Your Ultimate Guide To Landing Top AI roles

2.3.7

Space Complexity of Iterative & Recursive Programs

DECODE AiML

→ To Solve Any problem, there can be multiple algorithms to Solve it.

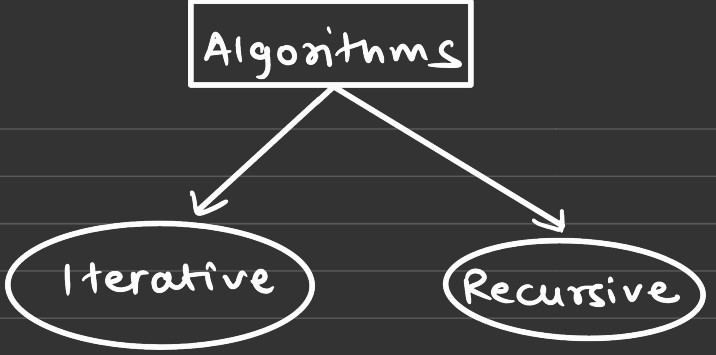
→ we need to find the best algorithm out of the K algorithm

→ To analyze the algorithm, we need to check Time and Space Complexity.

→ The algorithm which takes less time and less memory Compared to others will be Considered as better Algorithm.

→ After selecting the best algorithm, we write programs and run it on the Computer.

→ Let's Analyze the space Complexity of an algorithm.



→ When we talk about Space Complexity, we generally mean:

↳ The total memory used by the algorithm relative to n (size of input)

→ This includes

① Input Space (Memory to hold input data itself)

② Auxiliary Space (Temporary variables, Recursion stack etc)

However, in Practical Analysis, we often focus on Auxiliary Space. (excluding input space)

Space Complexity of Iterative Algorithms

Example 1:

```
def sum_n(n):
    total = 0
    for i in range(1, n+1):
        total += i
    return total
```

Input: n → O(1)
Auxiliary space: total, i → O(1)

Total space = O(1)
Auxiliary space = O(1)

Space Complexity = O(1)

Example 2:

```
def sum_list(data):
    total = 0
    for val in data:
        total += val
    return total
```

↑ list of n size

Input: data → O(n)
Auxiliary space: total, val → O(1)

Total space = O(n)
Auxiliary space = O(1)

Space Complexity = O(1)

Example 3 :

```
def sum_n(data):
    m = len(data)
    n = len(data[0])
    total = 0
    for i in range(m):
        for j in range(n):
            total += data[i][j]
    return total
```

Input: data → $O(n^2)$
Auxiliary space: total, m, n, i, j → $O(1)$

Total space = $O(n^2)$
Auxiliary space = $O(1)$

Space Complexity = $O(1)$

Space Complexity of Recursive Algorithms

Example 1:

```
def fact(n):
    if n==0:
        return 1
    return n * fact(n-1)
```

In Recursive program, we need to analyze depth of Recursion stack to analyze space Complexity.

↳ stack frame allocation

2·3·5

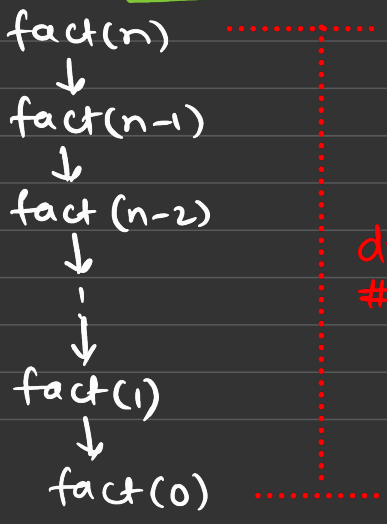
Input: n → O(1)

Auxiliary space: recursion call stack → O(n)

Total space = O(n)

Auxiliary space = O(n)

Space Complexity = O(n)

