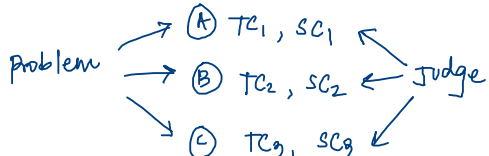
* Time complexity :




→ which solution is the best ?

min (TC1, TC2, TC3) ?respective of space

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

a. TC1 = $n^2$ iterations, SC1 = $\sqrt{n}$

b. TC2 = $\sqrt{n}$ iterations, SC2 = $n^2$ ✓

c. TC3 = n iterations, SC3 = n ✗

* Time and space are ?interms of given ?input n.


$n^2 > n > \sqrt{n}$

* Time is more important than space.

Why ? cpu is faster then program ever faster.

cpu are expensive. storage is cheaper.

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 }
```

1. Look for any loops and function calls, that is where your program spends most of the time.

2. TC = no. of instructions executed
= no. of iterations

no. of iterations = n
i = 0, 1, 2, ..... n-1, (n)^x

no. of iterations
= m
j = 0, 1, 2, ..... m-1, (m)^x

3. TC = Total no. of iterations
= n+m = O(n+m)
Big-oh

★ Big-oh; mathematical/asymptotic notation
→ It says that your program will not take more that n+m iterations / TC.
(upper bound/limit).

(2)

```
12  function solve2(n, m) {
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    }
```

(m = 3)

Total iterations = 3 + 3 + 3 + 3

= 4 * 3 = 12 iterations

generalize = n × m iterations

= O(n * m)

1. assume some example input, n = 4 and m = 3
2. Try to see how many iterations this example takes.

(n = 4)

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

(3)

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

(3)

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

(3)

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

(3)

i = 4 (i < n) X (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 iteration = 4 + 3 + 2 + 1 = 10

n = 7, 7 + 6 + 5 + 4 + 3 + 2 + 1 = 28

generalise, $n + n-1 + n-2 + n-3 + \dots + 3 + 2 + 1$

⇒ sum of 1st n Natural numbers = $\frac{n(n+1)}{2}$

★ TC = $O(n^2)$ = $\frac{n^2 + n}{2}$

Let say ⇒ n = 4

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

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

③ i = 2
→ j = 4
→ j = 3 } 2
→ j = 2 ( j > i, 2 > 2 ) ✗

④ i = 3
→ j = 4 } 1
→ j = 3 ( j > i, 3 > 3 ) ✗

⑤ i = 4 ( i < n, 4 < 4 ) ✗

★ pick highest degree term, ignore any constants

$\frac{n^2 + n}{2}$ ⇒ $n^2$

* Why to ignore constants and only pick highest degree terms ?

$$\frac{n^2 + n}{2}$$

for eg: $n = 10^8$

$$\frac{(10^8)^2 + 10^8}{2}$$

$$= \frac{10^{16} + 10^8}{2}$$

$10^8$ → negligible (less impact)

$2$ → negligible (less impact)

$$\simeq 10^{16}$$

→ you have 1 crore rupees,

10,000 / 1 lakh

⟹ negligible amount infront of 1 Cr.

(4)

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

eg: n = 20

⇒ 20/2 = 10 iterations

No. of Itr = n/2

TC = O(n)

eg: n = 10

(1) i=1, 1<=10
(2) i=3, 3<=10
(3) i=5, 5<=10
(4) i=7, 7<=10
(5) i=9, 9<=10
(6) i=11, 11<=10 X

(5) ⇒ 10/2 = 5

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

$1 \to 3 \to 5 \to \dots \to n$

⇒ O(n/2) = O(n)

---

$i = 1/0 \xrightarrow{+k} n$

$1 \to 1+k \to 1+2k \to \dots \to n$

⇒ O(n/k) = O(n)

you need 50/- and you have unlimited supply of 2/- coins and the current amount you have is 1/-. how many coins are required to make 50/-?

⇒ 25 coins * 2/- = 50/-

1/- + (25 coins) = 51/- ✓

⇒ 200/-

⇒ coins = 100 coins

⇒ 200/2 = 100

50/2 = 25

(5)

