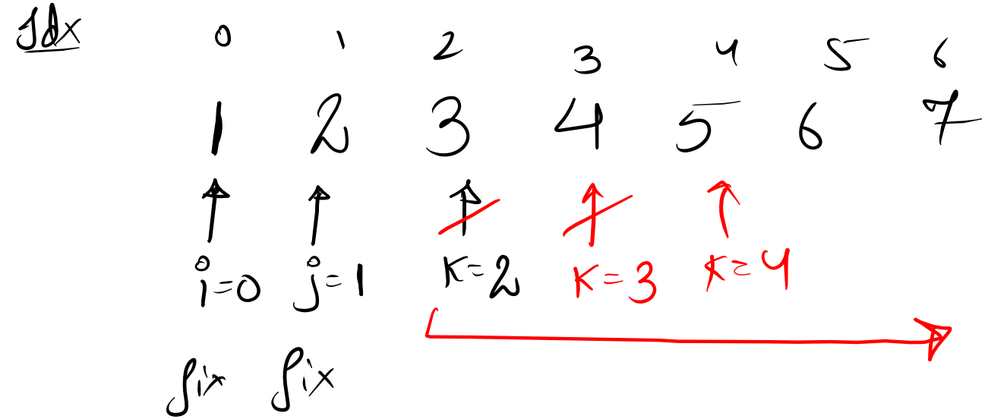
Triplet sum Equal to Target



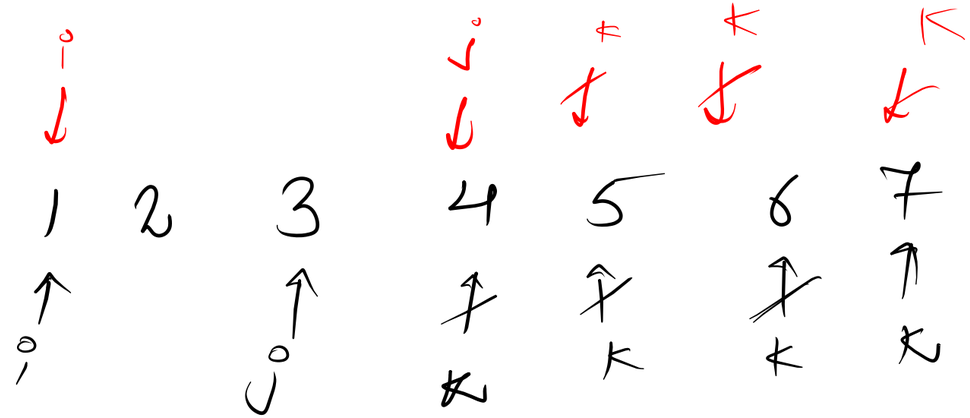
Target = 12

[1, 2, 3] ⇝ 1+2+3 ⇒ 6 == 12 ✗ cnt = 0

[1, 2, 4] ⇝ 1+2+4 ⇒ 7 == 12 ✗ cnt = 0

[1, 2, 5] ⇝ 1+2+5 ⇒ 8 == 12 ✗ cnt = 0

⋮



10^3 10^8 10^{12}

10^6 → $O(N^3)$

$O(N^2)$

[1,3,4] → sum == T if True

[1,3,5] " cnt++

[1,3,6] "

[1,3,7] "

Time complexity $O(N^3)$

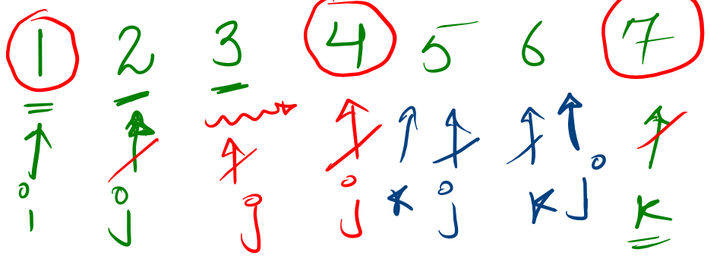
↓

TLE

optimise

↳ $\approx O(N^2)$

Sorted order → [1, 4, 7] [1, 5, 6]



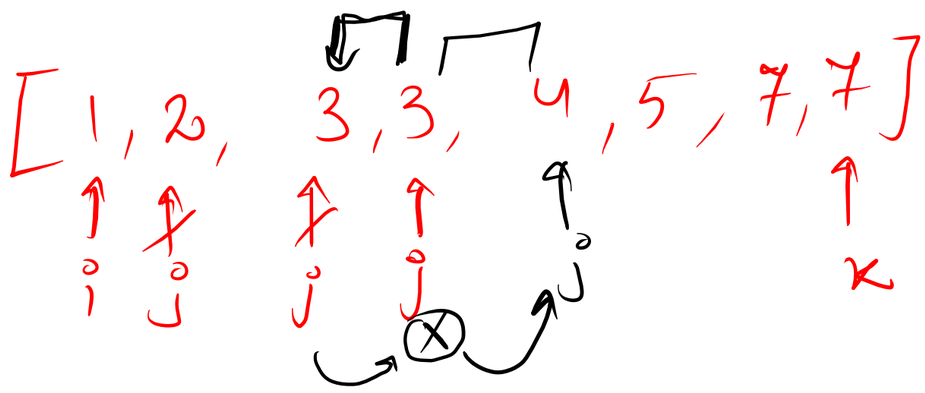
Target = 12

arr[i] + arr[j] + arr[k] = sum

1. sum == Target — j++, k--; cnt++;
2. ↑sum < Target — j++;
3. ↓sum > Target — k--;

