include <stdio.h>
int main()
{
	fun1();
	return 0;
}

fun1()
{
	fun2()
}
fun2()
{
	printf("hello");
}
fun3() // never called so not active function
{

}
```

Data Structure Training(Recursion)

Memory allocation for different segment through example

```
int squareofsum(int x,int y);
int square (int x);
int total;
int main()
{
        int a=4,b=2;
        total=squareofsum(a,b); 1000
        printf("square of sum=%d\n",total);
        return 0;
}
int squareofsum(int x,int y)
{
        static int count=0;
        printf("fun called %d times",++count);
        return square(x+y);
}
int square (int x)
{
        return(x*x);
}
```

Return Value
Return Address 1000
Local Variables
Actual argument a, b
Formal argument x, y

Code Segment	EXE
Data Segment	Total → Global Count → static
Heap Segment	
Stack Segment	
	Square()
	Squareofsum()
	Main()
Unmapped Area	

Data Structure Training(Recursion)

Recursion-

As we know that maximum computer concept(programming concept) comes from mathematics, Recursion is also one of them.

```c
int main()
{
	printf("can not stop programming till i die");
	main();
	return 0;
}
```

Let us discuss a problem of sum of N natural number.

$$\text{Sum}(N) = \begin{cases} N + \text{sum}(N-1) & \text{if } N>1 \\ 1 & \text{if } N=1 \end{cases}$$

Sum Of first N natural number=N + Sum Of first N-1 natural number +..........+1
When N becomes 1 sum(1) has fixed value 1.
It means we are capable to define sum in terms of sum itself, this is called recursion.

Data Structure Training(Recursion)

Recursion-

Now in programming language recursion defined as-

When a function call itself either directly or indirectly then it is called recursive function and process is called recursion.

Points-

1- Recursion is problem solving technique where solution of larger problems defined in terms of smaller instances of itself.

2- Recursion always have terminating condition otherwise infinite recursion.

3- Recursive function performs some part of task and delegate rest of it to recursive call.

How to write recursive Code

In recursion we are not solving complete problem, hence writing recursive code is simple and two step process.

1-Define larger solution in terms of smaller solution of the exact same type with narrow parameter.

2-Try to find terminating condition (base condition).

Data Structure Training(Recursion)

Sum Of n Natural Number

```
int sum(unsigned int n)
{
        if(n==1)
                return 1;
        else
                return n+sum(n-1);
}
```

This function has no issue for n>=1 but for n=0 it behaves in undefined manner. So in recursion always check boundary condition.

```
int sum(unsigned int n)
{
        if(n==0)
                return 0;

        if(n==1)
                return 1;
        else
                return n+sum(n-1);
}
```

Data Structure Training(Recursion)

Sum Of n Natural Number

Code should also written as

```
int sum(unsigned int n)
{
        return (n==0)? 0 :(n==1) ? 1 : n + sum(n-1);
}
```

When there is a choice between simple and complex code then go for simple one if time taken by both are same,otherwise go for lesser one.

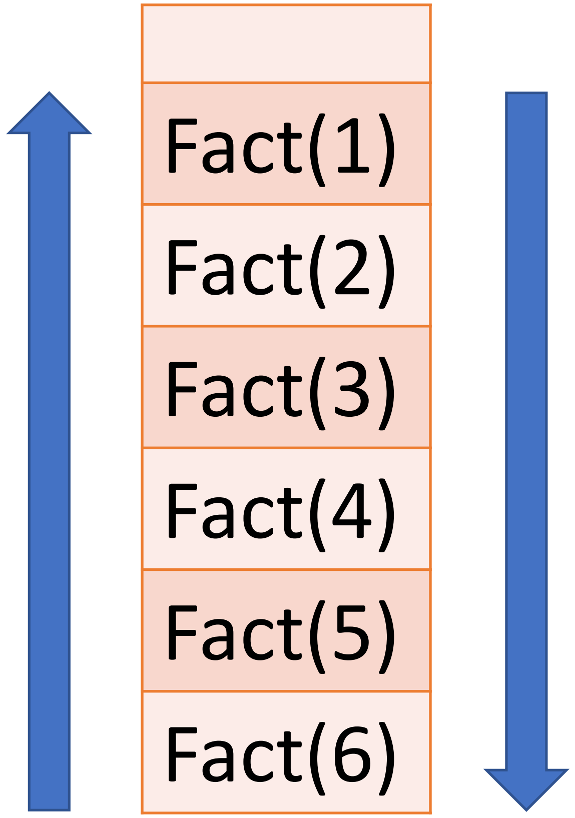
```
Iterative version-
int sum(unsigned int n)
{
        int sum=0,i=0;
        for(i=0;i<=n;i++)
                sum=sum+i;
        return sum;
}
```

When there is a choice between iterative and recursive code then go for iterative one if time taken by both are same, otherwise go for lesser one.

Data Structure Training(Recursion)

Factorial-

```c
#include<stdio.h>
int fact(int numb);
int main()
{
        int numb,f;
        printf("enter number");
        scanf("%d",&numb);
        f=fact(numb);
        printf("factorial=%d",f);
        return(0);
}
int fact(int x)
{
        if(x==0 || x==1)
                return 1;
        else
                return x*fact(x-1);
}
```



Data Structure Training(Recursion)

Power function

```
#include<stdio.h>
int power(int,int);
int main()
{
	int base,exp,f;
	printf("enter base and exp");
	scanf("%d%d",&base,&exp);
	f=power(base,exp);
	printf("power=%d",f);
	return(0);
}
int power(int x,int y)
{
	if(y==0)
	return 1;
	else
	return x*power(x,y-1);
}
```

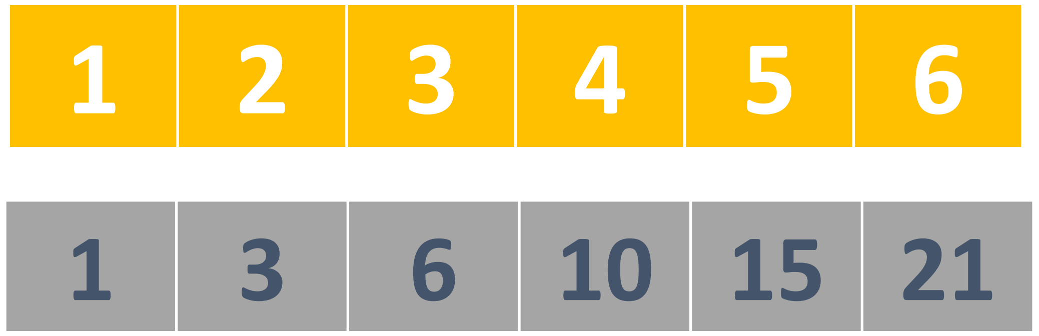
Data Structure Training(Recursion)

Problem

Given an array of integers, write a recursive code that add sum of all the previous numbers to each index of array.

Input- 1,2,3,4,5,6

Output- 1,3,6,10,15,21



Data Structure Training(Recursion)

Problem

Given an array of integers, write a recursive code that add sum of all the previous numbers to each index of array.

Input- 1,2,3,4,5,6

Output- 1,3,6,10,15,21

1	2	3	4	5	6

1	3	6	10	15	21

