Algorithms

Sequence of finite steps to perform some specific task.

a, b It is multiplication of two number.

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

Properties of an Algorithms.

1- Terminate after finite amount of time.
2- Produce atleast one output.
3- Independent of any programming language.
4- It should be unambiguous → Deterministic.

Unambiguous mean.

~~today~~ ~~tomorrow~~

2 * 3 = 6 ~~2 * 3 = 6~~

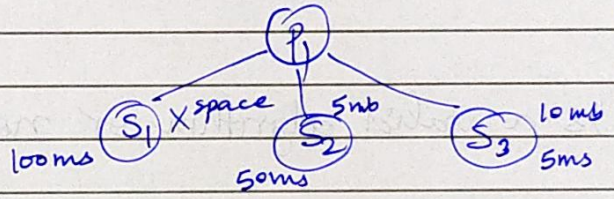
Question- Is a valid algrithm or not

```
while (True):
    print ("Narenda Yadav")
```

Ans- Not a valid algorithm because it is violating properties no 1. It will always true and keep on printing my name.

Steps to construct an algorithms.

1- Problem defination
2- Design algorithms
there are so many ways to design an algorithm like, Divide & Conquer, Greedy Algo, Dynamic programming & many more.
3- Draw the flow chart
4- testing the algorithms.
test your algorithms with various test cases for a particular input there should be a fixed input.
5- Implementation
6- Analysis
for solving a problem there are several ways so out of all solution which one is most correct.



which solution is best

with the help of analysis we will find the most optmised code.
In Analysis we have two component
1- Time complexity
2- Space complexity.

Time complexity - How much time required to execute the code.
Space complexity - How much extra space we are using to execute to code.

Question- Which one more preferable less time or less space?

Ans- there is a relation between time and space in this real world algorithms more space requried less time. And these the time complexity is more important for instant result and we have resource to increase the requried space. So time complexity minimum should always prefrable.

Example question- find the sum of 1st 10 natural number.

arr = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

Sol 1

one for loop

```
sum = 0
for i = 0 to n-1
    sum = sum + arr[i]
print(sum)
```

⇐ Correct ans-

Sol 2

using formula sum of n natural numbers

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

⇒ correct answer-

target ⇒ lesser time and space complexity. but time will be on higher priority.

time complexity and space complexity will be represented in terms of Asymptotic Notations.

Asymptotic Notations- there are three notation to represent the time and space complexity.

Big O → worst case scenario
Omega → best case scenario
theta → average case scenario.

Always trying to worry about worst case scenario because if your code performing well in worst case scenario then it always should be perform good in average and best case scenario.

There are two type of Analysis

1- Apostiary Analysis
↳ It's depend on language of compiler & type of hardware, we worry about exact result in case of Apostiary Analysis

2- Apriori Analysis
↳ It's idependent of both language and type of hardware. It is totally depend on the logic whic we will give. It's always take a bout approximate answer.
Using the Big O Notation we will give the approx answer of our code.

Question- Why do we focus on Apriori Analysis rather than Apostiary Analysis?
Ans- Becaus not everyone having mony to buy good harware but every having brain to built best logic.

Apriori Analysis — Order of magnitude of a statement.
It's mean has many number of time a statement is going to reapet.

Ex-

$x = y + z$ — Constant time (c)

$\Downarrow$

$O(1)$ constant time complexity.

Ex-

$x = y + z$ ——① 1 time

for (i → 0 to n-1):

$x = y + z$ ——② n times

$\Rightarrow (n+1)$ times

$(n+1) \Rightarrow O(n)$ time complexity

Always take higher cofficient value as time complixity beacuse we are taking about appriximation not for exact answer.

Ex- If time complixity is

$(n^2 + n + 1) \Rightarrow$ time complixity $\Rightarrow O(n^2)$

Ex-

$x = y + z$ ——① 1 times

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

$n^2 + n + 1 \Rightarrow O(n^2)$ time complexity.

Ex-

```
i = n
while (i > 1):
    i = i - 1
    print ("Narendra Yadav")      ---- n
```

⇒ O(n) is the time complexity.

Ex-

```
i = n
while (i > 1):
    i = i - 3
    print ("Narendra")
```

In this case $n/3$ times so ⇒ O(n) is the time complexity.