sum	Target	Process
[1, 2, 7]	10 < 12	j++
[1, 3, 7]	11 < 12	j++
[1, 4, 7]	12 == 12	cnt++; j++, k--;
[1, 5, 6]	12 == 12	cnt++; j++, k--;
j > k	loop Exit	

prev == current



Let assume

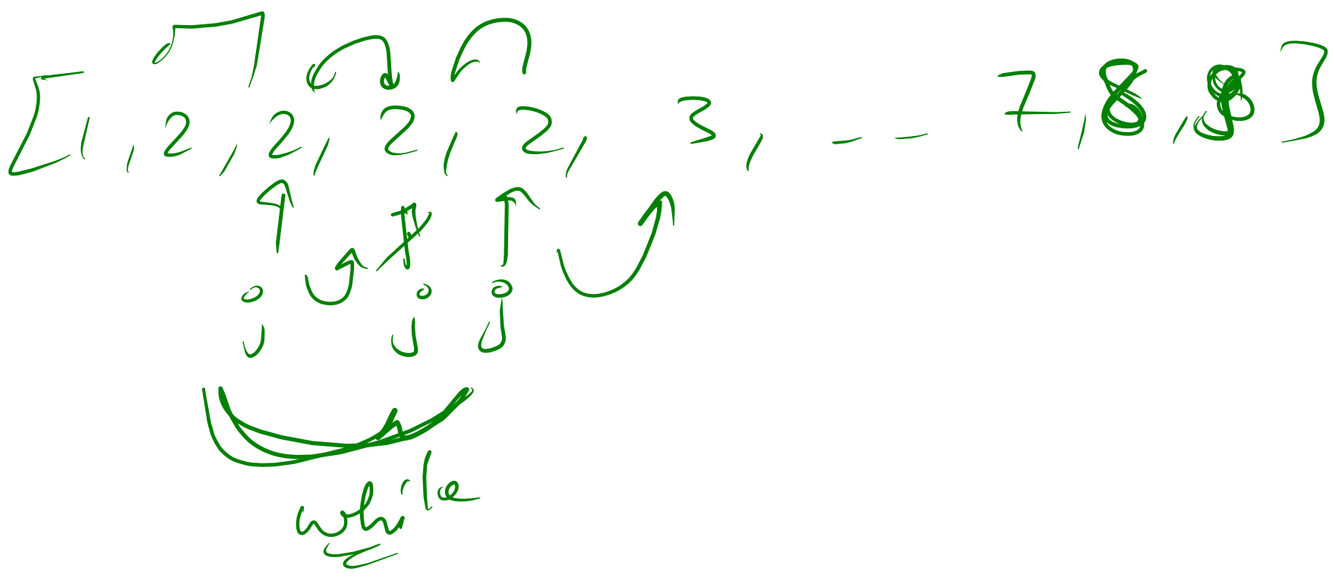
we are increasing cnt for the same subset.

[1,2,7] 10 <= 11 j++

[1,3,7] : 11 == 11 cnt++

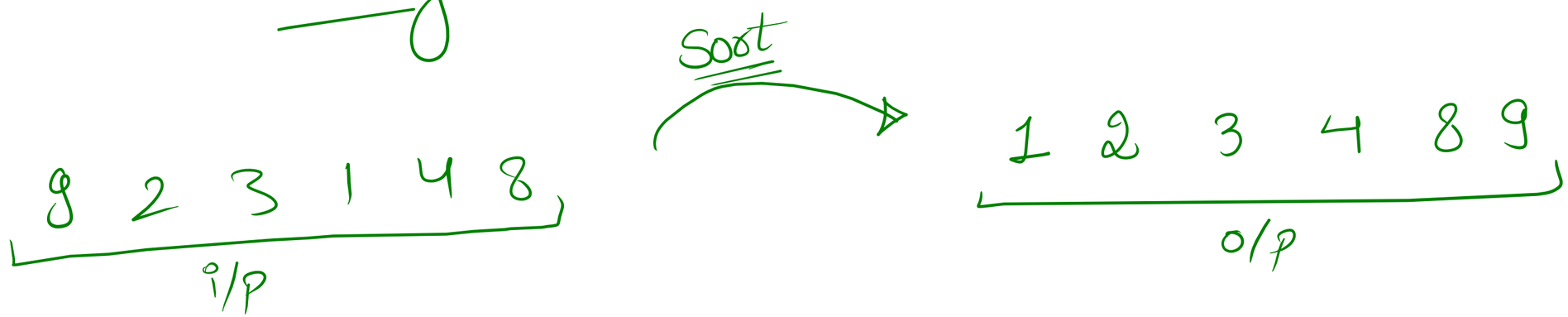
[1,3,7] 11 == cnt++

```
j<k &&
if (arr[j] == arr[j-1]) j++

j<k &&
if (arr[k] == arr[k+1]) k--;
```


[1, 2, 2, 2, 2, 3, -- 7, 8, 8]
i
j
while

Bubble Sort → Push max Element at last by adjecent Swapping

↳ sorting



idx idx+1

9 2 3 1 4 8

2 9 3 1 4 8

2 3 9 1 4 8

2 3 1 9 4 8

2 3 1 4 9 8

2 3 1 4 8 9

```
if(arr[idx] > arr[idx+1])
    swap(arr[idx], arr[idx+1])
}
else {
    order is correct
}
```

