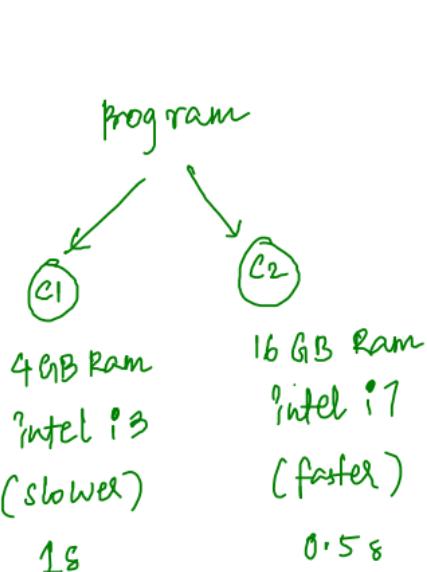


* Time Complexity :



* 1st preference is
 $\downarrow TC$ then only
go for $\downarrow SC$

— practical analysis —

* TC, SC are machine independent
(theoretical/mathematical analysis)

→ solution that has least time complexity
 $\min(TC_1, TC_2, TC_3)$ irrespective
of space.

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

$$TC_1 = 5 \text{ min} \quad SC_1 = 1 \text{ MB}$$

~~$TC_2 = 2 \text{ min} \quad SC_2 = 5 \text{ MB}$~~

$$TC_3 = 3 \text{ min} \quad SC_3 = 2 \text{ MB}$$

→ TC_2, SC_2 is a better
solution

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 }
```

Total no. of instructions

$$= 1 + 1 + 11 + 14$$

= 37 instructions

(precise/perfect calculation)

generalise,

$$\text{instructions} = 2 + 3m + 2 \\ + 3n + 2$$

$$= \beta m + \beta n + \beta$$

$$TC = O(m+n)$$

Let say, $m = 4, n = 3$

$$\text{let } a = 0; \quad -(1)$$

$$\text{let } b = 0; \quad -(1)$$

$$\textcircled{1} \quad \text{let } i = 0; \quad -(1)$$

$$\boxed{\begin{array}{l} 0 < g = (1) \\ a = a + i \\ = 0 + 0 = 0 \\ i++; \Rightarrow i = 1 - (1) \end{array}}$$

\textcircled{2}

$$\boxed{1 < 3 - (1)}$$

$$\boxed{\begin{array}{l} a = a + i \\ = 0 + 1 = 1 \\ i++; \Rightarrow i = 2 - (1) \end{array}}$$

\textcircled{3}

$$\boxed{\begin{array}{l} 2 < 9 - (1) \\ a = a + i = 1 + 2 = 3 \\ i++; \Rightarrow i = 3 - (1) \end{array}}$$

\textcircled{4}

$$3 < 3 - (1) \times$$

$m = 3$ (11 instructions)

$$m = 4 (4 * 3 + 2)$$

= 14 instructions

generalise,

$$\cancel{(m+1)} + \cancel{(1)} \\ + \cancel{(n+1)} + \cancel{(1)}$$

$+ C = \text{ignore all constants}$

$$= m + n$$

$$= O(m+n)$$

→ 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.

$$\begin{aligned} \text{TC} &= n + m \\ &= O(n+m) \end{aligned}$$

* 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.

```
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 timer

→ m timer

* If $m = n$
→ 1st - n times
→ 2nd - n times
 $n+n = 2n$
 $\Rightarrow O(n)$

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

→ $n^2 + n$ for very large n

let say $n = 10^8$

$$\begin{aligned} n^2 + n &= (10^8)^2 + 10^8 \\ &= (10^{16}) + (10^8) \rightarrow \text{negligible} \end{aligned}$$

