Algorithms

↳ Sequence of finite steps to perform some specific task.

a, b

Multiplication of two numbers.

1. Take two numbers → a & b
2. Take $c = a * b$
3. Return c.

Properties

1) Terminate after finite amount of time.
2) Produce atleast one output → multiplication ✓, Division ✓
3) Independent of any programming language.
4) Unambiguous ⇒ Deterministic

$2 * 3 = 6$ $\quad$ $2 * 3 = 6$

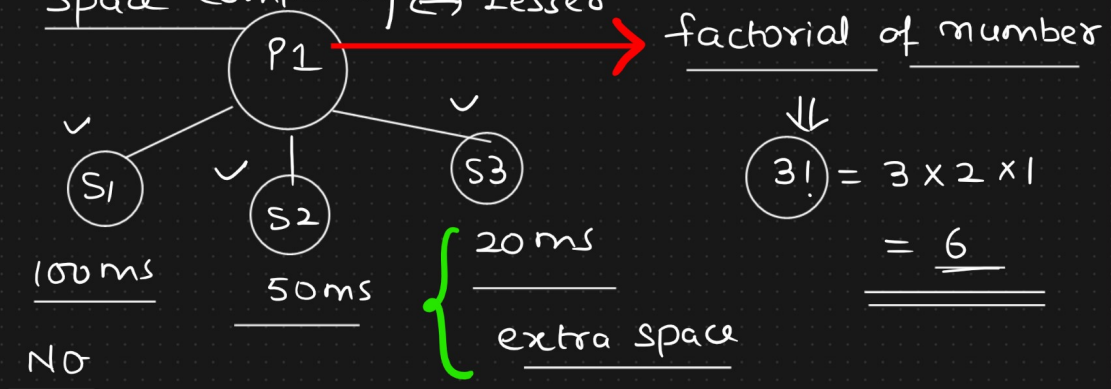
Not a valid algorithm

⇒

```
While (True):
    print("Priya Bhatia")
```

Steps construct an algorithm

1. Problem Definition
2. Design algorithm *
   - Approaches
     - Divide & conquer ←
     - Greedy Algo
     - Dynamic programming
     - & many more
3. Draw flow chart
4. Testing
   - Test cases
     - Input
     - output
5. Implementation
6. Analysis * ⇒ Time complexity ↔ Lesser
   - Space complexity ↔ Lesser



No extra space

arr ⇒ (1, 2, 3, 4, 5, 6, 7, 8, 9, 10)

→ Sum of 10 Natural Numbers

Dharun — one for loop

```
Sum = 0
for i = 0 to n-1
    Sum += arr(i)
```

⇓ correct

S1

Arvind — Sum of n natural numbers

S2

$\frac{n(n+1)}{2}$

$\frac{10 \times 11}{2} = 55$

⇓ correct

Target → lesser time & space complexity

Asymptotic Notations

Important ⇐ └ Big O → Worst case scenario
└ Omega → best case scenario
└ Theta ⇒ average case scenario

Analysis
- Apostiary Analysis → Exact results
  ↳ language of compiler & type of hardware
- Apriori Analysis → Independent
  (Nice Logic)

Big O Notation ⇒ ↳ Approximate answers

Apriori Analysis → Order of magnitude of a statement
↳ # times any statement is executing

Example 1

$(x = y + z)$ — (Constant) - c

No Loop → O(1)
↳ constant time complexity

Example 2

$x = y + z$ — ① — 1 time

```
for(i → 0 to n-1):
```

$x = y + z$ — ② → n times

$i=0$ $i=1$ - - - $i=n-1$

$x=y+z$ $x=y+z$ ... $x=y+z$ } n times

$O(n)$

$(n+1)$ times $\Rightarrow O(n)$

Example 3

```
x = y + z                  — ① — 1
for i → 0 to n-1:
    x = y + z              — ② — n times

for i → 0 to n-1:
    for j → 0 to n-1:
        x = y + z          — ③ — n^2 times
```