```
CumulativeSum(A[ ], N)
{
If(N==1)
        return A[0];
Else
        A[N-1] = CumulativeSum(A, N-1)+A[N-1]
        return A[N-1]
}
```

A[5] = A[5]+A[4]

A[4] = A[4]+A[3]

A[3] = A[3]+A[2]

A[2]=A[2]+A[1]

A[1]=A[1]+A[0]

A[0]

Data Structure Training(Recursion)

Focus On four Points-

1-Serve the purpose, for every possible parameters function should return expected value.
2-Time should be minimized
3-Memory should be minimized
4-Function should easy to understand(self explanatory)

Data Structure Training(Recursion)

Head and Tail Recursion-

1-Recursive function perform some task and call itself.

2-If call is made before function perform own task then called head recursion.

3-If function perform own task first and then call is made then tail recursion.

Head and tail Recursion example

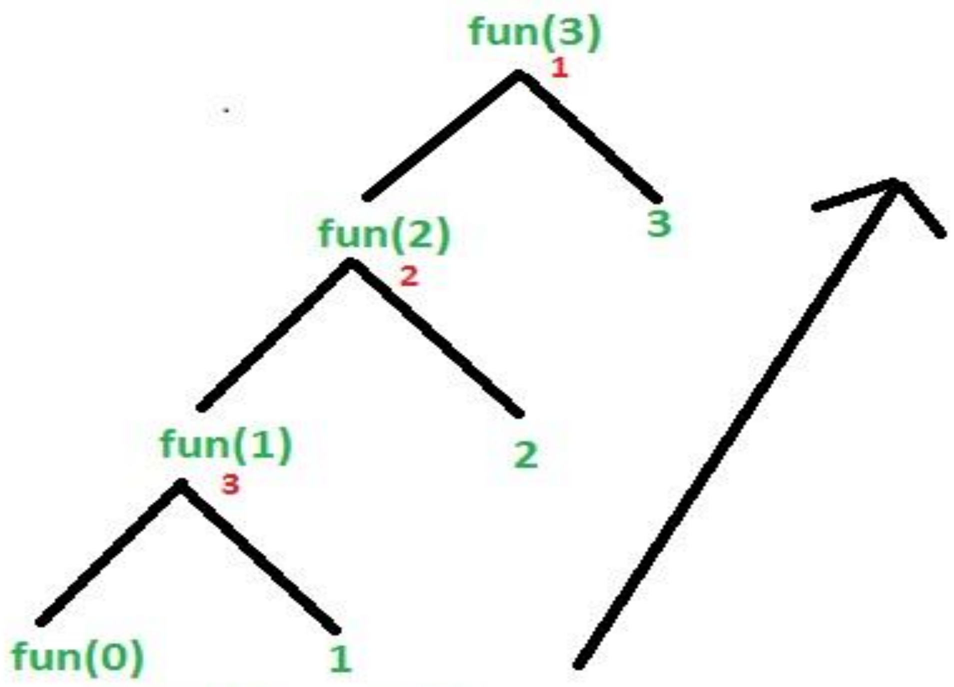
```
void disp(struct node *head)
{
        if(head != NULL)
        {
                disp(head->next);
                printf("%d",head->next);
        }
}
void disp(struct node *head)
{
        if(head != NULL)
        {
                printf("%d",head->next);
                disp(head->next);
        }
}
```

Data Structure Training(Recursion)

Head and tail Recursion example

Write the Recursive function here

Tracing Tree Of Recursive Function



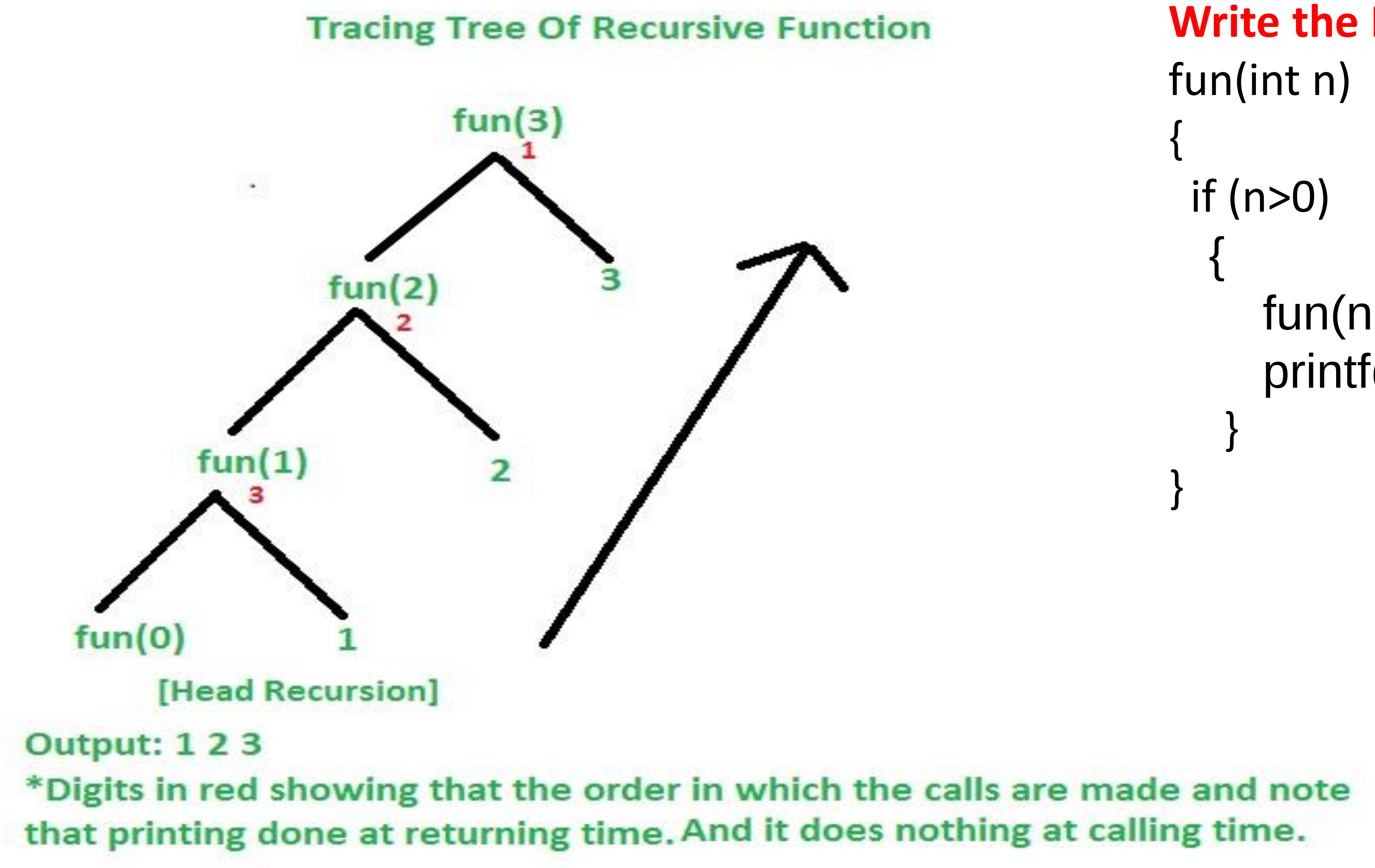
[Head Recursion]

Output: 1 2 3

