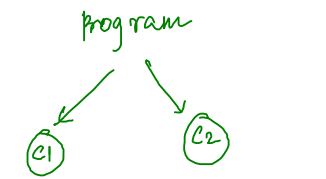
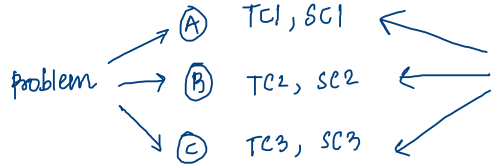
★ Time complexity :



practical Analysis

★ TC, SC are machine independent (theoritical/mathematical Analysis)



★ 1st preference is ↓tc then only go for ↓sc

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

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

TC1 = 5min SC1 = 1MB

TC2 = 2min SC2 = 5MB

TC3 = 3min SC3 = 2MB

→ TC2, SC2 is a better solution

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;
  }
}
```

Total no. of instructions

= 1 + 1 + 11 + 14

= 27 instructions

(precise / perfect calculation)

generalise,

instructions = 2 + 3m + 2 + 3n + 2

= 3m + 3n + 6

TC = O(m+n)

Let say, m = 4, n = 3

let a = 0; —(1)

let b = 0; —(1)

① let i = 0; —(1)

~~0 < 3 —(1)~~

a = a + i —(1)

= 0 + 0 = 0

i++; ⇒ i = 1 —(1)

② i < 3 —(1)

a = a + i —(1)

= 0 + 1 = 1

i++ ⇒ i = 2 —(1)

③ 2 < 3 —(1)

a = a + i = 1 + 2 = 3 —(1)

i++ ⇒ i = 3 —(1)

④ 3 < 3 —(1) ×

n = 3 (11 instructions)

m = 4 (4 * 3 + 2)

= 14 instructions

generalise,

~~2~~ + (m ~~* 3~~ + ~~2~~)

+ (n ~~* 3~~ + ~~2~~)

TC = ignore all constants

= m + n

= O(m+n)

⇒ TC is actually calculating instructions but you can follow this simple approach.

⇒ check out for loops and any function calls and calculate no. of iterations.

