# [ WELCOME ]

## SORTING - 1

→ Real World examples

- ↳ Contact list, Dictionary
- ↳ Product Amazon (Rating)
- ↳ Flights (Price)
- ↳ Fruit Shopping (quality)
- ↳ wardrobe
- ↳ Ordering on zomato
- ↳ Menu (A, B, ---)

Hybrid

→ Problems

Not used a lot

- → Bubble Sort
- → Selection Sort
- → Insertion Sort

} $O(N^2)$

- → Merge Sort
- → Quicksort
- → Comparators

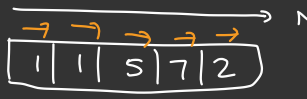
} $O(N \log N)$ ; $O(N^2)$ worst case

- → Counting Sort } $O(N + \text{Range})$
- → Radix Sort - ?
- → Heap Sort

Later

Quick idea

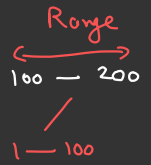
[ Counting Sorting ]





1, 1, 2, 5, 7

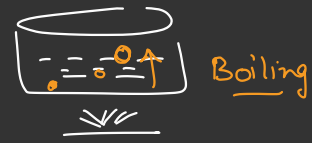
O(N + Range)



# Basic Sorts $O(N^2)$

## ① Bubble Sort

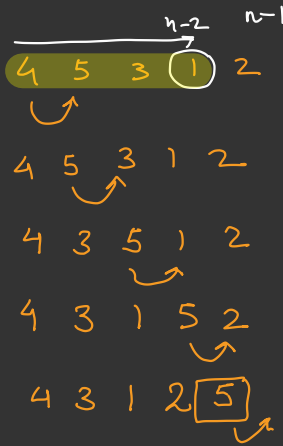
↳ in each iteration the bigger element moves to the end of the array by pairwise swapping.



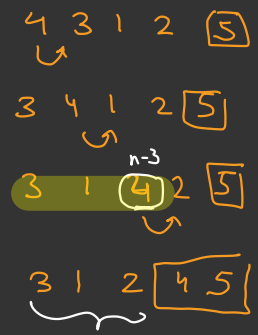

Input → [4, 5, 3, 1, 2]

[1, 2, 3, 4, 5] sorted

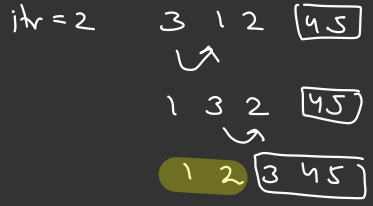
itr = 0



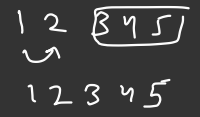

itr = 1




itr = 2

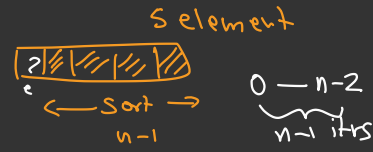



itr = 3



Code -

```
bubbleSort ( arr ) {
    n = arr.length,
    for ( itr = 0 , itr < n-1 , itr++ ) {
        for ( j=0 ; j <= n-2-itr , j++ ) {
            if ( arr[j] > arr[j+1] ) {
                swap ( arr[j] , arr[j+1] ),
            }
        }
    }
}
```




| itr | indx |
|---|---|
| 0 | n-2 |
| 1 | n-3 |
| 2 | n-4 |
| : | |

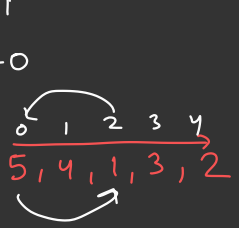
$n-1$ steps + $n-2 + n-3 + \dots + 1$

$\Rightarrow \frac{(n-1)(n)}{2} \Rightarrow O(n^2)$ time , $O(1)$ space

extra space

(2) Selection Sort → selects the min element & brings it to the front

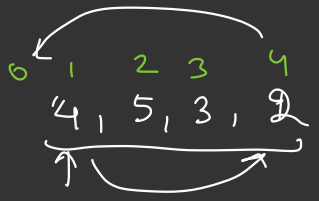

i=0




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

1, 4, 5, 3, 2

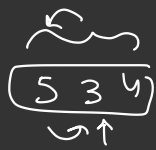
i=1




swap(arr(idx), arr(1))

