Complexity Analysis

Complexity : A factor involved in a complicated process or situation.

For a code/ solution to a problem, we want to figure out how good or bad is it?

Time Complexity

→ What is the time complexity of your code?

→ can you improve it?

$A_{algo} \longrightarrow T(n)$

n → input size

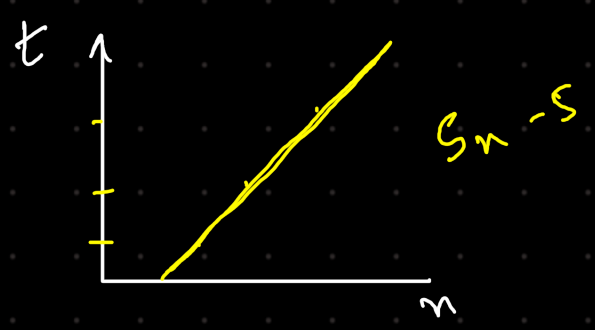
↓ how much time complexity

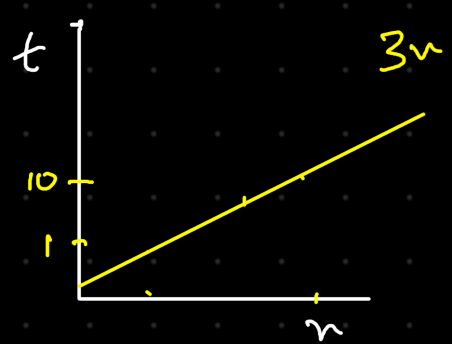
$f(n) = n^2$

$= 3n + 50$

n	HP (2016)	M1
1	1	1
100	5	2
1000	10	3
10000	15	4
100000	25	5

Time taken ≠ Time Complexity





Experimental Analysis

Not a correct way :-

1.] diff time on different m/c and diff run

2] diff implementation, time different

3] time vary for diff O/P

4] Generate test case

5] diff code for each

6] Large input very time consuming

Big Oh notation

1. want to evaluate when input is very big
2. want to calculate for worst case.
3. want to look for largest factor in run time.

$5n^2 + 10n + 2$

10^{12} 10^6

10^{18} 10^9

4. want to express run time as input size and we dont want to look for precise, rather 'order of' works.

$5n^2 + 10n + 2$

$10n^2 + 2n + 5$

Bubble sort : $2n^2 + 15n + 1$

for a large n , $n^2 >>> n$

so only n^2 will remain

We are fine with most dominating term : $2n^2$

⇒ no. of unit operation

time taken = order of input

1] Talk in terms of unit operation not time

2] highest power is our focus

3] don't care about coeff

$A_{f(n)}^{algo} \longrightarrow O(g(n))$

if time taken by A for input(n)

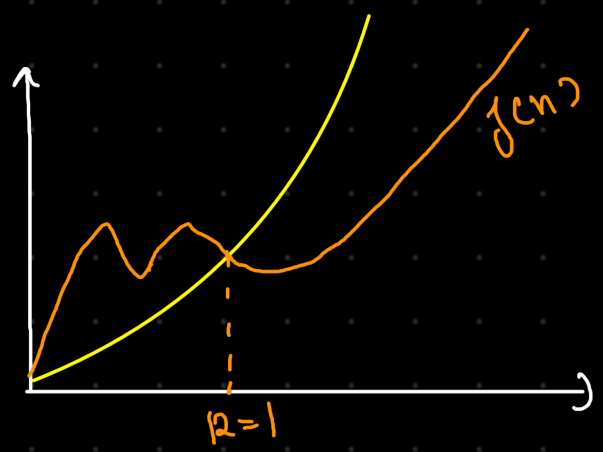
$\leq c * g(n)$

$f(n) = O(g(n))$

$f(n) = 2n^2 + n \leq c * g(n)$

$2n^2 + n \leq 3n^2$ → $g(n) = n^2$

$O(n^2)$



$n = 1$ $\quad c \cdot g(n) \geq f(n)$

Big Omega (Ω)

