int[] arr = new int[5];

= { 2, 18, 18, 12 }

datatype

ref variable

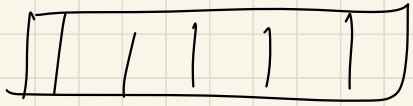
Compile time

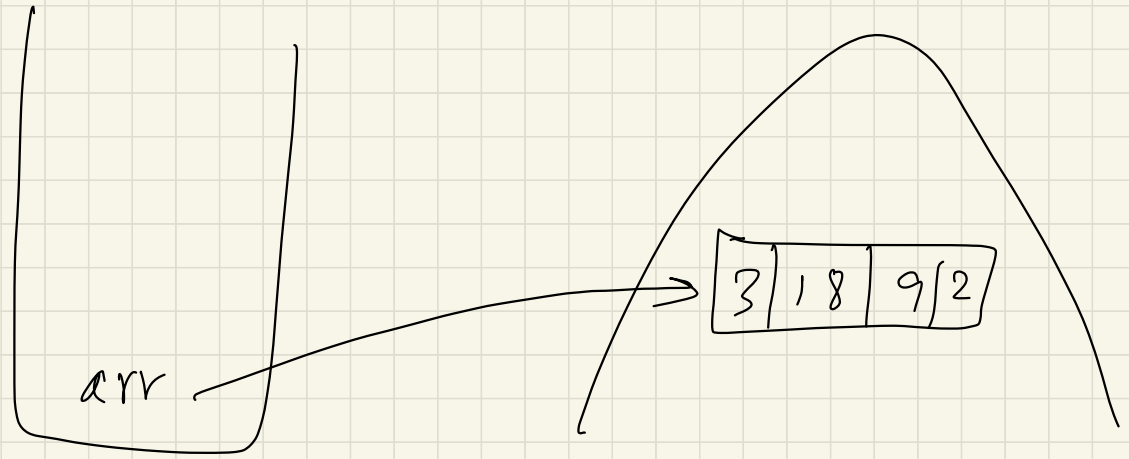
creating the object in heap memory.

⇒ Runtime

(Dynamic memory allocation)

new is used to create an object.





