③ Arrays :

stock = [298, 305, 320, 301, 292]

A = stock [2]

↳ Accessing via index

What happened when we Accessed ?

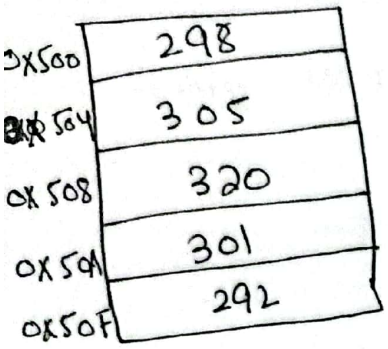
→ pointer is at beggining

stock [0] → 0x500

stock [2] → 0x500 + 2 * sizeof(int)

index ↑ ↓ 4

In Memory:



stock[2] → 0x500 + 2 * 4

stock[2] → 0x508

↳ Gives us the address.

Big O Notation in above case :

↳ So time complexity for looking at index will be :

⇒ O(1)

⇒ Consider a Scenario where we want to find index by giving a number.

```
for i in range (len (stock)):
    if stock [i] = 301:
        return i
```

↳ so if we have n element it will go through them to find it

⇒ O(n)

Look up by value Big O Notation :

Scenario 3: Print all prices

↳ we will traverse in whole array ca will print every value

→ for prices in stock:
    print(prices)

↳ so for n entries: For loop will run n times

Array Traversal → O(n)

↳ complexity is order of n.

Scenario 4: Insert new price 284 at index 1

⇒ stock.insert(1, 284)

↳ so after inserting 284 at 2nd location it will shift all elements one position a head so:

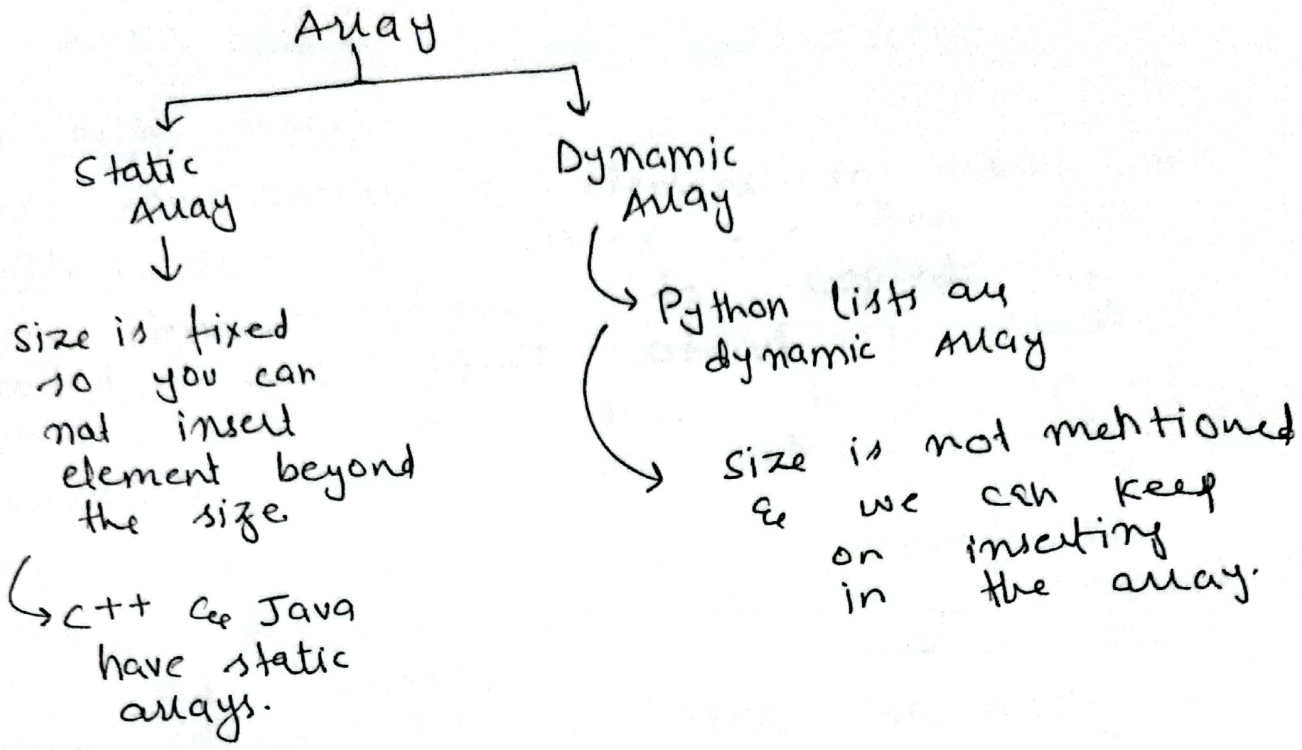
Array Insertion → O(n)

Scenario 5: Delete element at index 1

⇒ stock.remove(1)

↳ so after removing it will shift lower numbers up by 1 so:

Array deletion → O(n)



Dynamic Array

So basically when we make a dynamic array CPU gives us the limited size of (e.g 10) initially. After filling all the space & if we enter 11th element it will copy all the elements and will paste in new location that can be double of first. (So this process keeps on happening).

This is known as Geometric Propagation.