$n^2 + n + 1$

$= O(n^2)$

$i=0$

$j=0$ $j=1$ - - - - - - $j=n-1$ — n

$x=y+z$ $x=y+z$ $x=y+z$

$i=1$

$j=0$ $j=1$ — - - — $j=n-1$ — n

$x=y+z$ $x=y+z$ $x=y+z$

$i=n-1$

$n = 3$ → 9 times

$i < n$

$j < n$

n^2 times

$i = 0$

$j = 0$ $j = 1$ $j = 2$

$x = y + z$ $x = y + z$ $x = y + z$

$i = 1$

$j = 0$ $j = 1$ $j = 2$

$x = y + z$ $x = y + z$ $x = y + z$

$i = 2$

$j = 0$ $j = 1$ $j = 2$

$x = y + z$ $x = y + z$ $x = y + z$

Example 4

$n = 5$

```
i = n
while (i > 1):
    i = i - 1
    print("Priya")  ——①
```

$n = 5$

$i = 5$ $i = 4$

$i = 4$ $i = 3$

Priya Priya

$i = 3$ $i = 2$

$i = 2$ $i = 1$

Priya Priya

4 times

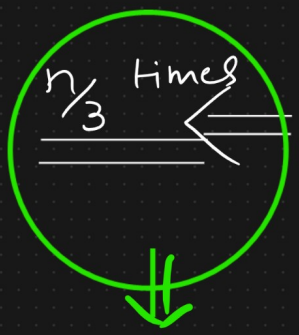
$i = 1$ ✗

$n - (n-1)$ times

$O(n)$

Example 5

```
i = n
While (i > 1):
    i = i - 3
    print("Pradeep")
```



$O(n)$

$n = 9 \quad - \quad 9/3 = 3$

$n = 9$	$i = 6$	$i = 3$
$i = 6$	$i = 3$	$i = 0$
Pradeep	Pradeep	Pradeep

$10/3 = 3.33$

- lower bound = 3
- upper bound = 4

$n = 5$ → Upper Bound $5/3 = 2$

⇓

Before decimal

Note :-

1. Time complexity is loop only
2. Not only loop, bigger loop
3. No loop at all – $O(1)$ ⇒ constant

Example 6

```
i = 1
While (i < n):
    i = 2 * i
    print(i)
```

$n = 64$

$O(\log_2 n)$

$i = 2 * i \longleftrightarrow i = 3 * i$

⇓

$O(\log_3 n)$

$O(\log n)$

$n = 64$

$\log_2 64$

$\log_2 2^6$

$6 \log_2 2$

1

$\Rightarrow 6$

$1 < 64$; $i = 2$; 2 ✓

$2 < 64$; $i = 4$; 4 ✓

$4 < 64$; $i = 8$; 8 ✓

$8 < 64$; $i = 16$; 16 ✓

$16 < 64$; $i = 32$; 32 ✓

$32 < 64$; $i = 64$; 64 ✓

$64 < 64$ — X

Example 7

```
i = n
While i > 2 :
    i = i^(1/2)
```

$n = 256$

$i = 256$

$256 > 2$

$i = (256)^{1/2}$

$i = 16$

$16 > 2$

$i = (16)^{1/2}$

$i = 4$

$4 > 2$

$i = (4)^{1/2}$

$i = 2$

$2 > 2$ * Stop

$n = 256$ — 3 times

$n^{1/2^k} = 2$ — Stopping Criteria

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

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

$\log_2 n = 2^k$

$\log_2 (\log_2 n) = k \log_2 2$

$k = \log_2(\log_2 n)$

$O(\log(\log n))$

$(256)^{1/2}$

$(256)^{1/2^2}$

$(256)^{1/2^3} = 2 \Leftrightarrow$ stop / terminate

$\log_2 n = 2^k$

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

$\log_2(\log_2 n) = k \log_2 2$ → $O(\log(\log n))$

$k = \log_2(\log_2 n)$

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

Apriori Analysis

1)
```
i = n
while i > 2 :
    i = i^(1/25)
    print(i)
```
⇒

2)
```
i = 29
while i < n :
    i = i^23
```
⇒

3)
```
i = 1
while i < n :
    i = 2 * i
    i = 3 * i
```
⇒