*Digits in red showing that the order in which the calls are made and note that printing done at returning time. And it does nothing at calling time.

Data Structure Training(Recursion)

Head and tail Recursion example



Tracing Tree Of Recursive Function

[Head Recursion]

Output: 1 2 3

*Digits in red showing that the order in which the calls are made and note that printing done at returning time. And it does nothing at calling time.

Write the Recursive function here

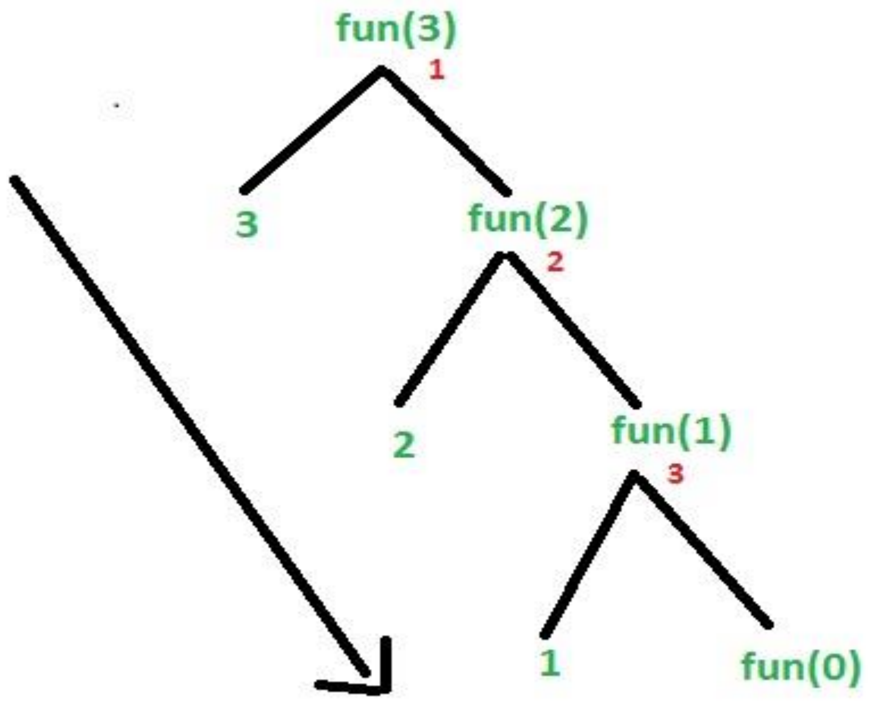
```
fun(int n)
{
 if (n>0)
   {
        fun(n-1);
        printf(“”%d”,n);
    }
}
```

Data Structure Training(Recursion)

Head and tail Recursion example

Write the Recursive function here

Tracing Tree Of Recursive Function

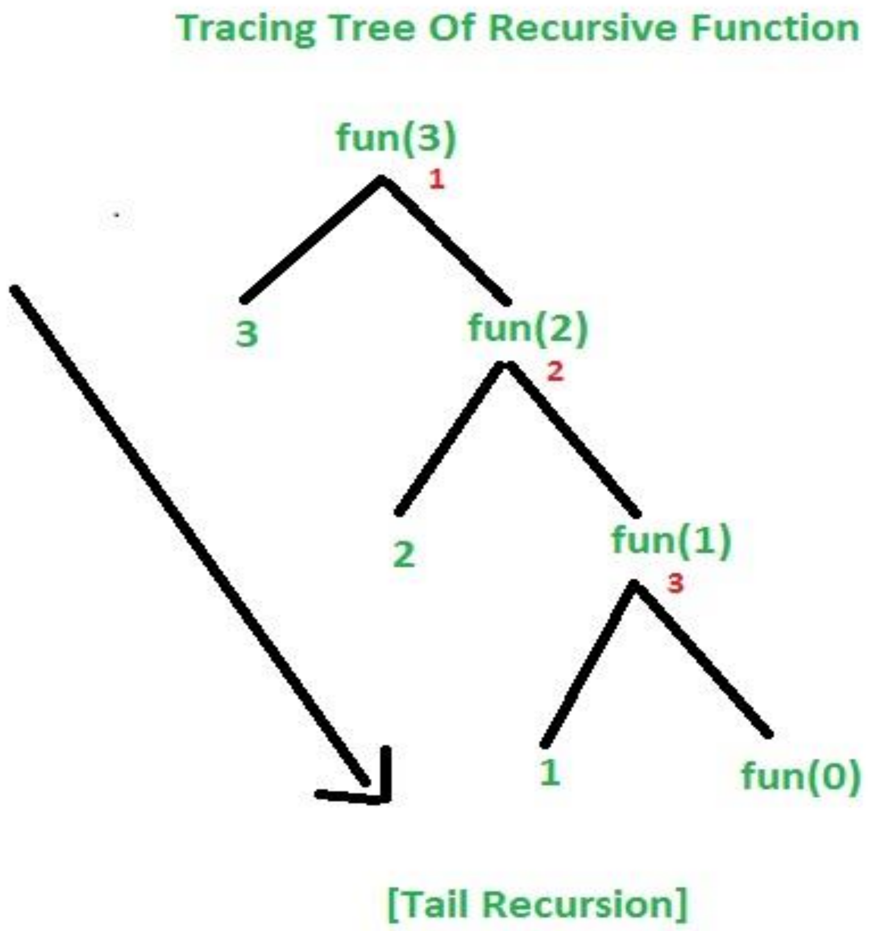


[Tail Recursion]

Output: 3 2 1
*Digits in red showing that the order in which the calls are made and according to the order of calling the output are printed on the screen.Note that for fun(0) it gives nothing as output.

Data Structure Training(Recursion)

Head and tail Recursion example



Output: 3 2 1
*Digits in red showing that the order in which the calls are made and according to the order of calling the output are printed on the screen.Note that for fun(0) it gives nothing as output.

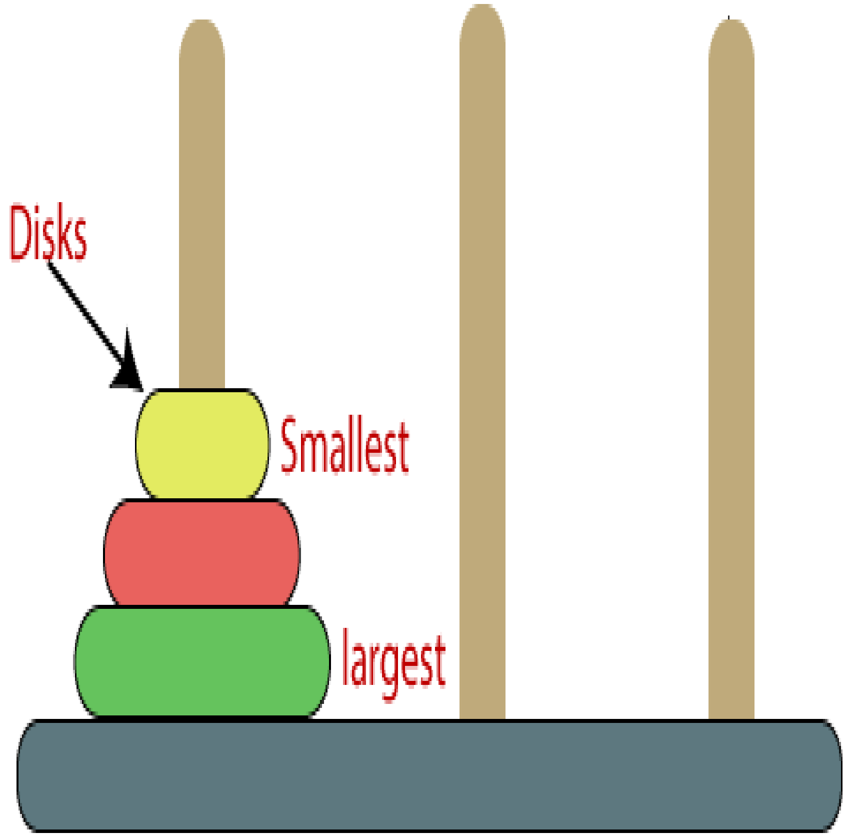
Write the Recursive function here