Ex-

```
i = n
while ( i > 50):
    print ("Narendra")
```

In this case also time will be O(n).

Note:

Time complexity is loop only.

It's mean to avoid the higher time complexity avoid the loop in code. Avoiding loop mean try to avoid highe order loop becaus we did not consider low order cofficient when calculating the time complexity.

If there is no loop then the time complixity will be constant. O(1) ⇒ C

Ex-

```
i = 1                    n = 64
while (i < n):
    i = 2 * i
    print (i)
```

$i < n$

$i = 1$ & $n = 64$

① $1 < 64$ → $i = 2$ → 2

② $2 < 64$ → $i = 4$ → 4

③ $4 < 64$ → $i = 8$ → 8

④ $8 < 64$ → $i = 16$ → $i = 16$ print

⑤ $16 < 64$ → $i = 32$ → 32

⑥ $32 < 64$ → $i = 64$ → 64

$64 < 64$ ✗

$\log_2 64$

$\log_2 2^6 \Rightarrow 6 \log_2 2 \Rightarrow 6$

So time complexity will be $O(\log_2 n)$

If you chang the statement

```
i = 3 * i        ⇒  O(log3 n)
print (i)
```

So we are not calculating exact answer then we can say

$O(\log n)$

Ex-

```
i = 1                    n = 64
while (i < n):
    i = 2 * i
    print (i)
```

$i < n$

$i = 1$ & $n = 64$

① $1 < 64$ ② $2 < 64$ ③ $4 < 64$

$i = 2$ $i = 4$ $i = 8$

2 4 8

④ $8 < 64$ ⑤ $16 < 64$ ⑥ $32 < 64$

$i = 16$ $i = 32$ $i = 64$

$i = 16$ print 32 64

$64 < 64$ X

$\log_2 64$

$\log_2 2^6 \Rightarrow 6 \log_2 2 \Rightarrow 6$

So time complexity will be $O(\log_2 n)$

If you chang the statement

```
i = 3 * i        ⇒   O(log3 n)
print (i)
```

So we are not calculating exact answer then we can say

$O(\log n)$

Ex- $i = n$

while $i > 2$:

$i = i^{1/2}$

Let's consider $n = 256$

So, $i = 256$

① $256 > 2$

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

$i = 16$

② $16 > 2$

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

$i = 4$

③ $4 > 2$

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

$i = 2$

$2 > 2$ ✗ stop

$256 \longrightarrow$ 3 times

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

$(256)^{1/2}$

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

$(256)^{1/2^3} = 2$

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

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

$\log_2 n = 2^k$

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

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

$O(\log(\log n))$ is the time complexity.

Ex-

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

$i = n$

$n^{1/25}$

$n^{1/25^2}$

$n^{1/25^3}$

⋮ k times to stop the loop

⋮

$n^{1/25^k} = 2$

$n^{1/25^k} = 2$

$\log_2 n^{1/25^k} = \log_2 2$

$\frac{1}{25^k} \log_2 n = 1$

$\log_2 n = 25^k$

$\log_{25}(\log_2 n) = k \log_{25} 25$

$\Rightarrow k = \log_{25}(\log_2 n)$

$\Rightarrow O(\log(\log n))$ is the time complexity.

Ex-

```
i = 29
while i < n:
    i = i^23
```

$i = 29$ so, $i = (29)^{23}$

$(29)^{23^2}$

$(29)^{23^3}$

⋮

$(29)^{23^k} = n$

$\log_n (29)^{23^k} = \log_n n$

$\log_n (29)^{23^k} = 1$

$23^k \log_n (29) = 1$

$\log_n (29) = \frac{1}{23^k}$

taking log again both side

$k = \log_{23} (\log_{29} n)$

$\Rightarrow O(\log(\log n))$ is the time complexity.

Ex- $i = 1$

while $i < n$:

$i = 2 * i$

$i = 3 * i$

$i = 2*i$
$i = 3*i$
$\Rightarrow i = i*6$ or $i = 6*i$

take $n = 40$

$1 < 40$	$6 < 40$	$36 < 40$
$i = 6*1$	$i = 6*6$	$i = 36*6$
$= 6$	$= 36$	$i = 216$

$i = 216$

$216 < 40$ X stop. $\Rightarrow \log_6 n$

$\Rightarrow O(\log n)$ is the time complexity.