As n becomes larger ($n \rightarrow \infty$), 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; } } → O(n*m)
17   }
18 }
```

 $\rightarrow (n, m)$ $O(n*m)$

\Rightarrow your program will not take more than $n*m$ iterations / etc.

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

① $i=0$ $\rightarrow j=0$ $\rightarrow j=1$ $\rightarrow j=2$ $\rightarrow j=3 \times$	② $i=1$ $\rightarrow j=0$ $\rightarrow j=1$ $\rightarrow j=2$ $\rightarrow j=3 \times$	③ $i=2$ $\rightarrow j=0$ $\rightarrow j=1$ $\rightarrow j=2$ $\rightarrow j=3 \times$	④ $i=3$ $\rightarrow j=0$ $\rightarrow j=1$ $\rightarrow j=2$ $\rightarrow j=3 \times$	⑤ $i=4$ $(4 < 4) \times$
--	--	--	--	-----------------------------

$$\begin{aligned} \text{no. of iterations} &= 3 + 3 + 3 + 3 \\ &= 4 * 3 = 12 \end{aligned}$$

generalise, $T(n) = O(n*m)$

for every i , inner loop runs m times
 $\Rightarrow m + m + m + \dots + (m \text{ times})$
 $\Rightarrow n * m$

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,
 $n = 4$

$$\begin{aligned}
 * TC &= \frac{n(n+1)}{2} \\
 &= \frac{n^2 + n}{\cancel{x}} = n^2 + n \rightarrow \text{pick highest degree} \\
 &= O(n^2)
 \end{aligned}$$

① $i=0$
 $\rightarrow j=4$ }
 $\rightarrow j=3$ } 4
 $\rightarrow j=2$
 $\rightarrow j=1$
 $\rightarrow j=0 (0>0) \times$

② $i=1$
 $\rightarrow j=4$ }
 $\rightarrow j=3$ } 3
 $\rightarrow j=2$
 $\rightarrow j=1 (1>1) \times$

③ $i=2$
 $\rightarrow j=4$ } 2
 $\rightarrow j=3$
 $\rightarrow j=2 (2>2) \times$

④ $i=3$
 $\rightarrow j=4$ } 1
 $\rightarrow j=3 (3>3) \times$
 $\rightarrow j=2 (2>2) \times$
 $\rightarrow j=1 (1>1) \times$

$$\text{Total iterations} = 4 + 3 + 2 + 1$$

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

$$\begin{aligned}
 &= \text{Sum of first } n \text{ natural numbers} = \frac{n(n+1)}{2}
 \end{aligned}$$

④

```

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

```

$$\textcircled{1} \quad 1 \leftarrow 10$$

$$\textcircled{2} \quad 3 \leftarrow 10$$

$$\textcircled{3} \quad 5 \leftarrow 10$$

$$\textcircled{4} \quad 7 \leftarrow 10$$

$$\textcircled{5} \quad 9 \leftarrow 10$$

$$\textcircled{6} \quad 11 \leftarrow 10 \times$$

iterations = 5

generalise to formula,

$\approx \frac{n}{2}$ iterations

$$TC = O(n)$$

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

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

$$\Rightarrow \frac{n}{2} \text{ jumps}$$

$$n = 15$$

$\Rightarrow 8$ iterations

$$\textcircled{1} \quad 1 \leftarrow 15$$

$$\textcircled{2} \quad 3 \leftarrow 15$$

$$\textcircled{3} \quad 5 \leftarrow 15$$

$$\textcircled{4} \quad 7 \leftarrow 15$$

$$\textcircled{5} \quad 9 \leftarrow 15$$

$$\textcircled{6} \quad 11 \leftarrow 15$$

$$\textcircled{7} \quad 13 \leftarrow 15$$

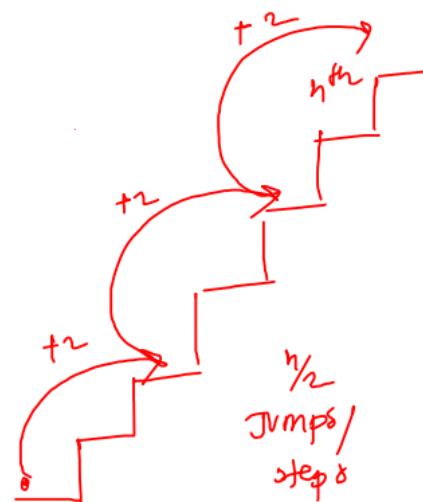
$$\textcircled{8} \quad 15 \leftarrow 15$$

$$\textcircled{9} \quad 17 \leftarrow 15 \times$$

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

$\Rightarrow 25$ $\textcircled{2}$ coins

$\Rightarrow 50/2 \Rightarrow \frac{n}{2}$



5