2 3 1 4 8 9

2 3 1 4 8 9

2 1 3 4 8 9

2 1 3 4 8 9

2 1 3 4 8 9

2 1 3 4 8 9

2 1 3 4 8 9

2 1 3 4 8 9

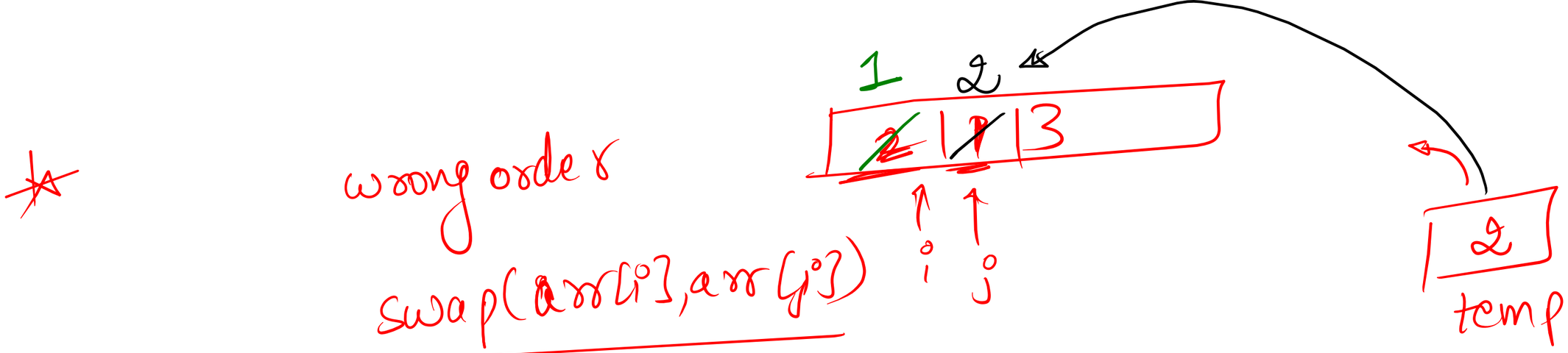
1 2 3 4 8 9

1 2 3 4 8 9

1 2 3 4 8 9

1 2 3 4 8 9

sorting done.



wrong order

swap(arr[i], arr[j])