```
fun(int n)
{
 if (n==0)
   return;
 else
 {
  printf(“”%d”,n);
  fun(n-1);
}
}
```

Data Structure Training(Recursion)

Tower Of Hanoi(Mixed Recursion)

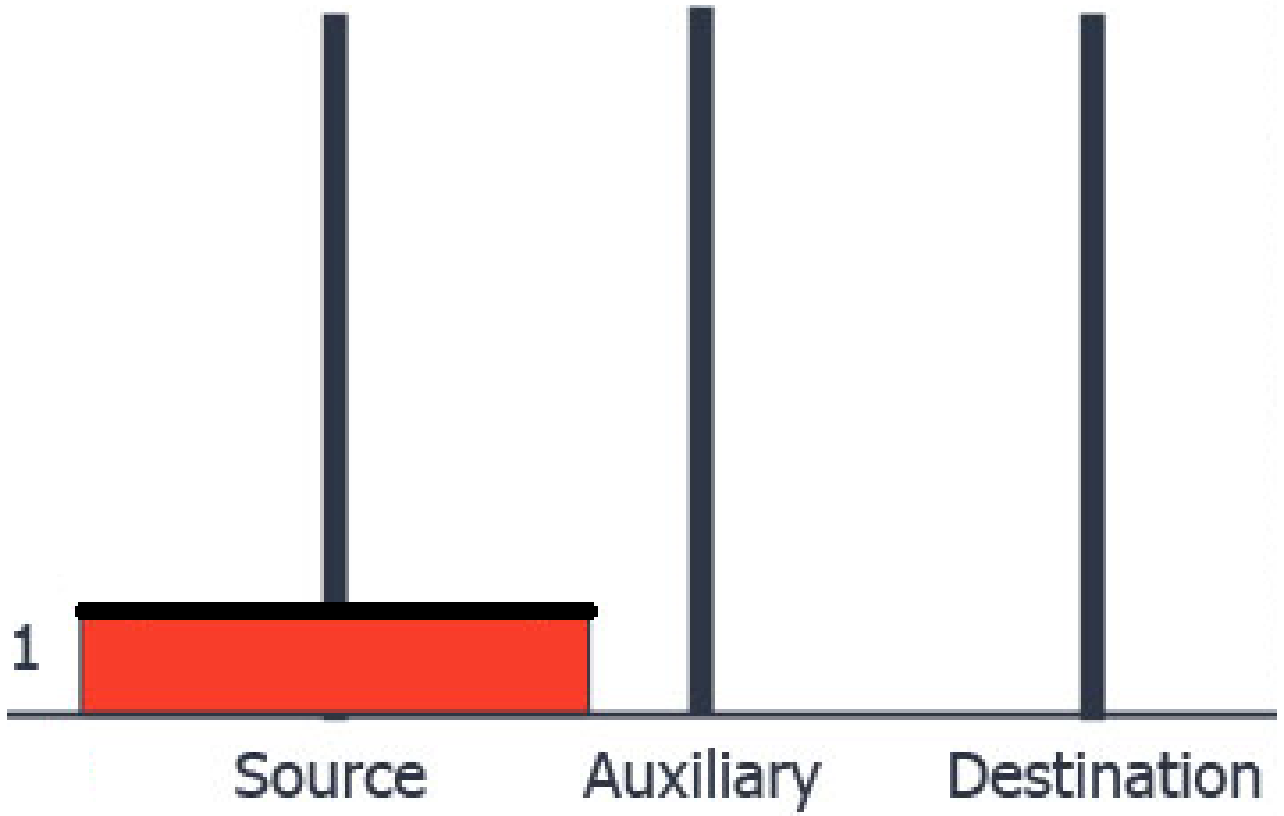
- There are three Pegs: Source, Destination and Mediator and there are n disks (of different sizes). All disks are initially at source (in the order of bigger to smaller).
- Our aim is to move all disks from Source to Destination using Mediator but never place larger disk on the top of smaller one.
- At a time only one move is allowed.



Logic-
1-Move n-1 disc source to Mediator using Destination.
2-Move nth disc from Source to Destination.
3-Move n-1 disc from Mediator to Destination using source

Solution

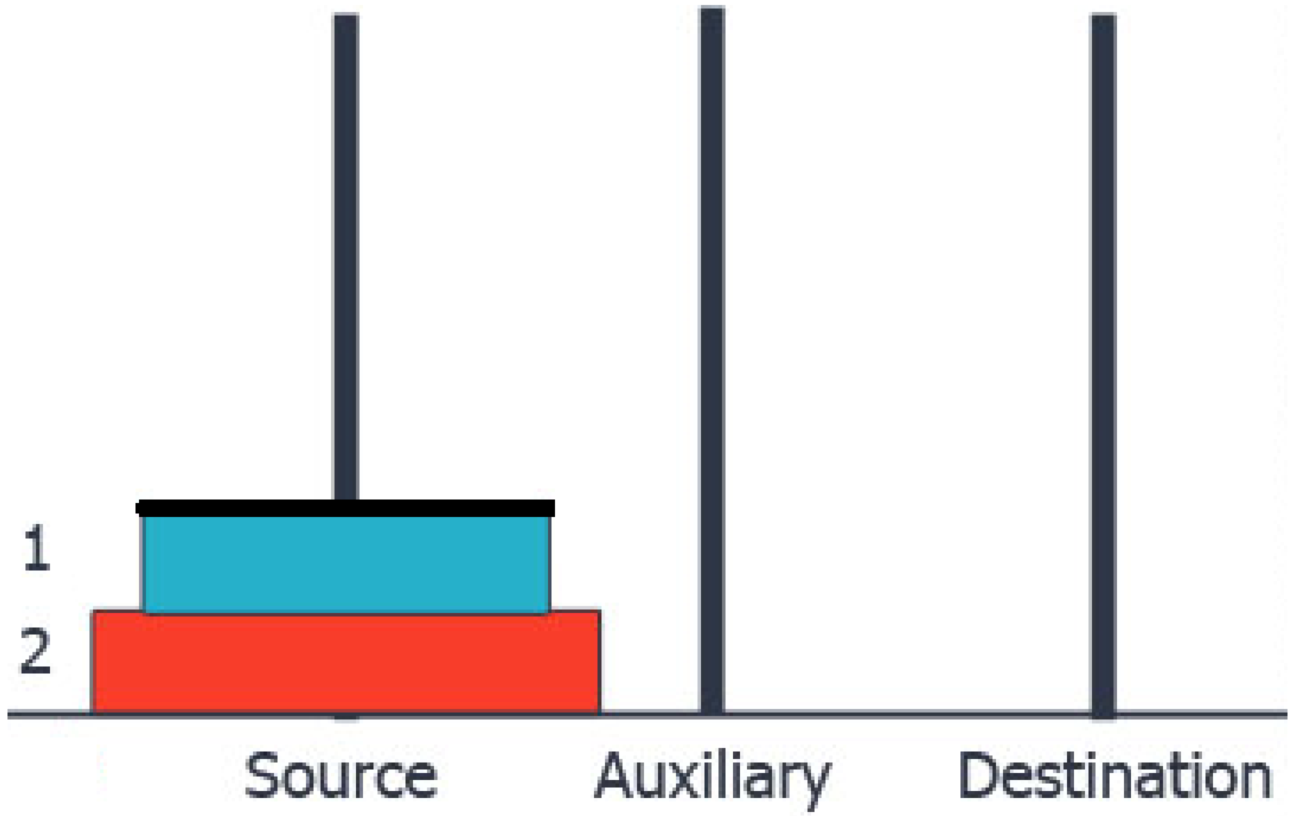
If there is only 1 disk at source Peg



Solution

If there are 2 disks at source Peg

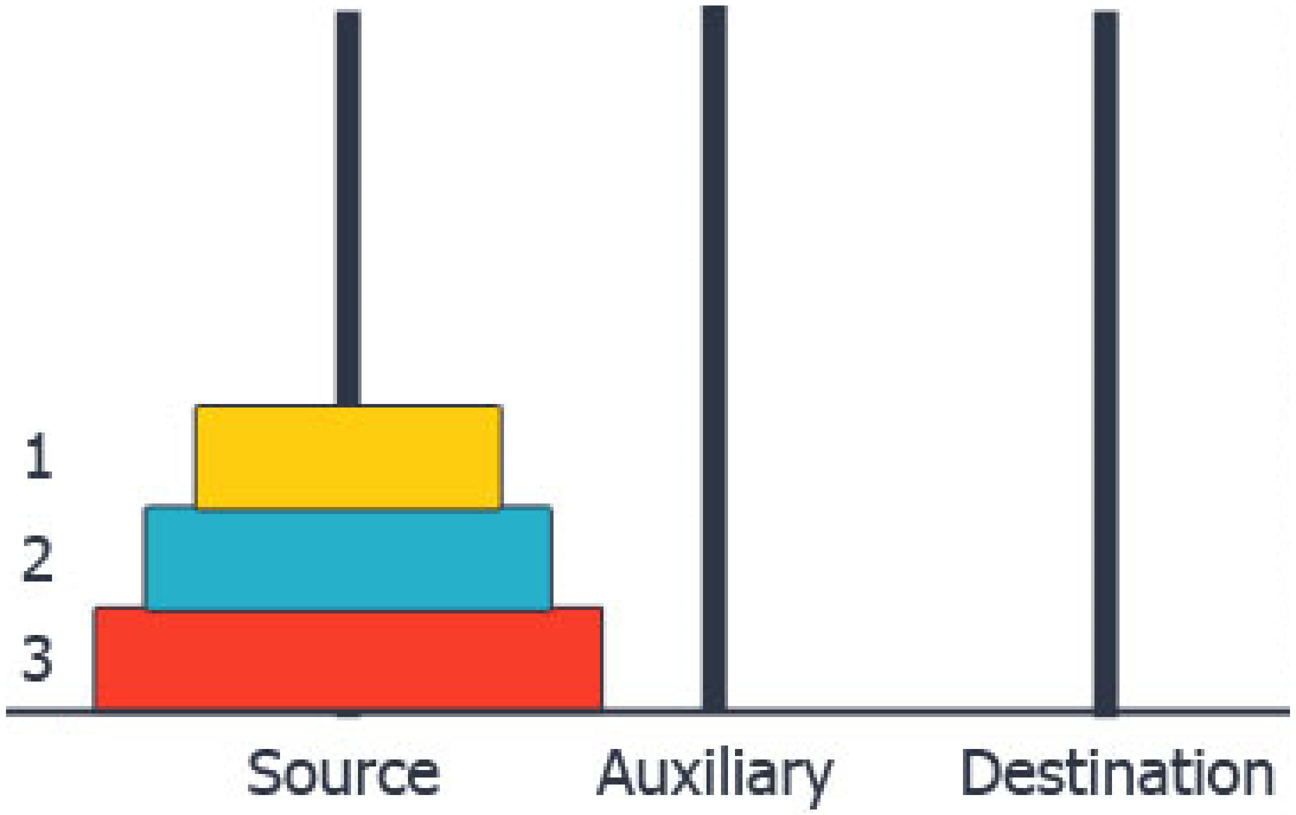
No of Moves:



Solution

If there are 3 disks at source Peg

No of Moves:



Solution

If there are N disks at source Peg **No of Moves:**

