1/03/2024

PAGE No.
DATE / /

Day 4 of DSA.

* Tasks!-

Check Box	
	Recursion.
☑	Understanding Recursion :- Basic Recursive Function (Factorial, Fibonacci)
☑	Recursive Backtracking : simple problems like generating subsets, permutations
☑	Implement recursive algorithms : Factorial, Fibonacci
☑	Solve problems involve recursive backtracking.

Theory Part

PAGE No.
DATE 1 1

* Recursion:-

It is a technique in which a function calls itself repeatedly until a given condition is satisfied.

Syntax:

return_type recursive_func()
{
// Base Condit°
// Recursive Case
}

* Recursive function:-

A function that calls itself is called a recursive function.

- Base Condit°:- For terminating the recursion
- Recursive Case:- It contain multiple recursive calls

* eg

```
int sum (int n)
{   if (n==0) {
        return 0;
    }
    int res = n + nsum(n-1);
    return res;
}
```

Dry Run

PAGE No.
DATE 1 1

Sum (3)

res = 3 + (3-1) ← 3+3 = 6

↓

Sum(2)

res = 2 + (2-1) ← 2+1 = 3

↓

Sum(1)

res = 1 + (1-1) ← 1+0 = 1

↓

Sum(0)

n == 0

return 0

Types of Recursion

- Direct Recursion
  - Head Recursion.
  - Tail Recursion.
  - Tree Recursion
- Indirect recursion

Page No.
Date 11

* Basic Examples :-

(1) Fibonacci series :-

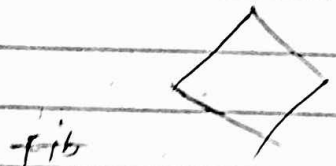
```
int Fib(int n){
if( n==0)  return 0;
if( n==1 || n==2)  return 1
else  return =( Fib(n-1) + F(n-2));
```

Fib. Series = 0 1 1 2 3 5 8 13
0 1 2 3

Fib(3)

Fib(3-1) + Fib(3-2)

Fib(2) + Fi(1)



return 1 =

return 1 + 1 = (2)

PAGE No.
DATE / /

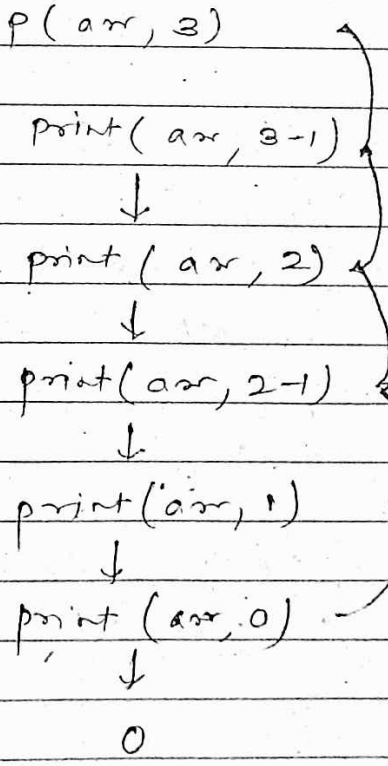
(2) Print array in reverse order using Recursion:-

```
void print (int * arr, int n)
{
    if (n == 0) return;

    print (arr, n-1);
    cout << arr[n-1] << " ";
}

arr[3] = {1, 2, 3};
```

1) p(arr, 3)



PAGE No.
DATE 1 1

(1) Program OF printing N to 1 Numbers in reverse order

Approach:-

Base condition = if (n <= 0) return;
else cout << n;

Recursive call = printReverse(n-1);

Time complexity : O(N)
Space complexity : O(N)

(2) Print 1 to N Numbers.

Approach:-

Base condit^n:- ⟶ if (n > 0)
{

Recursive call ⟶ print(n-1);
cout << n << " ";
}

Time complexity : O(N)
Space complexity : O(N)

PAGE No.
DATE / /

* Tail Recursive Funct" :-

It is defined as a recursive function in which the recursive call is the last statement that is executed by the function.