Stack

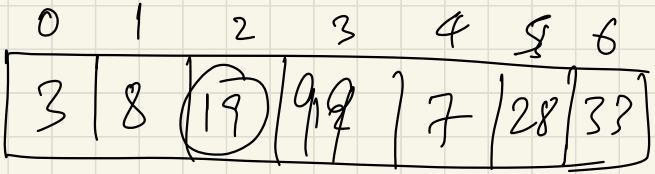
Heap

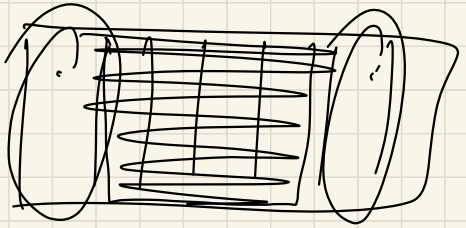
① array objects are in heap

② heap objects are, not continuos

③ DMA

Hence! may not be continuos → Depends on JVM





arr

print(arr[0]) ⟹ 3

arr[2] ⟹ 19

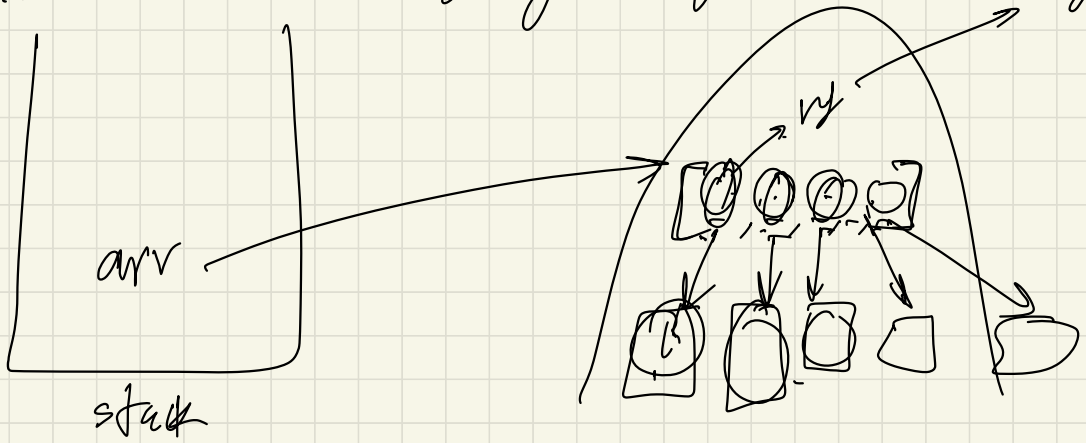
arr[3] = 99

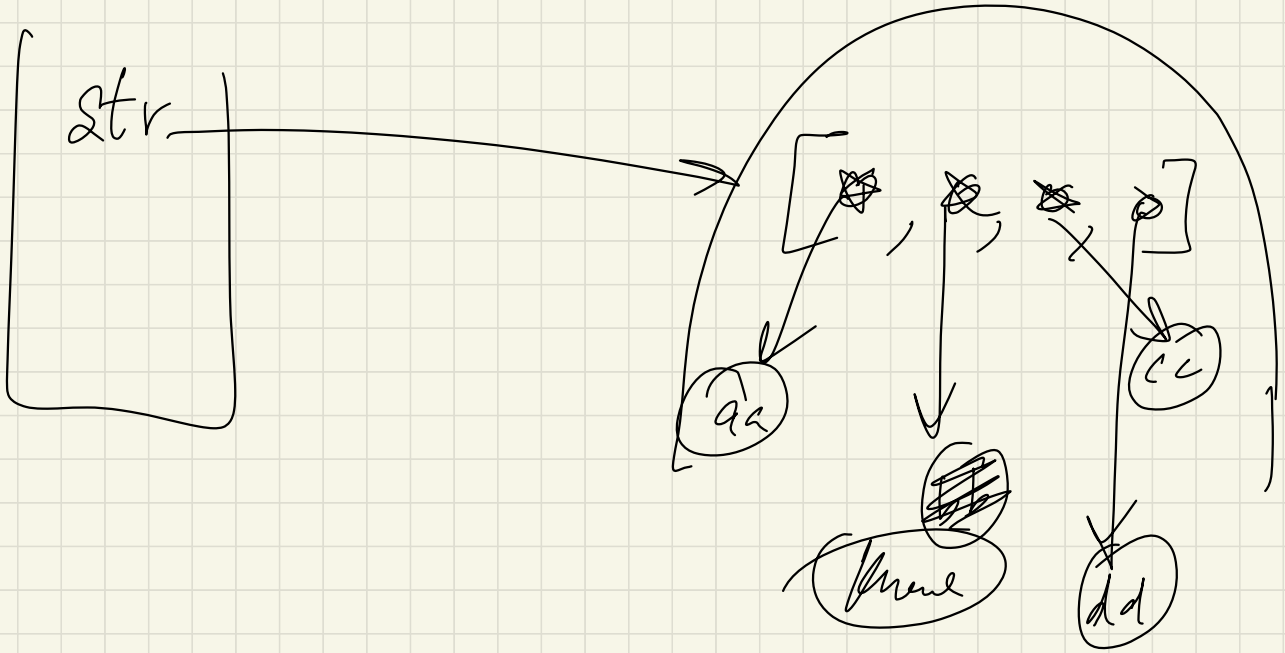
String [] arr = new String [5];