```
function solve5(n) {
  while (n > 1) {
    n = n + 20;
    n = n - 10;
    n = n - 15;
  }
}
```

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

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

* $1/0 \xrightarrow{+k} n = n/k$
* $n \xrightarrow{-k} 1/0 = n/k$


eg: $n = 30$

① $30 > 1$
$\Rightarrow n = 30+20 = 50$
$\Rightarrow n = 50-10 = 40$
$\Rightarrow n = 40-15 = 25$

② $25 > 1$
$\Rightarrow n = 25+20 = 45$
$n = 45-10 = 35$
$n = 35-15 = 20$

③ $20 > 1$
$\Rightarrow n = 20+20 = 40$
$\Rightarrow n = 40-10 = 30$
$\Rightarrow n = 30-15 = 15$

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

$\Rightarrow$ 6 iterations ($n/5$)

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

⑥

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

$\star$ $i = 1 \xrightarrow{*K} n$ (TC) $n \xrightarrow{/K} 1$

$\log_K n$ $\qquad$ $\log_K n$

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

Let say N = 128 ₹ coin

On 8th day it becomes 128.

Eg: $n = 128$ $\qquad$ $(128 < 128)$

$i = 1, 2, 4, 8, 16, 32, 64, 128$ ✗

no. of itr = 7

$\star$ on $\log_2 N + 1^{th}$ day the value = N

$\star$ TC = no. of itr = no. of days = $O(\log_2 N)$

| day | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | ..... | $K^{th}$ |
|---|---|---|---|---|---|---|---|---|---|---|
| ₹ | 1 | 2 | 4 | 8 | 16 | 32 | 64 | 128 | ..... | N |
| | $2^0$ | $2^1$ | $2^2$ | $2^3$ | $2^4$ | $2^5$ | $2^6$ | $2^7$ | | $2^{K-1}$ |

value on day $K_1$ = N

value on day K = $2^{K-1}$

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

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

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

$\Rightarrow K-1 = \log_2 N$

$\Rightarrow K = \log_2 N + 1$

1 ₹ coin triples every day

$i = 1 \xrightarrow{*2} n \Rightarrow \log_2 N$

$i = 1 \xrightarrow{*3} n \Rightarrow \log_3 N$

$3^0 \to 3^1 \to 3^2 \to \dots \to n = 3^{K-1}$ ($K^{th}$)

$3^{K-1} = N$

$\log_3 3^{K-1} = \log_3 N \Rightarrow K = \log_3 N + 1$

# Basic log propaties :

$2^x = 32$ ?

what is $x$ ? $\Rightarrow x = 5$

$2^x = 4294967296$

what is $x$ ! we use log,

$x = \log_2 4294967296$

$a^x = b \Rightarrow x = \log_a b$

$a^{\textcircled{1}x} = a \Rightarrow x = \log_2 2$

$\downarrow$ $\quad = 1$

$1$

① $\log_{b^n} a^m = \frac{m}{n} \log_b a$

tq: $\log_{2^1} 2^{k-1} = \frac{k-1}{1} \log_2 2$

② $\log_a a = 1$ tq: $\log_2 2 = 1$

③ $\log_{\textcircled{b}} a + \log_{\textcircled{b}} c = \log_b ac$

same base

(7)

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

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

$\Rightarrow \log_2 n$

① $i = n \rightarrow n/2^0$

② $i = n/2 \rightarrow n/2^1$

③ $i = n/4 \rightarrow n/2^2$

④ $i = n/8 \rightarrow n/2^3$

⋮

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

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

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

$\Rightarrow k = \log_2 n + 1$

* NOTE:

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

$i = n \xrightarrow{-k} 1$

$O(n/k)$

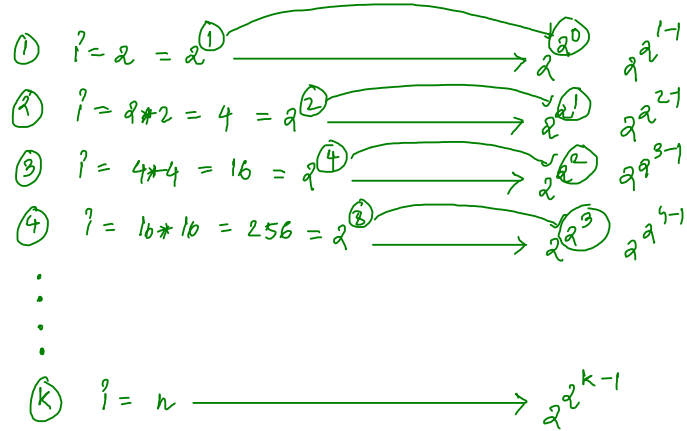
$i = 1 \xrightarrow{*k} n$

$i = n \xrightarrow{/k} 1$

$O(\log_k n)$

④

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




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

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

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

$$\Rightarrow \log_2 2^{k-1} = \log_2 (\log_2 N)$$

$$\Rightarrow k = \log_2 (\log_2 N) + 1$$

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

$k^{th} \; ? = n$

$k^{th} \; ? = 2^{2^{k-1}}$

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

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

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

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

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

$$Te = \log_2 (\log_2 n)$$

$\log_b a^{(m)}$ $(n)$

$a = 2 \quad m = 2^{k-1}$

$b = 2 \quad n = 1$

$\frac{m}{n} \log_b a$

$\Rightarrow \frac{2^{k-1}}{1} \log_2 2$

$\Rightarrow 2^{k-1}$

(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 $n = 4$

① $i = 1$
→ $j = 1$ } ①
→ $j = i + j = 1 + 1 = 2$ ($j < i$, $4 < 1$) ✗

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

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

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

⑤ $i = 5$ ($i < n$) ($5 < 4$) ✗

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

★ Do not fall in trap by looking 2 nested loops → $n^2$
(always take examples & conclude)

(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;
    }
  }
}
```

