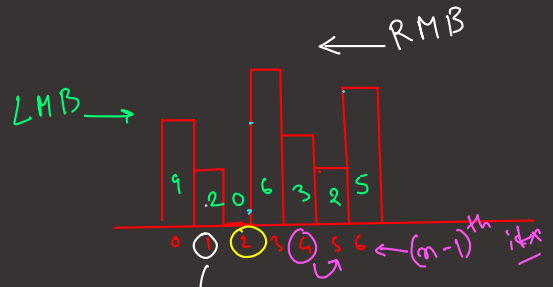


LMB = | 4 | 4 | 4 | 6 | 6 | 6 | 6 | → Left Max Boundary

0 1 2 3 4 5 6

RMB = | 6 | 6 | 6 | 6 | 5 | 5 | 5 | → Right Max Boundary.

0 1 2 3 4 5 6

Trapped water at idx i = $\min(LMB[i], RMB[i]) - height[i]$

(Water level)

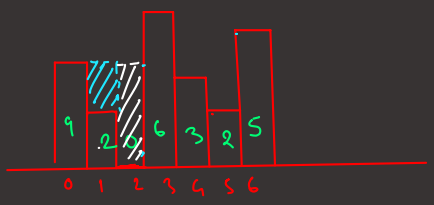
Total water = $\sum tw$

$LMB[i] = \max(LMB[i-1], height[i])$ if $i \neq 0$

for $i=0$, $LMB[0] = height[0]$

$RMB[i] = \max(RMB[i+1], height[i])$ if $i \neq n-1$

if $i = n-1$, $RMB[i] = height[i]$



$tw[0] = \min(LMB[0], RMB[0]) - height[0]$

$= \min(4, 6) - 4 = 4 - 4 = 0$

$tw[1] = \min(4, 6) - 2 = 4 - 2 = 2$

$tw[2] = \min(4, 6) - 0 = 4 - 0 = 4$

⋮

Sorting

① **Bubble Sort** : Large elements come to the end of array by swapping with the adjacent element.

