Time Complexity in Recursion

- Harsh Gupta

Goal

To understand:

- Substitution Method
- Master’s Method
- Time Complexity in Recursion

Substitution Method

- In this method we make a **guess for the solution** and then we use **mathematical induction** to prove if the guess is correct or incorrect.

Example:

Consider a recurrence relation →

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

Substitution Method (Contd…)

We guess the solution as **T(n) = O(nLogn)**. Now we use **induction** to prove our guess.

We need to prove that T(n) <= cnLogn (c is a constant). We can assume that it is true for values smaller than n.

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

$$<= 2cn/2Log(n/2) + n$$

$$= cnLogn – cnLog2 + n$$

$$= cnLogn – cn + n$$

$$<= cnLogn$$

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

$$T(n) = O(n \log n) \quad \text{(assume)}$$

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

where c is some constant

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

$$T(n) \leq cn \log\left(\frac{n}{2}\right) + n$$

$$T(n) \leq cn\log(n/2) + n$$

$$\log\left(\frac{a}{b}\right) = \log a - \log b$$

$$T(n) \leq cn\log n - cn\log 2 + n$$

$$\leq cn\log n$$

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

$$T(n) = O(n\log n)$$

Substitution Method (Contd…)

Now we will guess the solution as **T(n) = O(n)** and check whether it satisfies the relation or not.