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

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

$\simeq n \cdot n^K \simeq n^{K+1}$

$*TC = O(n^{K+1})$

$\simeq O(n^K)$

$n = 3, \; k = 2$

① $i = 1$

$p = 1 ** 2 = 1^2 = 1$

$j$ runs $p = 1$ time

② $i = 2$

$p = 2 ** 2 = 2^2 = 4$

$j$ runs $p = 4$ times

③ $i = 3$

$p = 3 ** 2 = 3^2 = 9$

$j$ runs $p = 9$ times

Total $= 1 + 4 + 9$

for $N$, and $K = 2$ $= 1^2 + 2^2 + 3^2 + 4^2 + \dots + N^2$ (sum of squares of 1st N natural Numbers)

$= \dfrac{n(n+1)(2n+1)}{6}$

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

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

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

(3) Exponential ($2^n, 5^n$) [ b ]

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

(5) Log Linear - $n\log_2 n$ [ c ]

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

a) $N^{K+9} \Rightarrow (\text{Input})^{\text{constant}}$

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

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

$\Rightarrow \log_2 N$

* Constant $\Rightarrow O(1)$

(no loops, no complex functions)

horrible/need to optimise

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

okayish | fair | good | Best

# Space Complexity :

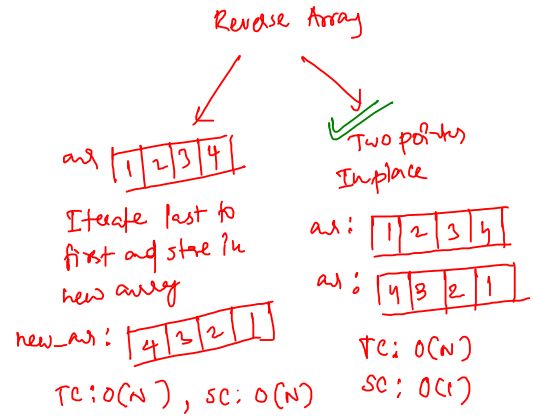
* If you are using any data structures apart from given user Input (In your solution)

$SC = O(\text{size(DS)})$

* Mostly you will use array of size N or $N^2$

$\Rightarrow SC = O(N)$ or $O(N^2)$

otherwise, $\Rightarrow SC : O(1)$



```
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;
      }
    }
  }
}
```