```

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

Let $n=15$,
 $n=n-5 \Rightarrow n=10$

$$\begin{array}{r}
 n = n + 20 \\
 n = n - 10 \\
 n = n - 15 \\
 \hline
 n = n - 5
 \end{array}$$

$$\begin{aligned}
 n &= n + 20 \\
 n &= n - 10 \\
 n &= n - 15 \\
 &= n - 5
 \end{aligned}$$

$$\textcircled{1} \quad 15 > 1$$

$$\textcircled{2} \quad 10 > 1$$

$$\textcircled{3} \quad 5 > 1$$

$$\textcircled{4} \quad 0 > 1 (\times)$$

$$\begin{aligned}
 n &= 15 + 20 \\
 &= 35
 \end{aligned}$$

$$\begin{aligned}
 n &= 10 + 20 \\
 &= 30
 \end{aligned}$$

$$\begin{aligned}
 n &= 5 + 20 \\
 &= 25
 \end{aligned}$$

$$\begin{aligned}
 n &= 35 - 10 \\
 &= 25
 \end{aligned}$$

$$\begin{aligned}
 n &= 30 - 10 \\
 &= 20
 \end{aligned}$$

$$\begin{aligned}
 n &= 25 - 10 \\
 &= 15
 \end{aligned}$$

$$\begin{aligned}
 n &= 25 - 15 \\
 &= 10 \quad (15-5)
 \end{aligned}$$

$$\begin{aligned}
 n &= 20 - 15 \\
 &= 5 \quad (10-5)
 \end{aligned}$$

$$\begin{aligned}
 n &= 15 - 15 \\
 &= 0 \quad (5-5)
 \end{aligned}$$

$$\star n \xrightarrow{-5} 10 \xrightarrow{-5} 0$$

$$\begin{array}{ccccccc}
 15 & \xrightarrow{-5} & 10 & \xrightarrow{-5} & 5 & \xrightarrow{-5} & 0 \\
 & & & & & & \\
 & & & & 3 \text{ jumps} & &
 \end{array}$$

iterations = 3

$$TC = \frac{n}{5} = O(n)$$

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

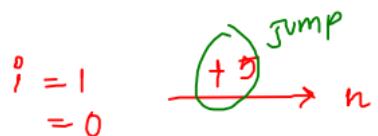
$$\Rightarrow \frac{50}{5} = 10 \text{ coins}$$

10 + 5 coins

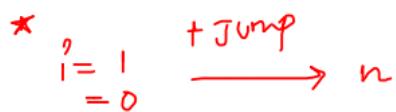
* Imp observation :