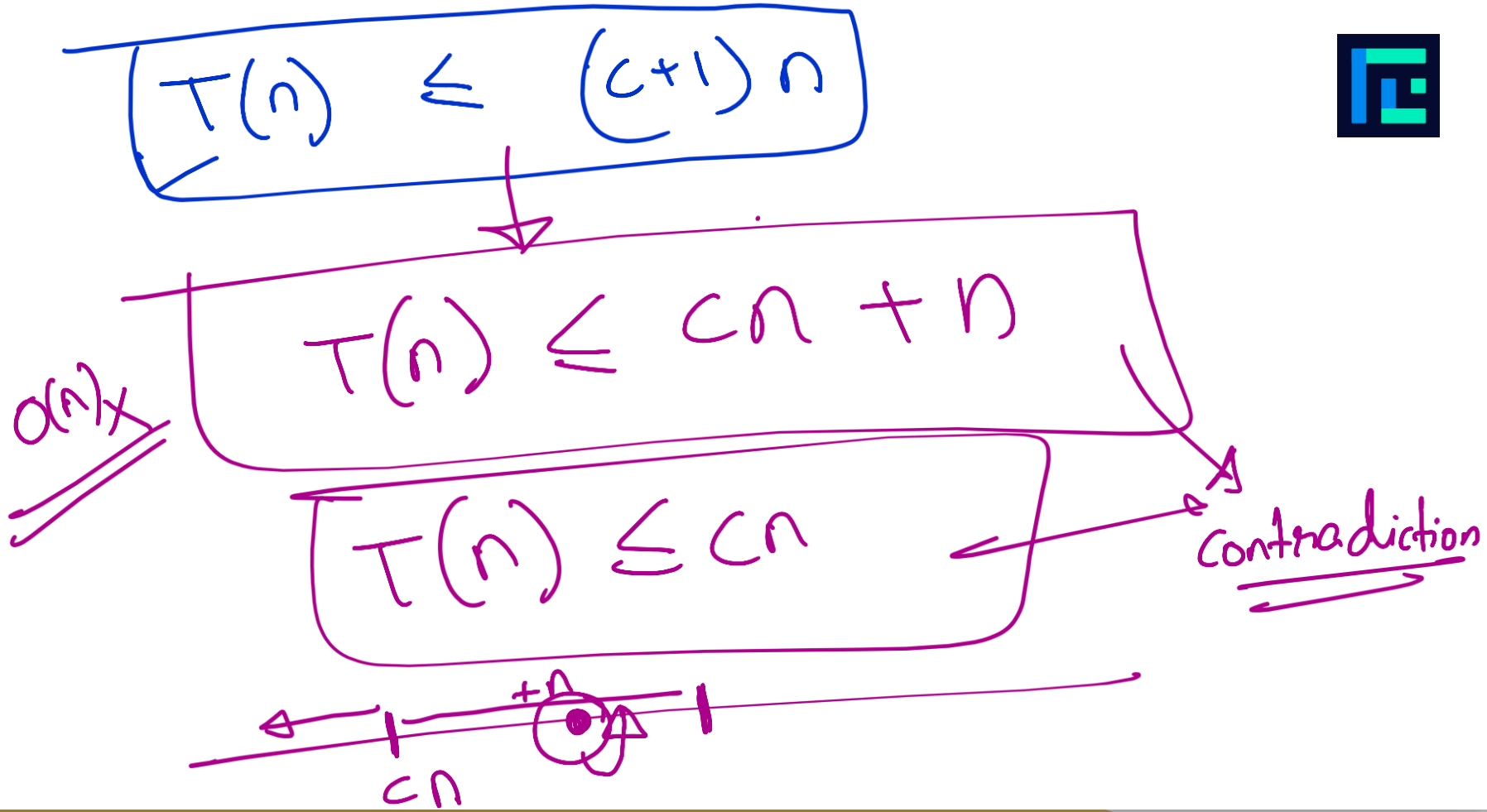
$T(n) = 2T(n/2) + n$ ✗ merge sort

$T(n) = O(n) \Rightarrow T(n) \leq cn$ ✓

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

$\Rightarrow T(n) \leq cn + n$

$nlogn$ ✗ n^2 ✓


T(n) ≤ (c+1) n
T(n) ≤ cn + n
a(n)x
T(n) ≤ cn
contradiction
cn
+n

$T(n) = O(n^2) \Rightarrow T(n) \leq cn^2$

for some constant c

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

$\Rightarrow T(n) \leq 2 \cdot \frac{cn^2}{k} + n$

$$T(n) \leq \frac{cn^2}{2} + n$$

$$T(n) \leq cn^2$$

Achieved

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

$n = 4$

$T(n) \leq 8c + 4$

$nlog(logn)$

Assumed

$T(n) \leq cn^2$

$T(n) \leq 16c$

$8c + 4 > 16c$

$8c \leq 4$

$c \leq 1/2$

0.25

Master’s Theorem

Master Theorem is a tool that allows us to get our answer without going through the recurrence relations each time. Let T(n) be a **monotonically increasing function** satisfying the conditions

Master's Theorem (Contd…)

T(n) = a * T(n/b) + f(n) ; where a > 0, b > 1 and f(n) ∈ O(n^d) and d ≥ 0

Then, depending upon the relation between a, b and d; we can have the following solutions:

- $T(n) = O(n^d)$ if $d > \log_b a$
- $T(n) = O(n^d * \log n)$ if $d = \log_b a$
- $T(n) = O(n \wedge \log_b a)$ if $d < \log_b a$

Example 1

Find time complexity of the given recurrence relation:

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

$T(n) = a * T(n/b) + O(n^d)$

$a = 1$

$b = 2$

$d = 1$

$\log_b a = \log_2 1 = 0$

$d > \log_b a$

$d = 1$

$O(n)$

Example 2

Find time complexity of the given recurrence relation:

$T(n) = 7 \cdot T(n/2) + O(n^2)$

$a = 7, \quad b = 2, \quad d = 2$

$\log_b a = \log_2 7 > d$

$O(n^{\log_b a}) = O(n^{\log_2 7})$

Example 3

Find time complexity of the given recurrence relation:

$T(n) = 3 \cdot T(n/4) + O(n)$

$a = 3, \; b = 4, \; d = 1$

$\log_4 3 < d$

$\Rightarrow O(n^d) = \boxed{O(n)}$

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

$a = 2, \quad b = 2, \quad d = 1$

$$\log_b a = \log_2 2 = 1 = d$$

$$O(n^d \log n) = O(n \log n)$$

Drawbacks of Master's Theorem

Master's Theorem cannot be applied for all recurrence relations and cannot be used if

- T(n) is **not a monotone**. (Eg :- T(n) = Cos(n))
- f(n) is **not a polynomial**. (Eg :- $f(n) = 2^n$)
- b cannot be represented as a **constant** (Eg :- T(n) = T(sqrt(n)))