- Nothing is left to execute after the recursion call

Eg.

```
void print (int n) {
    if (n<0) return;
    cout << " " << n;
    print (n-1);
}
```

Time Complexity : O(N)
Space complexity : O(N)

Time Complexity : O(logn)
Space complexity : O(logn)

PAGE No.
DATE / /

* Check if a number is palindrome:-

1) We make 3 Functi

2) int oneDigit(int num)
{
return (num >=0 && num<10);
}
checky is number is present bet" 0 to 10

3) Recursive Funct"
bool isPalUtil(int num, int * dupnum)
{
if(oneDigit(num))
return (num == (*dupNum) % 10);
Comparing 1st & last digit.

if (!isPalUtil(num/10, dupNum))
return false;
every time we decrease num/10 &
check 1st & last element of
num & dupNum respectively

* dupNum /= 10;

return (num % 10 == (*dupNum) % 10);
}

3) isPal(int num)
{ if(num<0) { num = -1; }
int * dupNum = new int(num);
return isPalUtil(num, dupNum);
}

Time Complexity : O(log N)
Space complexity : O(log N)

PAGE No.
DATE / /

* Sum of the digits of a given number :-

Input = 666
Output = 18

Approach :-

Base Case : if (no == 0) return 0;

Recursive call : return (no % 10) + sumdigits (no/10);

Dry Run.

666

① no = 666

(666 % 10) + sum (666/10)
6 + (66)

6 + 12 = 18

② no = 66

(66 % 10) + S (66/10)
6 + 6

6 + 6 = 12

③ no = 6

(6 % 10) + S (6/10)
6 + 0.6

6 + 0 = 6

④ no = 0

n == 0 return 0

PAGE No.
DATE 11

* Printing all subsets from a set.

Approach:-

Subset (string str, int index = -1, string curr = "",

Base case = if(index == n) return;

— cout << curr << "\n";

Recursive call - For(int i = index + 1; i < n; i++)

{

curr += str[i];

subset (str, i, curr);

curr.erase (curr.size() - 1);

}

return;

Time complexity : $O(2^n)$
space complexity : $O(n)$

PAGE No.
DATE / /

Tower of Hanoi :-

Approach :-

```
toh( int n, char from_rod, char to_rod,
                    char aux_rod)
{
```

Base case - if (n == 0) return;

Recursive calls - toh(n-1, from_rod, aux_rod, to_rod);

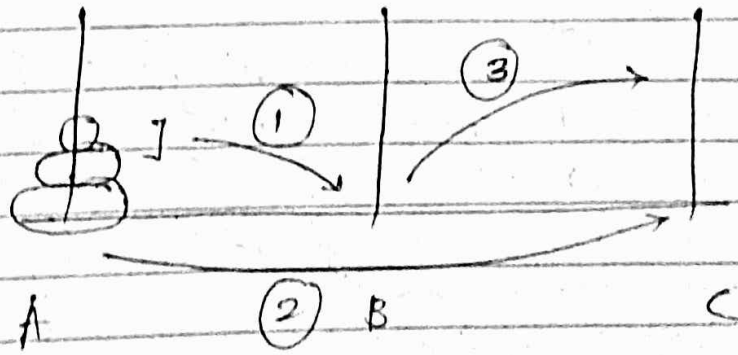
print statements -

cout << "Move disk " << n << " from rod " << from_rod << " to rod " << to_rod << endl;

toh(n-1, aux_rod, to_rod, from_rod);

}

Time complexity : O(2^n)
Space complexity : O(N)



PAGE No.
DATE 11

* Josephus problem:-

Base case:- if (n == 1) return 1;

Recursive call: return

j(n-1, k) + k-1) % n+1;

n = 5 , k = 2

① (j(4, 2) + 1) % 6 (4+1) % 6 = 5

↓ n = 4, k = 2

j(3, 2) + 1) % 5; (3+1) % 5 = 4

↓ n = 3; k = 2

j(2, 2) + 1) % 4; (2+1) % 4 = 3

↓ n = 2, k = 2

j(1, 2) + 1) % 3; (1+1) % 3 = 2.

↓

n == 1

1

Time complexity: O(N)
Space complexity: O(N)

1 2 3 4 5