Data Structure & Algorithms

2 - Months — Analysis

Data Structure → way to Structure/organize the data

Linear

1. Array
2. Tuple
3. Linked List
4. Stack
5. Queue
6. Dictionary
7. Set

Non-Linear

1. Tree
2. Graph
3. Trie

Algorithms

↓

sequence of **finite steps** to solve any particular problem

Algorithm

Multiplication of two numbers

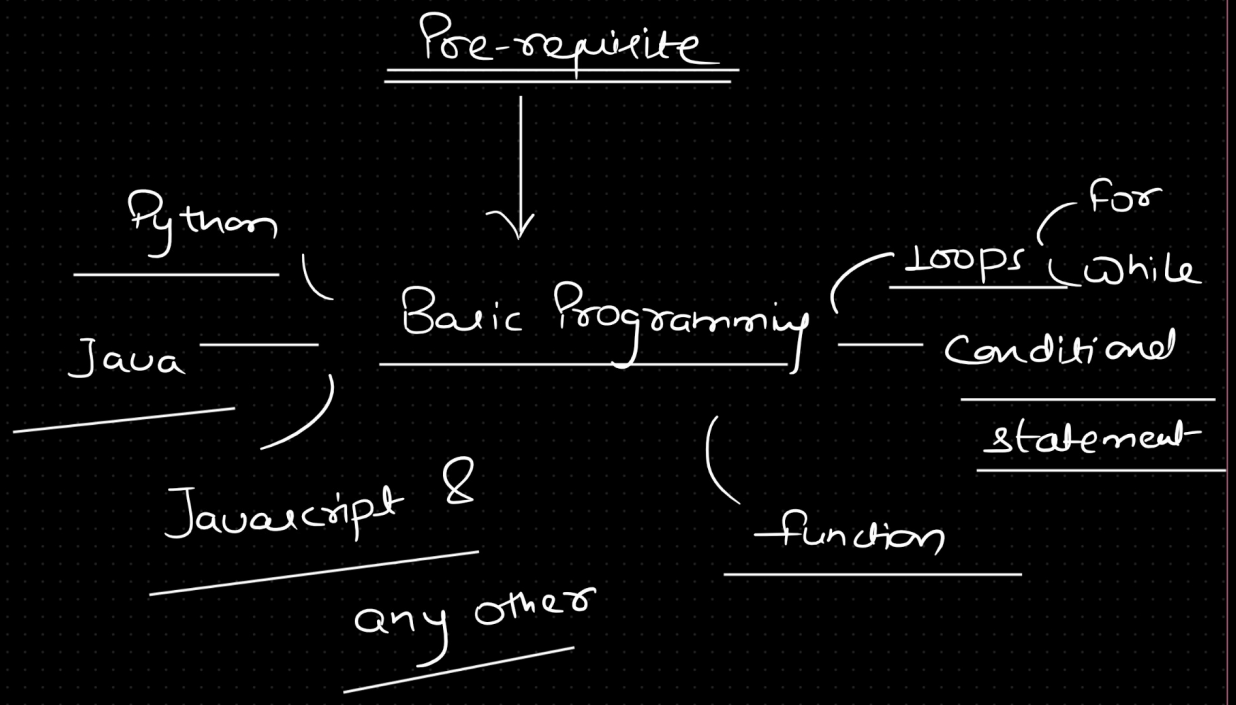
1. Take two num's (a & b)
2. Take third num (c ← store)
3. $c = a * b$ (2 * 3)