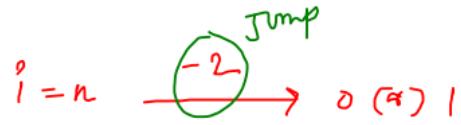
$$\text{iterations} = \frac{n}{2}$$



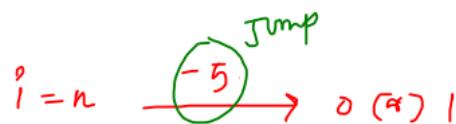
$$\text{iterations} = \frac{n}{5}$$



$$TC = \frac{n}{\text{Jump}} = O(n)$$



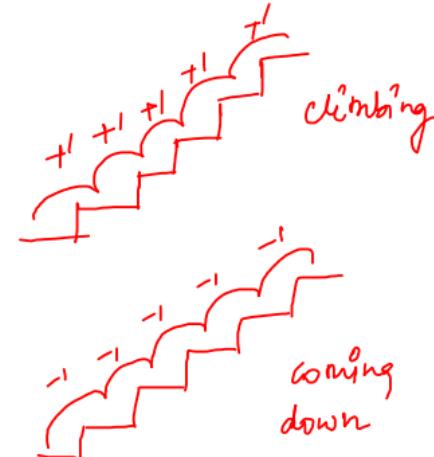
$$\text{iterations} = \frac{n}{2}$$



$$\text{iterations} = \frac{n}{5}$$



$$TC = \frac{n}{\text{Jump}} = O(n)$$



* Both are same no. of steps →

6

```

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

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

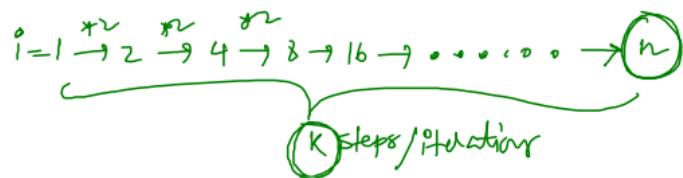
$n \rightarrow \text{destination}$

last iteration \rightarrow

Let say you want reach ' n '

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

Let us assume, iterations = $k = Tc$



$$* Tc = O(k) \xrightarrow{\text{iterations}} O(\log_2 n)$$

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

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

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

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

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

\vdots

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

(last iteration)

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

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

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

$$\Rightarrow \log_2 n = K-1$$

$$\Rightarrow K = \log_2 n + 1$$

$$\textcircled{1} \quad i = 1 = 2^0 = 2^{1-1}$$

$$\textcircled{2} \quad i = 2 = 2^1 = 2^{2-1}$$

$$\textcircled{3} \quad i = 4 = 2^2 = 2^{3-1}$$

$$\textcircled{4} \quad i = 8 = 2^3 = 2^{4-1}$$

$$\textcircled{5} \quad i = 16 = 2^4 = 2^{5-1}$$

:

:

:

:

$$\textcircled{k} \quad i = n = 2^{k-1}$$

WKT $i=n$ at last iteration

let say last iteration = \textcircled{k}

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

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

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

$$k-1 = \log_2 n$$

$$k = \log_2 n + 1$$

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

Basic log properties :

$$a^x = b^y$$

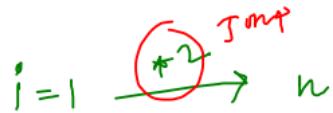
$$x = \log_b a$$

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

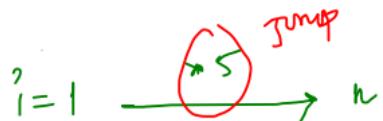
$$\Rightarrow \log_a a = 1$$

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

* Imp observation :

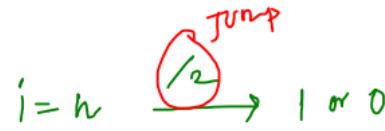


$$\text{Iterations} = \log_2^n$$



$$\text{Iterations} = \log_5^n$$

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



$$\text{Iterations} = \log_2^n$$



$$\text{Iterations} = \log_5^n$$

You have 1 Re. make n Rs using compound method ($\times 2$) how many days it will take.

$$\text{days } 1 \ 2 \ 3 \ 4 \ \dots \ k^{\text{th}}$$

$$\text{Rs } 1 \ 2 \ 4 \ 8 \ \dots \ n$$

$$2^0 \ 2^1 \ 2^2 \ 2^3 \ \dots \ 2^{k-1}$$

$$k^{\text{th}} \text{ day} = n$$

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

$$\Rightarrow k = \log_2^n + 1 \text{ days}$$

are required to earn
'N' Rs.

7