```
void toh(char S ,char M, char D, int n)
{
    if(n==1)
        printf("move %d disc from %d to %d",n,S,D);
    toh(S,D,M,n-1);
    printf("move %d disc from %d to %d",n,S,D);
    toh(M,S,D,n-1);
}
```

Solution

If there are N disks at source Peg **No of Moves:**

```
void toh(char S ,char M, char D, int n)
{
    if(n<=0)
        return;
    toh(S,D,M,n-1);
    printf("move %d disc from %d to %d",n,S,D);
    toh(M,S,D,n-1);
}
```

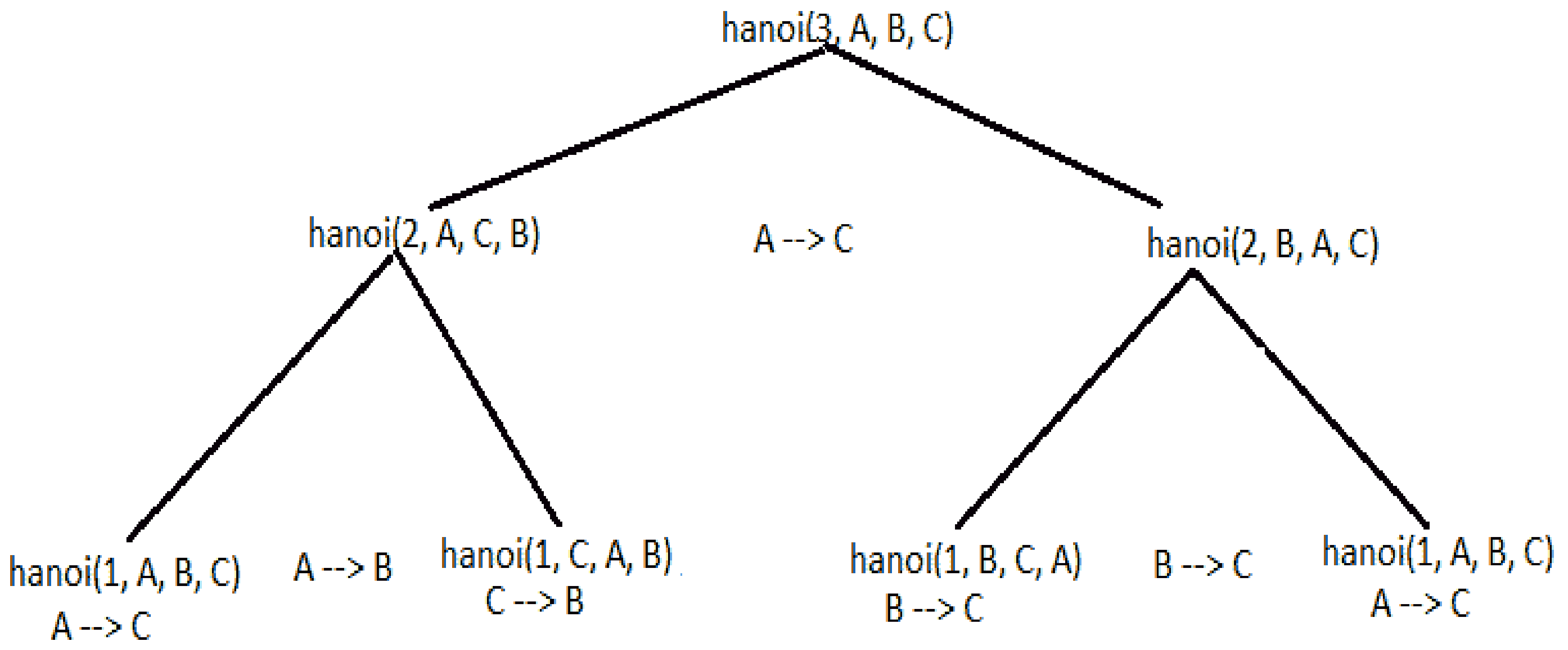
Recursion Tree

ToH(3,A,B,C)

List of Moves

Total Function Calls :
Total Push in Call Stack :
Total Pop in Call Stack :
Time Function :

Recursion Tree



Function Call stack

Total Function Calls : 7
Total Push in Call Stack : 7
Total Pop in Call Stack : 7
Time Function : 2^N-1
Space Function : N+1

Time Complexity: $O(2^N)$
Space Complexity O(N)

Data Structure Training(Recursion)

Tree Recursion