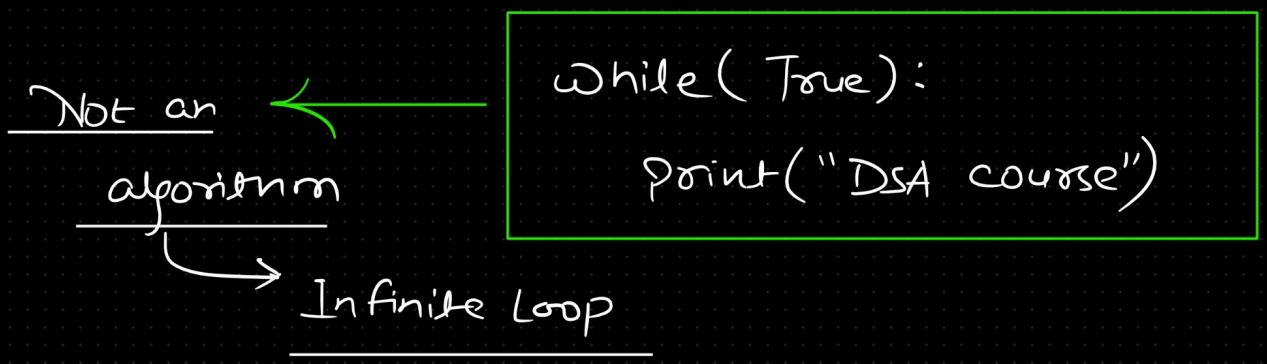
→ return c

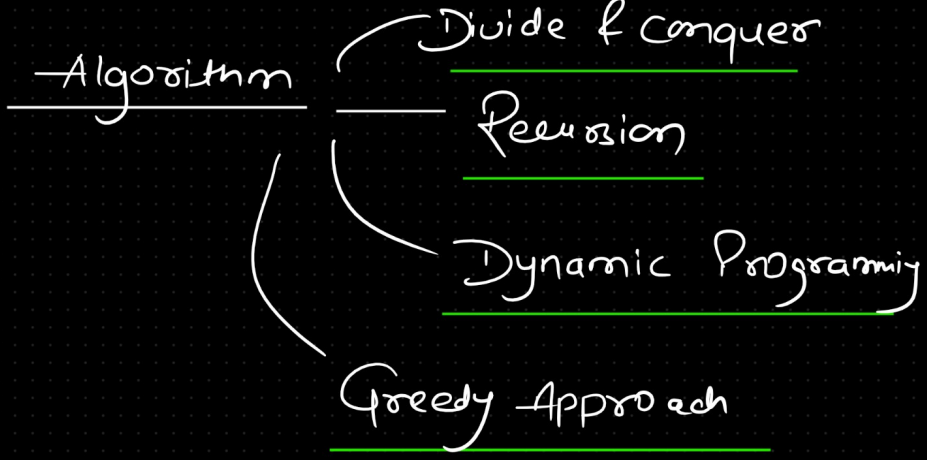
Properties of algorithms

1. It should produce atleast one output.
2. It should terminate after some point of time.
3. It should be unambiguous (Deterministic)
   - ↳ at every point of time, similar results we will be able to achieve
4. It should have finite steps