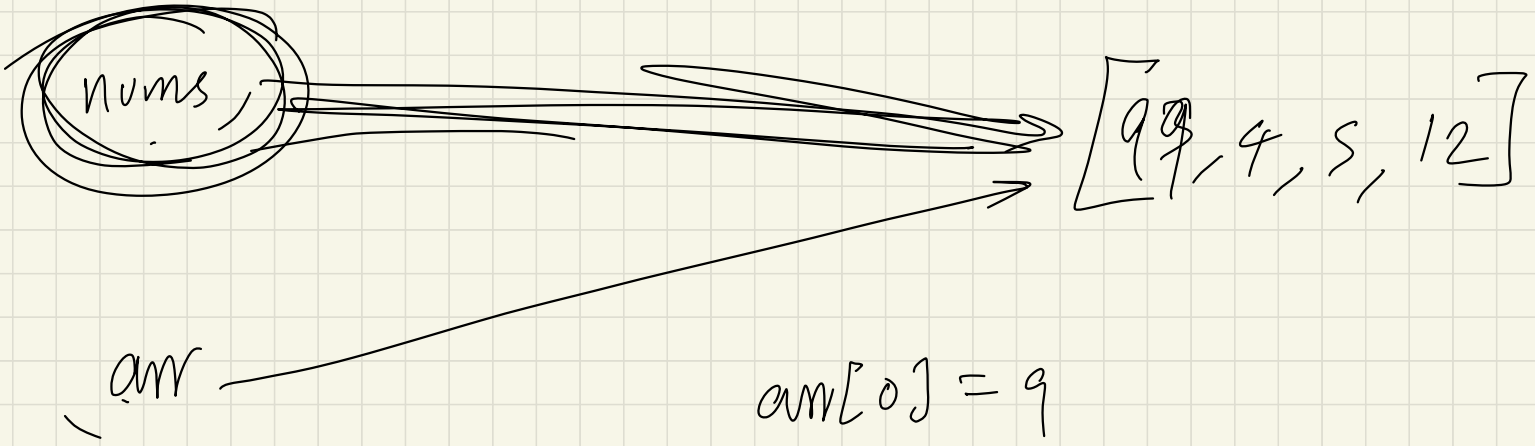
arr[0] = "S"

