Fibbonacci using recursive :

$F(0) = 0$ & $F(1) = 1$

$F(n) = F(n-1) + F(n-2)$

Recursive relation : $T(n) = T(n-1) + T(n-2) + O(1)$

Approximate : $T(n-1) \approx T(n-2)$

$$T(n) = 2T(n-2) + K$$
$$= 2\{2T(n-4) + K\} + K$$
$$= \cancel{4}\{2T(n-6) + K\} + 3K$$
$$= 8T(n-6) + 7K$$
$$= 2^3 T(n - 2\times 3) + 7K$$
$$= 2^{K_1} T(n - 2K_1) + \cancel{7}K(2^K - 1)$$

when, $n - 2K_1 = 0$

$$K_1 = \frac{n}{2}$$

$$T(n) = 2^{n/2} T(0) + K(2^{n/2} - 1)$$
$$= 2^{n/2}[T(0) + K] - 1$$

Lower bound : $\Omega(2^{n/2})$

if $T(n) = 2T(n-1) + K$
$$= 2^{K_1} T(n - K_1) + K(2^{K_1} - 1)$$

$n - K_1 = 0 \Rightarrow K_1 = n$

$$T(n) = 2^n T(0) + (2^n - 1)K$$

Running time & time complexity are two different things.

Running time : $T(n) = 3n^2 + 4n + 2$

Time complexity : $O(n^2)$

→ We calculate unit operations to find out Big-O & ignore constants.

→ For recursive functions, find out the ~~recursive~~ recurrence relation

Ex: Factorial of n

$$T(n) = T(n-1) + K$$
$$T(n-1) = T(n-2) + K$$
$$T(n-2) = T(n-3) + K$$

$$T(n) = T(n-3) + K + K + K + \ldots$$
$$T(n) = T(0) + n * K$$
$$T(n) = K_2 + nK$$
$$T(n) = nK \Rightarrow O(n)$$

Binary Search :

Recurrence relation: $T(n) = T(n/2) + K$

$$T(n/2) = T(n/4) + K$$

$$T(n) = K_2 + (\log n) K$$

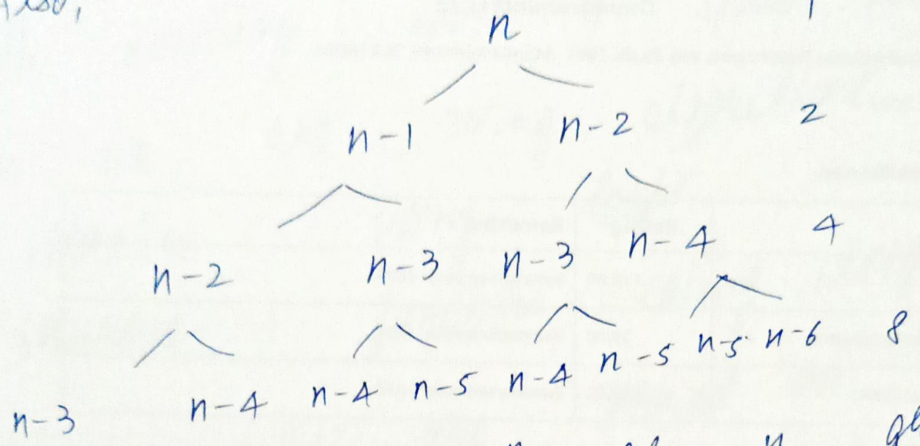
$\Omega(1)$, $O(\log n) \Rightarrow O(\log n)$

$$T(n) = 2^n (T(0) + K) - K$$

$\Rightarrow O(2^n)$ Exponential

If fibbonacci is iterative, then it is linear $O(n)$.

Also,



$= 2^n$ as n grows.

Space complexity:

Auxiliary space is the extra space used by the algorithm apart from the input space.

Recursive ⇩	Space Complexity	Auxiliary Space
Linear Search	$O(n)$	$O(1)$
Binary Search	~~O(Klogn)~~ $O(\log n)$	$O(1)$

Space complexity is the maximum space required at any point in time.

~~Dont~~ count the input space required in space complexity.

Hence space complexity includes & is the total space taken by the algorithm with respect to the input size. It includes both auxiliary space & space used by input.

when we are using algorithms like

iterative : using loops

recursive : function calling itself

Recursive calls are stored on stack, hence depends on the auxiliary space that determines the space complexity hence we can neglect the input space in case of recursion.

By neglecting, for iterative linear/binary search space complexity is $O(1)$.

For recursive fibonacci sequence:

$\vdots$

$F(n-1)$

$F(n)$

main()

Space complexity is $O(n)$

Iterative binary & linear search has a space complexity of $O(1)$.

<u>Examples</u>:

1. Array intersection

I $\Rightarrow$ $O(m * n)$ for 2 unsorted arrays

II if we sort each array first & then use merge sort to combine them : then $\Rightarrow$ $\underbrace{n\log n + m\log m}_{\text{sort}} + \underbrace{m+n}_{\text{merge}}$

$$\Rightarrow O(n\log n + m\log m)$$

III if $m < n$, sort array m

$$m\log m + \underbrace{n\log n}_{\text{merge}}$$

$$\Rightarrow O(m\log m + n\log n)$$

2. Power solution:

$$x^n = x^{n/2} * x^{n/2}$$

```
if n == 0:
    return 1

if n % 2 == 0
    return p(x, n/2) * p(x, n/2)
return x * p(x, n/2) * p(x, n/2)
```