```

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

$$i = n \xrightarrow{\text{1/2}} 0 \quad \text{for } i \text{ is same as}$$

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

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

$$\textcircled{1} \quad i = n = \frac{n}{2^0}$$

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

$$\textcircled{2} \quad i = \frac{n}{2} = \frac{n}{2^1}$$

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

$$\textcircled{3} \quad i = \frac{n}{4} = \frac{n}{2^2}$$

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

$$\textcircled{4} \quad i = \frac{n}{8} = \frac{n}{2^3}$$

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

$$\vdots$$

$$\vdots$$

$$\vdots$$

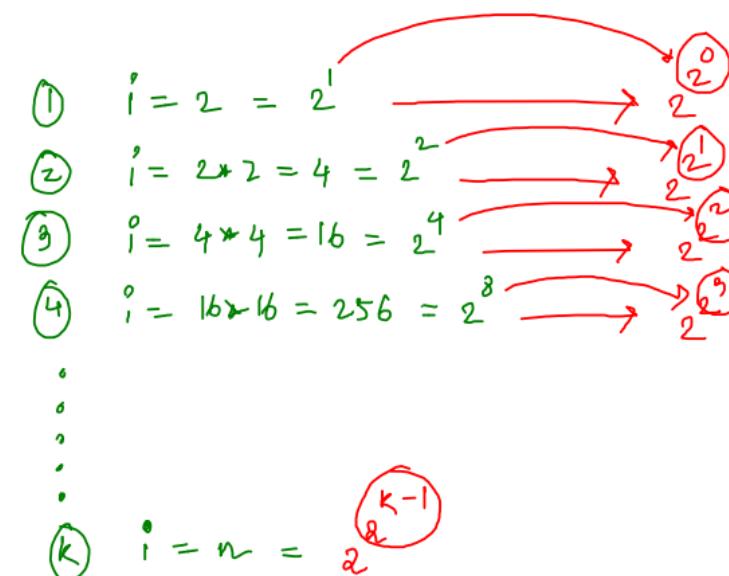
$$\textcircled{k} \quad i = 1 = \frac{n}{2^{k-1}}$$

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

*Advanced TC

④

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



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

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

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

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 }
```

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)

- | | | |
|-----------------|-----|------------------------------|
| (1) Linear | [] | a) N^{K+g} |
| (2) Logarithmic | [] | b) ζ^{N+2} |
| (3) Exponential | [] | c) $(N/4) \log_2(N/4000)$ |
| (4) Polynomial | [] | d) $3^{20} N + 10^5$ |
| (5) Log Linear | [] | e) $10N + 9(N/100) + 340N^2$ |
| (6) Quadratic | [] | f) $10^3 \log_2(N+3N)$ |

Space Complexity :

```
90  function hw1(n) {
91    for (let i = 0; i < n; i++) {
92      for (let j = 0; j < i; j++) {
93        console.log("*");
94        break;
95      }
96    }
97  }
98
99  function hw2(n) {
100  let i = 1;
101  while (i ** 2 <= n) {
102    i = i + 1;
103  }
104}
105
106 function hw3(m, n) {
107  while (m != n) {
108    if (m > n) {
109      m = m - n;
110    } else {
111      n = n - m;
112    }
113  }
114}
```

```
116  function hw4(n) {
117    let i = 1;
118    while (i < n) {
119      let j = n;
120      while (j > 0) {
121        j = parseInt(j / 2);
122      }
123      i = i * 2;
124    }
125  }
126
127  function hw5(n) {
128    for (let i = 0; i < parseInt(n / 2); i++) {
129      for (let j = 1; j < n - parseInt(n / 2); j++) {
130        let m = 1;
131        while (m <= n) {
132          m = m * 2;
133        }
134      }
135    }
136  }
```