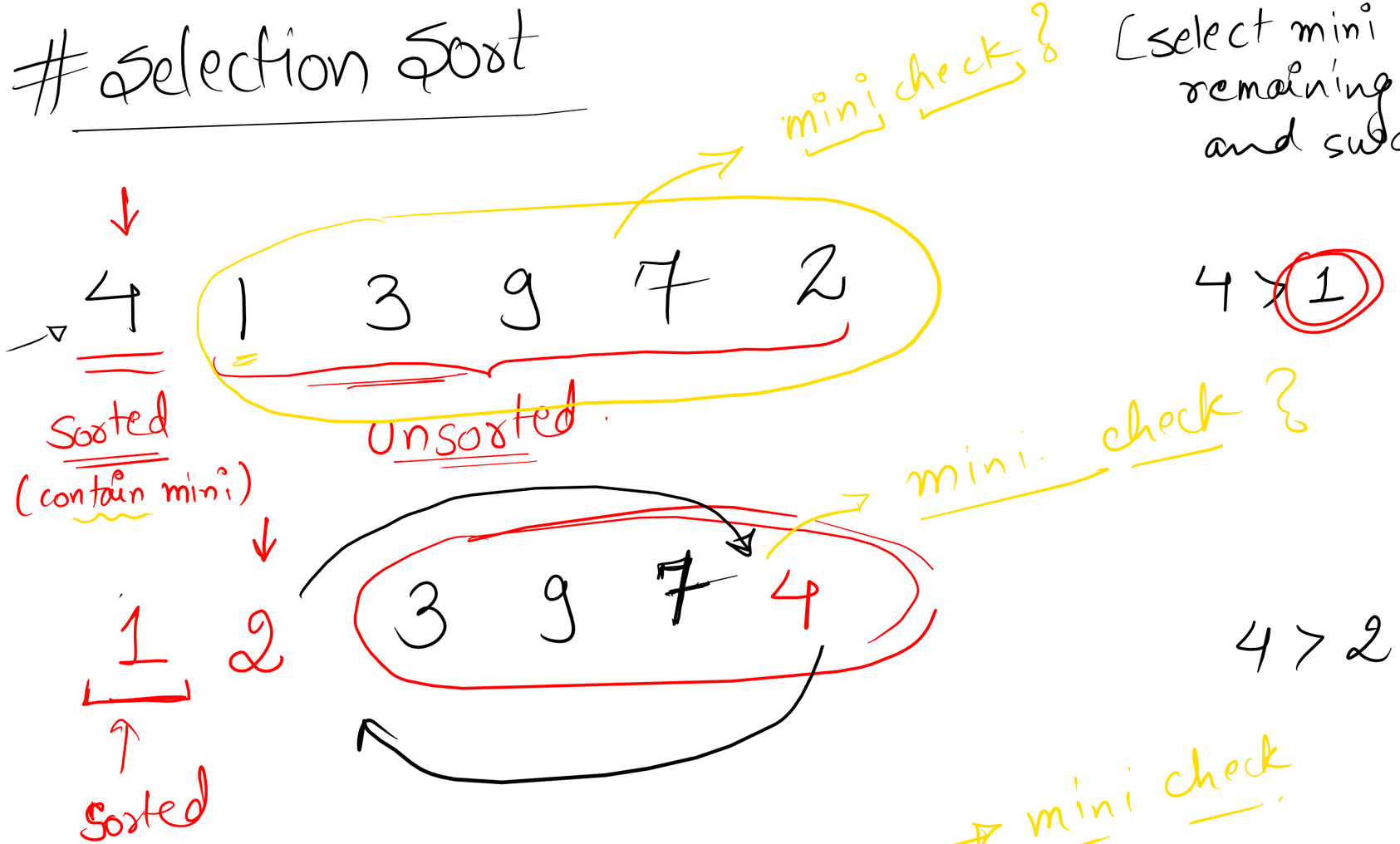
Java

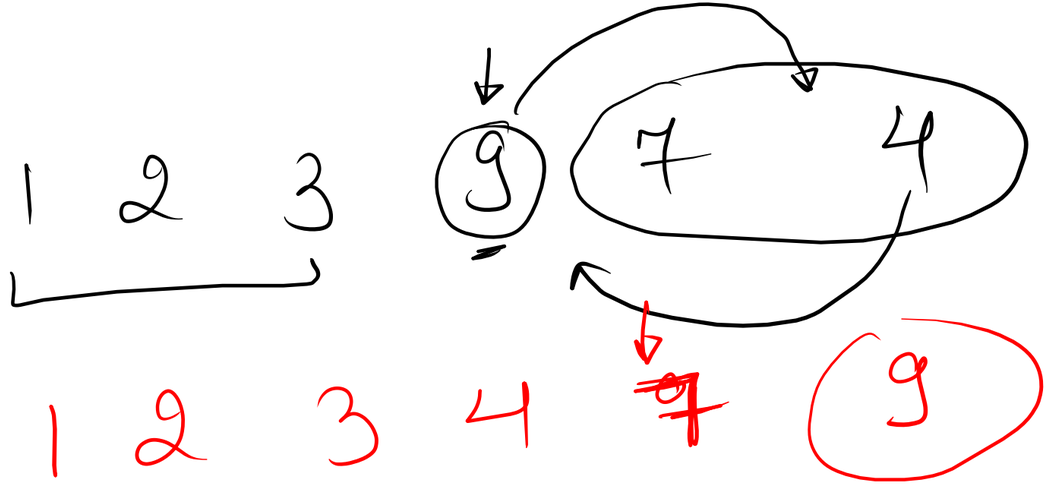
Internal working of swap

```
int temp = arr[i]
arr[i] = arr[j]
arr[j] = temp
```

Selection Sort

[select mini from the remaining arr and swap it]




1 2 3
9
7
4
9 > 4
1 2 3 4
7
9

Maximum Occurance

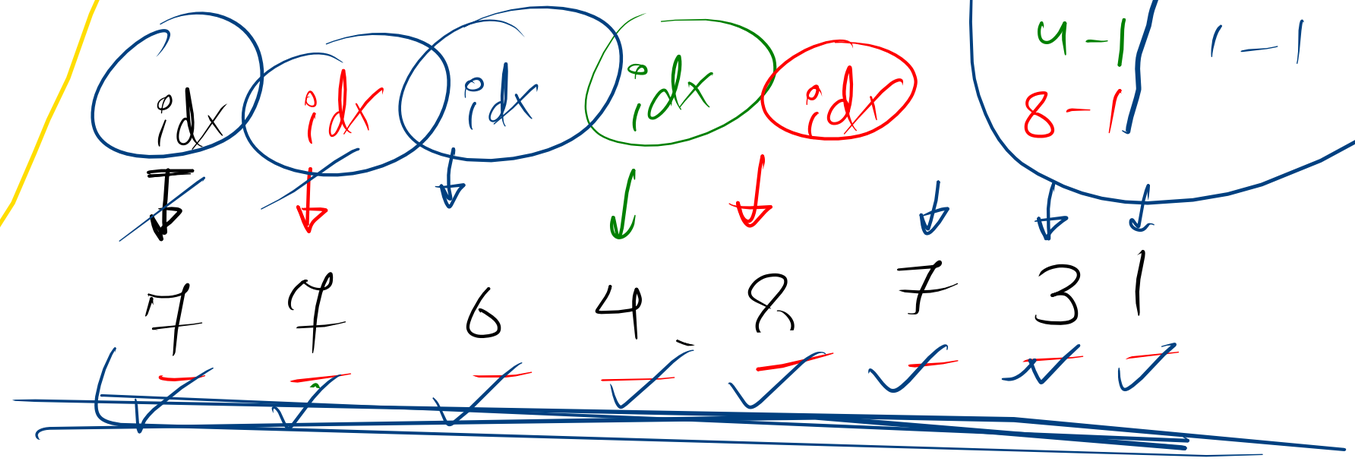
7 - 0 1 2 (3)

```
if(arr[idx] == arr[idx+x]) cnt++ }

if (cnt > maxCnt) {
    maxCnt = cnt;
    res = arr[idx]
}
```