$O(n^d)$

$f(n) = 2^n$

$$T(n) = aT(n/b) + O(n^d)$$

$$T(n) = aT(\sqrt{n}) + O(n^d)$$

Master Theorem

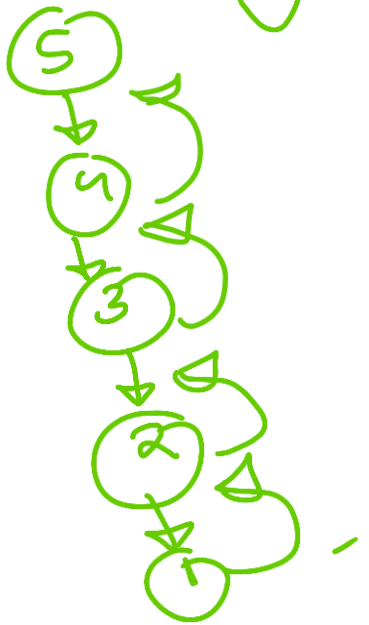
Using Recursion Tree

$$T(n) = T(n-1) + O(1)$$

Time complexing

5 → 4

$n = 5$

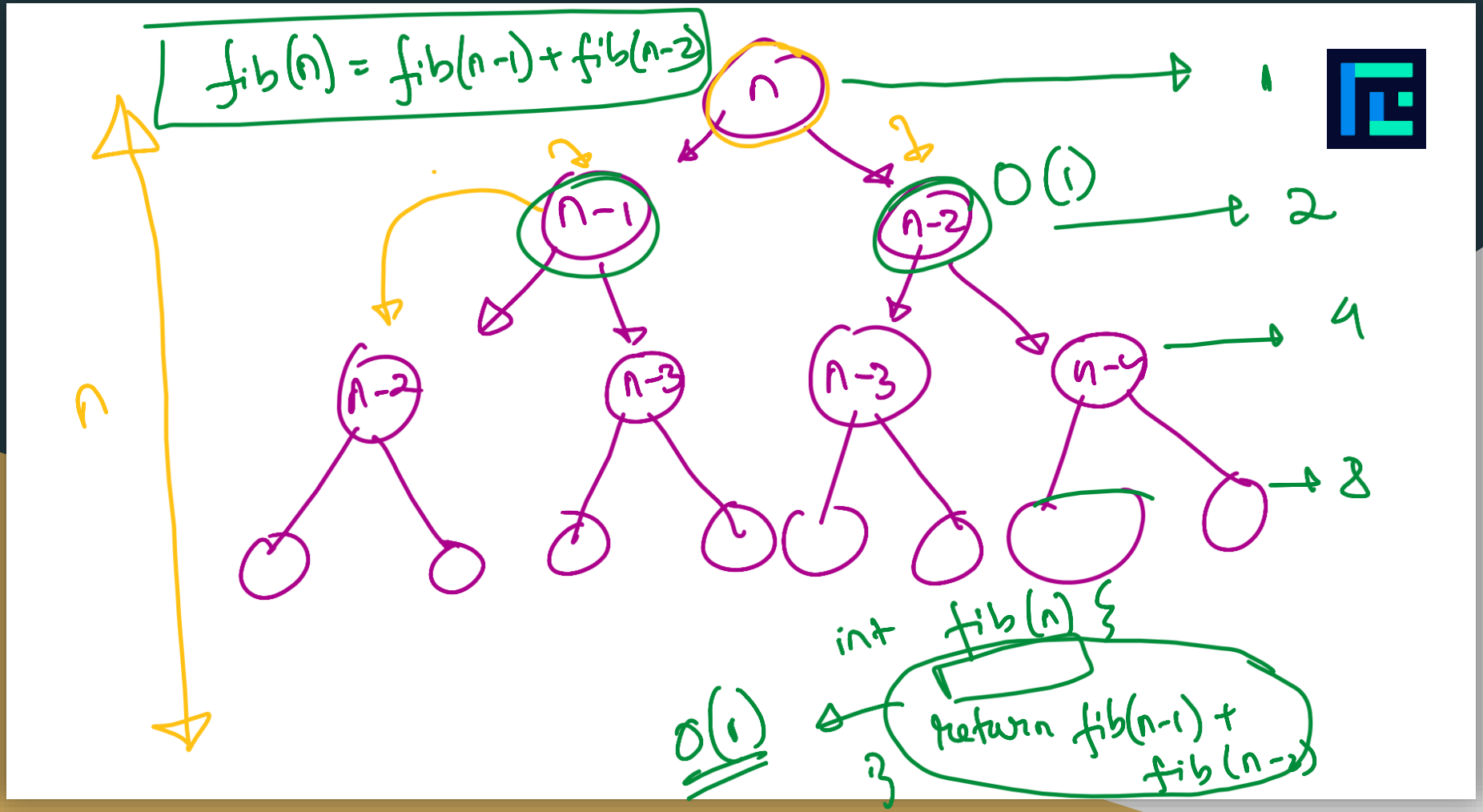


$$T(n) = O(n)$$

for (int i = 0 ; i < n ; i++) → O(n)

for (int i=0 ; i < n ; i+=2) → O(n/2)

fib(n) = fib(n-1) + fib(n-2).


fib(n) = fib(n-1) + fib(n-2)
n
n-1
n-2
n-2
n-3
n-3
n-4
1
O(1)
2
4
8
n
int fib(n) {
return fib(n-1) +
fib(n-2)
}
O(1)