```
Fibonacci Series: 0 1 1 2 3 5 8 13 21 …
int fib(int n)
{
    if(n==1)
        return 0;
    if(n==2)
        return 1;
    else
        return fib(n-1)+fib(n-2);
}
```

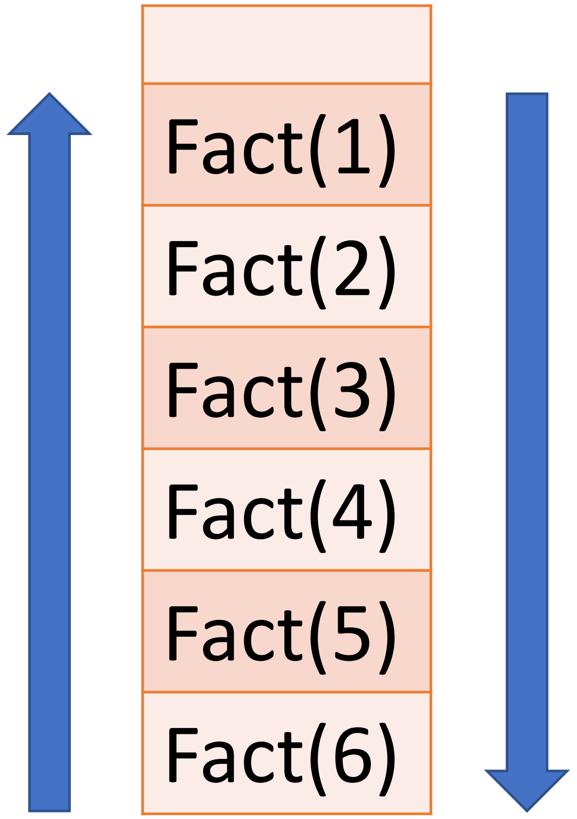
Data Structure Training(Recursion)

Explanation-

Data Structure Training(Recursion)

Complexity Of Recursive Function