A → | 8 | 5 | 7 | 3 | 2 |

i = 0

1st pass

8 5 7 3 2

5 8 7 3 2

5 7 8 3 2

5 7 3 8 2

5 7 3 2 8

result for pass : 1

comp = 4

max swap = 4

j = 0 to n-1

i = 1

2nd Pass

5 7 3 2 8

5 7 3 2 8

5 3 7 2 8

5 3 2 7 8

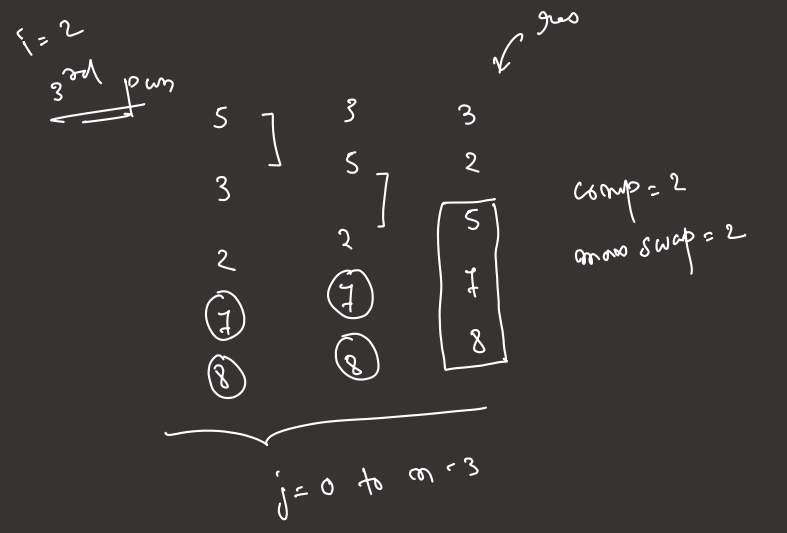
result of 2nd pass

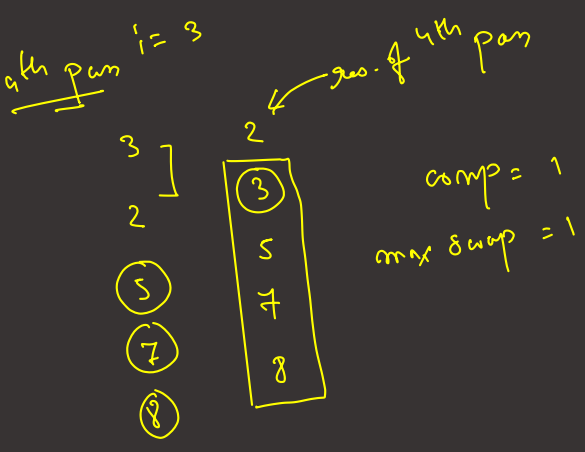
comp = 3

swap = 2

max swap = 3

j = 0 to n-2





∴ Total no of comp = $(n-1) + \cdots + 3 + 2 + 1$

$= \frac{n(n-1)}{2}$ ← max swap

Time Complexity $= O(n^2)$ $\left[O\left(\frac{n^2}{2} - \frac{n}{2}\right) = O(n^2/2) = O(n^2) \right]$

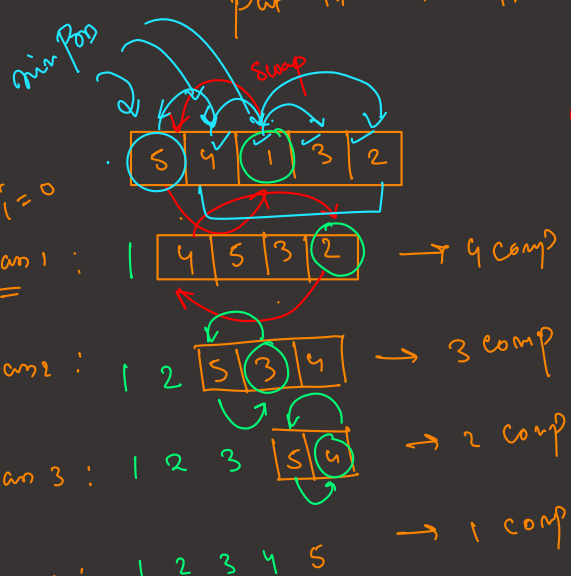
Note: Bubble sort → adaptive (not by default)

↓ stable (HW)

```
for (int i = 0 ; i < n-1 ; i++)
    for (int j = 0 ; j < n-1-i ; j++)
        // start of code.
```

↓ comp or swap

Selection Sort — pick the smallest (from unsorted part), put it at the beginning.



① check for the min of unsorted array.

② else [1st index] ↔ [min position]

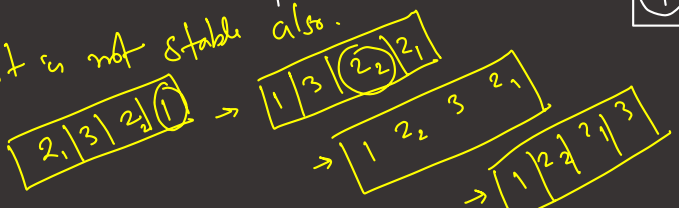
Total no. of swap $= n-1$

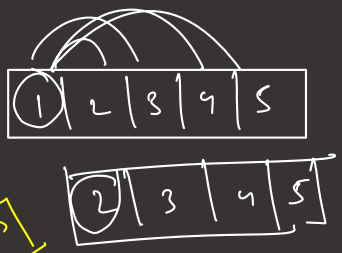
$= O(n)$

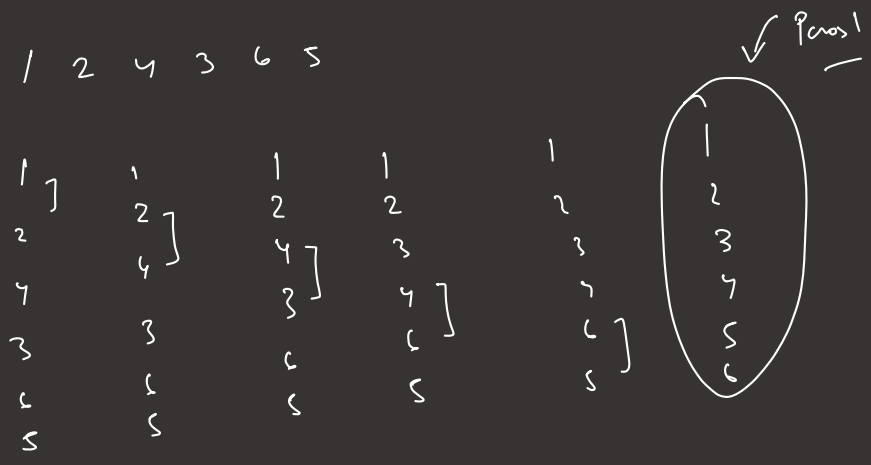
Total comp $= \frac{n(n-1)}{2} = O(n^2)$ ← Time Complexity

