

Computational Model

Mathematical Abstractions which help us analyze the efficiency of a program

→ RAM model

- One of the most common model used for basic binary computation devices.

* Time Complexity

$\text{sum} = 0$	- 1 time
for $i = 0 \dots n-1$	- $2n$ ($i++$ & $i \leq n$ for each cycle)
$\text{sum} += a(i)$	- $3n$ (load $a(i)$, $a(i) + \text{sum}$, store in sum)
print (sum)	- 1

$$\text{time complexity} = 1 + 2n + 3n + 1$$

$$= 2 + 5n$$

→ Constant is not the property of algorithm but property of how we write code.

$$\therefore O(n)$$

$$\exists n_0, c \quad \forall n \geq n_0 : f(n) \leq c \cdot g(n) \quad \text{--- Eq ①}$$

$$\hookrightarrow 2 + 5n \Rightarrow f(n) = 2 + 5n$$

$$g(n) = n \quad \text{such that } f(n) = O(g(n)) \quad \text{--- Eq ②}$$

$$\text{Thus } n_0 = 2$$

$$c = 6$$

$$\hookrightarrow 2 + 5n \leq 6n \quad \text{Based on Eq ①}$$

$$2 \leq n$$

$O(n)$ → Upper bound of complexity or
the worst case complexity

$\Omega(n)$ → Lower bound of complexity or
the best case complexity.

If $O(n)$ and $\Omega(n)$ are same then $\Theta(n)$

↓
Asymptotic
Notation

Practice Q

for $i = 0 \dots n-1 - n$
for $j = 0 \dots n-1 - n$
...
..

$$O(h) = n \times n = n^2$$

Practice Q

$i=0 - 1$
while $i \cdot i \leq n - \sqrt{n}$
 $i++ - 1$

$$O(n) = \sqrt{n}$$

Practice Que

$i=1$
while $i \leq n$
 $i*=2$

After K iterations $i = 2^K$

$$2^K \geq n$$
$$K \geq \log_2 n$$

$$O(\log_2 n) \Rightarrow O(\log n)$$

base only denotes the slope
of logarithmic curve and
is also a constant.

For Recursion

```
def : f(n)
    if n=0 - 1
        return
    else f(n-1) - n-1
```

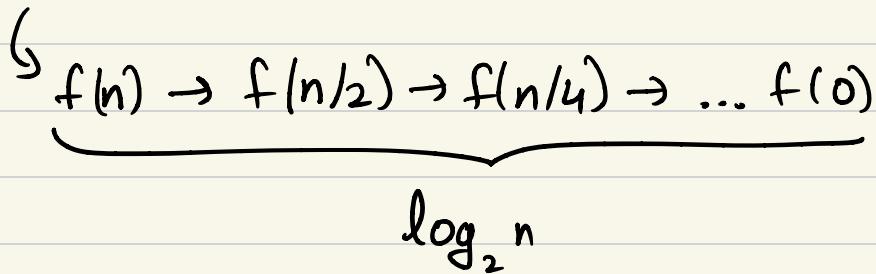
↳ $f(n) \rightarrow f(n-1) \rightarrow f(n-2) \dots$

n

$$\therefore O(n)$$

Practice Q.

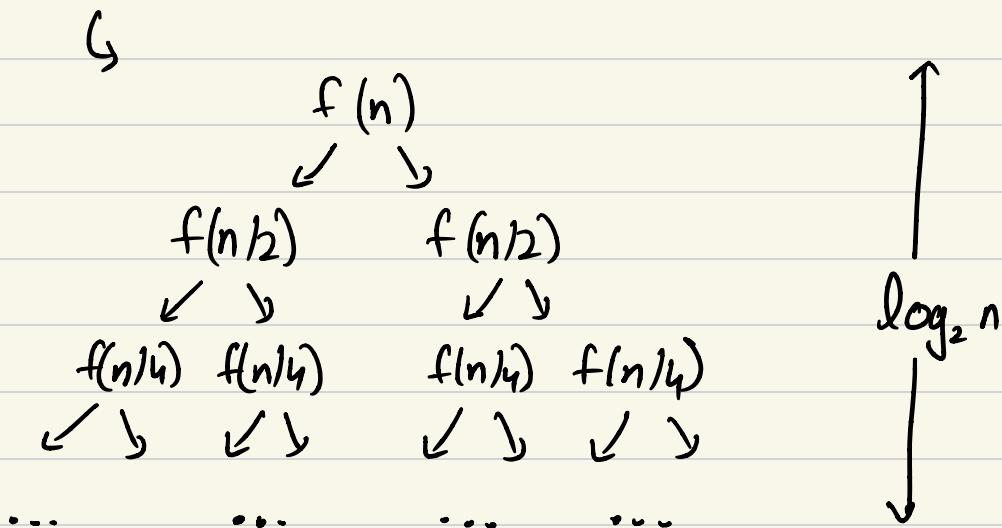
```
def : f(n)
  if n = 0
    return
  else f(n/2)
```



$O(\log n)$

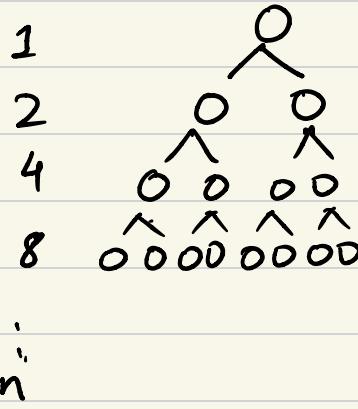
Practice Q

```
def : f(n)
  if n = 0 - 1
    return
  f(n/2)
  f(n/2)
```