1 2 5, 3, 4

i=2



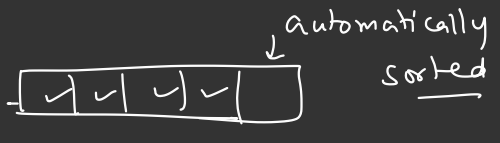

swap

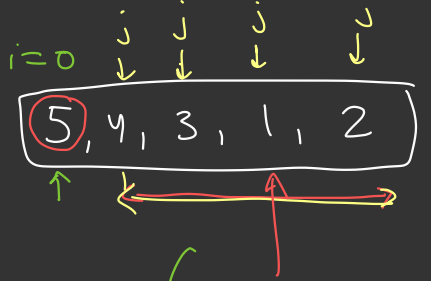
1 2 3 5 4

i=3


1 2 3 4 5

N elements



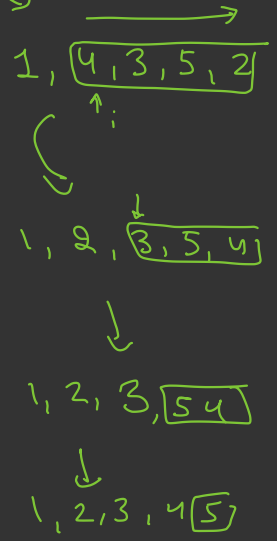




```
for ( i=0 ;   i < n-1 ;  i++ ) {
    Smallest = a(i),  idx = i
    for (j= i+1)   j<= n-1; j++) {
        if ( arr[j] <  arr[idx] ) {
            Smallest = arr(j)
            idx = j
        }
    }
    • Swap (arr(i), arr(idx)),
}
```

Linear search for smallest element

| Smallest | | idx |
|---|---|---|
| 5 | ← | 0 |
| 4 | ← | 1 |
| 3 | ← | 2 |
| 1 | ← | 3 |

2 < 1 ✗



Time = $n-1 + n-2 + \dots + 1$

= $O(N^2)$ time

$O(1)$ space

$N = 100$

$N^2 = 10^4$

$N = 10^4$

$N^2 \Rightarrow 10^8 \rightarrow$ 1s

$N = 10^5$

$N^2 = 10^{10}$ steps

= 100 seconds

→ Bubble

→ Selection

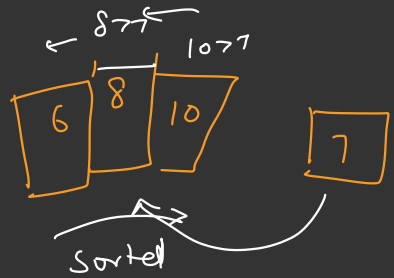
→ Insertion Sort

$O(N^2)$

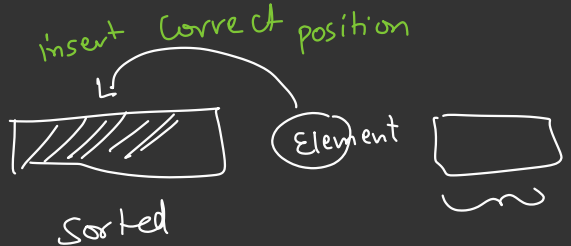
# Insertion Sort

Stable ?

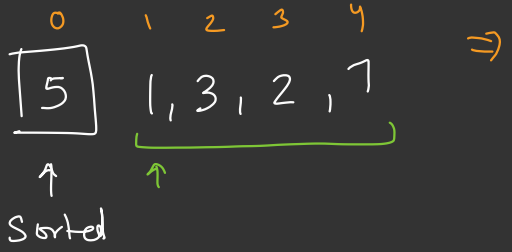

Cards Game

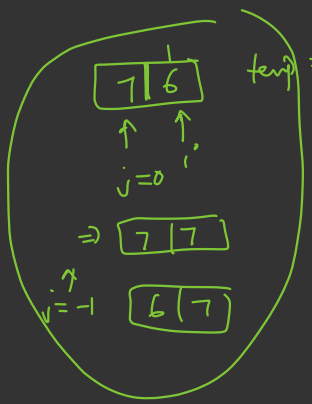



Idea




5, 1, 3, 2, 7




5 > 2

temp = 2

1, 3, 5, 5, 7, 6

1 3 5 5 7 6

1 3 3 5 7 6

1 2 3 5 7 6

1 > 2

insertion in middle

j = i-1 temp = arr(i)