→ Best case

→ At least

$$f(n) = \Omega(g(n))$$

$$f(n) \geq c \cdot g(n)$$

↓ greatest lower bound

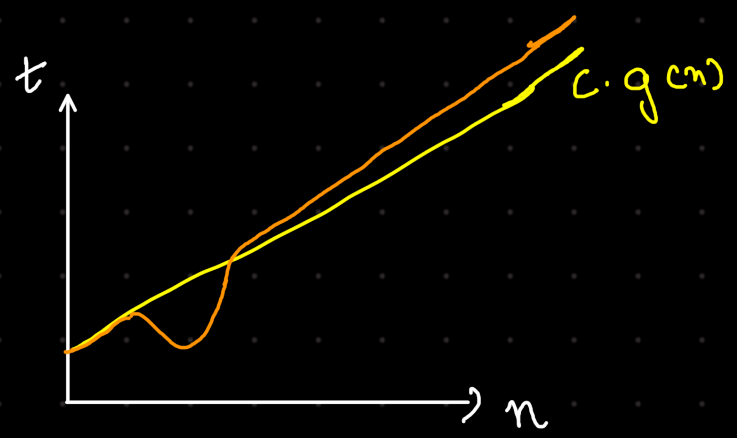
$$2n^0 + n \geq c\, n^2$$

$\Downarrow$

$0, -1, 1$

$$2n^2 + n \geq 2n^2$$

$$n \geq 0$$



Theta (Θ)

$$c_1 g(n) \leq f(n) \leq c_2 g(n)$$

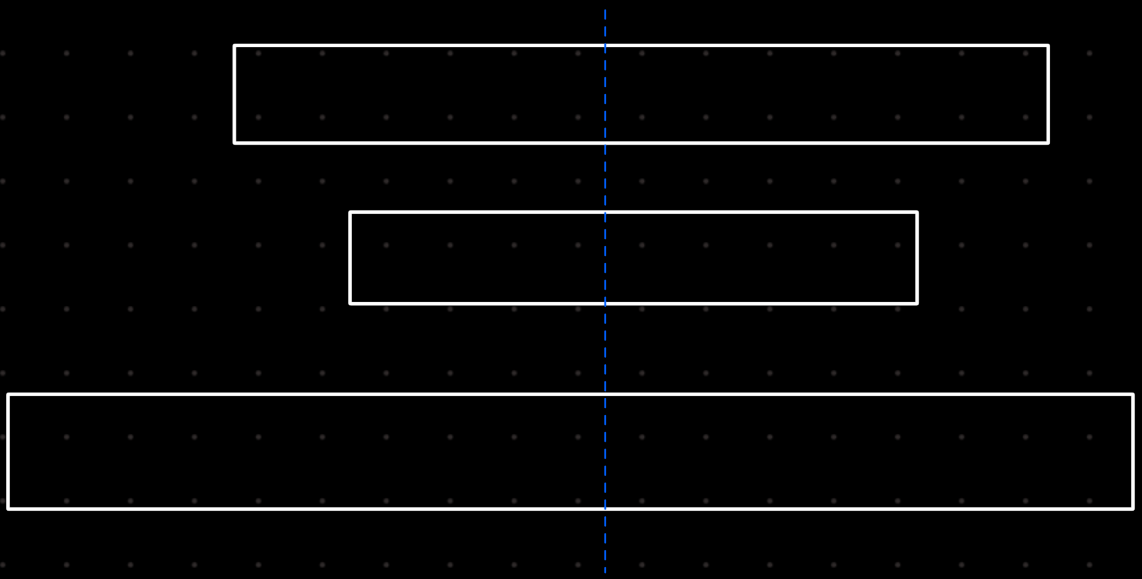
$$2n^2 \leq 2n^2 + n \leq 3n^2$$

little Θ no equal to

little Ω no equal to

Time taken $\neq$ Time Complexity

Middle element of an array



$f(n) = 5 \quad \leq c \cdot O(n^0)$

$O(n^0) = O(1)$

Largest in an array of size (n)

max = a[0]

for i → n

} k operations

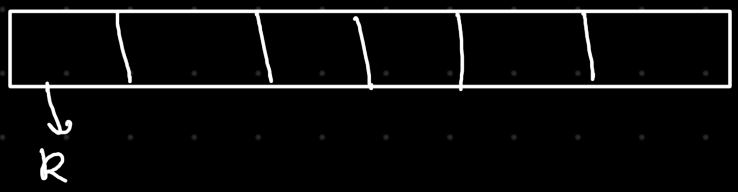
$T(n) = k + n + n + k \quad \ldots n \text{ times}$

$= kn \qquad < c * g(n)$

$(n+1)\ (n) \rightarrow$ upper bound

$O(n)$

Bubble sort



1st element → R(n-1)

