Date :
Page No. :

⇒

Space Complexity —

Order of growth of memory (or RAM) space in terms of input size.

→
```
int getSum(int n)
{
    return n*(n+1)/2
}
```
[O(1)] ✓
Or
[θ(1)] ✓

→
```
int getSum2(int n)
{
    int sum = 0;
    for(int i=1; i<=n; i++)
        sum = sum+i;
    return sum;
}
```
→ [O(1)] ✓ as we need only 3 variables i.e → n, sum & i

→
```
int arrSum(int arr[], int n)
{
    int sum = 0;
    for(int i=0; i<n; i++)
        sum = sum + arr[i];
    return sum;
}
```

O(n) as the space is growing linearly
[n of arr[]] ✓

⇒ Auxiliary Space :- Order of growth of extra space or temporary space in terms of input size.

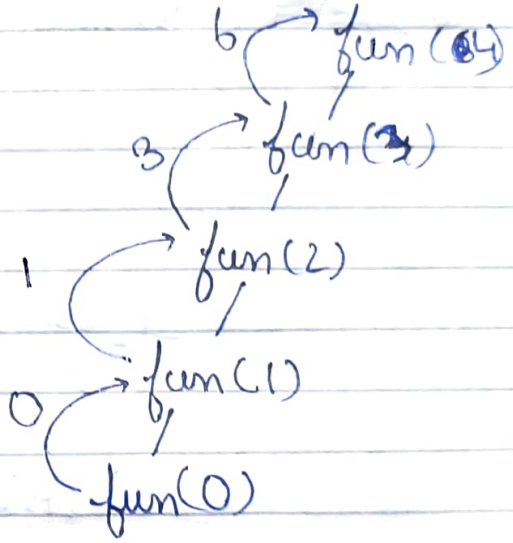
Date :
Page No. :

→
```
int arrSum (int arr[]; int n)
{
    int sum = 0;
    for (int i = 0; i < n; i++)
        sum = sum + arr[i];
    return sum;
}
```

Space complexity = O(n)
Auxiliary space = O(1) as no other extra space is used.

→
```
int fun (int n)
{
    if (n <= 0)
        return 0;

    return n + fun(n-1);
}
```



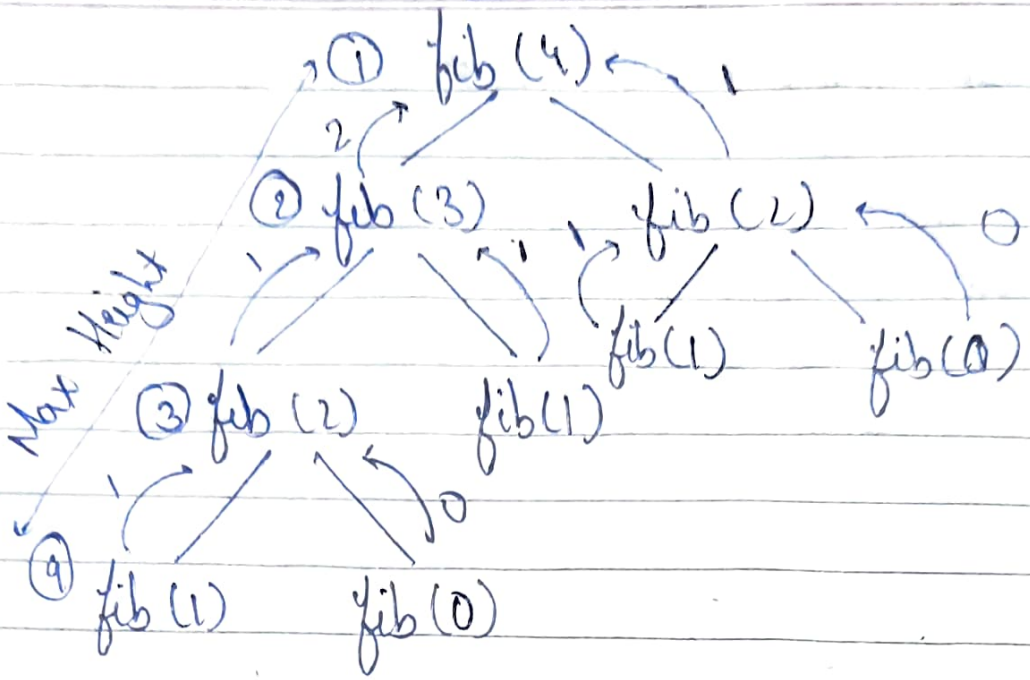
∴ [O(n)] ✓

→
```
int fib (int n)
{
    if (n == 0 || n == 1)
        return n;

    return fib(n-1) + fib(n-2);
}
```

0, 1, 1, 2, 3, 5 ...
n=0 n=1 n=2 →
Fibonacci No's

3



→ ~~Exact~~ formula

(*) Auxiliary space → Max length of root to leaf path.

∴ It is growing linearly
[O(n)] ≈