```
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;          — (n)
6    }
7    for (let j = 0; j < m; j++) {
8      b = b + j;          — (m)
9    }
10 }
```

→ n times

→ m times

★ If m = n

⇒ 1st — n times

⇒ 2nd — n times

n + n = 2n

⇒ O(n)

T.C = n + m

= 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 / iterations

★ TC is always in terms user input.

Q: Why we need to ignore constants and pick highest degree?

⇒ $n^2 + n$ for very large n

let say $n = 10^8$

$n^2 + n = (10^8)^2 + 10^8$

$= 10^{16} + 10^8$ → negligible

As n becomes larger ($n \to \infty$), the lowest degree terms will not have much impact on overall Time complexity.

(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)

(n*m)

O(n*m)

⇒ your program will not take more than n*m iterations/tc.

Let say, n = 4 and m = 3 (you can take any example just for your understanding)

① i = 0
→ j = 0
→ j = 1
→ j = 2
→ j = 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) ✗

no. of iterations = 3 + 3 + 3 + 3
= 4 * 3 = n

generalise, TC = O(n * m)

for every i, inner loop 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, $n = 4$

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

$= \frac{n^2+n}{2} = n^2+n \rightarrow$ pick highest degree

$= O(n^2)$

(1) $i = 0$
- $\rightarrow j = 4$
- $\rightarrow j = 3$
- $\rightarrow j = 2$
- $\rightarrow j = 1$

} 4

$\rightarrow j = 0 \ (0>0) \times$

(2) $i = 1$
- $\rightarrow j = 4$
- $\rightarrow j = 3$
- $\rightarrow j = 2$

} 3

$\rightarrow j = 1 \ (1>1) \times$

(3) $i = 2$
- $\rightarrow j = 4$
- $\rightarrow j = 3$

} 2

$\rightarrow j = 2 \ (2>2) \times$

(4) $i = 3$
- $\rightarrow j = 4$

} 1

$\rightarrow j = 3 \ (3>3) \times$

(5) $i = 4 \ (4<4) \times$

Total iterations $= 4 + 3 + 2 + 1$

generalize for any n, iterations $= n + n-1 + n-2 + \dots + 3 + 2 + 1$

$=$ Sum of 1st n natural numbers $= \frac{n(n+1)}{2}$

④

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

Let,

$n = 10$

$\Rightarrow$ 5 iterations

$n = 15$

$\Rightarrow$ 8 iterations

① $1 <= 15$

② $3 <= 15$

③ $5 <= 15$

④ $7 <= 15$

⑤ $9 <= 15$

⑥ $11 <= 15$

⑦ $13 <= 15$

⑧ $15 <= 15$

⑨ $17 <= 15$ ✗

① $1 <= 10$

② $3 <= 10$

③ $5 <= 10$

④ $7 <= 10$

⑤ $9 <= 10$

⑥ $11 <= 10$ ✗

iterations = 5

generalise to formula,

$\simeq n/2$ iterations

$TC = O(n)$

★ $i = 1 \xrightarrow{+2} n$

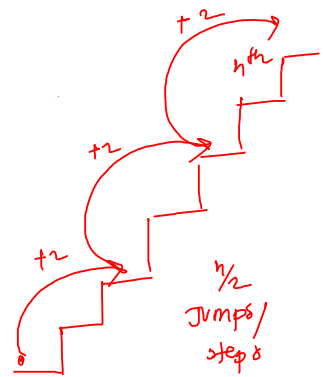
$1 \xrightarrow{+2} 3 \xrightarrow{+2} 5 \xrightarrow{+2} 7 \xrightarrow{+2} 9 \xrightarrow{+2}$ ✗ $(n = 10)$

$\Rightarrow n/2$ jumps

★ you need 50/- but you have all 2/- coins, how many 2/- coins are required to make 50/- ?

$\Rightarrow 25$ ② coins

$\Rightarrow 50/2 \Rightarrow n/2$



(5)

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

⇒ $n = n-5$

Let $n = 15$,

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

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

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 $0 > 1$ (✗)

iterations = 3

TC = $n/5 = O(n)$

* you need 50/-, you only have 5/- coins ?

⇒ $50/5 = 10$ coins

10 + (5) coins

* $n \xrightarrow{-5} 1(or)\ 0$

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

3 jumps

* Imp observation :

i = 1 → (+2, Jump) → n

= 0

iterations = $n/2$

i = 1 → (+5, Jump) → n

= 0

iterations = $n/5$

* i = 1 —(+ Jump)→ n

= 0

TC = n/Jump = O(n)

i = n → (-2, Jump) → 0 (or) 1

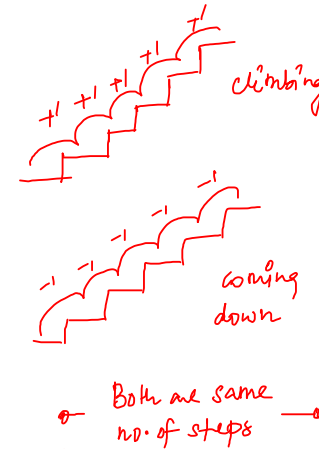
iterations = $n/2$

i = n → (-5, Jump) → 0 (or) 1

iterations = $n/5$

* i = n —(- Jump)→ 0 (or) 1

TC = n/Jump = O(n)



⑥

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

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

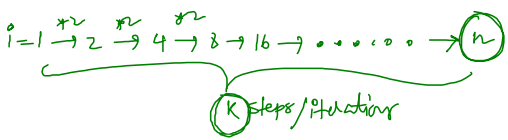
$n \rightarrow$ destination

last iteration $\rightarrow$

Let say you want reach 'n'

$\rightarrow$ how many iterations will it take to reach 'n'

Let us assume, iterations = K = TC



$\star$ TC = $O(K) = O(\log_2 n)$ (iterations)

itr ① $\Rightarrow$ $i = 1 = 2^0 = 2^{1-1}$

itr ② $\Rightarrow$ $i = 2 = 2^1 = 2^{2-1}$

itr ③ $\Rightarrow$ $i = 4 = 2^2 = 2^{3-1}$

itr ④ $\Rightarrow$ $i = 8 = 2^3 = 2^{4-1}$

itr ⑤ $\Rightarrow$ $i = 16 = 2^4 = 2^{5-1}$

$= 2^{itr-1}$

itr (K) $\Rightarrow$ $i = n = 2^{K-1}$ (last iteration)

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

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

$\rightarrow \log_2 n = K-1 \log_2 2$

$\Rightarrow \log_2 n = K-1$

$\Rightarrow K = \log_2 n + 1$

① $i = 1 = 2^0 = 2^{1-1}$

② $i = 2 = 2^1 = 2^{2-1}$

③ $i = 4 = 2^2 = 2^{3-1}$

④ $i = 8 = 2^3 = 2^{4-1}$

⑤ $i = 16 = 2^4 = 2^{5-1}$

⋮

Ⓚ $i = n = 2^{k-1}$

WKT $i = n$ at last iteration

let say last iteration = Ⓚ

$2^{k-1} = n$

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

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

$k-1 = \log_2 n$

$k = \log_2 n + 1$

$TC = O(\log_2 n)$

Basic log properties :

$2^x = 32$

$x = \log_2 32$

$\Rightarrow \log_b a^m = \frac{m}{n} \log_b a$

$\Rightarrow \log_a a = 1$

$\Rightarrow \log_b a + \log_b c = \log_b ac$

* Imp observation :

iterations = $\log_2 n$

iterations = $\log_5 n$

iterations = $\log_2 n$

iterations = $\log_5 n$

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

You have 1 Re. make n Rs using compound method ($*^{k-1}$) how many days it will take.

days	1	2	3	4	....	k^{th}
Rs	1	2	4	8	.....	n
	2^0	2^1	2^2	2^3	....	2^{k-1}

k^{th} day = n

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

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

are required to earn 'N' Rs.

(7)