Height of the tree is n

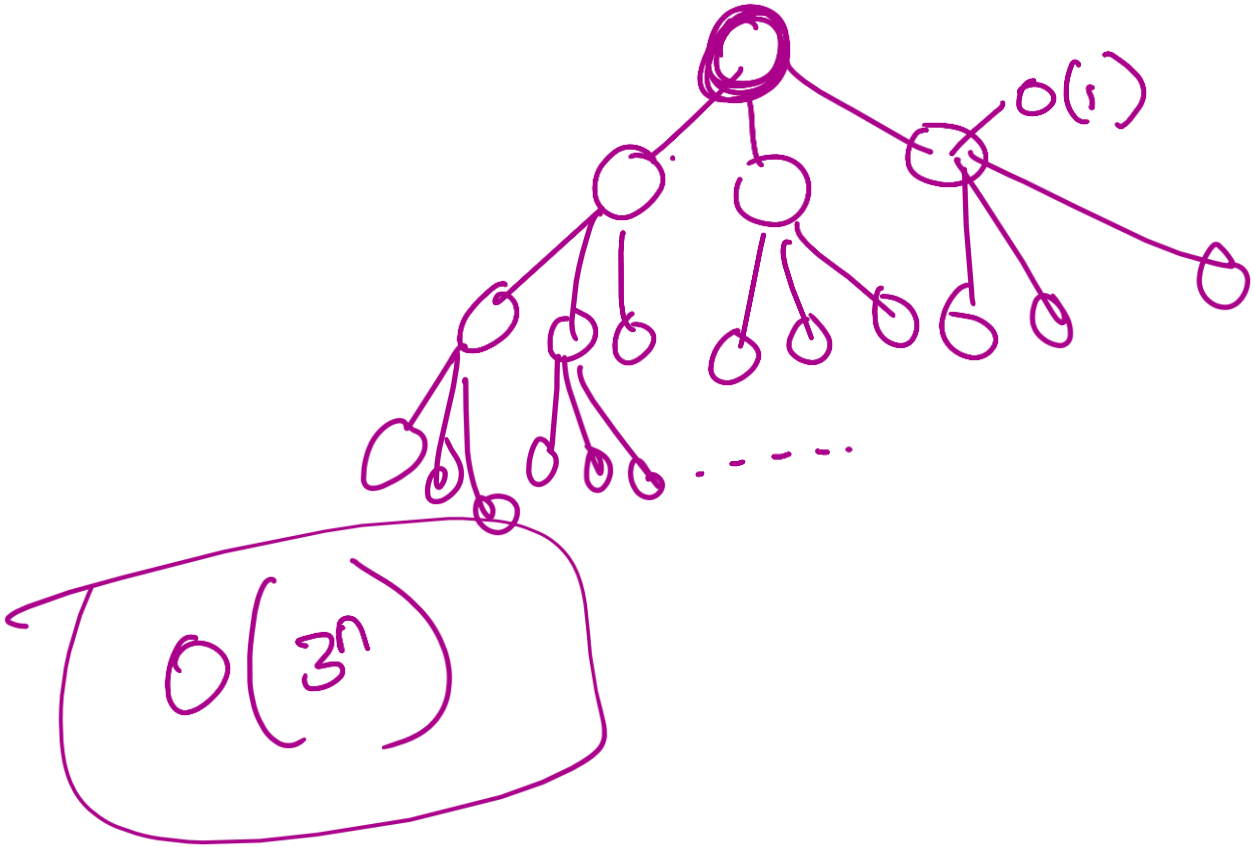
Time complexity = Total no. of nodes in the tree

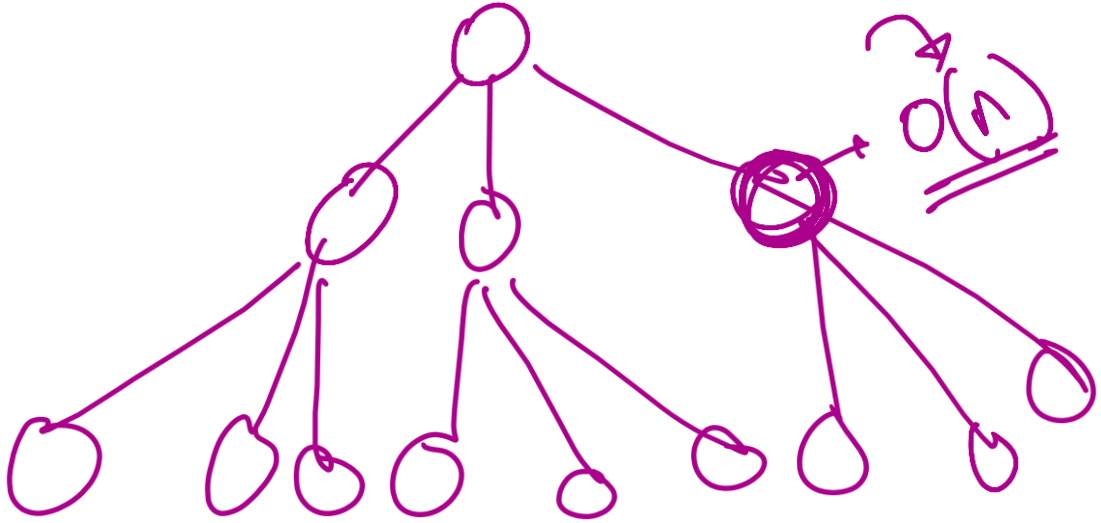
$1 + 2 + 4 + \cdots + 2^{n-1}$

GP

$a \cdot \left(\frac{r^n - 1}{r - 1}\right) = 1 \cdot \left(\frac{2^n - 1}{2 - 1}\right)$