(5) Language Independent

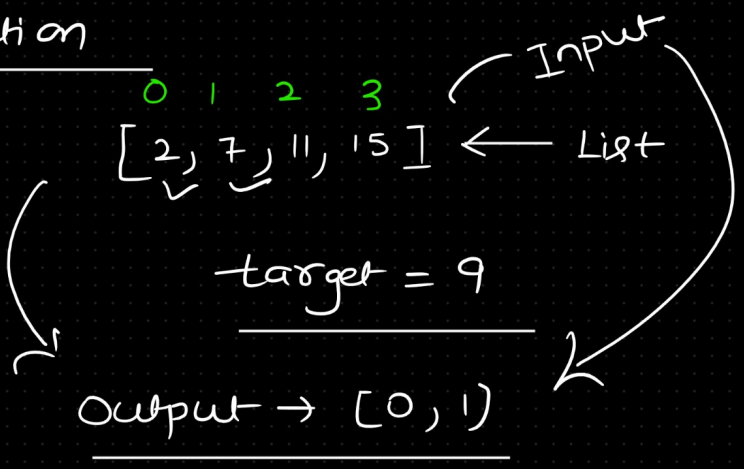






Steps required to construct an algorithm

1. Problem Definition



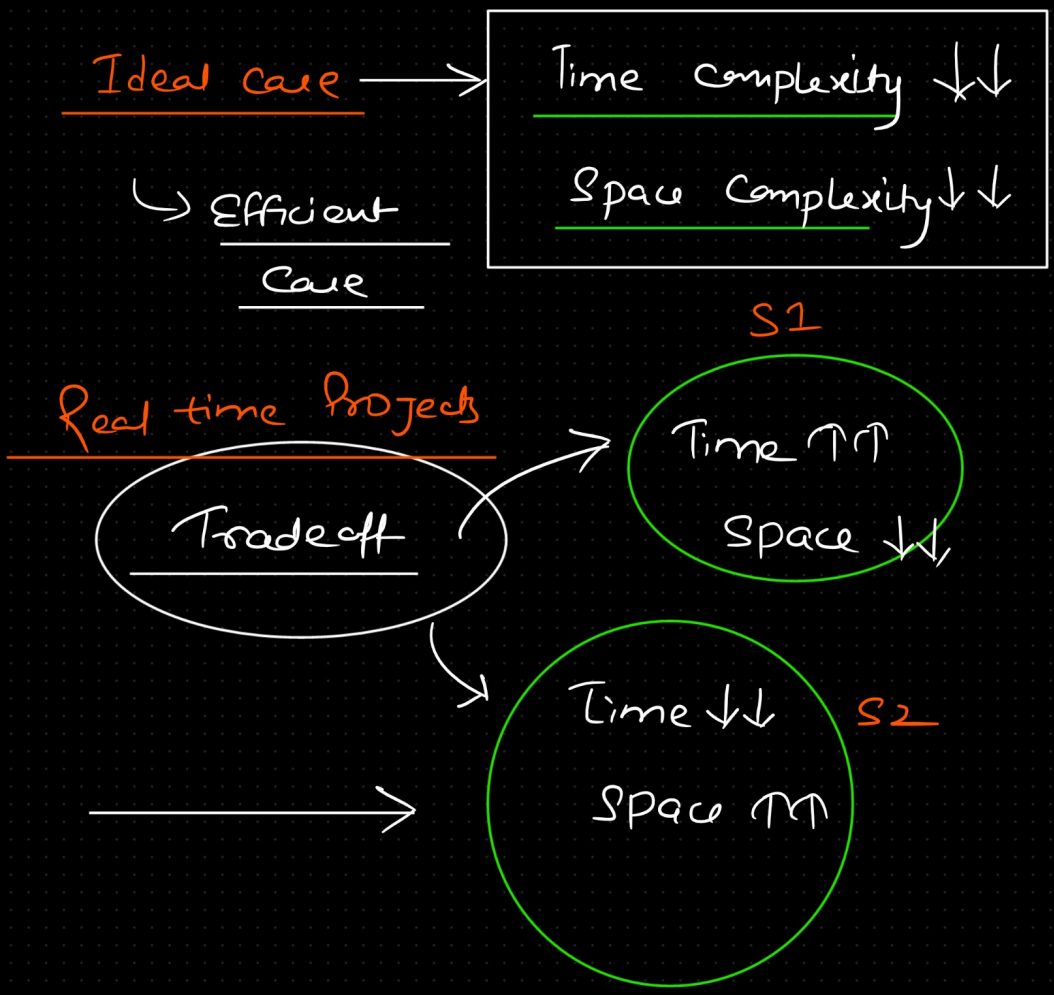
2. Design an algorithm
   - D & C
   - DP
   - (Logic)
   - Recursion & many more

3. Test (Available test Cases)

4. Implementation

5. Analysis
   - time Complexity ↓↓
   - Space Complexity ↓↓

Efficient algorithm



DSA

Book → Cormen (E-book)