2nd element → R(n-2)

3rd → R(n-3)

⋮

gud least → R

$1+2+3 \cdots n = \frac{(n)(n+1)}{2}$

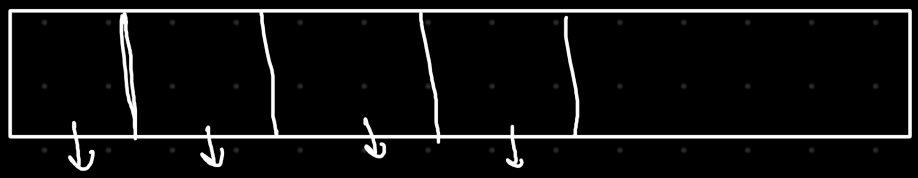
$R(n-1) + R(n-2) + R(n-3) \cdots \cdots R \cdot 1$

$$\frac{(n-1)(n)}{2} = \frac{n^2}{2} - \frac{n}{2}$$

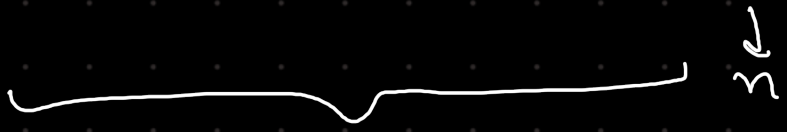
$$\frac{n^2}{2} \leq c \cdot g(n^2)$$

$O(n^2)$

Insertion Sort



1 1
2 1 2
3 1 2 3
4

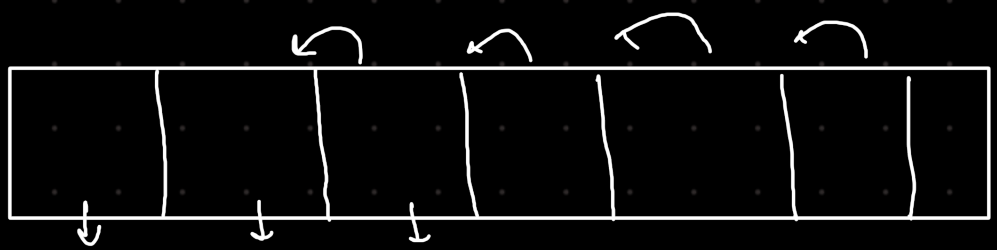


$n-1$

$1+2+3 \ldots (n-1)$ $\quad \frac{n(n-1)}{2}$

$\frac{n^2}{2} - \frac{n}{2}$

$O(n^2)$

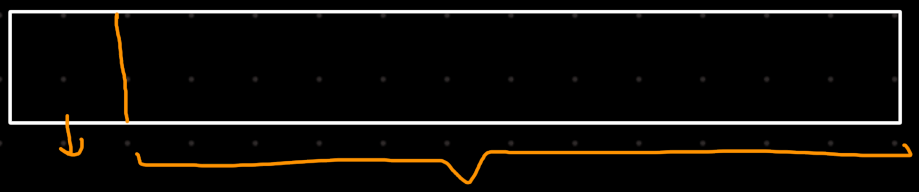


k 1
k 1
k 1
1
1

$O(n)$

$kn \quad \leq (k+1)n$

Selection Sort



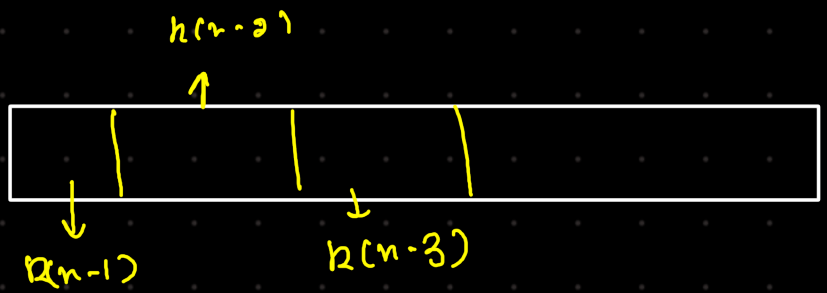
$R(n)$
$R(n-1)$
$R(n-2)$
$\vdots$
1

$R(n + n-1 + n-2 \cdots 1)$

$$\frac{Rn^2}{2} + \frac{nR}{2} \leq \frac{(R+1)n^2}{2}$$

$O(n^2)$

Best case



$$\sum_{i=1}^{n} R(n-i) = R\frac{n(n-1)}{2}$$

$$= \frac{Rn^2}{2} - \frac{Rn}{2}$$

$$= O(n^2)$$

Time Complexity - Recursive

For recursion, we follow a different approach.