```
function solve7(n) {
  let i = n;
  while (i > 0) {
    i = i / 2;
  }
}
```

$i = n \xrightarrow{/2} 0$ this is same as $i = 1 \xrightarrow{*2} n$

$TC = O(\log_2 n)$

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

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

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

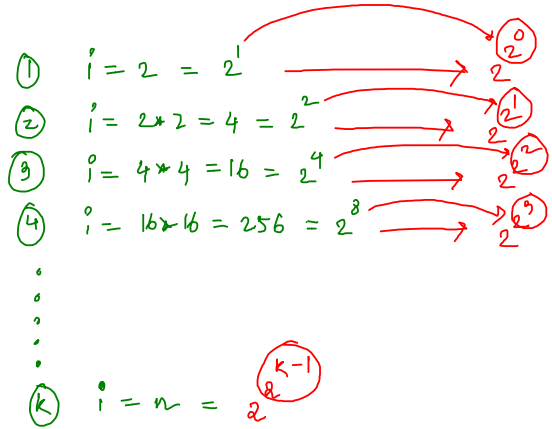
$\Rightarrow \log_2 n = \log_2 2^{k-1}$

$\Rightarrow k = \log_2 n + 1$

$TC = O(\log_2 n)$

⑧ ★Advanced TC

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



$2^{k-1} = n$

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

★ TC = $O(\log_2 \log_2 n)$

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

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

(9)

```
function solve9(n) {
  let a = 0;
  for (let i = 1; i <= n; i++) {
    for (let j = 1; j <= i; j = i + j) {
      a = a + i + j;
    }
  }
}
```

let n = 4,

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

* for every i,
→ j is running only once

① i = 1
→ j = 1, 1 <= 1 } 1
→ j = i + j = 1 + 1 = 2, 2 <= 1 ✗

② i = 2
→ j = 1, 1 <= 2 } 1
→ j = i + j = 2 + 1 = 3, 3 <= 2 ✗

③ i = 3
→ j = 1, 1 <= 3 } 1
→ j = i + j = 3 + 1 = 4, 4 <= 3 ✗

④ i = 4
→ j = 1, 1 <= 4 } 1
→ j = i + j = 4 + 1 = 5, 5 <= 4 ✗

⑤ i = 5
5 <= 4 ✗

(10)

```
80  function solve10(n) {
81    let a = 0;
82    for (let i = 1; i <= n; i++) {
83      let p = i ** k;
84      for (let j = 1; j <= p; j++) {
85        a = a + i + j;
86      }
87    }
88  }
```

→ (n, k)

Let

$n = 9,\ k = 2$

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

$\simeq n^k + n^k + n^k + \cdots\cdots + n^k$

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

→ your program will not take $> n^k$

① $i = 1$

$p = i^k = 1^2 = 1$

$j = 1 \to p = 1 \to 1$

→ ① iteration

② $i = 2$

$p = i^k = 2^2 = 4$

$j = 1 \to p = 1 \to 4$

→ ④ iterations

③ $i = 3$

$p = i^k = 3^2 = 9$

$j = 1 \to p = 1 \to 9$

→ ⑨ iterations

$TC = 1 + 4 + 9$

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

(when $k = 2$)

(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 $O(n \log_2 n)$ [c]

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

a) $N^{K+9} \rightarrow N^{\text{constant}}$

b) $5^{N*2} \rightarrow 5^n$

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

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

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

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

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

★ Constant → O(1)

(A) → n^2
(B) → $n \log n$
(C) → n

optimised

Space Complexity :

→ If you are using any data structures apart from input
⇒ O(size of ds) ↘ in your solution

→ we will mostly come across only arrays of size 'N'
⇒ SC = O(N)

→ else ⇒ SC = O(1)

Reverse an Array

↙ copy the given arr by iterating in reverse (R → L) into a new arr
SC = O(N)
TC = O(N)

↘ ✓ Inplace
Two pointer
S, e, swaps
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;
      }
    }
  }
}
```