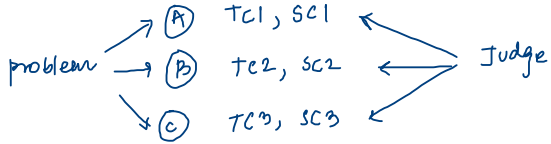
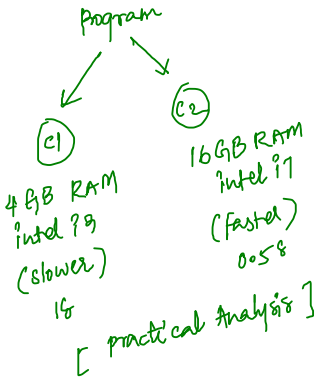
★ Time complexity :



- 1st preference should be ↓TC then only go for ↓SC

→ Solution which has least Time complexity
min (TC1, TC2, TC3)

→ If TC1 = TC2 = TC3, min (SC1, SC2, SC3)



★ Time complexity is machine Independent (Theoritical / Mathematical Analysis)

★ why are we ignoring constants and picking highest degree ?

→ apart highest degree all others do not make much impact on TC they are negligible as $n \to \infty$.

①

```
1  function solve1(m, n) {
2    let a = 0;
3    let b = 0;
4    for (let i = 0; i < n; i++) {
5      a = a + i;
6    }
7    for (let j = 0; j < m; j++) {
8      b = b + j;
9    }
10 }
```

m, n

2, 3

5, 2

7, 8

always dependent on user ⇒ not constant

no. of instructions ≃ n + m + 2 → constant

Time Complexity = O(n+m)

* ignore all the constants, pick the highest degree term

Big-oh (Asymptotic notation)

It tells that your program will not take more than "n+m" T.C.

```
function solve(a, b) {
    return a+b; — (1)
}
solve(2, 5);
```

no. of instructions ≃ 1

⇒ TC = O(constant) = O(1)

~~TC = O(one)~~

(2)

```
12  function solve2(n) {
13    let a = 0;
14    for (let i = 0; i < n; i++) {
15      for (let j = 0; j < m; j++) {
16        a = a + j;
17      }
18    }
```

(n, m)

(1)

(n*m)

* always your TC lies in iterations, remaining all will be ignored as they will be constants most of the cases.

take some example input,

n = 4 and m = 3

(1) i = 0
- → j = 0
- → j = 1
- → j = 2
- → ~~j = 3~~ (3 < 3)

} 3

(2) i = 1
- → j = 0
- → j = 1
- → j = 2
- → ~~j = 3~~

} 3

(3) i = 2
- → j = 0
- → j = 1
- → j = 2
- → ~~j = 3~~

} 3

(4) i = 3
- → j = 0
- → j = 1
- → j = 2
- → ~~j = 3~~

} 3

(5) ~~i = 4~~ (4 < 4)

TC = 3 + 3 + 3 + 3

= 4 * 3

= O(n * m)

for every i, innerloop runs m times,

⇒ m + m + m + ... + (n times)

⇒ n * m

(3)

```
26  function solve3(m, n) {
27    let a = 0;
28    for (let i = 0; i < n; i++) {
29      for (let j = n; j > i; j--) {
30        a = a + j;
31      }
32    }
33  }
```

Let $n = 4$

$n^2 + n$

$\Rightarrow$ let $n = 10^6$

$\Rightarrow n^2 + n = (10^6)^2 + 10^6$

$= 10^{12} + 10^6$ (10^6 → << small (negligible))

① $i = 0$

→ $j = 4$
→ $j = 3$
→ $j = 2$
→ $j = 1$ } 4
→ ~~$j = 0$~~ $(0 > 0)$

② $i = 1$

→ $j = 4$
→ $j = 3$
→ $j = 2$ } 3
→ ~~$j = 1$~~ $(1 > 1)$

③ $i = 2$

→ $j = 4$
→ $j = 3$ } 2
→ ~~$j = 2$~~

④ $i = 3$

→ $j = 4$ } 1
→ ~~$j = 3$~~

⑤ ~~$i = 4$~~

$\frac{n^2+n}{2}$ → $n^2 + n$ (layer) → n^2

$TC = 1 + 2 + 3 + 4$

$= 1 + 2 + 3 + 4 + \dots + n$

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

$= O(n^2)$

④

```
function solve4(n) {
  let i = 1;
  while (i <= n) {
    i = i + 2;
  }
}
```

Let $n = 10$

→ you need 50Rs but you have all 2Rs coins, how many coins will it take?

