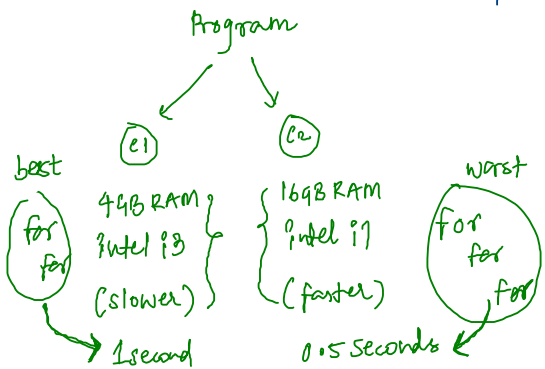
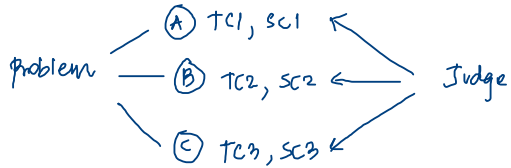
* Time complexity :



• —— pratical Analysis —— •

* TC, SC are machine Independent
(theoritical / Mathematical Analysis)



* 1st preference to ↓TC then only go for ↓SC

→ Solution that has least time complexity
min (TC1, TC2, TC3) irrespective of space

→ TC1 = TC2 = TC3, then pick the solution with min (SC1, SC2, SC3)

a. TC1 = 5 min, SC1 = 1MB
b. TC2 = 2 min, SC2 = 5MB
c. TC3 = 9 min, SC3 = 2MB

⇒ Solution (b) is better due to least TC

①

```
function solve1(m, n) {
  let a = 0;
  let b = 0;
  for (let i = 0; i < n; i++) {
    a = a + i;
  }
  for (let j = 0; j < m; j++) {
    b = b + j;
  }
}
```

★ Look for any loops, function calls.

★ TC = no. of instructions
= no. of iterations

no. of iterations = n

no. of iterations = m

+

n + m

→ m = 5, n = 3
5 + 3 = 8 itr

→ m = 10, n = 4
10 + 4 = 14 itr

→ program depends on m, n
→ inputs from user

⇒ TC = no. of iterations
= n + m
= O(n + m)

★ TC will be in terms of input size.

Big-oh, Mathematical notation, Asymptotic notation.

meaning: Your program will not take more than n+m iterations.

(2)

(n, m)

```
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    }
```

Total Iterations = 3 + 3 + 3 + 3
= 4 * 3

generalise, TC = O(n * m)

<= n * m > n * m

Let say n = 4 and m = 3 (take an example to understand no. of Iterations)

(1) i = 0
→ j = 0
→ j = 1
→ j = 2 } 3
→ j = 3 (3 < 3) ✗

(2) i = 1
→ j = 0
→ j = 1
→ j = 2 } 3
→ j = 3 (3 < 3) ✗

(3) i = 2
→ j = 0
→ j = 1
→ j = 2 } 3
→ j = 3 (3 < 3) ✗

(4) i = 3
→ j = 0
→ j = 1
→ j = 2 } 3
→ j = 3 (3 < 3) ✗

(5) i = 4
(4 < 4) ✗

(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  }
```

Total iterations = 4 + 3 + 2 + 1

generalise, TC = n + n-1 + n-2 + ... + 3 + 2 + 1

= sum of 1st n natural numbers

= $\frac{n(n+1)}{2}$ = $\frac{n^2+n}{2}$ = $O(n^2)$ (negligible → most impactful)

Let say n = 4

① i = 0
→ j = 4
→ j = 3
→ j = 2
→ j = 1 } 4
→ j = 0 X
(0 > 0)

② i = 1
→ j = 4
→ j = 3
→ j = 2 } 3
→ j = 1 X
(1 > 1)

③ i = 2
→ j = 4
→ j = 3 } 2
→ j = 2
(2 > 2) X

④ i = 3
→ j = 4 } 1
→ j = 3 X
(3 > 3)

⑤ i = 4
(4 < 4) X

★ when writing Big-oh we ignore any constants and also pick highest degree term

④