while(j>=0 && arr(j) > temp){

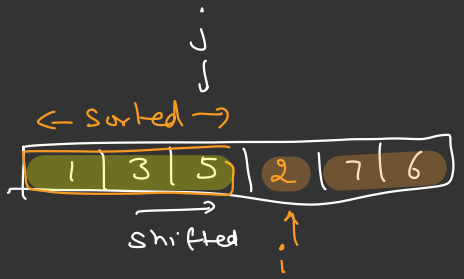
⇒ arr[j+1] = arr[j]

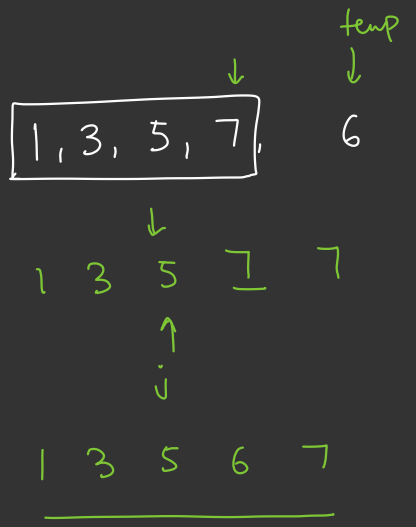
j--

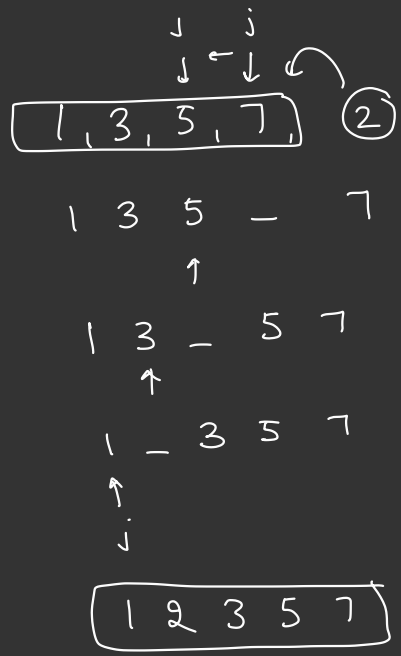
}

shifting

arr[j+1] = temp

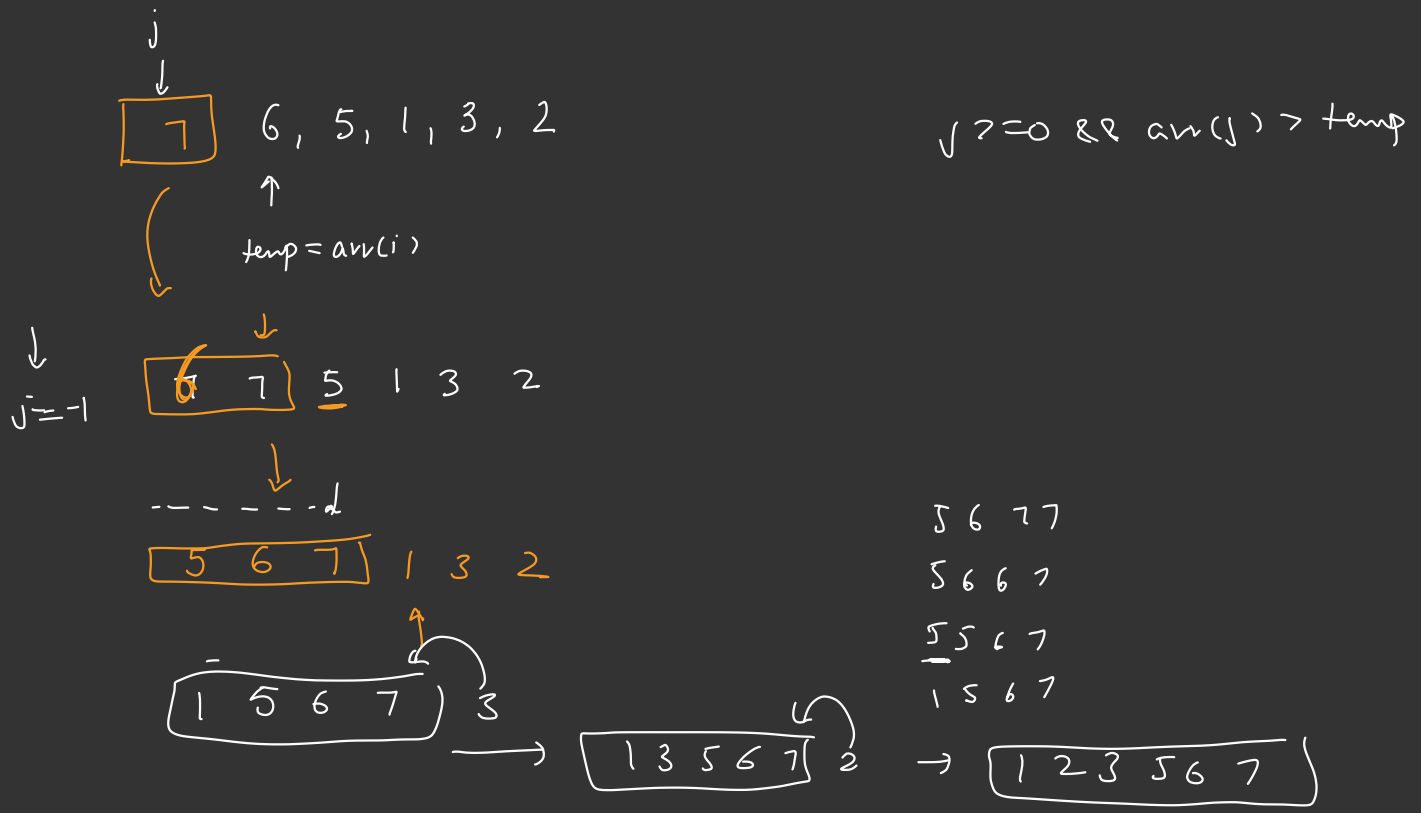



temp
1, 3, 5, 7,
6
1 3 5 7 7
i
1 3 5 6 7


j i
1, 3, 5, 7,
2
1 3 5 _ 7
1 3 _ 5 7
1 _ 3 5 7
i
1 2 3 5 7

Complete Array

7, 6, 5, 1, 3, 2



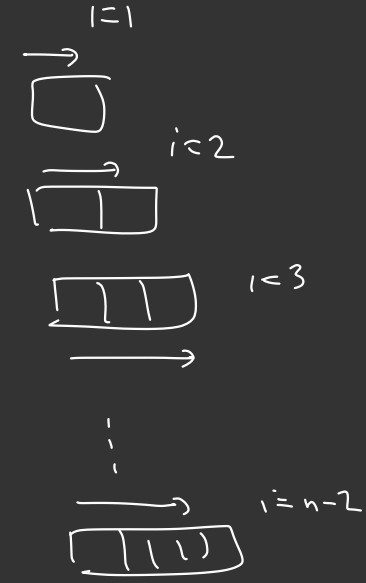
code

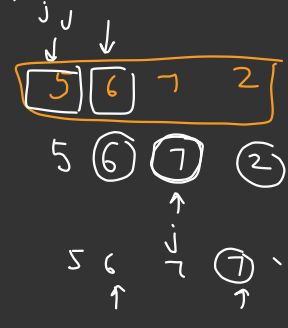
```
for (i=1 ——— n-1) {
    Put arr(i) in the corret idx.

    j = i-1        temp = arr(i)

    while(j>=0 && arr(j) > temp){
        ⇒ arr[j+1] = arr[j]
          j--
    }

    arr[j+1] = temp;
}
```




$1+2+3+\dots n-1$

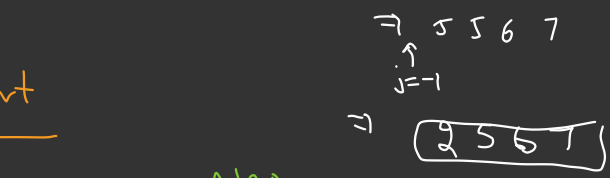
$= O(N^2)$ steps

$= O(1)$ space

# Stable Sort

↓

Property of sorting Also




- Bubble Sort → stable (will maintain same relative ordering for same keys)
- Selection → unstable