// internal working of object

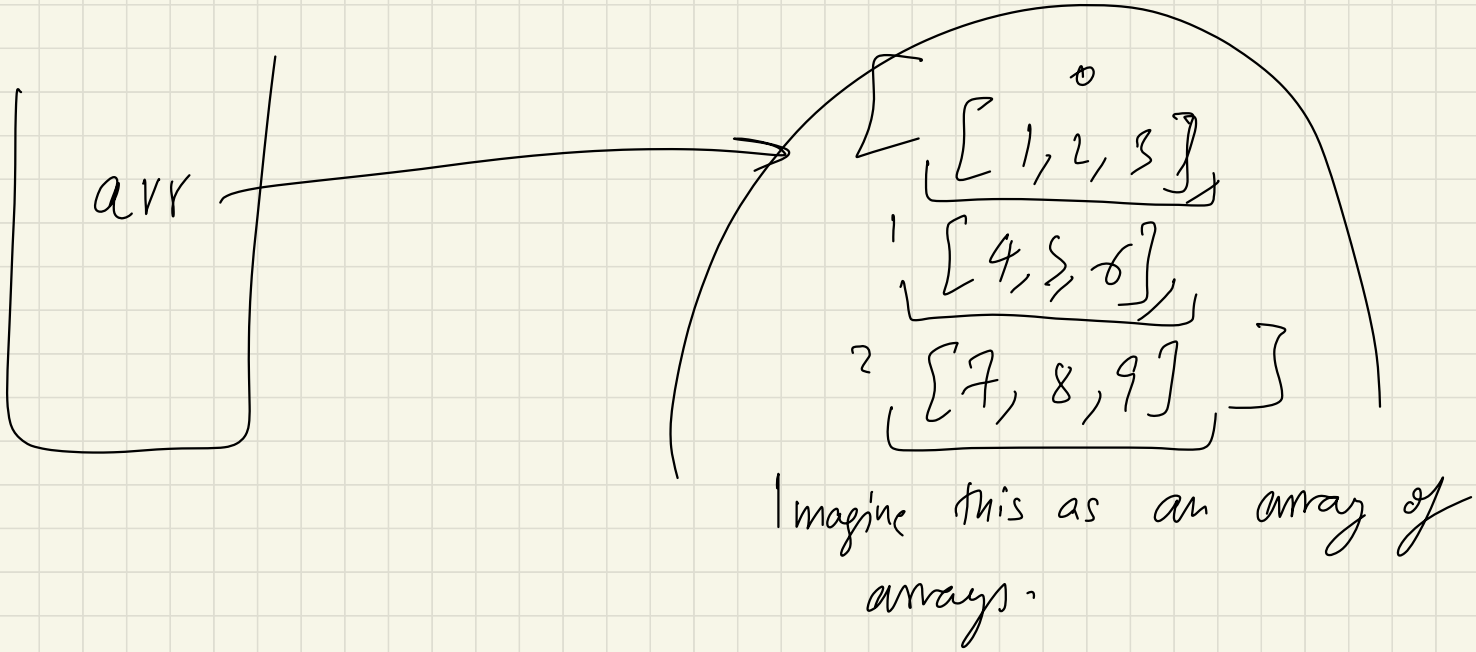
arr[0] = null



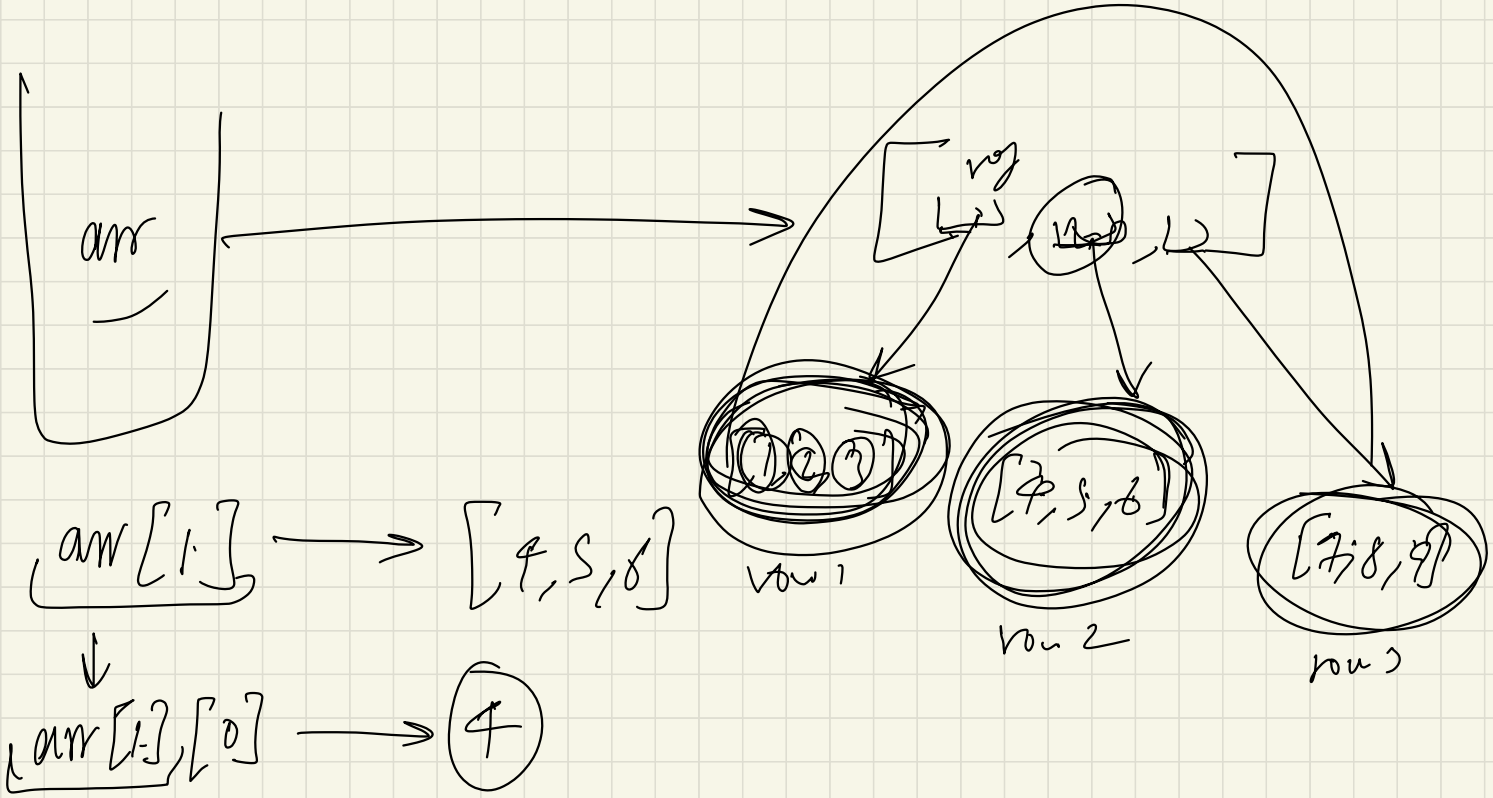

str
aa
cc
dd

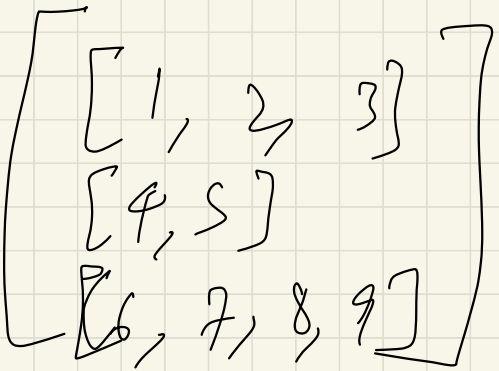


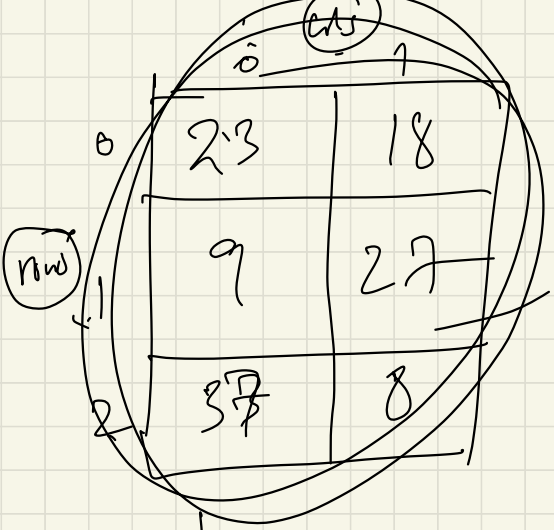
mutable behaviors



Imagine this as an array of arrays.


arr
arr[1:]
[4,5,6]
arr[1:][0]
4
row 1
row 2
row 3

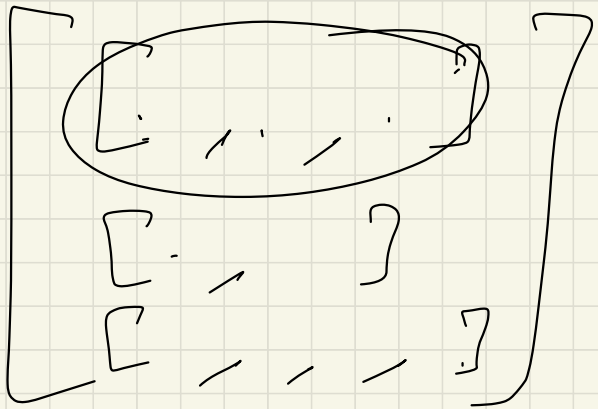