cnt = 0 1 2 3

cnt = 0 1

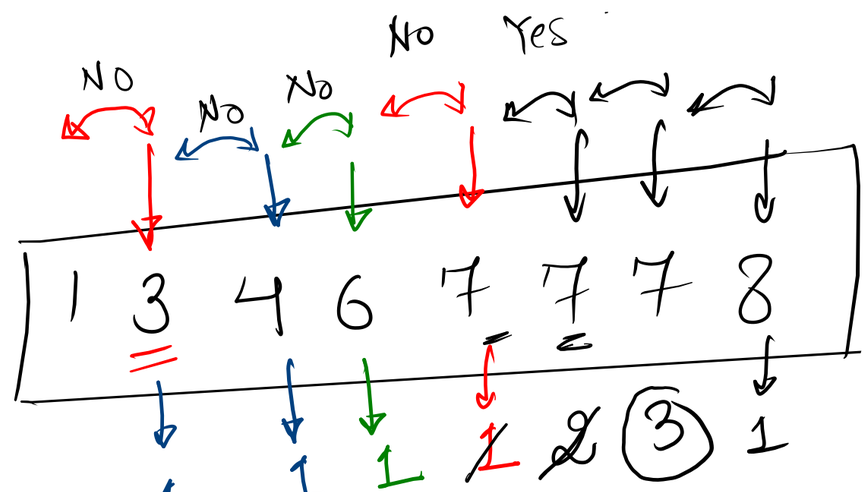
7 - 3	7 - 3
7 - 3	
6 - 1	3 - 1
4 - 1	1 - 1
8 - 1	



10^6 → Sorting

```
if(arr[i] == arr[i-1]){
    cnt++;
}
else{
    cnt = 1;
}
```

77648731 → sorting →



cnt :

```
if(cnt > maxcnt){
    maxcnt = cnt;
    answer = arr[i]
}
```

answer = 7

MaxCnt = 1; Cnt = 1;

9 8 1 1 2 2 3 —sort→ 1 1 2 2 3 8 9

same

Yes

1 1 2 2 3 8 9

2 1 2 1 1 1

2 2 ✗

2 2

1

Cnt > maxCnt

maxCnt = 2

ans = 1

✗

[2,3,7]

while(j<=k)

1 2 3 4 5 6 7

i j j j k k k

[2,3,7] 12 == 12 cnt++

j++, k--;

[2,4,6] 12 == 12 cnt++;

[2,5,5] j >= k exit

1 2 3 4 5 6 7

i j k k k

i = 0

j = i+1

k = n-1

sum > Target

k--;

[3,4,7] 14 > 12

k--;

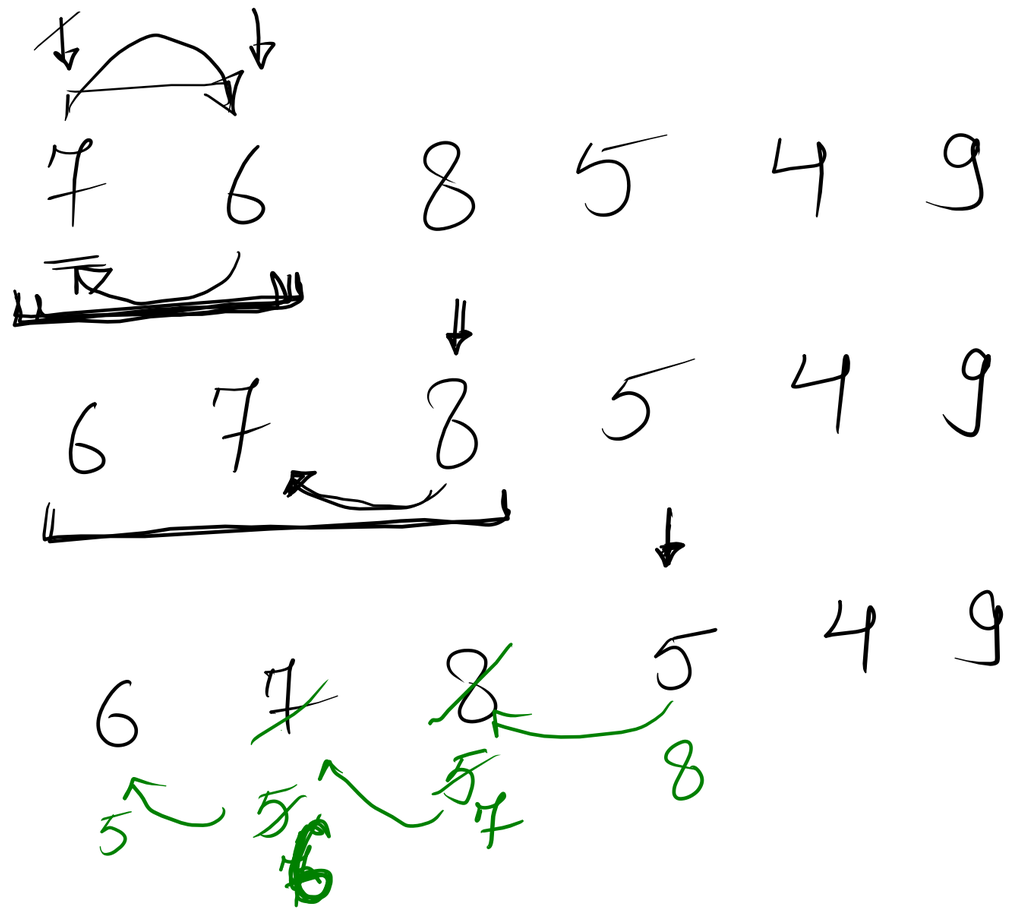
[3,4,6] 13 > 12

k--;

