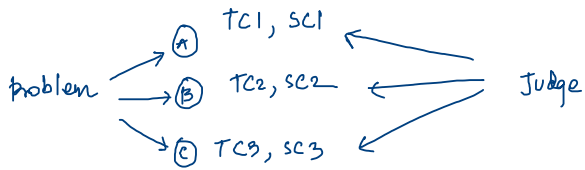
★ Time complexity :



1st pref
should be ↓TC
after that ↓SC

→ sol^n with less time (TC_1, TC_2, TC_3) min

→ If $TC_1 = TC_2 = TC_3$, (SC_1, SC_2, SC_3) min

→ sol^n should be correct (all testcases handled)

→ ~~less lines of code~~

★ no. of instructions / iterations
it is always expressed in terms
of input size (n)

Solution (program)

C1
4GB RAM
(slower)
Intel i3

C2
16GB RAM
(faster)
Intel i7

[practical analysis]

★ TC should not be machine dependent (theoritical analysis)

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)
(1)
(n)
(m)

$TC = 1 + 1 + 1 + n + n + n + 1 + m + m + m$

$= 4 + 3n + 3m$

$= O(n+m)$

$TC = 1 + 1 + n + m$

$= n + m + 2$

$= O(n+m)$

↓

Big-oh
(Asymptotic notation)

→ my program will not take more than $n+m$ time

* (Ignore all the constants, pick highest degree term)

```
Function solve0 (a, b) {
    return a + b;   — (1)
}
```

$TC = 1$

$= O(\text{constant}) = O(1)$

(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    }
```

* always look for iterations

take some example input,

n = 4

m = 3

(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

TC = 3 + 3 + 3 + 3

= 4 * 3

= n * m

for every i, m times

for every i

TC = m + m + m + m + ... + n times

= n * m

= O(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 $n = 4$

① $i = 0$
- $j = 4$
- $j = 3$
- $j = 2$
- $j = 1$
- ~~$j = 0$~~

4

② $i = 1$
- $j = 4$
- $j = 3$
- $j = 2$
- ~~$j = 1$~~

3

③ $i = 2$
- $j = 4$
- $j = 3$
- ~~$j = 2$~~

2

④ $i = 3$
- $j = 4$
- ~~$j = 3$~~

1

⑤ ~~$i = 4$~~

$$TC = 1 + 2 + 3 + 4$$

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

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

upperbound (Hard limit)

(4)

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

Let $n = 10$,

① $1 <= 10$ → $i = i + 2 = 1 + 2 = 3$

② $3 <= 10$ → $i = i + 2 = 3 + 2 = 5$

③ $5 <= 10$

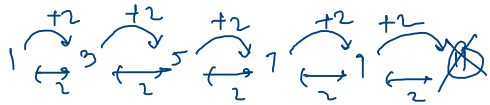
④ $7 <= 10$

⑤ $9 <= 10$

⑥ $11 <= 10$ ✗

TC = 5 iterations

$\simeq n/2 = O(n)$



50 Kgs
2 Rs coins
⇒ 50/2
⇒ 25 coins

10^8 → iteration → 1 second

10^7

1 second → 10^8 / 10^7 at max