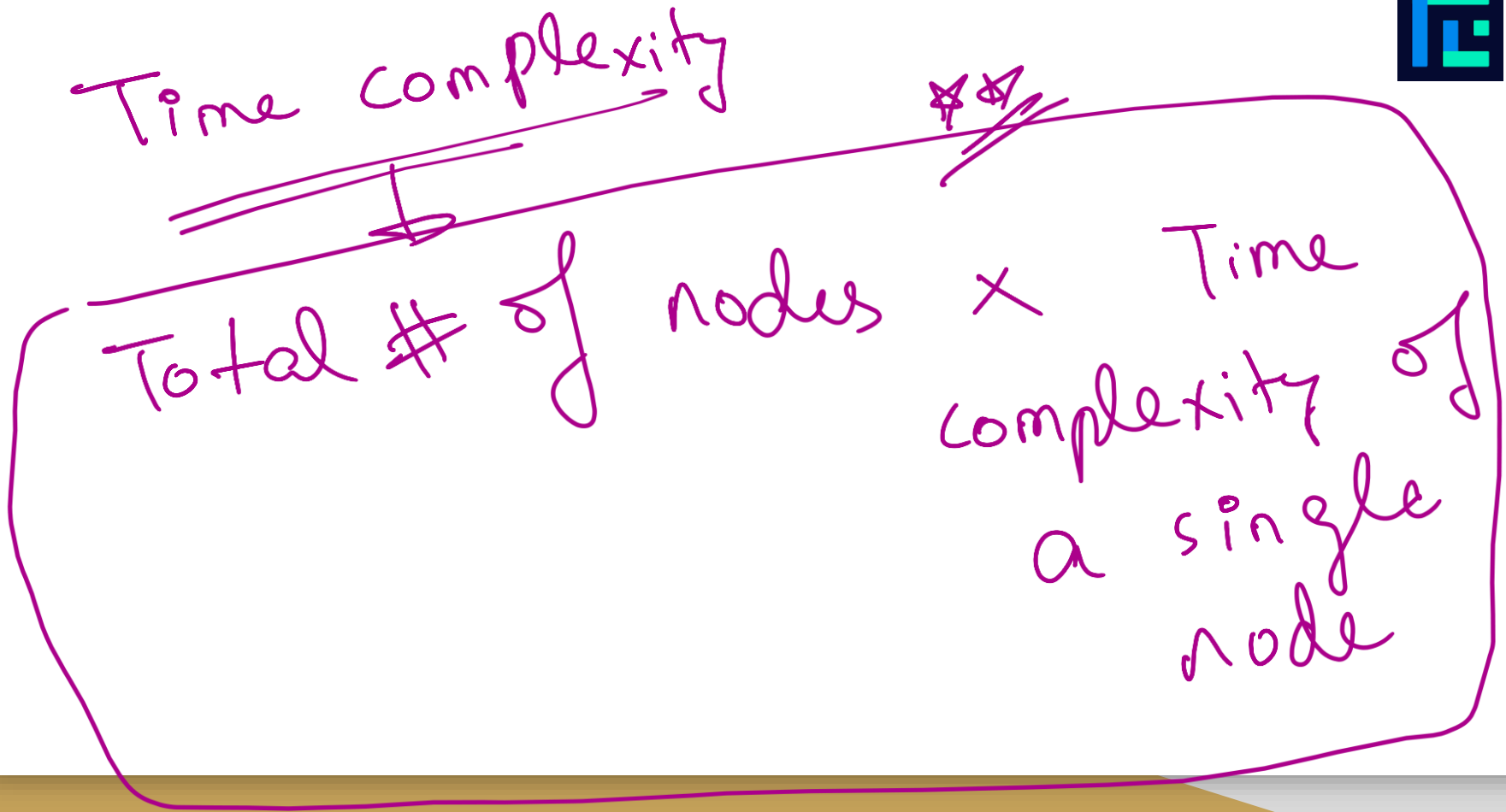
$T(n) = O(2^n)$


O(1)
O(3^n)



Time complexity = $O(n \cdot 3^n)$

$3^n \times n$

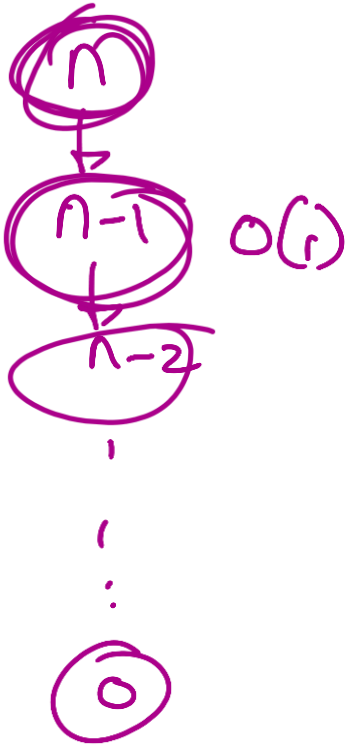

Time complexity
Total # of nodes x Time complexity of a single node

```
int f(n) {
    if (n<=0) return 0;
    return f(n-1) + f(n-2) + f(n-3);
}
```

