Algorithmic Analysis - 2

20 June 2022 20:08

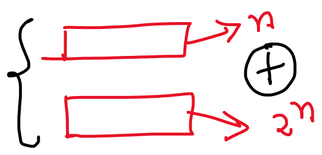
1. Algo. Analysis
2. goal of algo. anal.
3. Types of analysis → (1) Experimental analysis (2) Asymptotic analysis
4. Types of measurements
   - → Best
   - → Avg
   - → worst
5. Big oh notation

→ Multiple Examples

general rules:-

1. Ignore Constants:- ~~2~~n
2. $1 < \log n < n < n\log n < n^2 < 2^n < n!$

TC => $n\log n + n^2$ => n^2 (Ex + 2)



$x + 2^n$ → 2^n

$n * n \Rightarrow n^3$

O ⇒ Input Size ∝ Computer Steps

n = 5	10
n = 20	40

~~2~~ * n → O(n)

Q. function foo(n) { — O(1) — Worst → O, Best → Ω, Avg → Θ

```
let ans = 0;
for(let i=0; i<n; i++) {
    for(let j=1; j<Math.log(i); j++) {
        ans += i;
    }
}
```

- outer loop: n
- inner loop: log n
- O(n log n)
- Ω(1)
- Θ(n log n)

$\left(\frac{n \log n}{2}\right)$

Math.log(2)

Best → O(1)

ln(x)

$\log_e(x)$

1 → log 1
2 → log 2
3 → log 3
n → log n

m = 2n

function foo(n, m) { — TC: O( ) — n+m — O(n)/O(n)

```
let a=0, b=0;
for(i=1; i<n; i++) {
    a += i;
}
for(j=1; j<m; j++) {
    b += i;
}
```

O(n) + O(m)

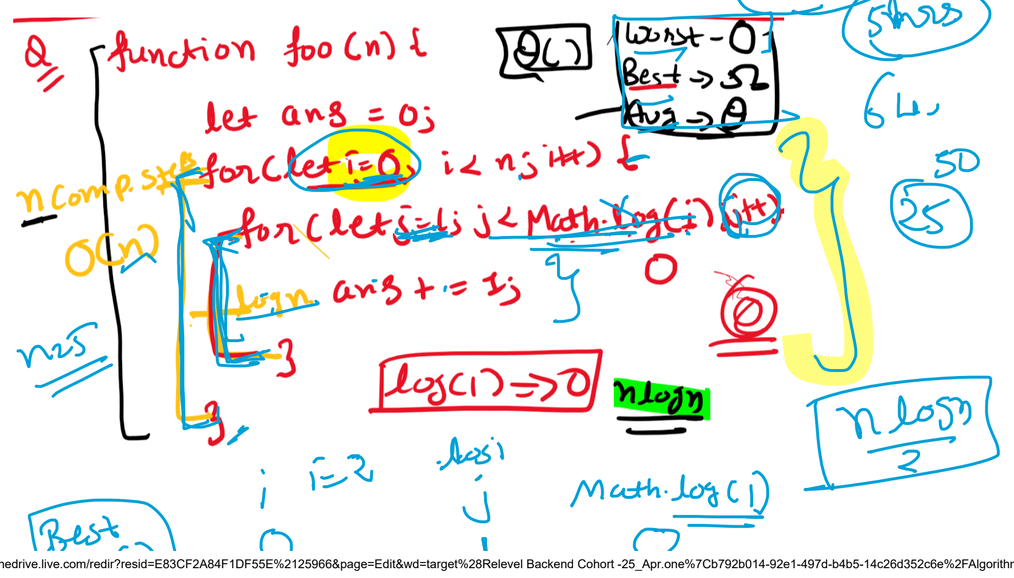
O(n+m)

O(n+2n)

O(2n)

O(n)

Q:-



Ω(1)

80% / 90% => Best -> Ω(1)
Avg =>

{ Select
bubble
Insert }

1 cs -> log1
2 cs => log2
3 cs => log3
:
n cs -> logn

Q
```
function foo(n) {
    let ans=0;
    for(i=1; i<n; i++)
    {
        for(j=n; j>i; j--) {
            ans += 1;
        }
    }
    print(ans);
}
```

i=1 { j=n => j=1 } n-1
i=2 j=n => j=2 n-2

TC
O( )

i=1 => j=n ---- j=1 } n-1 }
i=2 => j=n ---- j=2 } n-2 }
i=3 => j=n ---- j=3 } n-3 }
:
i=n => j=n ---- n } 1 =>

(n-1) + (n-2) + (n-3) + ---- n terms
n^2

Interview => 80% Explaining }
2-8% code
2-8% Time complexity

Q
```
function foo(n)
{
    for(i=2; i<=Math.sqrt(n); i++){
        {if(n%i==0) {
            return false;}
        }
    }
    return true;
}
```