[3,4,5] 12 == 12

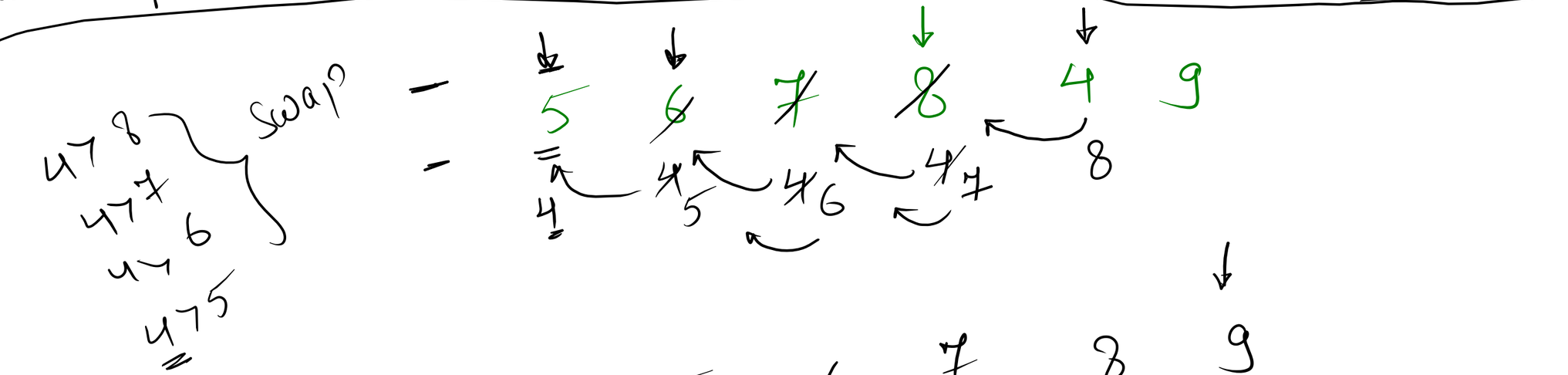
cnt++;

Insertion Sort



while (curVal < prevVal) → swap & index--;

Condition index > 0



9 7 8 ✓

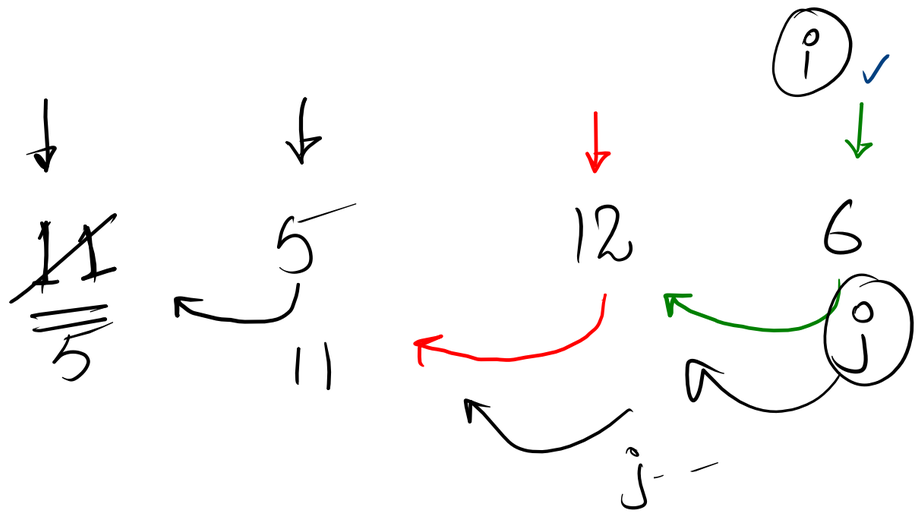
True
↓
No swapping

(curVal > prevVal)
↳ No swapping

Example - 2

if(arr[i] > arr[i-1])