```
35  function solve4(n) {
36    let i = 1;
37    while (i <= n) {
38      i = i + 2;
39    }
40  }
```

Let say n = 10

① i = 1 <= 10
② i = 3 <= 10
③ i = 5 <= 10
④ i = 7 <= 10
⑤ i = 9 <= 10
⑥ i = 11 <= 10 ✗

⑤

generalise,

≃ n/2 iterations

TC = O(n)

* i = 1 —(+2, +K)→ n

1 —+2→ 3 —+2→ 5 —+2→ 7 —+2→ 9 —+2→ 11 ✗

⇒ n/2 ⇒ n/K ↘ Jump

* you need 50/- and you have many 2/- coins, how many coins are required to make 50/- ?

⇒ 25 coins + (2) = 50/-

⇒ total Amt/2 ⇒ 50/2 = 25

(5)

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

$n = n + 20$
$n = n - 10$
$n = n - 15$
$n = n - 5$

$x + 20 - 10 - 15$
$\Rightarrow x + 10 - 25$
$\Rightarrow x - 5$

* $n \xrightarrow{-5} 1$

$15 \xrightarrow{-5} 10 \xrightarrow{-5} 5 \xrightarrow{-5} 0$

$\Rightarrow$ 3 iterations

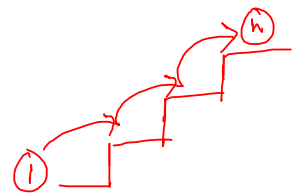
$\Rightarrow n/5$ iterations

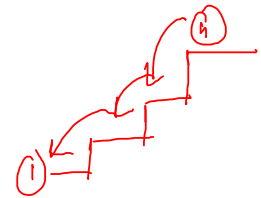
$30 \xrightarrow{-5} 25 \xrightarrow{-5} 20 \xrightarrow{-5} 15 \xrightarrow{-5} 10 \xrightarrow{-5} 5 \xrightarrow{-5} 0$

$\Rightarrow$ 6 iterations

$\Rightarrow 30/5 = 6$

* $1 \xrightarrow{+5} n \Rightarrow n/5$





no of steps will be same

⑥

```
50  function solve6(n) {
51    let i = 1;
52    while (i < n) {
53      i = i * 2;
54    }
55  }
```

In this program,

If $n = 128$

no. of iterations $= 8$

$\Rightarrow$ It will take K days to reach N.

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

$\Rightarrow \log_2 2^{K-1} = \log_2 N$

$\Rightarrow K-1 \log_2 2 = \log_2 N$ $\quad \Rightarrow K = \log_2 N + 1$

$\log_2 2 = 1$

$\star$ TC $= O(\log_2 N)$

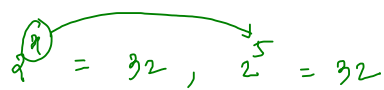
$\star$ $i = 1 \xrightarrow{*2} n$ $\quad \star$ $i = 1 \xrightarrow{*K} n$, $\log_K n$

You have a magical 1 Re. coin which doubles itself every day. How many days will it take to become N Rs.

Let say $N = 128$ Rs. $\Rightarrow$ 8 days to reach 128

day	1	2	3	4	5	6	7	8	.....	K th
Rs	1	2	4	8	16	32	64	128	.....	n
	2^0	2^1	2^2	2^3	2^4	2^5	2^6	2^7		2^{K-1}

Basic Log properties :



what is x ? $x = 5$

$2^x = 4294967296$

what is x ? we use log

$x = \log_2 4294967296$

↓

use log table to find the value

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

eg: $\log_{2^1} 2^{(k-1)m} = k-1 \log_2 2$

★ $\log_a a = 1$

eg: $\log_2 2 = 1$

★ $\log_b a + \log_b c = \log_b ac$

(7)

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

* $i = n \xrightarrow{/2} 1 \ (or) \ 0$

* $TC = \log_2 n$

(1) $i = n = n/2^0$

(2) $i = n/2 = n/2^1$

(3) $i = n/4 = n/2^2$

(4) $i = n/8 = n/2^3$

⋮

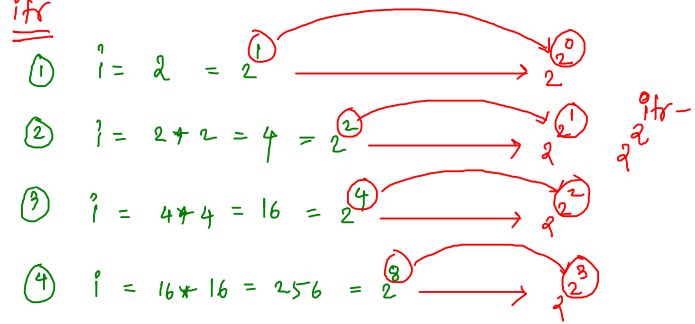
(k) $i = 1 = n/2^{k-1}$ $\Rightarrow 2^{k-1} = n \Rightarrow k = \log_2 n + 1$

Note:	
$i = 1 \xrightarrow{+K} n$ $i = n \xrightarrow{-K} 1$ Both are same n/K iteration $O(n/K) \simeq O(n)$	$i = 1 \xrightarrow{*K} n$ $i = n \xrightarrow{/K} 1$ Both are same $\log_K n + 1$ iteration $O(\log_K n)$

④