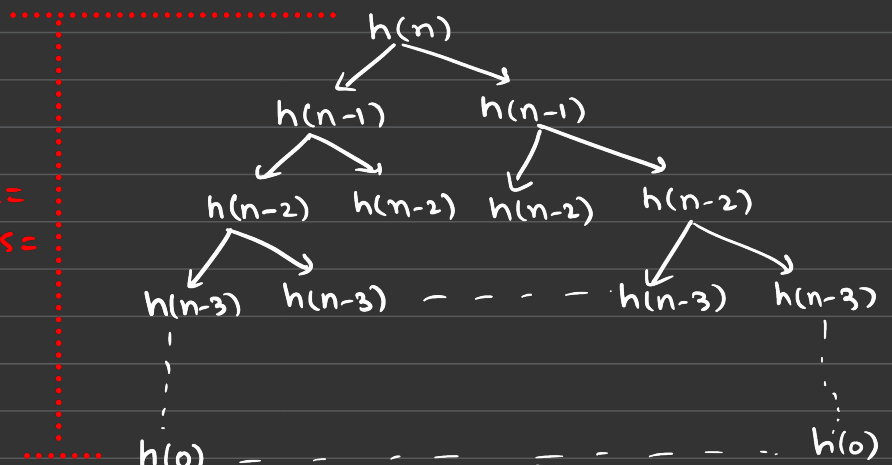


Each recursive function call takes memory in the stack segment of Main Memory (RAM)

Example 2:

```
def helper(n):
    if n>=1:
        helper(n-1)
        print(n)
        helper(n-1)
```

→ Let's find the Recursion depth.



Input: n → O(1)

Auxiliary space: recursion call stack → O(n) (depth)

Total space = O(n)

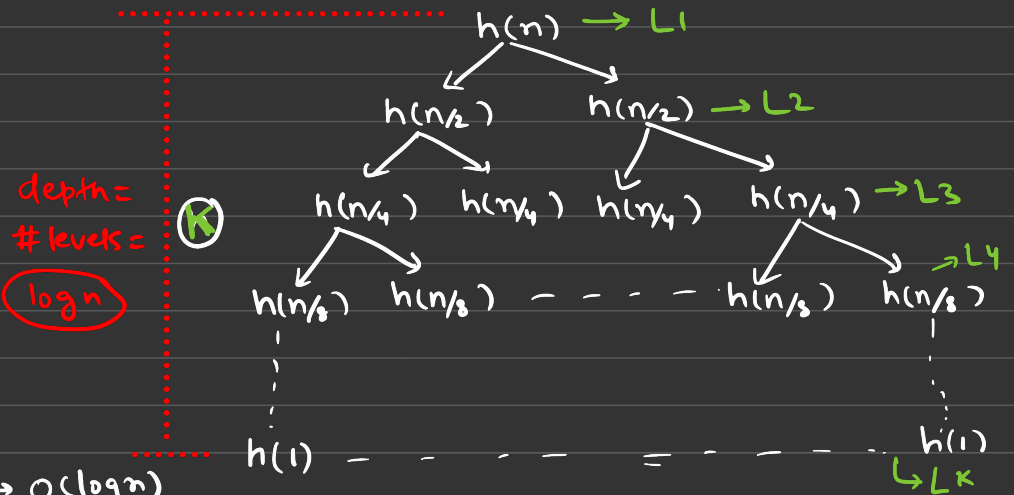
Auxiliary space = O(n)

Space Complexity = O(n)

Example 2:

```
def helper(n):
    if n>1:
        helper(n/2)
        print(n)
        helper(n/2)
```

→ Let's find the Recursion depth.



depth = # levels = K = $\log n$

Input: $n \rightarrow O(1)$

Auxiliary space: recursion call stack (depth) $\rightarrow O(\log n)$

Total space = $O(\log n)$

Auxiliary space = $O(\log n)$

Space Complexity = $O(\log n)$

→ $n/2^0, n/2^1, n/2^2, n/2^3, \ldots n/2^{k-1}$

$\frac{n}{2^{k-1}} = 1 \Rightarrow 2^{k-1} = n$

$\Rightarrow \boxed{k = 1 + \log n}$


Like
Subscribe