We have many methods.

1. Recurrence relation method. ⇐
2. Master theorem
3. Akra Bazzi theorem
4. Iterative approach

Factorial of a number

```
def fact(n):
    if (n <= 1):
        return 1          } R1
    return fact(n-1) * n
                        R2
```

$$T(n) = \overbrace{R_1 + R_2}^{R} + T(n-1)$$

$$\boxed{T(n) = R + T(n-1)}$$

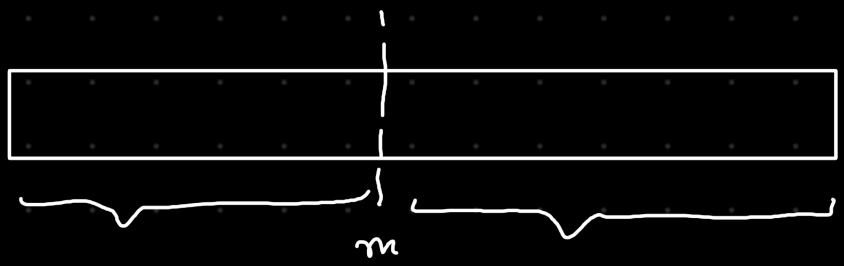
$$T(n) = R + T(n-1)$$
$$T(n-1) = R + T(n-2)$$
$$T(n-2) = R + T(n-3)$$
$$\vdots$$
$$T(1) = R$$