```
int fact(int x)
{
        if(x==0 || x==1)
                return 1;
        else
                return x*fact(x-1);
}
```



Time Complexity= Number of function call = n = O(n)

Space complexity=number of maximum pending activation record = n=O(n)

Data Structure Training(Recursion)

Complexity Of Recursive Function cont…

Calculate time complexity of following function-

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

For n=6

total function call=

Data Structure Training(Recursion)

Complexity Of Recursive Function cont…

Calculate time complexity of following function-

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

For n=1 Total calls = 1

For n=2 Total calls= 1

For n=3 Total calls = 3

For n=4 Total function calls =5

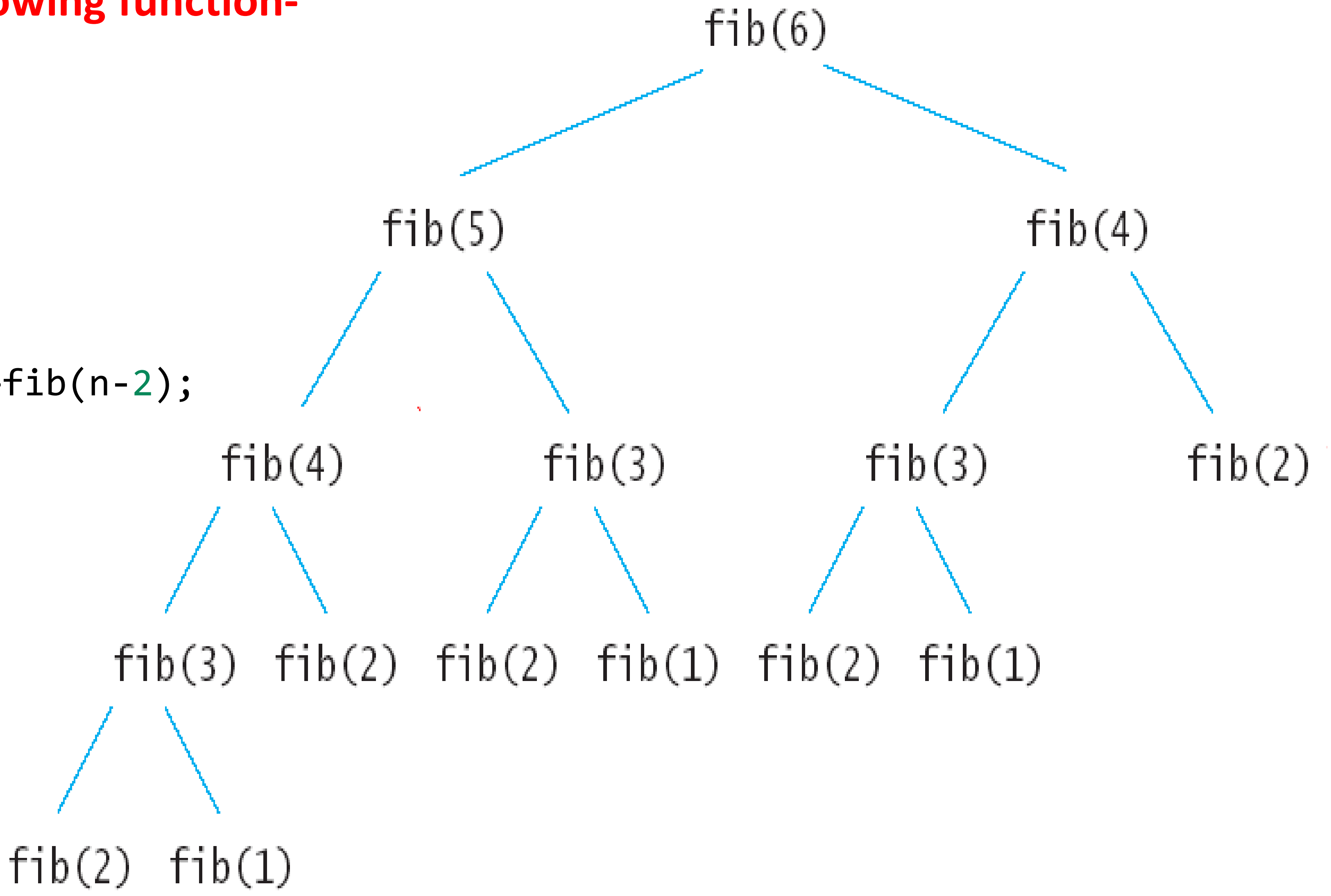
For n=5 total function call = 9

For n=6 total function call = 15

For n=7 total function call = 25

For n=8 Total function call = 41

$T(N) = T(N-1) + T(N-2) + 1 = O(2^N)$



Data Structure Training(Recursion)

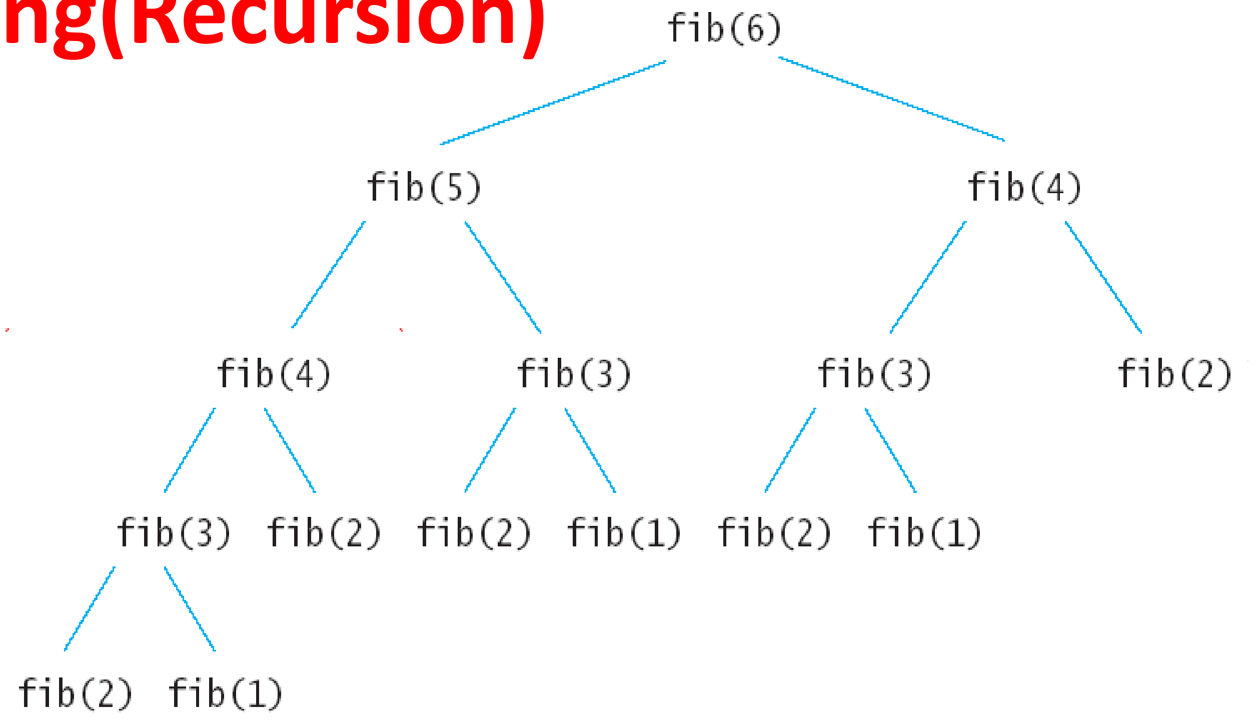
Time Function

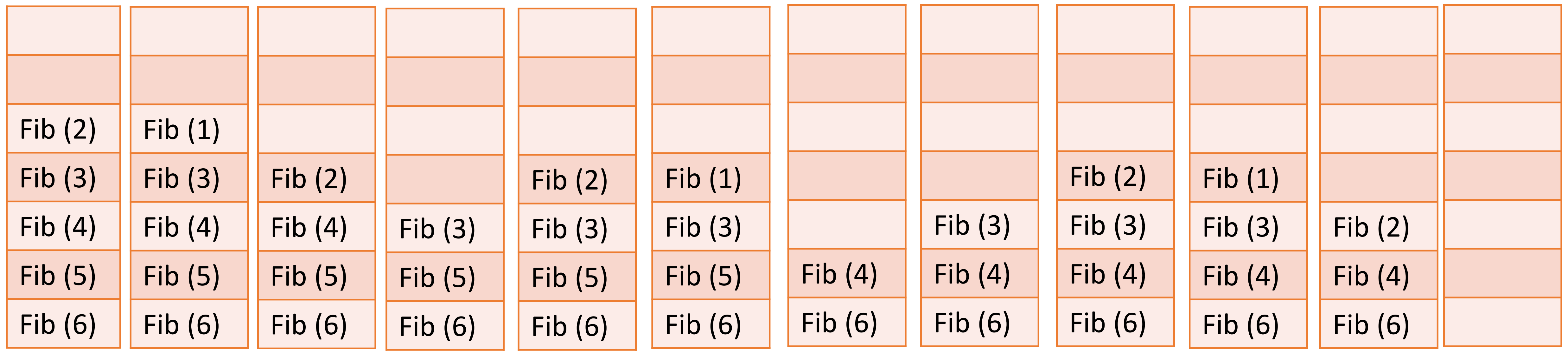
$T(N) = T(N-1) + T(N-2) + 1 = O(2^N)$

Space Function

Maximum Pending Activation records = N-1

Space Complexity = O(N)





Data Structure Training(Recursion)

what is the return value of fun(345,10)