```
for(let i=0; i<100; i++) {
  for(let j=0; j<n; j++) {
  }
}
```

⇒ 100 * n ⇒ O(n)

$n = 10^8$

⇒ $100 * 10^8$

10^{10} (TLE)

Observation =

★ $i = 1$ —Jump +2→ n

iterations = $n/2$

$i = 1$ —Jump +5→ n

iterations = $n/5$

(5)

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

* Jump -5

n ⟶ 1

n = n + 20
n = n - 10
n = n - 15

n = n - 5

Let n = 15

(1) 15 > 1

n = 15 + 20
= 35

n = n - 10
= 35 - 10
= 25

n = n - 15
= 25 - 15
= 10

(2) 10 > 1

n = 10 + 20
= 30

n = n - 10
= 30 - 10
= 20

n = n - 15
= 20 - 15
= 5

(3) 5 > 1

n = 5 + 20
= 25

n = 25 - 10
= 15

n = 15 - 15
= 0

(4) 0 > 1 ✗

iterations = n/5

TC = O(n)

⑥

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

Let n = 10

* 1 —(*2, Jump)→ n iterations = $\log_2 n$

1 —(*10)→ n iterations = $\log_{10} n$

4th iteration = k = 4

① 1 < 10 ⇒ i = i*2 = 1*2 = 2 (2^1)

② 2 < 10 ⇒ i = i*2 = 2*2 = 4 (2^2)

③ 4 < 10 ⇒ i = i*2 = 4*2 = 8 (2^3)

④ 8 < 10 ⇒ i = i*2 = 8*2 = 16 (2^4)

⑤ ~~16 < 10~~

Let iterations = k

(at k^{th} iteration, $i = 2^k$ (last iteration)) ⇒ $2^k = n$

⇒ $\log_2 2^k = \log_2 n$

⇒ $k \log_2 2 = \log_2 n$ ⇒ $k = \log_2 n$

WKT, at the last iteration i >= n (condition break)

T.C = $O(\log_2 n)$

⇒ $k - 1 = \log_2 n$

⇒ $k = \log_2 n + 1$

⇒ no. of itr = $\log_2 n + 1$

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

day	1	2	3	4	5	6	.....	K^{th} day
	1Re	2Rs	4Rs	8Rs	16Rs	32Rs	.....	2^K Re
	1Re	10Rs	100Rs	10^3	10^4			10^K Rs

you have (1Re)
you want to make (nRe)
using doubling method
how many days you will take?

Let it takes K days

$\Rightarrow K^{th}$ day $= n$ Re

$\Rightarrow 2^K = n$

$\Rightarrow \log_2 2^K = n$

$\Rightarrow K = \log_2 n$ days

$i = 1 \xrightarrow{*10} n$

K^{th} day $= n$ Re

$\Rightarrow 10^K = n$

$\Rightarrow \log_{10} 10^K = \log_{10} n$

$\Rightarrow K = \log_{10} n$

log properties :

$\log_b^a$ — b → base

* ① $\log_b a^m = m \log_b a$

② $\log_{b^n} a^m = \frac{m}{n} \log_b a$

* ③ $\log_a a = 1$

④ $\log_b a + \log_b c = \log_b ac$

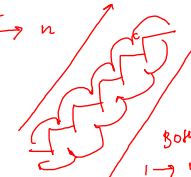
(7)

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

Time

$i = n \xrightarrow{/2} 0$ ⇔ $i = 1 \xrightarrow{*2} n$

$O(\log_2 n)$



Both $1 \to n$, $n \to 0$ (/2) are same

Let $n = 10$

① $10 > 0$ $n/2^0$ } $10/2^0$

② $5 > 0$ $n/2^1$ } $5/2 \Rightarrow 10/2^1$

③ $2 > 0$ $n/2^2$ } $2/2 \Rightarrow 10/2^2$

④ $1 > 0$ $n/2^3$ } $1/2 \Rightarrow 10/2^3$

~~⑤ $0 > 0$~~ $n/2^4$ } $1/2 \Rightarrow 10/2^n$

Let say,

iterations = k

k^{th} iteration = 1

$n/2^k = 1$

$n = 2^k$

$k = \log_2 n$

$TC = O(\log_2 n)$

WKT,

at last iteration

at k^{th} iteration,

$i = n/2^k$

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

$n \xrightarrow{/2} 0$

$1 \xrightarrow{*10} n$ } $\log_{10} n$

$n \xrightarrow{/10} 0$

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

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

$k = \log_2 n + 1$

① $\xrightarrow{*5}$ (n) $\log_5 n$

(n) $\xrightarrow{/5}$ ① $\log_5 n$

① $\xrightarrow{*243}$ (n) $\log_{243} n$

④

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

Let iterations = k

at k^{th} iteration = n (last)

WKT at k^{th} iteration $i = 2^{2^k}$

$\Leftrightarrow$

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

$2^{2^k} = n$

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

$2^k \log_2 2 = \log_2 n$

$2^k = \log_2 n$

$k \log_2 2 = \log_2 \log_2 n$

$k = \log_2 \log_2 n$

① $i = 2 = 2^1 \rightarrow 2^{2^0}$

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

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

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

⑤ $i = 256*256 = 2^{16} \rightarrow 2^{2^4}$

⋮

k $i = \rightarrow 2^{2^{k-1}} \simeq 2^{2^k}$

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 $i = 1$

→ $j = 1$

→ $j = 1 + 1 = 2$ } 1

② $i = 2$

→ $j = 1$

→ $j = 2 + 1 = 3$ } 1

③ $i = 3$

→ $j = 1$

→ $j = 3 + 1 = 4$ } 1

④ $i = 4$

→ $j = 1$

→ $j = 4 + 1 = 5$ } 1

⑤ ~~$i = 5$~~

TC = 4 iterations

= n iterations

= 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;
    }
  }
}
```

i → 2
j → K

assume,

$k = 2$

$n = 3$

① $i = 1,\ p = 1^2 = 1$

→ j = 1

~~→ j = 2~~

② $i = 2,\ p = 2^2 = 4$

→ j = 1

→ j = 2

→ j = 3

→ j = 4

~~→ j = 5~~

③ $i = 3,\ p = 3^2 = 9$

→ j = 1

→ j = 2

⋮

→ j = 9

~~→ j = 10~~

iterations = $1 + 4 + 9 = 1^2 + 2^2 + 3^2 + 4^2 + \dots + n^2$

TC = $1^k + 2^k + 3^k + \dots + n^k$ — a_1

$\simeq n^k + n^k + n^k + \dots + n^k$ — a_2

$\simeq n \cdot n^k = n^{k+1}$

$a_1 < a_2$

$= O(n^{k+1})$

$= O(n^k)$

⇒ this soln will not take more than n^k time

⇒ $1^k + 2^k + 3^k + \dots + n^k < n^{k+1}$ (accurate)

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

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

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

(4) polynomial (n^3, n^2, n^4) [a]

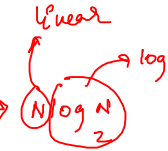
(5) Log linear $(n \log_2 n)$ [c]

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

a) N^{K+9}

b) 5^{N*2}

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



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

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

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

$\Rightarrow \log_2 N$

* linear is a special polynomial
degree = 1

* quadratic $\Rightarrow$ polynomial degree = 2

* constant $\Rightarrow O(1)$

Space Complexity :

→ If you are using any data structures (apart from given input and expected output)

⇒ O(size(ds))

arr → size n ⇒ O(N)

→ else ⇒ O(1)

rotate array

- We copied arr to temp (additional space) → O(N) (size of ds = size of arr) → TC = O(N), SC = O(N)
- ✓ inplace (3 reversals) (only original arr) → O(1) → TC = O(N), SC = O(1)

① reverse → O(N)
② rem → O(3N) = 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;
      }
    }
  }
}
```

(5) (2, 1)　　(A) (1, 2)

X (B) (1, 1)　　X (C) (1, 1)