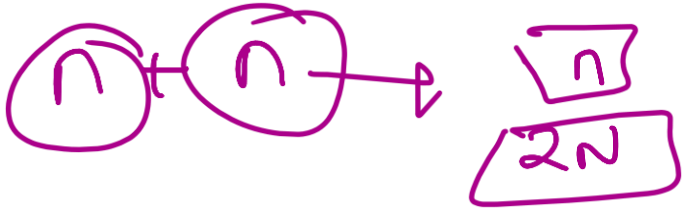
Quiz 1

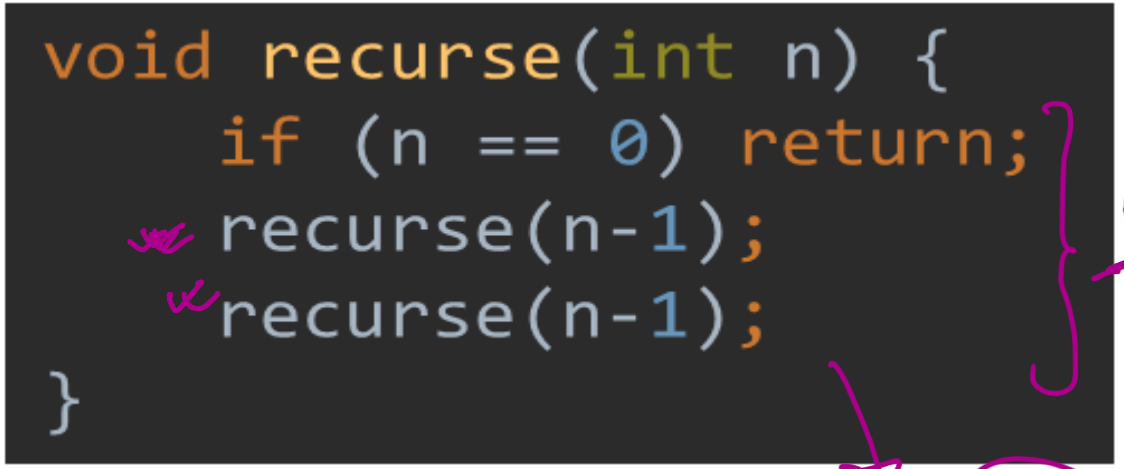
```
void recurse(int n) {
    if (n == 0) return;
    recurse(n-1);
}
```

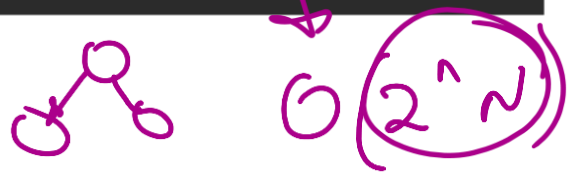
O(N)