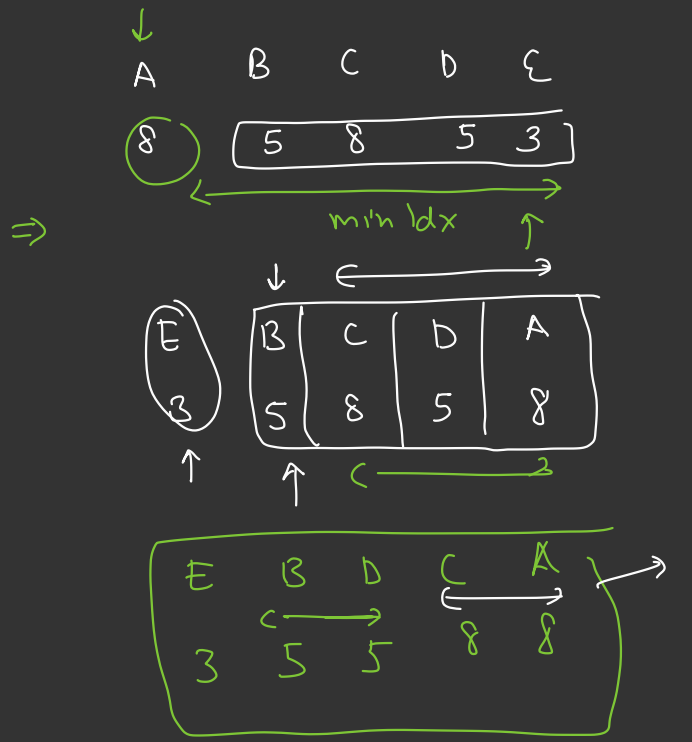
| Students | A | B | C | D | E |
|---|---|---|---|---|---|
| Marks | 8 | 5 | 3 | 5 | 8 |

A → E
8 8 NO

RankList → C, B, D, A, E

Bubble Sort? (Stable Sort) low to High

Selection Sort ⇒

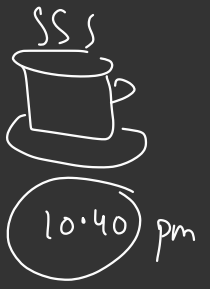



Relative order is not Preserved

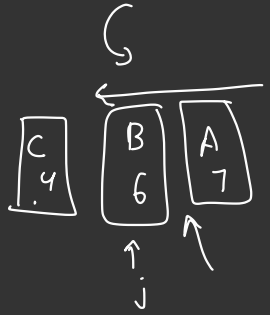
(Unstable Sort)