⇒ $50/2 = 25$ coins

① $1 < = 10$
$i = i + 2$
$= 1 + 2$
$= 3$

② $3 < = 10$
$i = i + 2$
$= 3 + 2$
$= 5$

③ $5 < = 10$
$i = i + 2$
$= 5 + 2$
$= 7$

④ $7 < = 10$
$i = i + 2$
$= 7 + 2$
$= 9$

⑤ $9 < = 10$
$i = i + 2$
$= 9 + 2$
$= 11$

⑥ ~~$11 < = 10$~~

TC = 5 iterations
$\simeq n/2$ iterations
$= O(N)$

$i = 1 \xrightarrow{+2} n$

$1 \xrightarrow{+2} 3 \xrightarrow{+2} 5 \xrightarrow{+2} 7 \xrightarrow{+2} 9 \xrightarrow{+2}$ ~~11~~

⇒ $n/2$

* if $n = 20$,

① $1 < = 20$
② $3 < = 20$
③ $5 < = 20$
④ $7 < = 20$
⑤ $9 < = 20$
⑥ $11 < = 20$
⑦ $13 < = 20$
⑧ $15 < = 20$
⑨ $17 < = 20$
⑩ $19 < = 20$
⑪ ~~$21 < = 20$~~

(5)

```
42  function solve5(n) {
43    while (n > 1) {
44      n = n + 20;
45      n = n - 10;
46      n = n - 15;
47    }
48  }
```

Let $n = 15$,

$\Rightarrow n = n - 5$

$n = n + 20$

$n = n - 10$

$n = n - 15$

$n = n - 5$

$n = n + 20 - 10 - 15$

$= n - 5$

① 15 > 1

$n = 15 + 20$

$= 35$

$n = 35 - 10$

$= 25$

$n = 25 - 15$

$= 10$

$(15 - 5)$

② 10 > 1

$n = 10 + 20$

$= 30$

$n = 30 - 10$

$= 20$

$n = 20 - 15$

$= 5$

$(10 - 5)$

③ 5 > 1

$n = 5 + 20$

$= 25$

$n = 25 - 10$

$= 15$

$n = 15 - 15$

$= 0$

$(5 - 5)$

④ ~~0 > 1~~

TC = 3 iterations

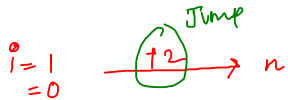
$\simeq n/5$ iterations

$= O(n)$

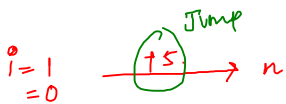
$\Rightarrow$ you need 50Rs, you have all 5Rs coins, how many coins you need?

$\Rightarrow 50/5 = 10$ coins

* Imp observation,

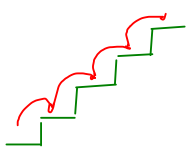


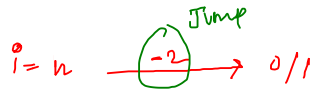
Iterations = $n/2$



Iterations = $n/5$

$= n/\text{Jump}$

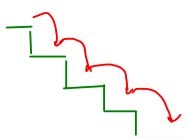




Iterations = $n/2$

Iterations = $n/5$

$= n/\text{Jump}$



9:15 PM

– 9:30 PM

BREAK

(6)

```
function solve6(n) {
  let i = 1;
  while (i < n) {
    i = i * 2;
  }
}
```