$$T(n) = R + R + R + R \quad \ldots \quad n \text{ times}$$

$$T(n) = \boxed{kn} \qquad \leq (n)(n+1)$$

$$O(n)$$

Binary Search



finding middle

$$T(n) = k + T(n/2)$$

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

$$T(n/4) = k + T(n/8)$$

$$\vdots$$

$$T(1) = k$$

$$T(n) = k + k + k + k \ldots \quad x \text{ times}$$

10 → 5 → 2 → 1 → 1/4

100 → 50 → 25 → 12 → 6 → 3 → 2 → 1 → 1/8

1000 → ... → 1/2

input = operation +

$n \quad \frac{n}{2} \frac{n}{4} \frac{n}{8} \quad \ldots \quad 1 \quad ?$

x times

$$\frac{n}{2^x} = 1$$

$$n = 2^x$$

$$\log_2 n = x \log_2 2$$

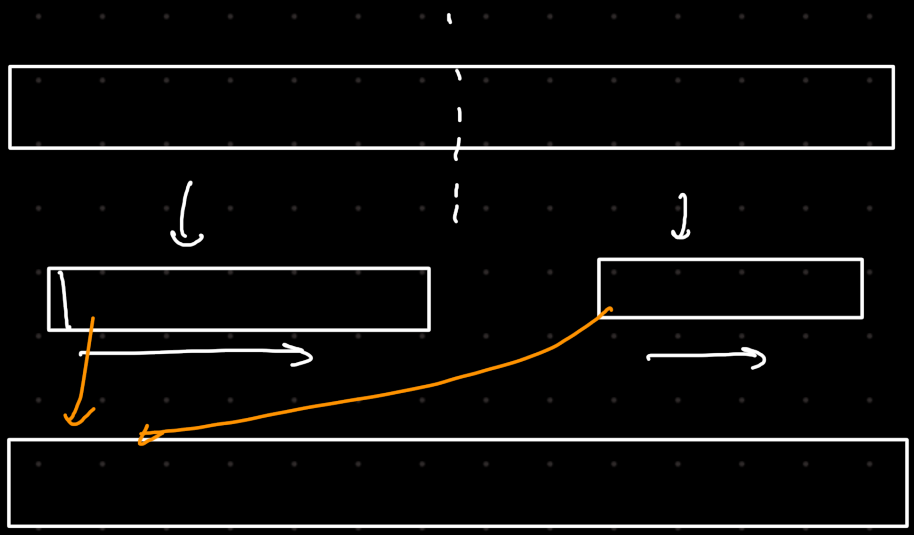
$$\log_2 n = x$$

$$\log n = x$$

$$T(n) = \log_2 n$$

10^5

Merge sort



$$T(n) = R + 2T(n/2) + R_2 n$$

$$= R + \underline{2T(n/2)} + R_2 n$$

$$T(n) = \cancel{R}^{0} + R_2 n + 2T(n/2)$$

$$T(n) = 2T(n/2) + Rn \leftarrow \text{final relation}$$

$$2 \cdot T(n/2) = 4T(n/4) + R\frac{n}{2} \quad \cdot 2$$

$$4\,T(n/4) = 8T(n/8) + Rn \quad \cdot 4$$

$$T(n/8) = 2T(n/16) + R\frac{n}{8} \quad \cdot 8$$

$$\vdots$$

$$T(1) = R$$

$$T(n) = Rn + Rn + Rn \quad \ldots x$$

$kn + kn \quad \ldots \quad x \text{ times}$

$n \quad \frac{n}{2} \quad \frac{n}{4} \quad \ldots \quad x \text{ times}$

$\frac{n}{2^x} = 1 \qquad x = \log n$

$k n * x$

$= kn \log n \leq (k+1)\,[n \log n]$

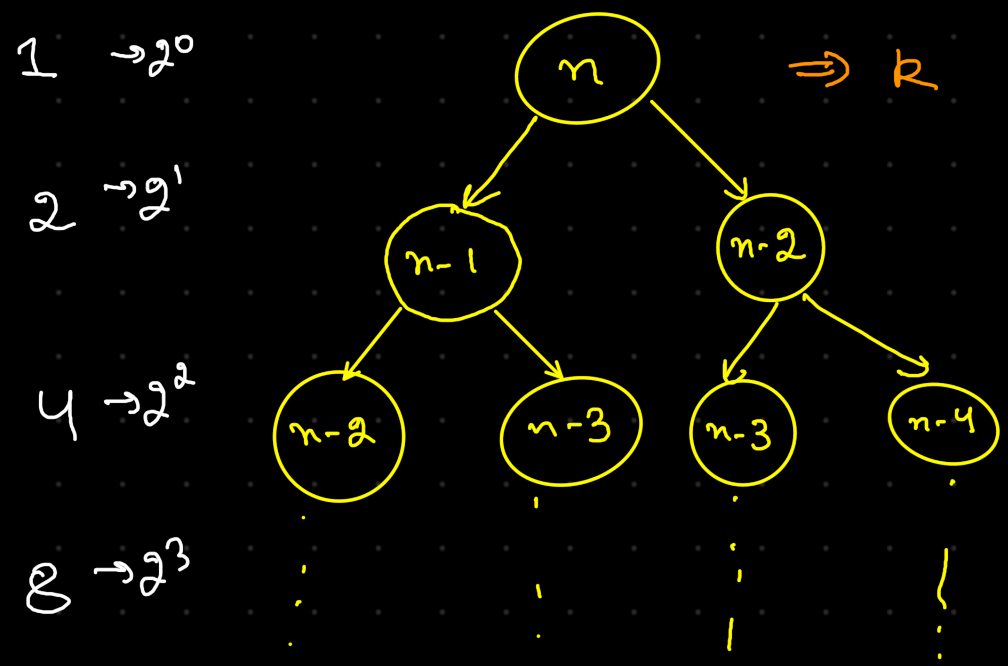
$O(n \log n)$

Fibonicci Number

```
def fib(n)
    if (n <= 1)
        return 1          } R
    return fib(n-1) + fib(n-2)
```

T(n-1) T(n-2)

$$T(n) = R + T(n-1) + T(n-2)$$



We will assume that the tree are ending all at the bottom most 1 together

$R(2^0 + 2^1 + 2^2 \cdots 2^n)$

$R(2^{n+1} - 1)$ (1 → 0)

$2^{n+1} R = 2 \cdot 2^n \leq c \, g(n)$ ($g(n) = 2^n$, $c = 2+1$)

$O(2^n)$

n	$O(\log n)$	$O(n)$	$O(n \log n)$	$O(n^2)$	$O(n^3)$	$O(2^n)$	$O(n!)$
10	3.32	10	33.22	100	1,000	1,024	3,628,80
100	6.64	100	664.39	10,000	1,000,000	Good Luck!	Good Luck!
200	7.64	200	1,528.88	40,000	8,000,000	Good Luck!	Good Luck!
500	8.97	500	4,484.12	250,000	125,000,000	Good Luck!	Good Luck!
1,000	9.97	1,000	9,966.57	1,000,000	1,000,000,000	Good Luck!	Good Luck!
5,000	12.29	5,000	61,434.69	25,000,000	125,000,000,000	Good Luck!	Good Luck!
10,000	13.29	10,000	132,877.12	100,000,000	1,000,000,000,000	Good Luck!	Good Luck!
20,000	14.29	20,000	285,783.99	400,000,000	8,000,000,000,000	Good Luck!	Good Luck!