# Insertion Sort ?

Stable or Not




Break

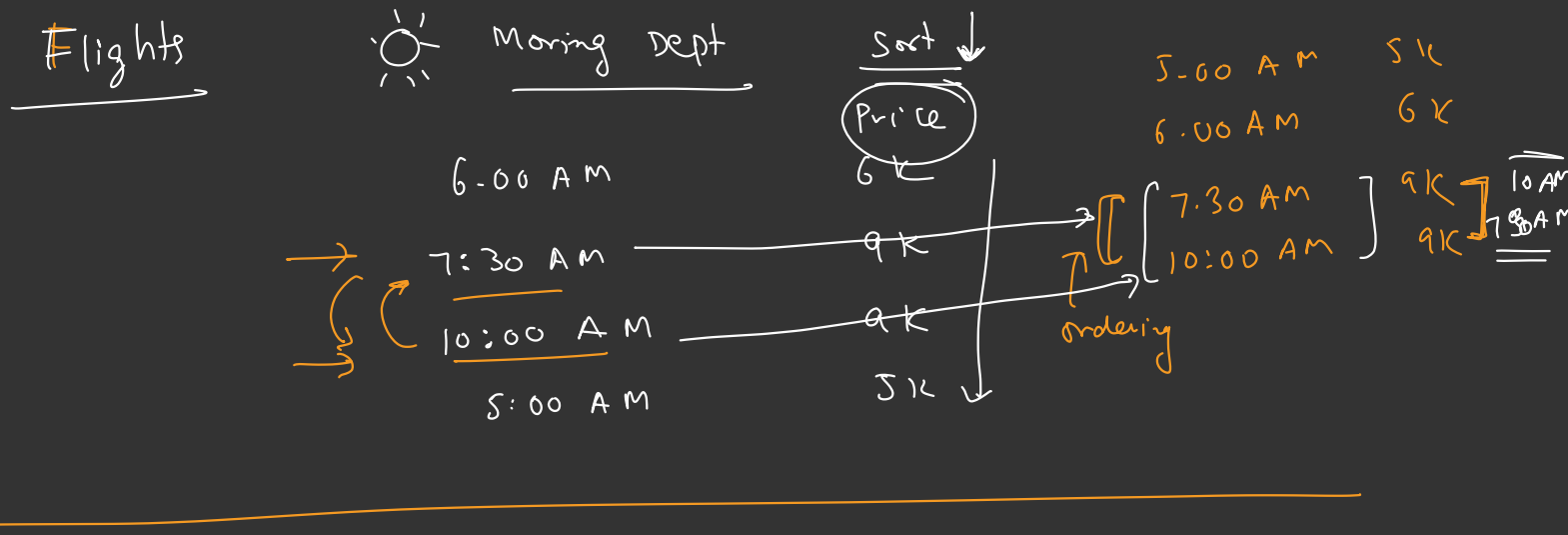




Stable Sort

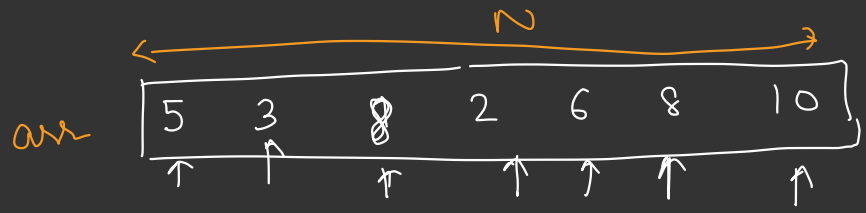

a(j) > temp

7, A  D, 6

6, B  } D, 6

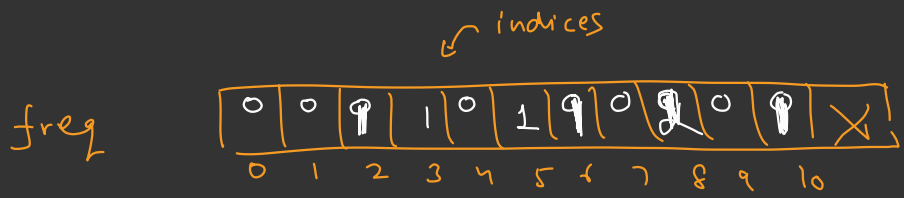

Flights
Morning Dept
Sort
Price
6K
6:00 AM
7:30 AM
9K
10:00 AM
9K
5:00 AM
5K
ordering
5:00 AM 5K
6:00 AM 6K
7:30 AM
10:00 AM
9K
9K
10 AM
9 AM

# Counting Sort + Applications




No are positive

↳ idea. count & maintain freq of each element




```
freq[MAX+1] = {0},
for( x : arr) {
    freq[x] += 1
```

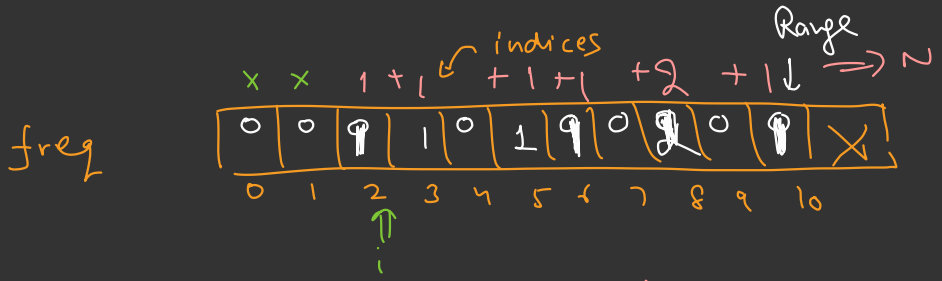
Step-1

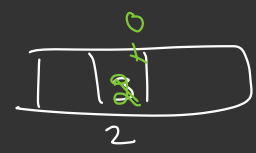
x = a[i]

O(N)

3

Step-2 iterate over freq array.