[1, 2, 3]
[4, 5]
[6, 7, 8, 9]



row	col
~~0~~	~~0 1 2~~
~~1~~	~~0 1~~ 2
~~2~~	~~0~~ 1 2
~~3~~	

```
for (row=0; row<=3; row++){
    // now we take very row
    for (col=0; col<=2; col++){
        arr[r][c] = input
    }
}
```

arr[row].length

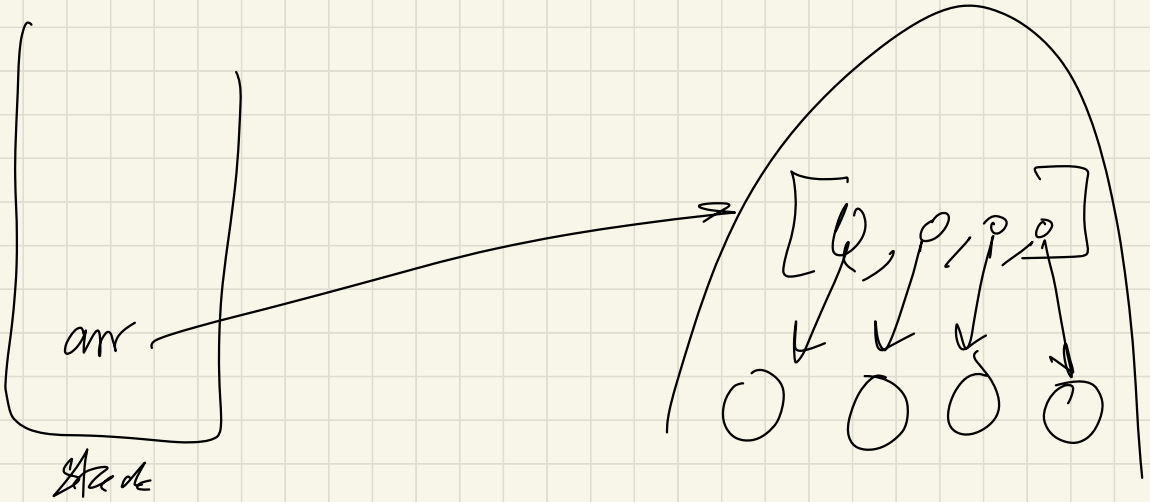


```
for(r=0; r<3; r++)
  for(c=0; c<size(r); c++)
    input
  }
}
```