5 > 11 swap



13

13 > 12 True

3

12 > 11 True No swapping

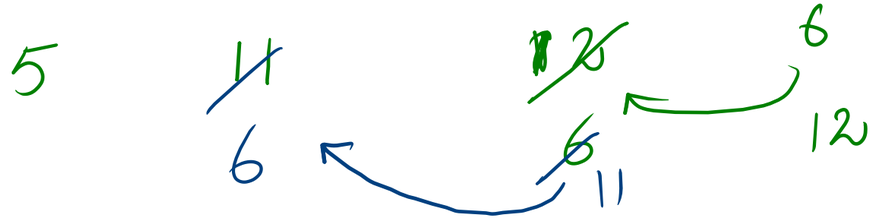
6 > 12 → swap

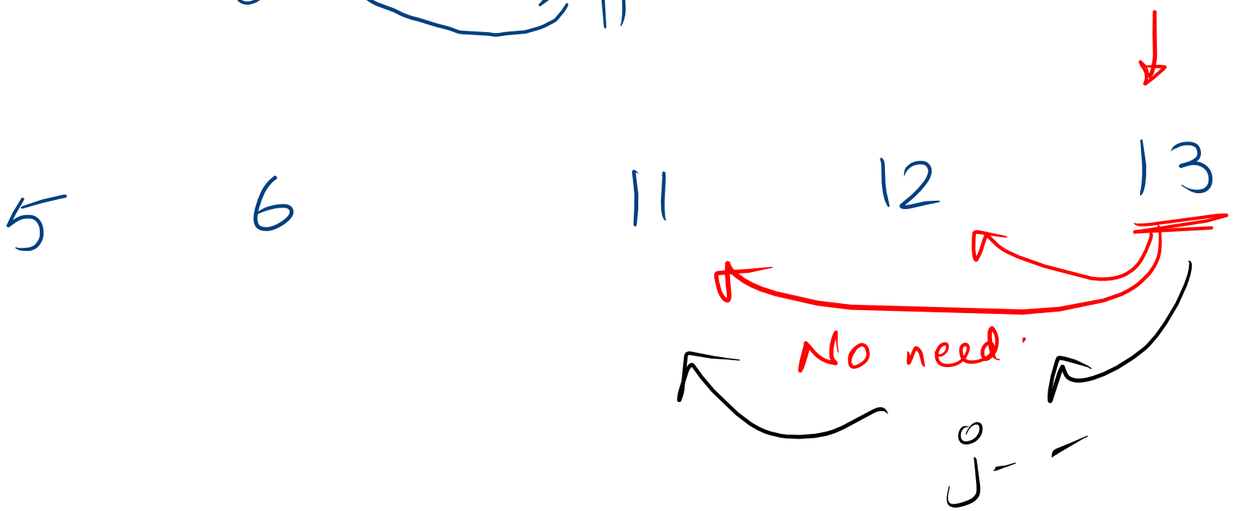
6 > 11 → swap

6 > 5

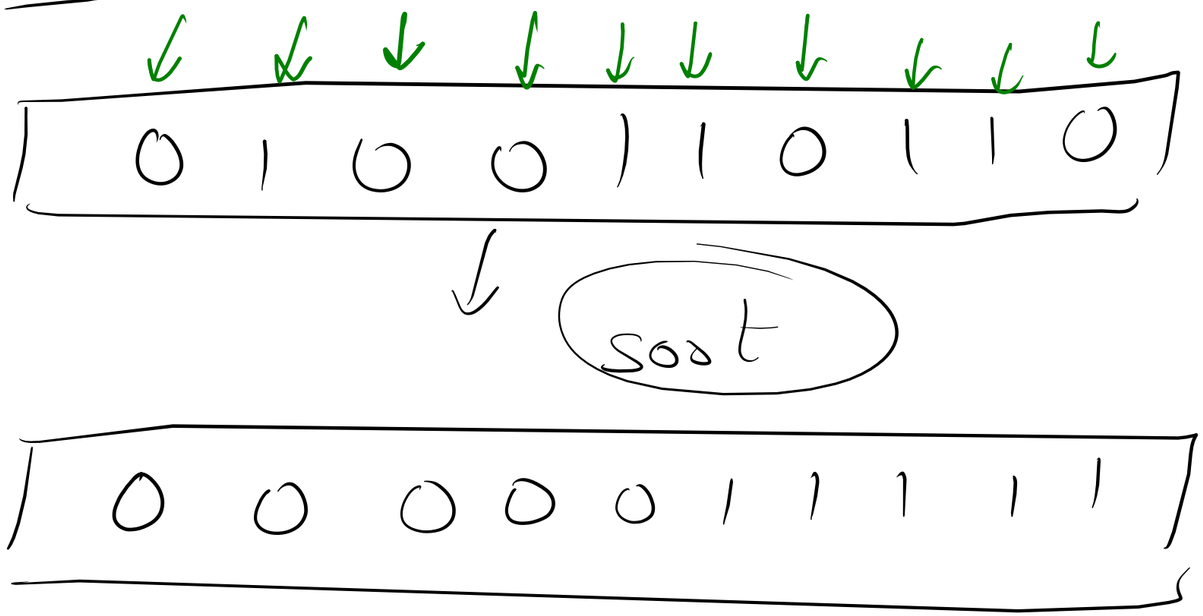
false

True





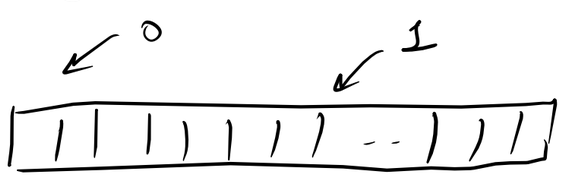
0-1 Sorting → O(N)



BS, SS, IS } TLE, O(N²)

Approach-1 : Arrays.sort(arr) ✓

Approach2 : result :



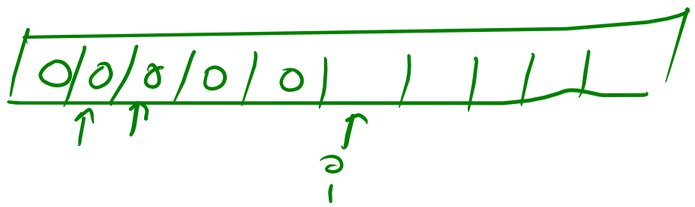
cntZero = ~~0~~ 1 ~~2~~ ~~3~~ ~~4~~ (5)

cntOne = ~~0~~ ~~1~~ ~~2~~ ~~3~~ ~~4~~ (5)