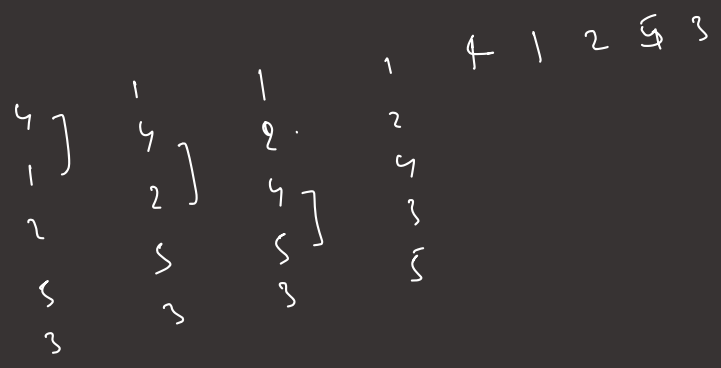
① Selection Sort is not adaptive

② Selection Sort is not stable also.

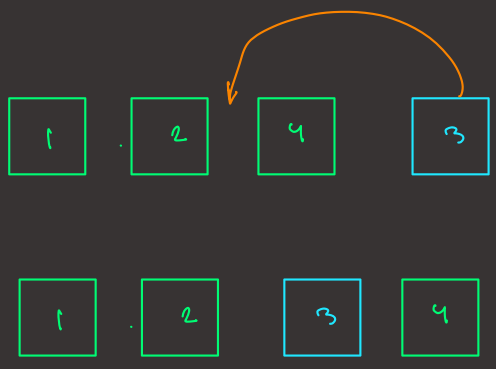




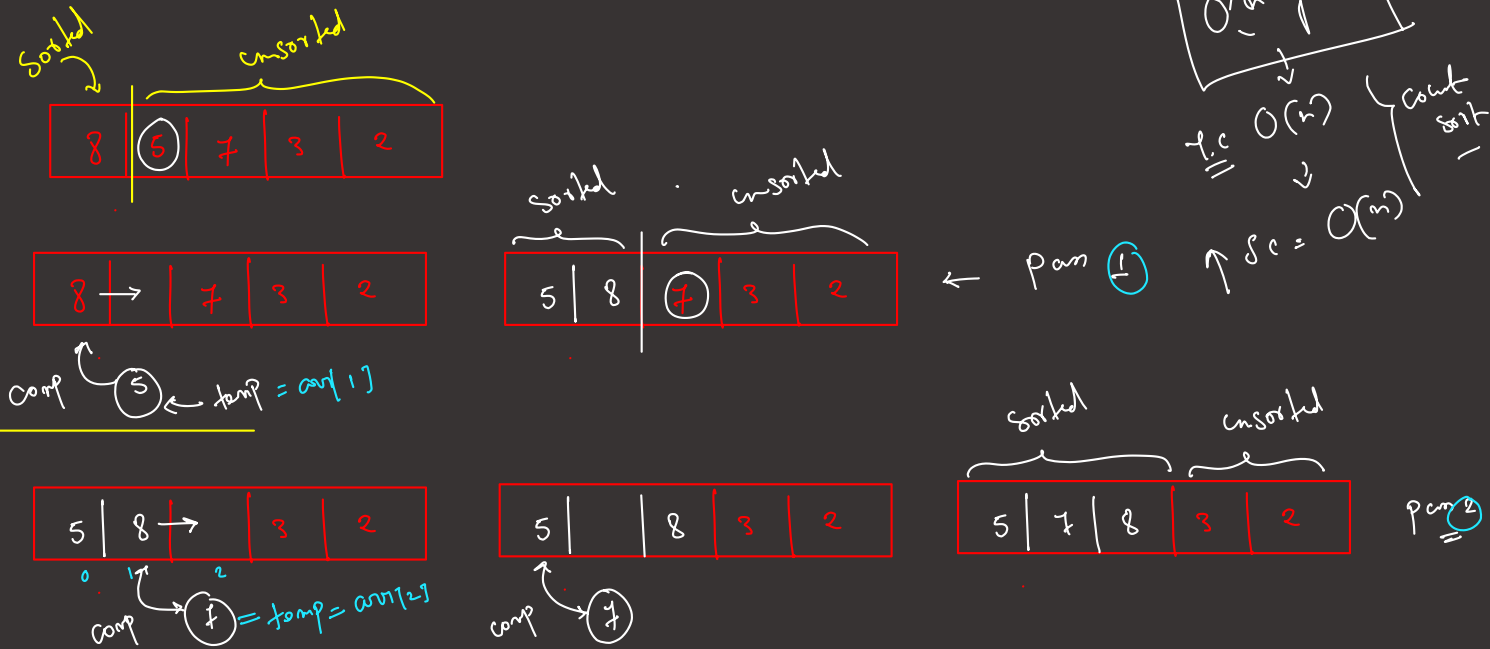


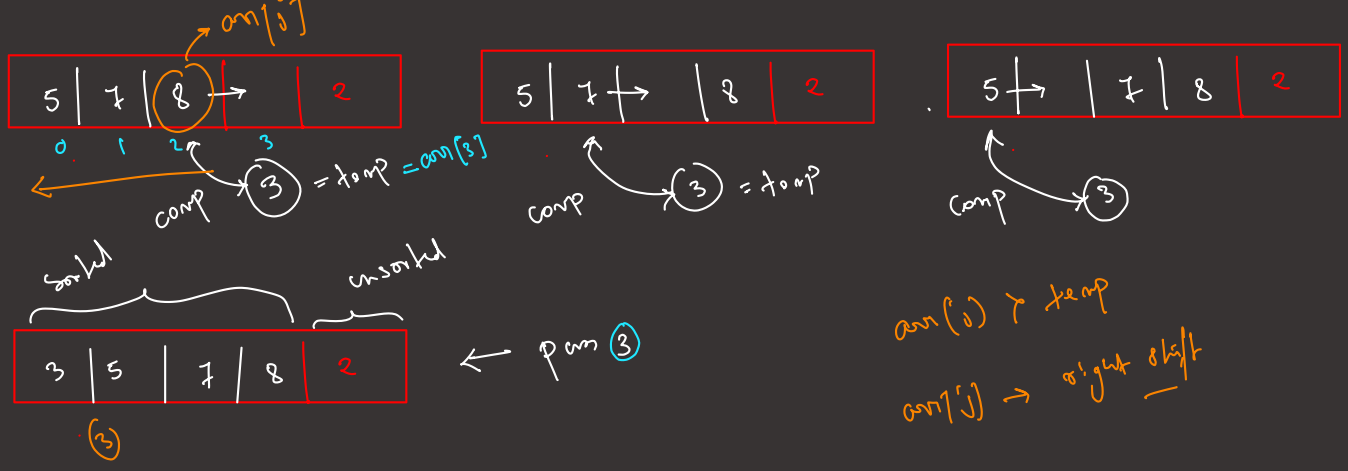


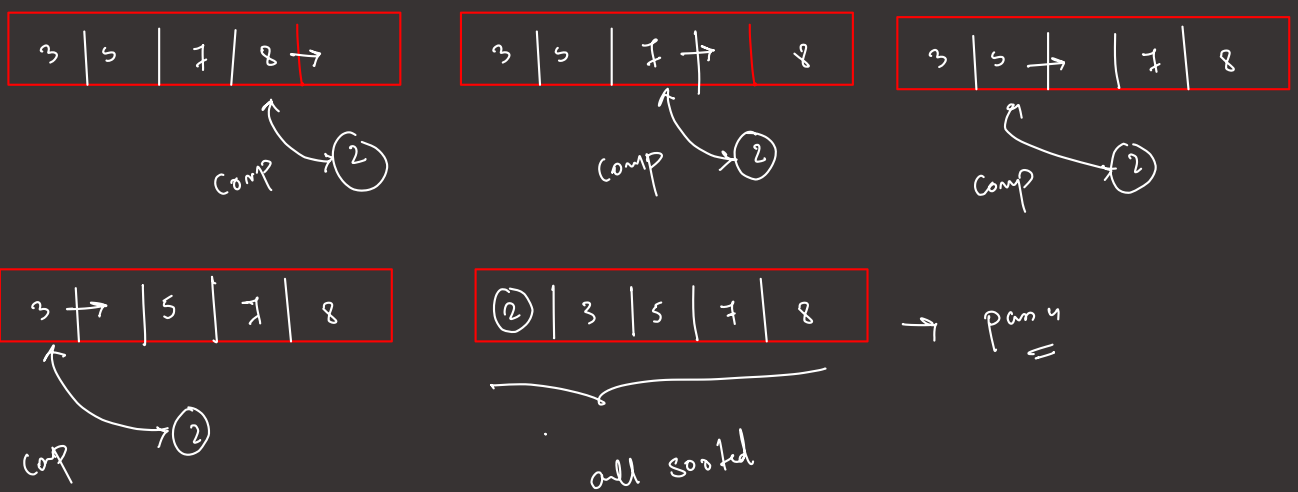
Insertion Sort



Pick an element from unsorted part and place it at the right position of the sorted array.







$$\#\text{pass} = (n-1)$$

$$\#\text{comp} = 1+2+3+\cdots+(n-1) = \frac{n(n-1)}{2}$$