Time Complexity :-
O( )

n = 25

4 { i=2, i=3, i=4, i=5

n = 16

i=2, i=3, i=4

sqrt(16) → 4 − 1 } sqrt(n)

sqrt(25) ⇒ 5

logn = C sqrt(n)

n

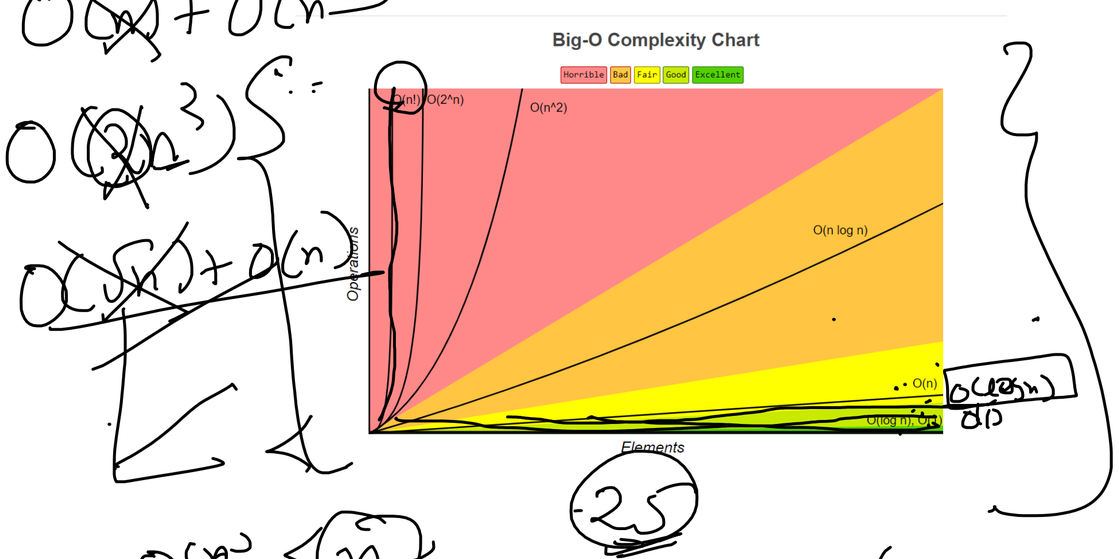
$\log_2 16 \Rightarrow 4$

$\log_3 16$

$\log_2 25 \Rightarrow 5$

$\log_2 36 \Rightarrow 6$

O(n) + O(n²)



O(n³) : O(n) + O(n) ⇒ O(n)

O(sqrt(n)) < n

O(n)! = O(f(n))

n!

O(n) + O(n) ⇒ O(n)

O(f(n)) O(1) + O(2n)

O(n)

for(i=1; i < f(n); i++) { }

f(n) = 2n²

n³

2n

n

n = 1 ⇒ 2

n = 2 ⇒ 8

n = 3 ⇒ 18

n³

matter of degree

n ~ n³

1. ignore lower order

$O(n) + O(n^2)$

$\{ O(n^1) + O(n^2)$

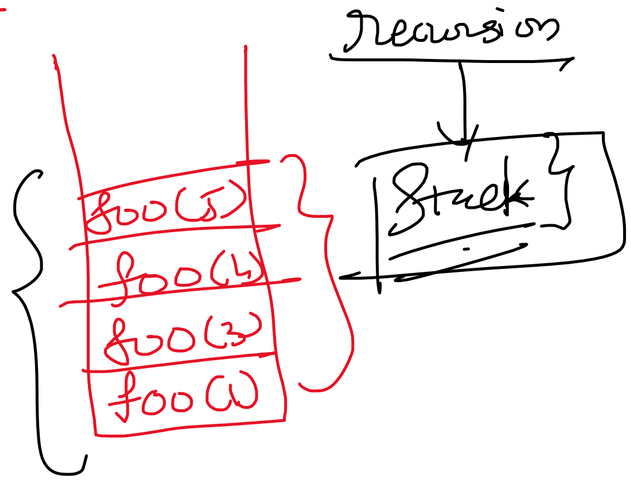
$O(\sqrt{n})$

$\Rightarrow O(n^{\frac{1}{2}}) + O(n^1)$

② Time complexity for recursion

1. Substitution method
2. Master theorem
3. Recursive tree

① Recurrance relation



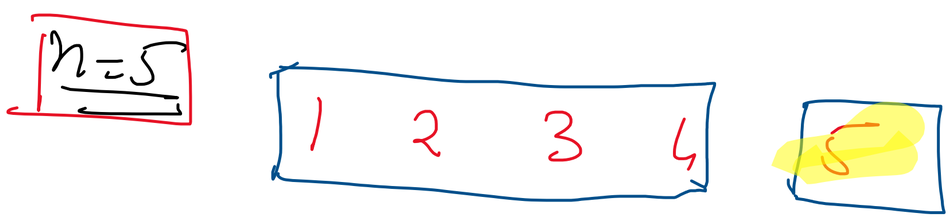
Q. Print first n natural no.