While doing questions on leetcode, OA, codechef we face time limit exceeded.

Brute force normally not accepted.

⇒ no. of operation allowed per unit second

$\sim 10^8$ ops/s

In problems on internet, the same is also provided in question defn.

How to determine the solution by looking at constraint ?

Constraint	Worst time Complexity	Algo
$n \leq 12$	$O(n!)$	Backtracking recursion
$n \leq 25$	$O(2^n)$	Bit manip, recurs/Bachtr.
$n \leq 100$	$O(n^4)$	DP
$n \leq 500$	$O(n^3)$	DP
$n \leq 10^4$	$O(n^2)$	DP, graph tree
$n \leq 10^8$	$O(n \log n)$	Sorting, Divide & conquer
$n \leq 10^8$	$O(n)$	Mathematical Greedy
$n > 10^8$	$O(1)$ / $O(\log n)$	Mathematical, greedy

One more way to remember,

whatever algo we choose, if we plug in n we should get around $\sim 10^8$

Above is a lot helpful in remembering.

Space Complexity

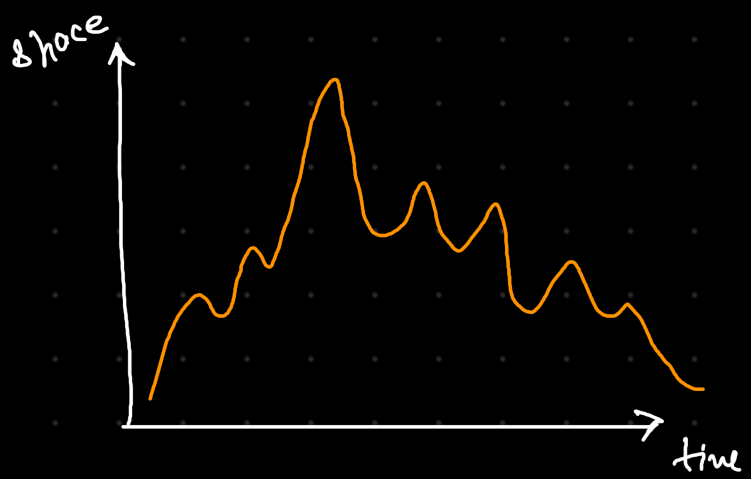
Space complexity is space required as a function of input size.

Space complexity

1. Source code space ~c
2. Space of variables ~c
3. input space
4. Auxiliary space.

merge (l1, l2)

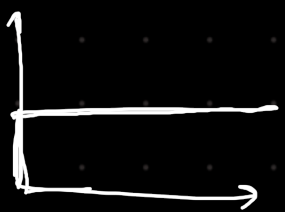
We majorly need to worry about auxilliary space.



while (i < n): n = 20

a = 5
a += 1 → 28

2 & n
x



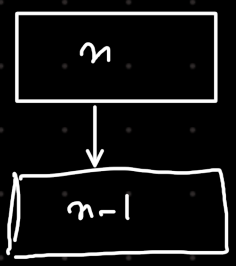
Bubble sort, selection sort, Insertion sort

No extra space which is order or dependent on n

$\Rightarrow O(n^0)$

Recursive algorithm

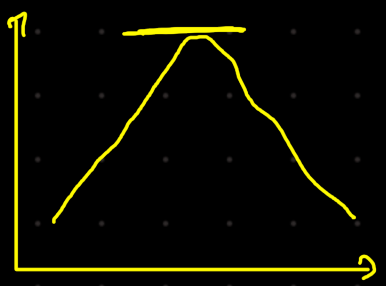
```
def fact(n):
    if (n<=1)
        return 1
    return n* fact(n-1)
```