Let $n = 10$,

① $1 < 10$
$i = i * 2$
$= 1 * 2$
$= 2$

② $2 < 10$
$i = i * 2$
$= 2 * 2$
$= 4$

③ $4 < 10$
$i = i * 2$
$= 4 * 2$
$= 8$

④ $8 < 10$
$i = i * 2$
$= 8 * 2$
$= 16$

⑤ $16 \not< 10$

$n = 10$
$\Rightarrow$ iterations $= 4$

Let, iterations $= k$ for given some 'n'

itr = ① $i = 1 = 2^0$
itr = ② $i = 2 = 2^1$
itr = ③ $i = 4 = 2^2$
itr = ④ $i = 8 = 2^3$
itr = ⑤ $i = 16 = 2^4$
⋮
itr = (k) $i = 2^{k-1}$ (last iteration)

value of i $= 2^{itr-1}$

value of i in last iteration $\simeq n$

$2^{k-1} = n$

$\log_2 2^{k-1} = \log_2 n$

$k-1 \log_2 2 = \log_2 n$

$k = \log_2 n + 1$

$TC = O(\log_2 n)$

* Imp observation,

Iterations = $\log_2^n$

Iterations = $\log_2^n$

Iterations = $\log_5^n$

Iterations = $\log_5^n$

* TC = $O(\log_{Jump}^n)$

you have 1 Rs, make it n Rs using compound method, how many days will it take? *2

day	1	2	3	4	5	k^{th}
Rs	1	2	4	8	16	n
	2^0	2^1	2^2	2^3	2^4	2^{K-1}

K^{th} day = n

$\Rightarrow 2^{K-1} = n$

$\Rightarrow K = \log_2^n$ days to earn a total of n Rs.

log properties :

$$2^{x} = 32$$

$$x = \log_{2} 32$$

$$\rightarrow \log_{b^n} a^m = \frac{m}{n} \log_b a$$

$$\rightarrow \log_a a = 1$$

$$\rightarrow \log_b a + \log_b c = \log_b a \cdot c$$

(7)

```
57  function solve7(n) {
58    let i = n;
59    while (i > 0) {
60      i = i / 2;
61    }
62  }
```

$i = n \xrightarrow{/2} 0$

$O(\log_2 n)$

① $i = n$

② $i = n/2$

③ $i = n/2^2$

④ $i = n/2^3$

⋮

(k) $i = n/2^{k-1}$

i at last iteration $\simeq 1$

$$\frac{n}{2^{k-1}} = 1$$

$$n = 2^{k-1}$$

$$\log_2 n = \log_2 2^{k-1}$$

$$\log_2 n = k-1 \log_2 2$$

$$k = \log_2 n + 1$$

$TC = O(\log_2 n)$

(8) *

```
function solve8(n) {
  let i = 2;
  while (i < n) {
    i = i * i;
  }
}
```

value of i at last iteration $\simeq n$

Let iterations = k for some given n

value of i at k^{th} iteration?
↓
(last)

* $TC = O(\log_2 \log_2 n)$

① $i = 2 = 2^{1} \longrightarrow 2^{2^0}$

② $i = 2 * 2 = 4 = 2^{2} \longrightarrow 2^{2^1}$

③ $i = 4 * 4 = 16 = 2^{4} \longrightarrow 2^{2^2}$

④ $i = 16 * 16 = 256 = 2^{8} \longrightarrow 2^{2^3}$

⋮

Ⓚ $i = 2^{2^{k-1}}$

$$2^{2^{k-1}} = n$$

$$\log_2 2^{2^{k-1}} = \log_2 n \quad (\text{apply } \log_2)$$

$$2^{k-1} \log_2 2 = \log_2 n$$

$$\log_2 2^{k-1} = \log_2 \log_2 n \quad (\text{apply } \log_2)$$

$$k = \log_2 \log_2 n + 1$$

(9)

```
71  function solve9(n) {
72    let a = 0;
73    for (let i = 1; i <= n; i++) {
74      for (let j = 1; j <= i; j = i + j) {
75        a = a + i + j;
76      }
77    }
78  }
```

let $n = 4$

① $i = 1$
→ $j = 1$ } 1
→ $j = 1 + j$
$= 1 + 1 = 2$
$(j <= i)$ ~~$(2 <= 1)$~~

② $i = 2$
→ $j = 1$ } 1
→ $j = 2 + 1 = 3$
~~$(3 <= 2)$~~

③ $i = 3$
→ $j = 1$ } 1
→ $j = 3 + 1 = 4$
~~$(4 <= 3)$~~

④ $i = 4$
→ $j = 1$ } 1
→ $j = 4 + 1 = 5$
~~$(5 <= 4)$~~

⑤ ~~$i <= 4$~~

$TC = 1 + 1 + 1 + 1$
$= 4 = n$
$= O(n)$

* for every i,
j is running only once.