```
int fun(int n, int r)
{
      if(n>0)
            return ((n%r)+fun(n/r,r));
      else
            return 0;
}
```

Data Structure Training(Recursion)

what is the return value of fun(345,10)

```
int fun(int n, int r)
{
      if(n>0)
            return ((n%r)+fun(n/r,r));
      else
            return 0;
}
```

```
Output- 12
```

Data Structure Training(Recursion)

what is the return value of fun(16384) and also calculate time complexity-

```
int fun(int n)
{
      if(n<=2)
            return 1;
      else
            return (fun(floor(sqrt(n)))+n);
}
```

Data Structure Training(Recursion)

what is the return value of fun(16384) and also calculate time complexity-

```
int fun(int n)
{
      if(n<=2)
            return 1;
      else
            return (fun(floor(sqrt(n)))+n);
}

Output-
Fun(16384)=16527
Time complexity=O(√n)
```

Data Structure Training(Recursion)

Nested Recursion-

When function calling itself with calling itself in parameter as well as.

```
A()
{
        A(A());
}
```

Ackermann function

$$a(m,n)=\begin{cases} n+1 & \text{if } m=0 \\ a(m-1,1) & \text{if } n=0 \\ a(m-1,a(m,n-1)) & m>0,\ n>0 \end{cases}$$

Calculate a(1,2)

Data Structure Training(Recursion)

Ackermann function

$$A(m,n)= \begin{cases} n+1 & \text{if } m=0 \\ a(m-1,1) & \text{if } n=0 \\ a(m-1,a(m,n-1)) & m>0,\ n>0 \end{cases}$$

Calculate A(1,2)

$$\begin{aligned} A(1,2) &= A(0,A(1,1)) \\ &= A(0,A(0,A(1,0))) \\ &= A(0,A(0,A(0,1))) \\ &= A(0,A(0,2)) \\ &= A(0,3) \\ &= 4. \end{aligned}$$

Data Structure Training(Recursion)

Ackermann function

$$A(m,n) = \begin{cases} n+1 & \text{if } m=0 \\ a(m-1,1) & \text{if } n=0 \\ a(m-1,a(m,n-1)) & m>0,\ n>0 \end{cases}$$

Calculate A(1,5)

Data Structure Training(Recursion)

Ackermann function

$$A(m,n)=\begin{cases} n+1 & \text{if } m=0 \\ a(m-1,1) & \text{if } n=0 \\ a(m-1,a(m,n-1)) & m>0,\ n>0 \end{cases}$$

Calculate A(1,5)

Answer=7

Data Structure Training(Recursion)

Indirect Recursion-

```
a(int n)
{
        if(n<=1)

                return;
        else
        {
                b(n-2);
                printf("%d",n);
                b(n-1);
        }
}
b(int n)
{
        if(n<=1)
                return;
        else{
                printf("%d",n-1);
                a(n-1);
                a(n-2);
                printf("%d",n);
           }
}
```

for n=4 what will be the output?

Time Complexity ?

Space Complexity ?

Data Structure Training(Recursion)

Indirect Recursion-

```
a(int n)
{
        if(n<=1)
                return;
        else
        {
                b(n-2);
                printf("%d",n);
                b(n-1);
        }
}
b(int n)
{
        if(n<=1)
                return;
        else
        {
                printf("%d",n-1);
                a(n-1);
                a(n-2);
                printf("%d",n);
        }
}
```

```
for n=4 output 1 2 4 2 2 3
Time complexity=O(2^n)
Stack Space O(n)
```

Data Structure Training(Recursion)

Permutation Problem-(Recursion with in loop)
Print all possible permutation of an array elements.

Write Your Code here

Data Structure Training(Recursion)

Permutation Problem-(Recursion with in loop)

Print all possible permutation of an array elements.

```
void perm(int *arr, int n, int l)
{
     if(l==n-1)
     {
          display(arr,n);
          return;
     }
     for(int i=l;i<n;i++)
     {
          swap(&arr[i],&arr[l]);
          perm(arr,n,l+1);
          swap(&arr[i],&arr[l]);
     }
}
```

Data Structure Training(Recursion)

Explanation-

Data Structure Training(Recursion)

Important Points

From today onwards try to solve every problem with the help of recursion because recursion is very powerful problem solving tool once u master in recursion then surely you all are capable to solve problem of backtracking and dynamic programming.

Try to write recursive formula for every problem. Without Recursive formula you can not solve problem of dynamic programming.

no recursive formula= no dynamic programming

“if you know what to do but don’t know how to do then create a function”
Believe me your 80% problem solved and after that try to convert this function into recursive function.

Data Structure Training(Recursion)

What will be the output of the following code ?

```
f(int x)
{
    if(x > 0)
     {
          f(--x);
          printf("%d", x);
          f(--x);
     }
}

int main()
{
        int a=2;
        f(a);
        return 0;
}
```

A 01

B 012

C 12

D 120

Data Structure Training(Recursion)

What will be the output of the following code ?

```
f(int x)
{
    if(x > 0)
     {
           f(--x);
           printf("%d", x);
           f(--x);
     }
}

int main()
{
        int a=2;
        f(a);
        return 0;
}
```

[✓ A 01]

B 012

C 12

D 120

Data Structure Training(Recursion)

What will be the output of the following code ?

```
int fun(int n, int k)
{
        if (n == 0)
                return 0;
        else if(n % 2)
                return fun(n/2, 2*k) + k;
        else
                return fun(n/2, 2*k) - k;
}

int main()
{
        printf("%d", fun(10, 1));
        return 0;
}
```

A 9

B 5

C 12

D Error

Data Structure Training(Recursion)

What will be the output of the following code ?

```c
int fun(int n, int k)
{
        if (n == 0)
                return 0;
        else if(n % 2)
                return fun(n/2, 2*k) + k;
        else
                return fun(n/2, 2*k) - k;
}

int main()
{
        printf("%d", fun(10, 1));
        return 0;
}
```

A 9

[✓ B 5]

C 12

D Error

Data Structure Training(Recursion)

What will be the output of the following code ?

```
#include <stdio.h>
int fun(int*,int);
int main(void)
{
        int a[]={12,7,13,4,11,6};
        printf("%d",fun(a,6));
        return 0;
}
int fun(int *a,int n)
{
        if(n<=0)
                return(0);
        else if(*a%2==0)
                return(*a+fun(a+1,n-1));
        else
                return(*a-fun(a+1,n-1));
}
```

A 20

B 15

C 12

D Error

Data Structure Training(Recursion)

What will be the output of the following code ?

```c
#include <stdio.h>
int fun(int*,int);
int main(void)
{
        int a[]={12,7,13,4,11,6};
        printf("%d",fun(a,6));
        return 0;
}
int fun(int *a,int n)
{
        if(n<=0)
                return(0);
        else if(*a%2==0)
                return(*a+fun(a+1,n-1));
        else
                return(*a-fun(a+1,n-1));
}
```

A 20

[✓ B 15]

C 12

D Error