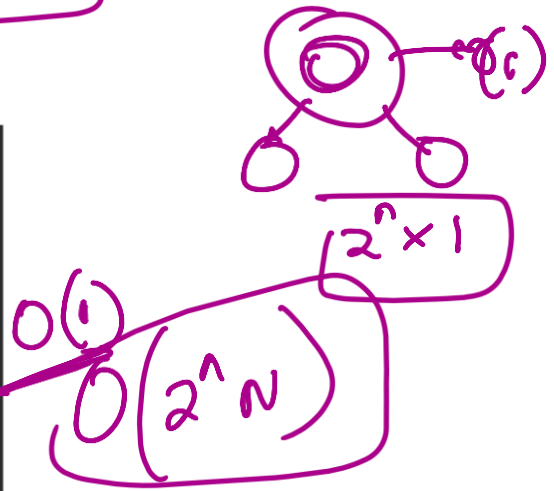


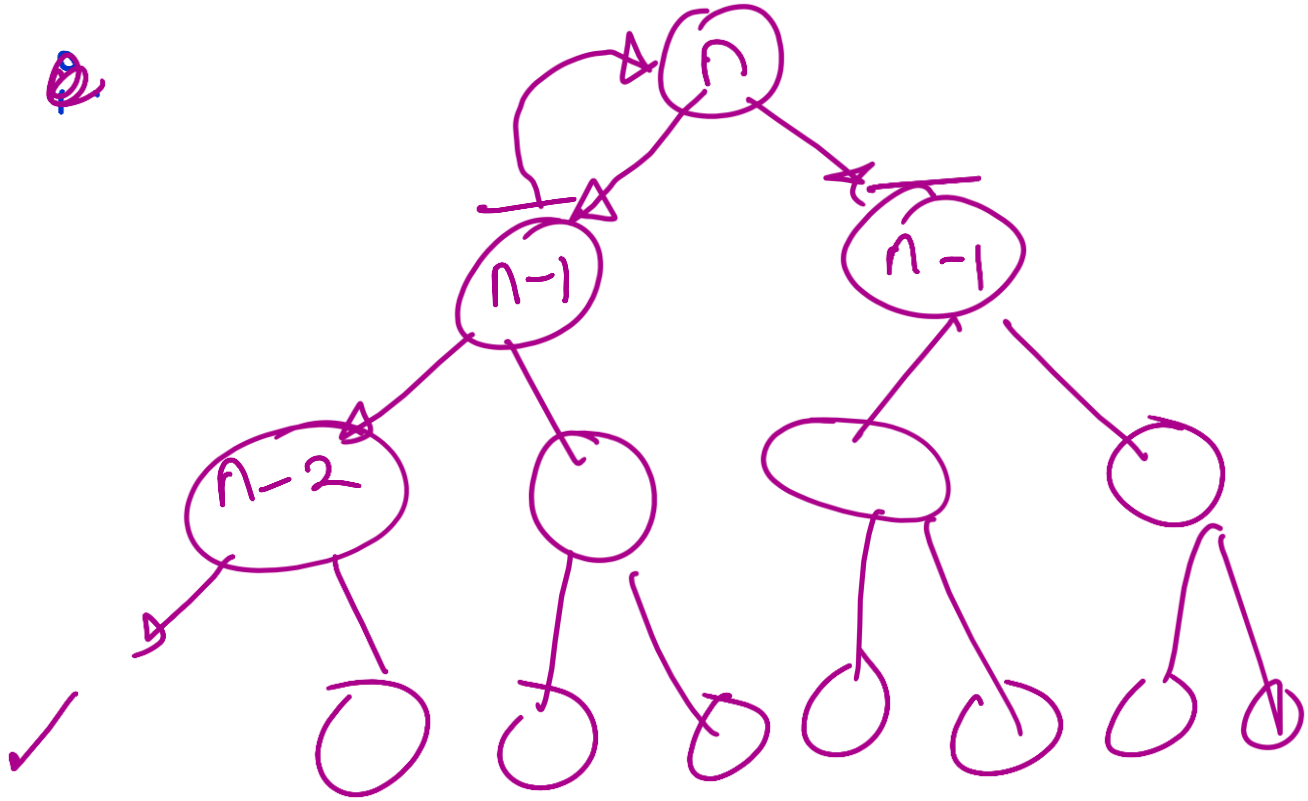
Quiz 2

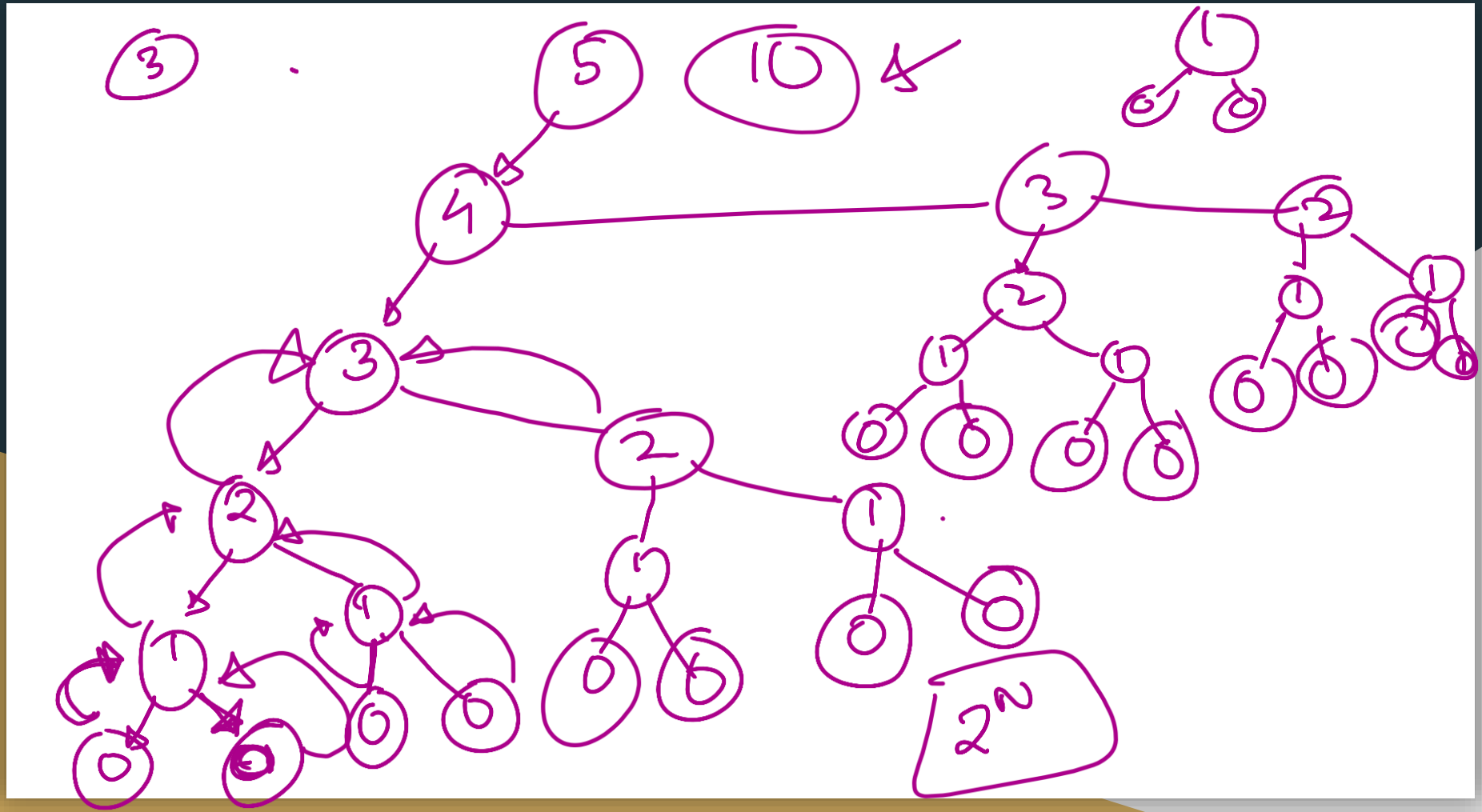



void recurse(int n) {
if (n == 0) return;
recurse(n-1);
recurse(n-1);
}


O(2^N)


O(1)
O(2^N)
2^N x 1


n
n-1
n-1
n-2


3
5
10
1
4
3
2
3
2
1
1
2
1
1
0
0
0
0
0
0
0
2
1
1
1
0
0
0
0
0
0
0
0
2^n

Quiz 3

```
void recurse(int n) {
    if (n == 0) return;
    recurse(n/2);
}
```