```
64  function solve8(n) {
65    let i = 2;
66    while (i < n) {
67      i = i * i;
68    }
69  }
```

itr



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

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

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

$\log_2 2^{K-1} = \log_2(\log_2 n) \Rightarrow K-1 = \log_2(\log_2 n)$

(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 $n = 4$

Total = $1 + 1 + 1 + 1$

$= 4 = n$

* Do not fall in trap by looking 2 nested loops $\rightarrow n^2$

$TC = O(n)$

(1) $i = 1$

$\rightarrow j = 1,\ 1 <= 1$ } (1)

$\rightarrow j = i + j = 1 + 1 = 2,\ 2 <= 1$ ✗

(2) $i = 2$

$\rightarrow j = 1,\ 1 <= 2$ } (1)

$\rightarrow j = i + j = 2 + 1 = 3,\ 3 <= 2$ ✗

(3) $i = 3$

$\rightarrow j = 1,\ 1 <= 3$ } (1)

$\rightarrow j = i + j = 3 + 1 = 4,\ 4 <= 3$ ✗

(4) $i = 4$

$\rightarrow j = 1,\ 1 <= 4$ } (1)

$\rightarrow j = i + j = 4 + 1 = 5,\ 5 <= 4$ ✗

(5) $i = 5,\ 5 <= 4$ ✗

(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 $n = 3$, $K = 2$

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

$\simeq n^K + n^K + n^K + \dots + n^K$

$\simeq n \times n^K \simeq n^{K+1} \Rightarrow TC = O(n^{K+1})$

$\simeq O(n^K)$

① $i = 1$

→ $p = i^K = 1^2 = 1$

→ $j = 1 \to p$

① iteration

② $i = 2$

→ $p = i^K = 2^2 = 4$

→ $j = 1 \to 4$

④ iterations

③ $i = 3$

→ $p = i^K = 3^2 = 9$

→ $j = 1 \to 9$

⑨ iterations

④ $i = 4$ ✗

★ $TC = O(n^3)$

$TC = 1 + 4 + 9$ (when $K = 2$)

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

= Sum of squares of 1^{st} n natural numbers $= \dfrac{n(n+1)(2n+1)}{6}$

$= (n^2 + n)(2n + 1) = 2n^3 + n^2 + 2n^2 + n$

(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 $O(n \log n)$ [c]

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

a) $N^{K+9} \Rightarrow N$ (Constant)

b) $5^{N*2} \Rightarrow 5^N \Rightarrow$ (Constant)^N

c) $(N/4) \log_2 (N/4000) \Rightarrow N \log_2 N$

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

e) $10N + 9(N/100) + 340N^2 \Rightarrow N^2$ (highest degree)

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

* Constant $\rightarrow O(1)$ (no loops, no functions)

horrible / need to optimize | Bad | fair | good | Best

$O(n!) > O(2^n) > O(n^2) > O(n \log n) > O(n) > O(\log n) > O(1)$

take $n = 10^8$ and verify

Space Complexity :

* If you are using any datastructures | apart from input, | In your solution
  $\Rightarrow O(\text{size of DS})$

* Mostly you will come across arrays of size 'N'
  $\Rightarrow SC = O(N)$

* otherwise, $\Rightarrow SC = O(1)$

Reverse an Array

Copy all elements into new array by iterating (R → L) (reverse)

arr: 1 2 3 4

new-arr: 4 3 2 1

SC = O(N)

TC = O(N)

Two pointer start, end swap

arr: 1 2 3 4

arr: 4 3 2 1

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;
      }
    }
  }
}
```