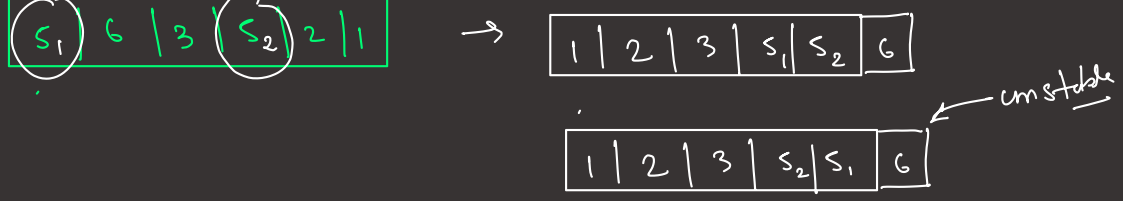
time comp^n : $O(n^2)$

$$\text{max } \#\text{swap} = \frac{n(n-1)}{2}$$

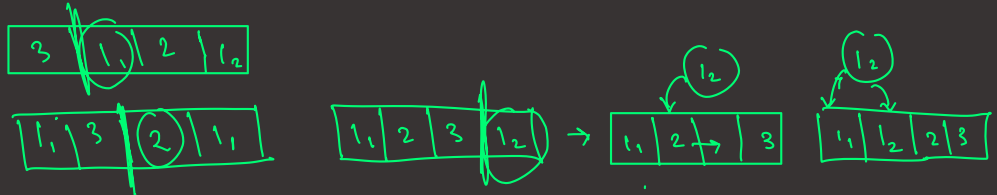
Insertion sort → Adaptive. → $O(n)$

Stability

[Relative position will be maintained]



Insertion sort is also stable.



Inbuilt Sort

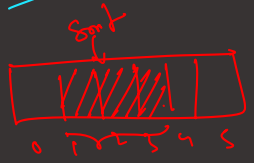
import java.util.Arrays ← class

int arr[] = { ..... }

Arrays.sort(arr) // asc order. // T.C = O(nlogn)

Arrays.sort(arr, s, e) → exclusive

s, e



1 to 3 sort

import java.util.Collections ← class

Arrays.sort(arr, Collections.reverseOrder()); // desc order sort

Arrays.sort(arr, s, e, Collections.reverseOrder());

↑

Integer arr[] = { ... }

{ int arr[] ✗

class

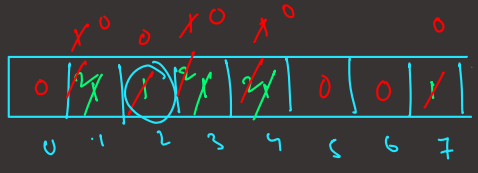
S.C = O(N) ↑

T.C = O(n)

Count Sort

Step (1)

Step (2) ✓ int count[] = new int[max+1]



← count[arr[i]]++

element → position of count arr.

Step (3) // Sorting