$O(\log n)$

height = ? $\log_2 n$

$n/2^k = 1$

$n = 2^k$

$k = \log_2 n$

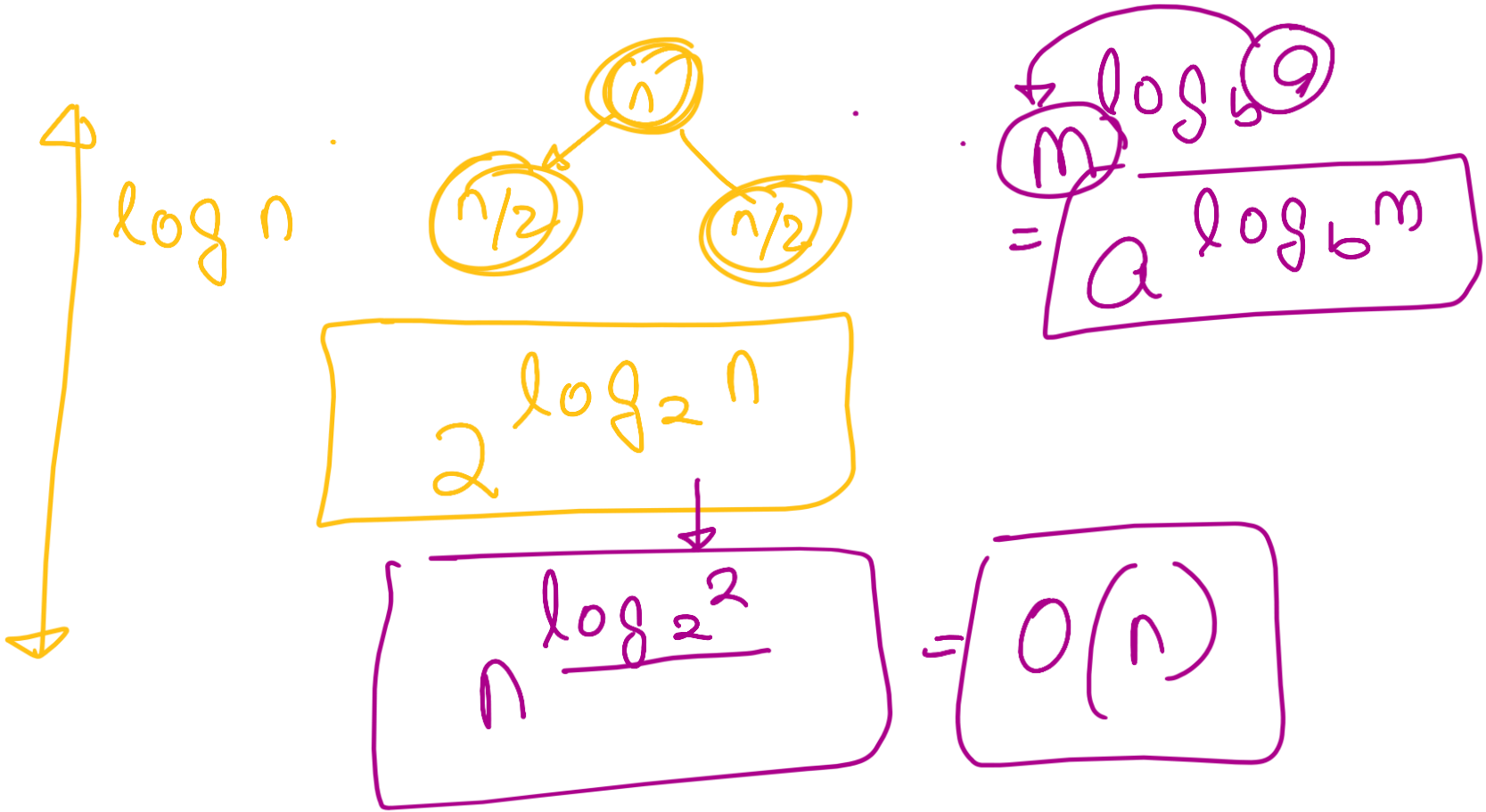
```
for(int i=0; i<=n; i++)
```
→ O(n)

```
for(int i=1; i<=n; i+=2)
```
→ O(n)

* ```
for(int i=1; i<=n; i*=2)
```
= O(log n)
```

Quiz 4

```
void recurse(int n) {
    if (n == 0) return;
    recurse(n/2);
    recurse(n/2);
}
```


log n
n
n/2
n/2
2^log2 n
n^log2 2
= O(n)
m
log_b a
= a^log_b m

Quiz 5

$f(n) = 2 \cdot f(n/2) + O(n)$

```
void recurse(int n){
    if(n==0){
        return;
    }
    recurse(n/2);
    recurse(n/2);
    for(int i=0;i<n;i++){
        cout<<"Hello"<<endl;
    }
}
```

Total # of nodes → $2^{\log_2 n} = O(n)$

and Time in one node → $O(n)$

$O(n^2)$

$O(n \log n)$