size(arr[0])

arr[1]

r	c
0	0
1	


arr
Stack

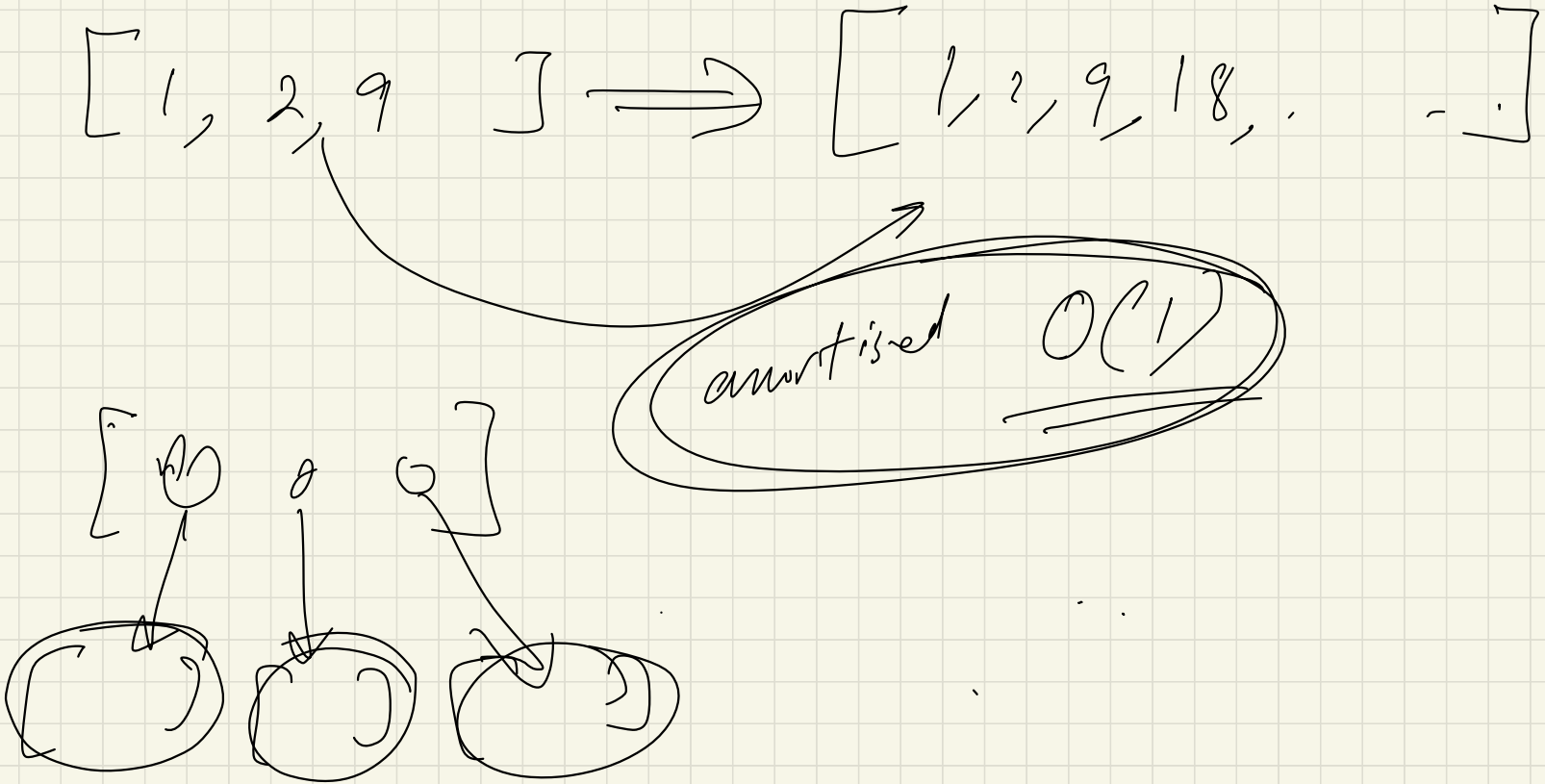
① Size is fixed Internally

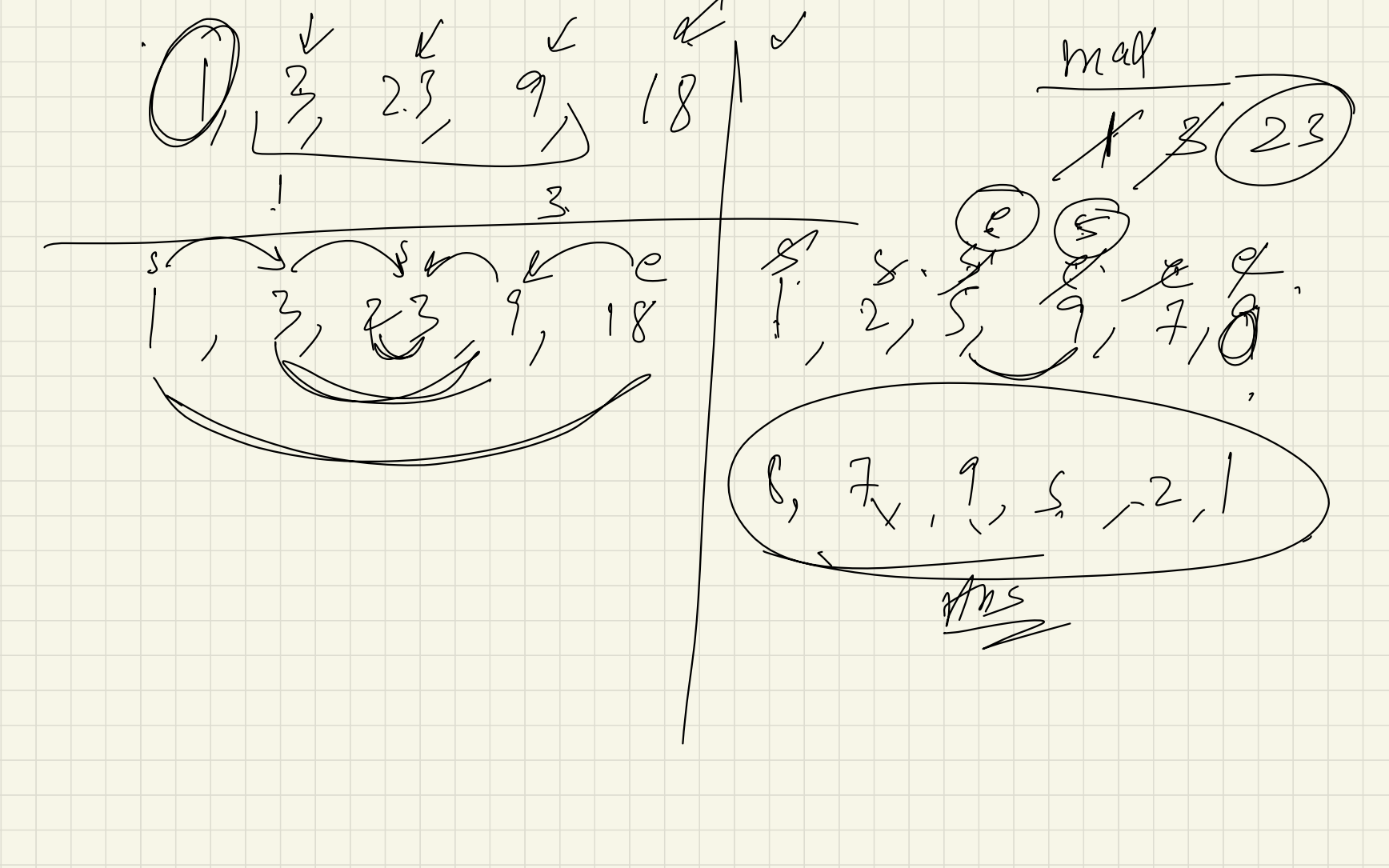
② Say arraylist fills by some amount

⟹ It will a new arraylist of say, double the size

⇒ old elements are copied in new one

⇒ old one is deleted


amortised O(1)


1
3
2 3
9
18
max
23
1 3 2 3 9 18
1 2 5 9 7 8
6, 7, 9, 5, 2, 1