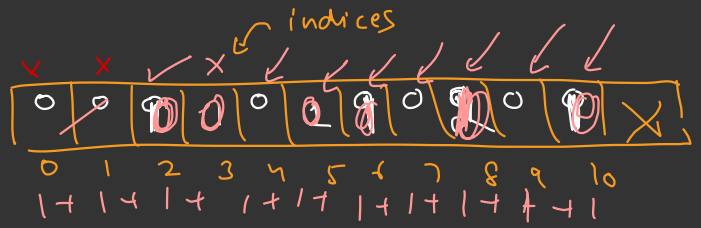

2, 2, 2

```
for (i=0 ——— Range) {
    while( freq[i] > 0) {
        print(i),
        freq[i]--,
    }
}
```

N times total for all ith.

$O(\frac{Range + N}{7})$

freq




2, 3, 5, 6, 8, 8, 10

0 to $\infty$ or $-\infty$ to $\infty$ $\Rightarrow$ Don't use Counting Sort.

(1, 100, 5, 10000, ---- , $10^6$, 4)

(N log N using Merge Sort)

X Problem

1, 100, 1000, $10^6$, 4

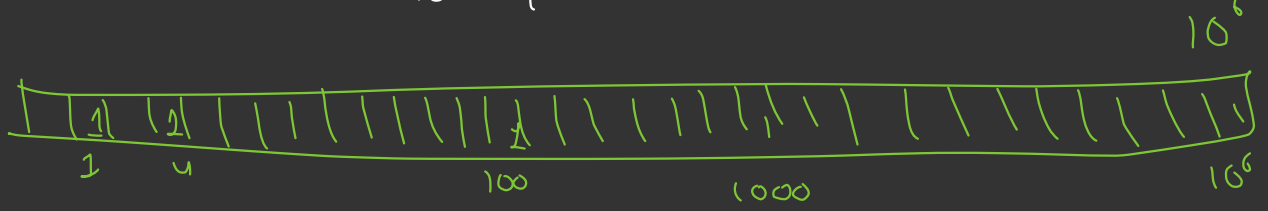
N = 5

Range = $10^6$

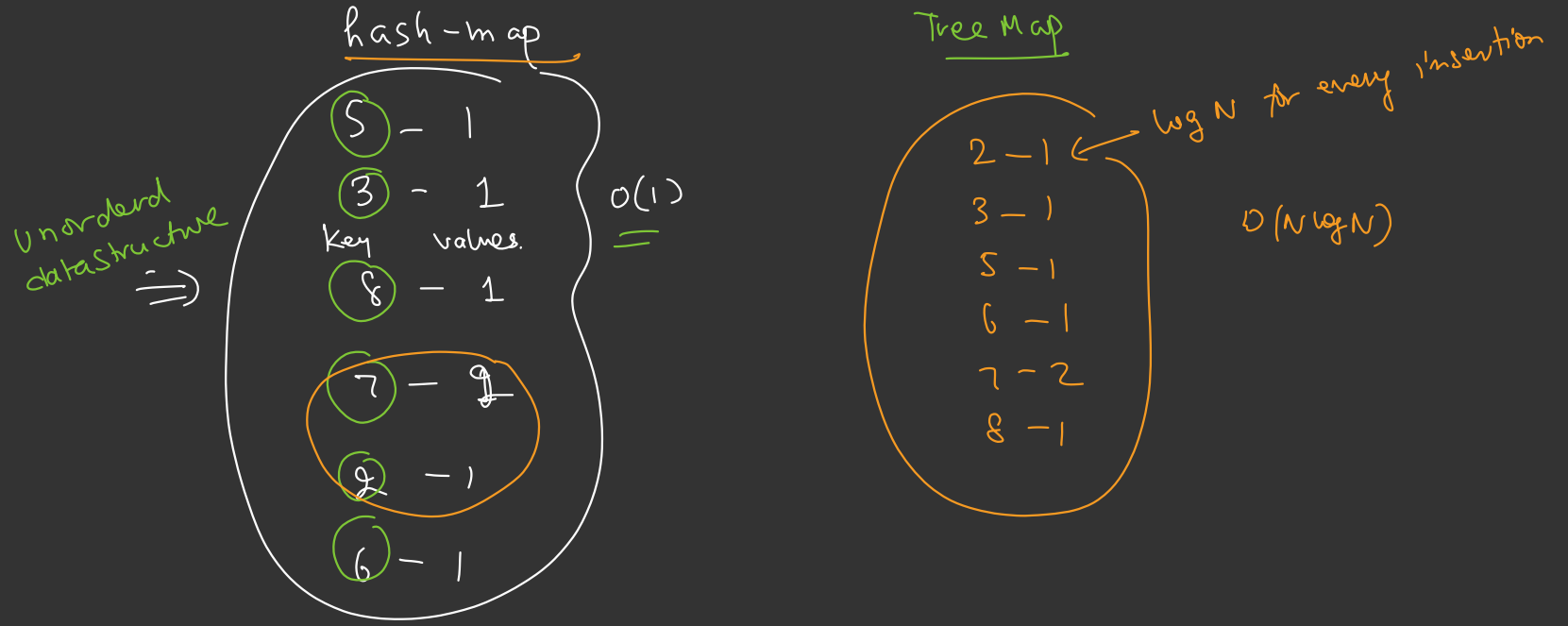
$O(N + Range)$

↳ $10^6$ steps

↳ Bubble Sort $N^2 = 25$ steps



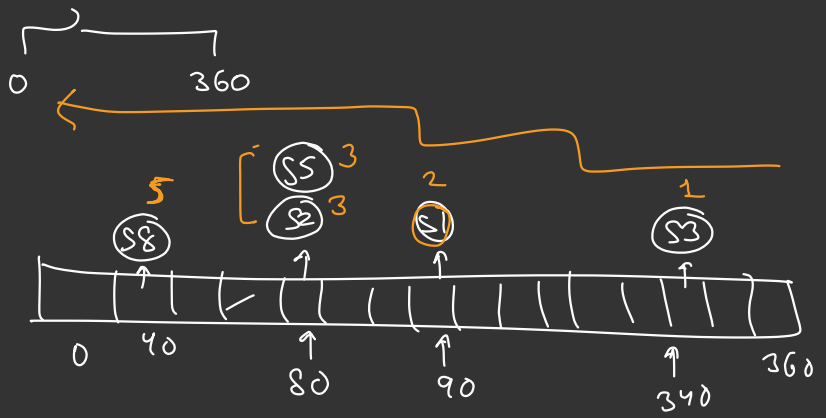
3, 8, 5, 7, 2, 7, 6


hash-map
Unordered datastructure
Key values
5 - 1
3 - 1
8 - 1
7 - 2
2 - 1