$R \rightarrow f^n$

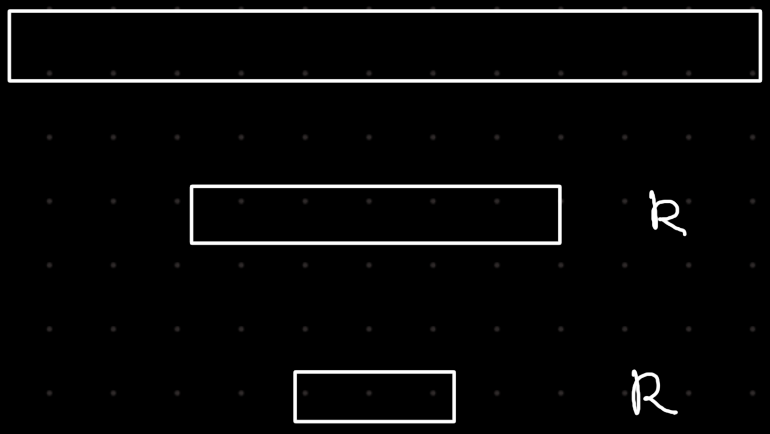
$n \rightarrow R * n$ space

$O(n)$ space



So, recursion is not free, some extra space is required

Binary Search



R = (no. of function)

$$n + \frac{n}{2} + \frac{n}{4} + \frac{n}{8} \cdots$$

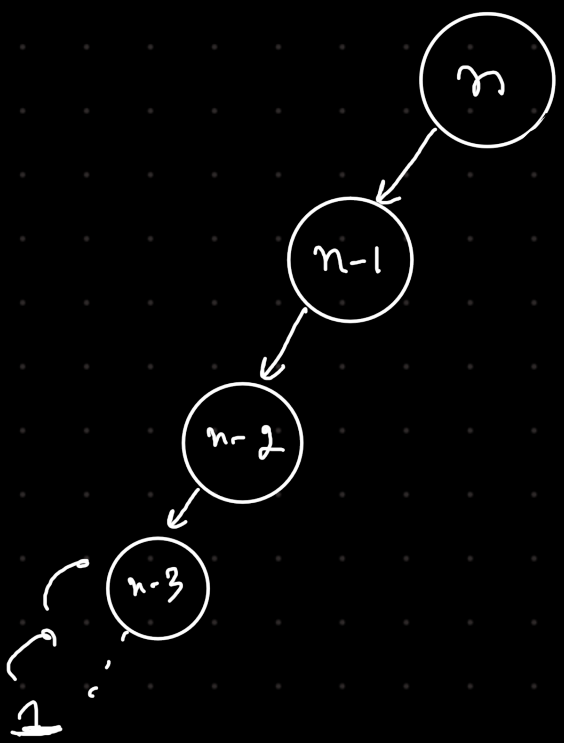
$$2^x = n \qquad x = \log_2 n$$

$\log_2 n$ functions in the call stack

fibonicci Number

```
def fib(n)
    if n ≤ 1:
        return 1

    return fib(n-1) + fib(n-2)      = fib(n)
```