(10) *

→ (n, k)

```
function solve10(n) {
  let a = 0;
  for (let i = 1; i <= n; i++) {
    let p = i ** k;
    for (let j = 1; j <= p; j++) {
      a = a + i + j;
    }
  }
}
```

Let

$n = 3,\ k = 2$

$TC = 1^k + 2^k + 3^k + \dots + n^k$ (accurate)

$\simeq n^k + n^k + n^k + \dots + n^k$

$\simeq n \cdot n^k = n^{k+1} = O(n^k)$ (approx)

→ your program will not take $> n^k$

① $i = 1$

→ $p = i^k = 1^2 = 1$

→ $j = 1$ } 1

→ ~~$j = 2$~~

TC = $1 + 4 + 9$

$= 1^2 + 2^2 + 3^2 + \dots + n^2$

when $(k = 2)$

② $i = 2$

→ $p = i^k = 2^2 = 4$

→ $j = 1$

→ $j = 2$

→ $j = 3$ } 4

→ $j = 4$

→ ~~$j = 5$~~

③ $i = 3$

→ $p = i^k = 3^2 = 9$

→ $j = 1$

→ $j = 2$

→ $j = 3$ } 9

⋮

→ $j = 9$

→ ~~$j = 10$~~

(1) Linear $O(N)$ [d]

(2) Logarithmic $O(\log_2 N)$ [f]

(3) Exponential $(2^n, 3^n)$ [b]

(4) polynomial (n^3, n^4, n^5) [a]

(5) Log linear $(n \log_2 n)$ [c]

(6) Quadratic (n^2) [e]

a) $N^{K+9} \rightarrow N^{\text{constant}}$ $\quad (\text{constant})^N$ - exp

b) $5^{N*2} \rightarrow 5^{2N}$ $\quad N^{\text{const}}$ - poly

c) $(N/4) \log_2 (N/4000)$ → N (linear) $\log_2 N$ (logarithmic)

d) $3^{20} N + 10^5 \rightarrow N$ (ignore constants)

e) $10N + 9(N/100) + 340N^2 \rightarrow n + n + n^2$
$\rightarrow O(n^2)$

f) $10^3 \log_2 (N + 3N) \rightarrow \log_2 4N$
$\rightarrow \log_2 N$

* constant → $O(1)$

* linear is also polynomial with degree = 1
quadratic is also polynomial with degree = 2

Space Complexity :

→ If you are using any data structures (apart from given input) (in your solution)

→ O(size of DS)

* In general, you will mostly come across arrays of size n

→ SC = O(N)

→ else, SC = O(1)

rotate array

we take a temp arr we copied last k and the 1st (n-k)

SC = O(N)

TC = O(N)

inplace (3 reversals) (original Arr)

SC = O(1)

TC = O(N)

```
function hw1(n) {
  for (let i = 0; i < n; i++) {
    for (let j = 0; j < i; j++) {
      console.log("*");
      break;
    }
  }
}

function hw2(n) {
  let i = 1;
  while (i ** 2 <= n) {
    i = i + 1;
  }
}

function hw3(m, n) {
  while (m != n) {
    if (m > n) {
      m = m - n;
    } else {
      n = n - m;
    }
  }
}
```

```
function hw4(n) {
  let i = 1;
  while (i < n) {
    let j = n;
    while (j > 0) {
      j = parseInt(j / 2);
    }
    i = i * 2;
  }
}

function hw5(n) {
  for (let i = 0; i < parseInt(n / 2); i++) {
    for (let j = 1; j < n - parseInt(n / 2); j++) {
      let m = 1;
      while (m <= n) {
        m = m * 2;
      }
    }
  }
}
```