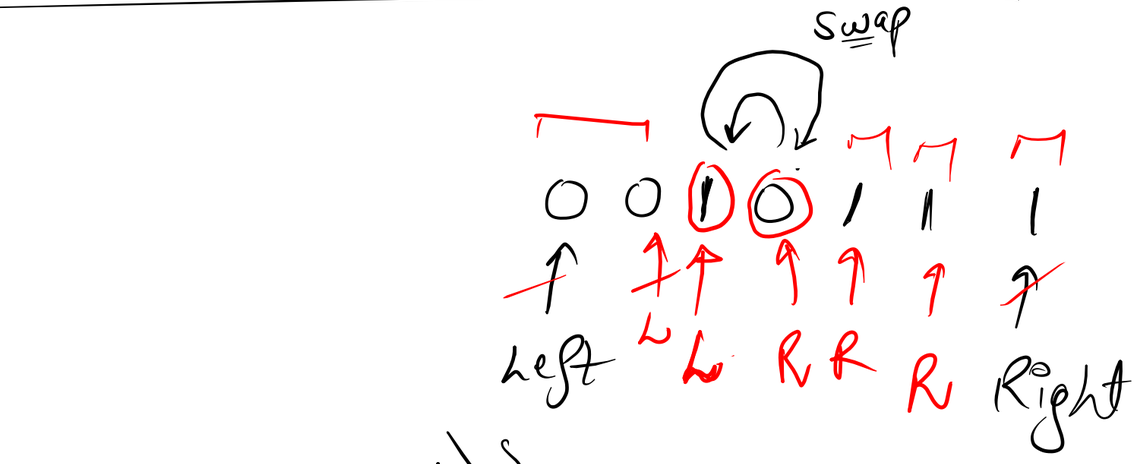
```
while (cntZero > 0) {
    result[i] = 0;
    i++; cntZero--;
}

while (cntOne > 0) {
    result[i] = 1;
    i++;
    cntOne--;
}
return results
```

cntZero = 5 4 3 .... 1 0

cntOne = 5



Best/ optimal Approach



Left pointer => 0 point

Right pointer => 1 point

2-pointer

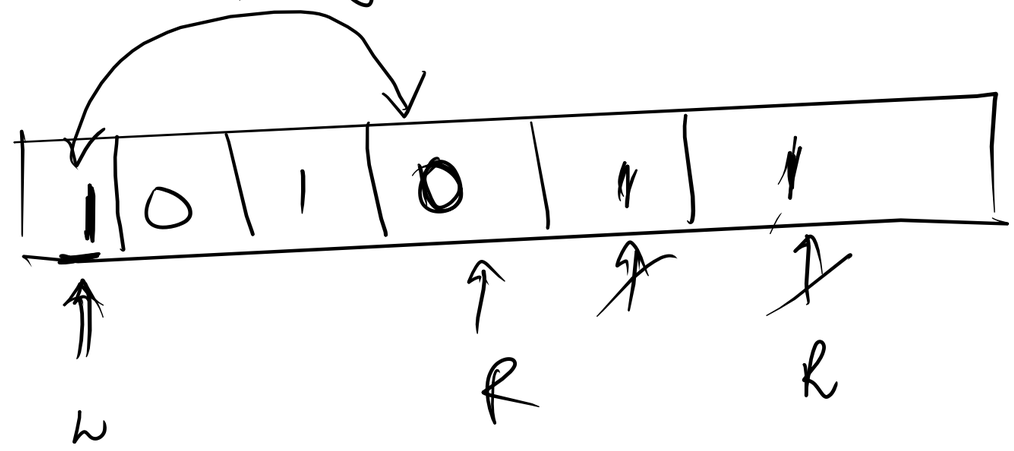
↓

if Conflict

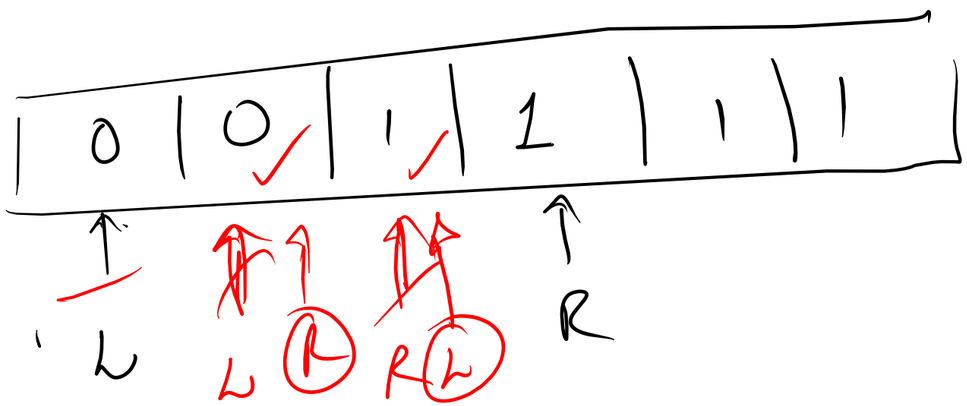
↓

swap

Swapping done



→ 0 0 1 1 1



while (L < R) false

▷ exit

Sorting 01

LC → Sort 012

→ Beautiful array

→ 2sum & 3sum

→ H/W

* 1W = 7day.
  ↳ 4day Test
  (8:30 pm)

* Camera → ✓ ✓

* Practice