R

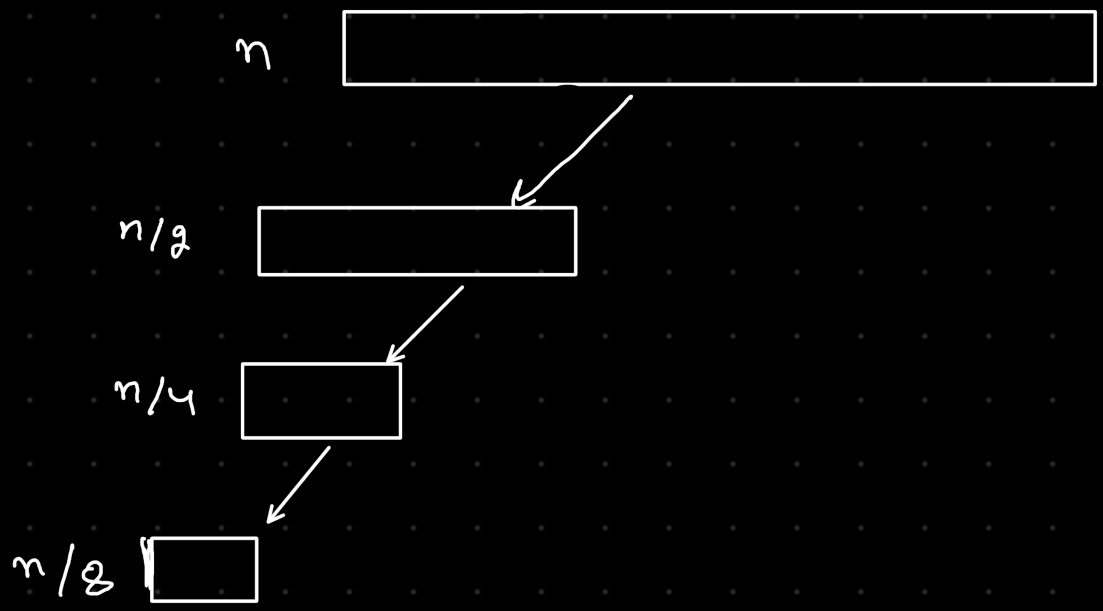
Call at one point only & not both

every f^n has constant space say R

and at a point in time maxm space will be Rn as n f^n will be in stack.

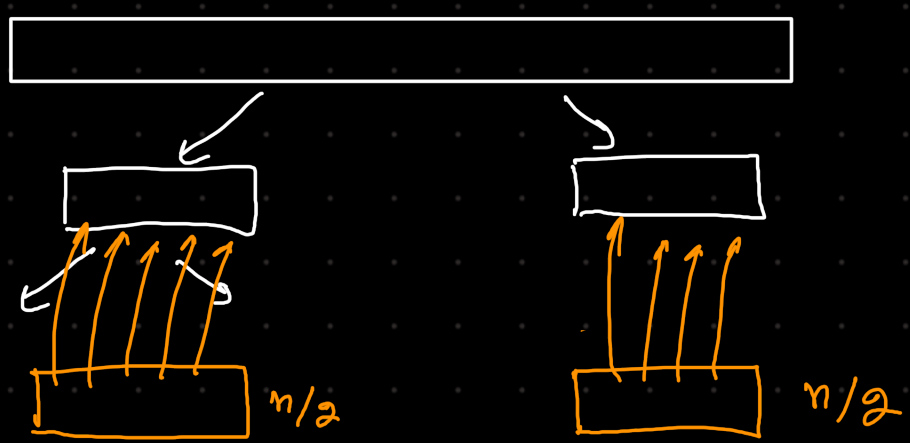
$O(n)$

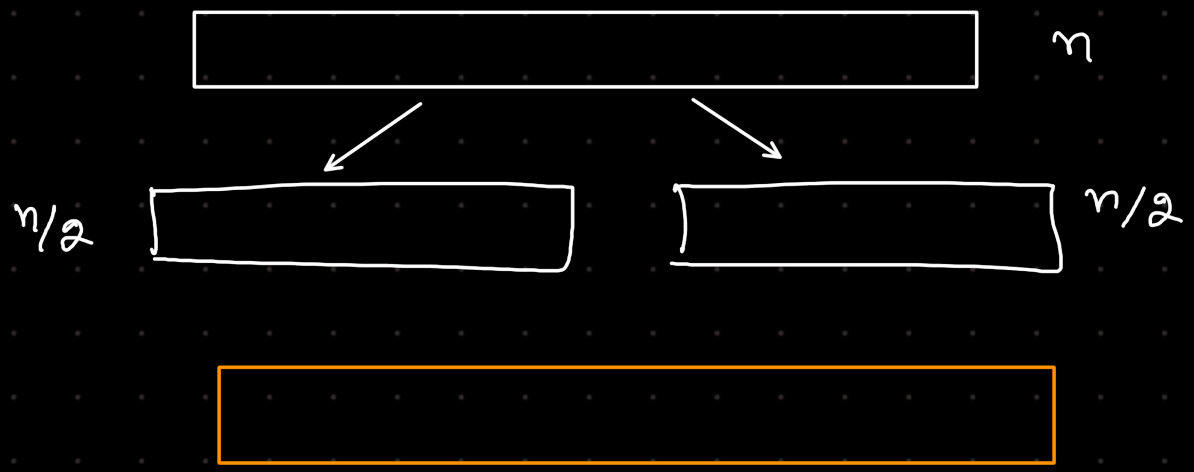
Merge sort



$$n \quad \frac{n}{2} \quad \frac{n}{4} \quad \frac{n}{8} \quad \dots \quad 1$$

$$\log_2 n$$





$$O\left(\underbrace{\frac{n}{2}}_{\text{via auxiliary array}} + \underbrace{\log n}_{\text{via } f^n \text{ stack}}\right)$$

$$\simeq O(n)$$

Eg 6

4 ✗
4 ✗
2 ✗
2 ✗
8 ✓