using recursion

$n=5 \Rightarrow$ 1, 2, 3, 4, 5

$n=2 \Rightarrow$ 1, 2

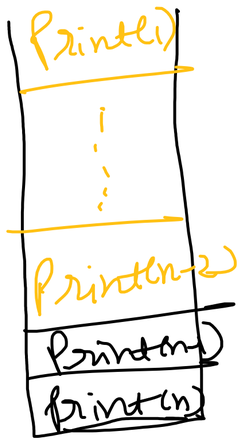
1. Base case
2. Recursive call
3. self work




```
function Rprint(n)
{
  if(n==0)
    return;
  Rprint(n-1);
  if(n==1){
    console.log(n);
    return;
  }
}
```

```
    console.log(n);
}
```

no. of fun calls = value of n



O(n) ≈ no. of calls in call stack.

n natural no. → n fun calls in call stack.

O(n)

```
Print(n)
{
    for(i=1; i<n; i++)
    {
      console.log(i)
    }
}
Print(5)
```

	Recursive	Iterative
TC	O(n)	O(n)
SC	O(n)	O(1)

Q
```
function sum(list, i) {
    if(i == list.length-1)
        return list[i]
    let res = sum(list, i+1);
    return res + list[i]
}
```

n, n-1

T(n) = T(n-1) + c

T(n) = T(n-1) + c — sub (1)

T(n-1) = T(n-2) + c

T(n) = T(n-2) + c + c — (2)

T(n-2) = T(n-3) + c

T(n) = T(n-3) + c + c + c — (3)

T(n) = T(n-1) + c

T(n) = T(n-2) + 2c

T(n) = T(n-3) + 3c

n-1, n-2, n-3, ...

T(1) = 0

$T(n) = T(1) + nC$

$T(n) = T(n-i) + iC$ ... $n-i$

$T(n) = T(1) + nC$

$T(n) = 0 + nC$

$T(n) = n(C)$

$T(n) = O(n)$

Q Fiboracci

```
function fibo(n) {
    if(n == 0 || n == 1) {
        return n;
    }
    return fibo(n-1) + fibo(n-2);
}
```

4, 5

4, 5

3

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

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

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

$T(n) = 2 * T(n-1) + C$

$T(n-1) = 2 * (n-2) + C$

$T(n) = 2 * (2 * (n-2)) + C) + C$

$T(n) = 4 * T(n-2) + 3C$

$= \int 2 * 1 (n-3) + 1C$

$= 1C + T(n-4) + 1C$

2^k Q $k=3$

$= 2^k * T(n-k) + (2^k - 1)$

$T = 1$

$n - k = 1$

$k = n - 1$

$n - k = 1$

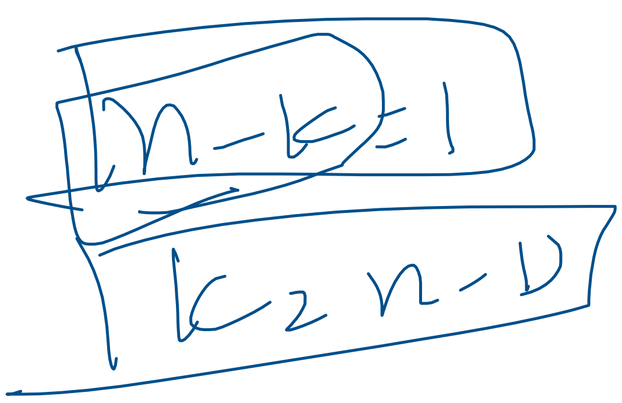
$2^k * T(n-k) + (2^k - 1)$

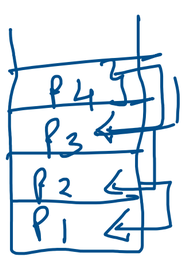
≥ 0

$O + 2^k (1)$

$O(n) = 2^k$

$T(n-k) = T(1)$






```
function permutation(str, l, r)
{
    if (l == str.length-1)
         console.log(str);
       else
       {
           for (let i = l; i <= str.length-1; i++)
           {
              str = swap(str, l, i);
              permutation(str, l + 1, r);
              str = swap(str, l, i);
           }
       }
}

function swap(a, i, j)
{
    let temp;
let charArray = a.split("");
temp = charArray[i] ;
charArray[i] = charArray[j];
charArray[j] = temp;
return (charArray).join("");
}

let str = "ABC";

permutation(str, 0, str.length-1);
```