6 - 1
O(1)
Tree Map
log N for every insertion
2 - 1
3 - 1
5 - 1
6 - 1
7 - 2
8 - 1
O(N log N)

# "Amazon Interview Problem"

IITJEE exam ⇒ $10^6$ students. (1M)

Score ⇒ ⇐ Ranklist ⇒




Array of Arraylist

Arrays.sort ⇒ N log N

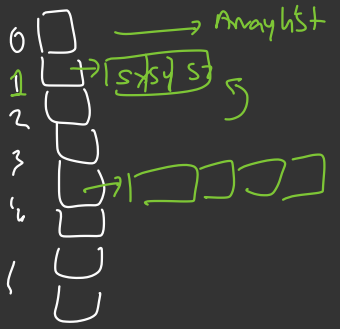
$10^6 \log 10^6$

O(N + Range)

$10^6$ 360

S1 – 90
S2 – 90
S3 – 340
S4 – 60
S5 – 80

Linear in terms of N



# Which sorting Technique to use?

N ✓ Range ✓ Datatype ✓

Smallest O(N) ↓ Largest O(N) ↓

Range = 300

-100 to 200

-100 +100
5 +100
-50 +100

Shifting → 1 | | | | | | 0 50 105 300 ← Shift

Shift Back

-100, -50, 5

O(N + ~~Largest~~)

O(N + Range)

6.4
3.28
4.46
4.41

AbC
def
batman

→ Merge Sort

800 — 2000 O(2000)

-800 -800

0 — 1200 O(1200) ✓