↳ $1 \times \text{no. of nodes}$

no. of nodes \rightarrow



no. of nodes =

$$1 + 2 + 4 + \dots + 2^H \quad \leftarrow$$

$$= 2^{H+1} - 1$$

$$O(2^H)$$

$$\Rightarrow 2^{\text{height}} = 2^{\log_2 n} = n$$

$$O(n)$$

Practice Q

def : $f(n)$

if $n=0$

return

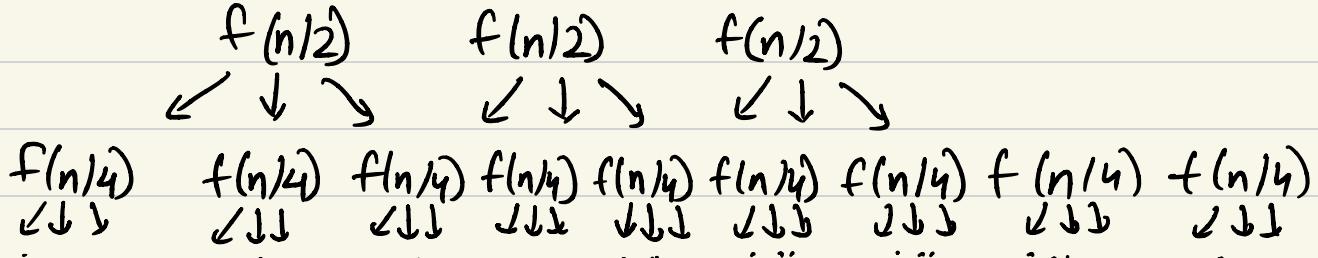
$f(n/2)$

$f(n/2)$

$+ (n/2)$



$f(n)$



$$\begin{aligned} \text{no. of nodes} &= 1 + 3 + 9 + \dots + 3^H \\ &= 3^{H+1} - 1 \\ &= 3^H \end{aligned}$$

$$\Rightarrow O(n^{\log_2 3})$$

Sorting Algorithms

↳ Insertion Sort

for $i=0 \dots n-1$

$j = i$

while $j > 0$ and $a[j] < a[j-1]$:

swap($a[j]$, $a[j-1]$)

$j--$

Best case - $O(n)$

Worst case - $O(n^2)$

↳ Merge Sort

merge(a, b):

$n = \text{len}(a)$

$m = \text{len}(b)$

int $c[n+m]$

$i = 0, j = 0, k = 0$

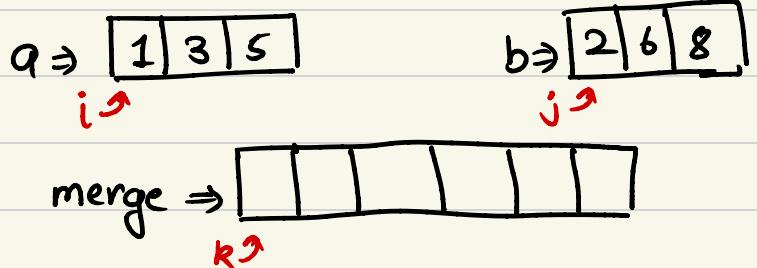
while $i < n$ or $j < m$:

if $j = m$ or($i < n$ and $a[i] < b[j]$):
 $c(k++) = a(i++)$

else

$c(k++) = b(j++)$

return c



sort(arr):

```

if len(arr) < 2: return arr
a = arr[0...n/2-1]
b = arr[n/2...n-1]
a = sort(a) - T(n/2)
b = sort(b) - T(n/2)
return merge(a,b) arr - O(n)
    
```

$$T(n) = 2 \cdot T(n/2) + c \cdot n$$

$$T(n) \leq c n \log n$$

$$T(n) \leq 2 \cdot c \frac{n}{2} \log \frac{n}{2} + cn$$

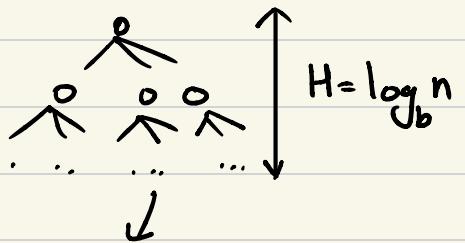
$$\leq cn(\log n - 1) + cn$$

$$T(n) \leq cn \log n$$

$$O(n \log n)$$

Master's Theorem

$$T(n) = a \cdot T\left(\frac{n}{b}\right) + f(n)$$



$$a^{\log_b n} \leftarrow n^{\log_b a} \leftarrow a \begin{cases} g(n) \\ g(n/b) \\ g(n/b^2) \end{cases}$$

$$f(n) = O(n^{\log_b a - \epsilon})$$

$$\Downarrow$$
$$T(n) = O(n^{\log_b a})$$

$$f(n) = O(n^{\log_b a + \epsilon})$$

$$\Downarrow$$
$$T(n) = O(n^{\log_b a + \epsilon})$$

$$f(n) = O(n^{\log_b a})$$

$$\Downarrow$$
$$T(n) = O(n^{\log_b a} \cdot \log n)$$