Standard Lectures → MIT Lectures

Analysis

Apostiary Analysis

↳ Dependent on hardware machine

(Super computers)

↳ Exact numbers

Apriori Analysis

↳ Independent on hardware machine

(Quality Logic)

↳ Approximation

Time & Space

Example 1

main() y = 3, z = 2

```
x = y+z
```

(1)

Big O → Worst case scenario

time complexity = O(1)

Constant

Example 2

```
main()
x = y+2
```

(1)

```
for( i in range(n)):
    x = y+2
```

(4)

n = 4 → m = 10

10

i = 0	i = 1	i = 2	i = 3
x = y+2	x = y+2	x = y+2	x = y+2

time complexity = O(1) + O(n)

= O(n)

Linear Time Complexity

Notation

- Best case — Ω (omega)
- Average Case — Theta (Θ)
- Worst case — Big O (O) — n is very large

Example 3

```
main()
    x = y + 2                          ①
    for (i = 0 to n-1):                n
        x = y + 2
```

① $O(n^2)$

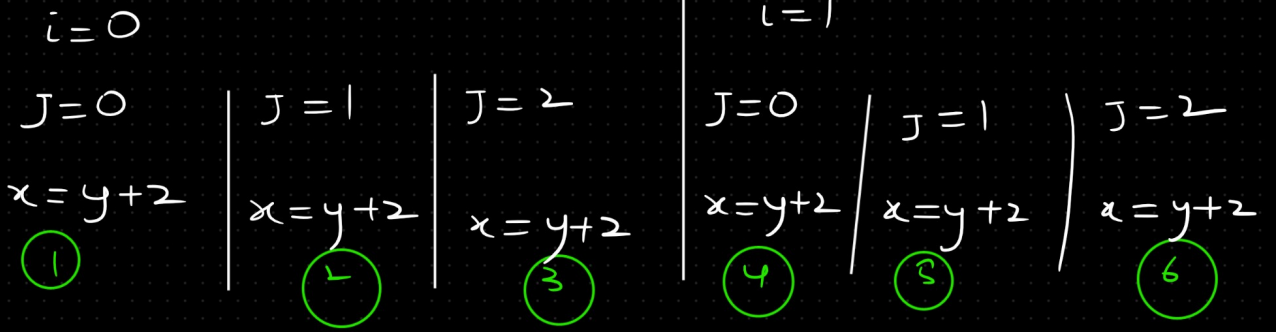
② $O(n) + O(n^2)$ ✓

③ $O(1) + O(n) + O(n^2)$ ✓

④ All of the above ✓

```
for (i = 0 to n-1):                    n^2
    for (j = 0 to n-1):
        x = y + 2
```

n = 3 → $3^2 = 9$



i=2

J=0	J=1	J=2
(1) x=y+2	(2) x=y+2	(3) x=y+2

Nested Loop → multiplication (true)

individual Loops → add

```
for(i=0 to n-1):
    for(j=0 to n+1):
        a = b+c
        d = y+2
        c = a+x
```

Constant → O(1)

O(n)

O(n²)

O(n) + O(m)

O(n) + O(m) ✓

* O(n+m)

```
for(i=0 to n-1):        } O(n)
    x = y+2

for(i=0 to m-1):        } O(m)
    x = y+2
```

MergeProcedure

MergeSort

```
for(i=0 to n+m-1):
    print(Hi)
```

n = 3	m = 3

```
for(i=0 to n):
    for(j=0 to m):
        print(Hi)
```

$O(n*m)$

Note:

Iterative code → Loops

1. Time complexity → loop
2. Higher loop

Sum of natural numbers

n = user

$0+1+2+3+4+\cdots+n$

S1

```
for(i=0 to n):
    Sum += i
return Sum
```

Time Complexity = $O(n)$

n = 20

S2

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

$\frac{20\times21}{2} = 210$

```
return Sum
```

Time Complexity = $O(1)$