Bubble Sort - Sorting

* **Sorting** : Sorting is the process of arranging the data in ascending and descending order.

Ex :- 3, 1, 5, 4, 2
⇒ [1, 2, 3, 4, 5] ⇐ ascending ordered sorted

* **Bubble Sort** :

>>> It is basically a Comparison-sort method.
[Comparison sort means sorting step-by-step]

>>> It is also known as Sinking / Exchange sort.

>>> In every step, we are comparing adjacent elements.

* **Why Bubble Sort ?**

With Every Pass / Step, the largest element comes in the end.

Means

>>> With Pass no. 1, the largest element came at the end.

>>> With Pass no. 2, the 2nd largest element came at 2nd index from the last
and so on-----

Example

i = 0, j = 1

3 , 1 , 5 , 4 , 2

Pass 1 (i=0)

>>> Initially, $\boxed{i = 0 , j = 1}$

check, if [j] < [j-1] ⇒ Here, $\boxed{1 < 3}$ ⇒ Yes ⇒ $\boxed{\text{Swap}}$

⇒ 1 , 3 , 5 , 4 , 2

>>> Now, $\boxed{j = 2}$

Check, if [j] < [j-1] ⇒ Here, $\boxed{5 < 3}$ ⇒ No ⇒ No swapping

>>> Now, $\boxed{j = 3}$

⇒ 1 , 3 , 5 , 4 , 2

Check, if [j] < [j-1] ⇒ Here, $\boxed{4 < 5}$ ⇒ Yes ⇒ $\boxed{\text{Swap}}$

⇒ 1 , 3 , 4 , 5 , 2

>>> Now, $\boxed{j = 4}$

check, if [j] < [j-1] ⇒ Here, $\boxed{2 < 5}$ ⇒ Yes ⇒ $\boxed{\text{Swap}}$

⇒ 1 , 3 , 4 , 2 , 5

Now, j will index out of bound.

# Here, i = counter
j = internal loop

- Now, j will start again for second pass (i.e for i = 1)

Pass-2 (for i = 1)

$\overset{0}{1}, \overset{1}{3}, \overset{2}{4}, \overset{3}{2}, \overset{4}{5}$

» i = 1 , j = 1
check, if [j] < [j-1] ⇒ Here, $\boxed{3 < 1}$ ⇒ No ⇒ No swapping

⇒ 1 , 3 , 4 , 2 , 5

» j = 2
Check, if [j] < [j-1] ⇒ Here, $\boxed{4 < 3}$ ⇒ No ⇒ No swapping

⇒ 1 , 3 , 4 , 2 , 5

» j = 3
check, if [j] < [j-1] ⇒ Here, $\boxed{2 < 4}$ ⇒ Yes ⇒ $\boxed{\text{Swap}}$

⇒ 1 , 3 , 2 , 4 , $\boxed{5}$

Now, j = 4 , Now, we know that 5 is the largest element and After 2nd pass , 4 and 5 will be sorted

1 , 3 , 2 , $\boxed{4 , 5}$

So, we don't need to compare again & again.

So, with every loop , j is starting from index 1 and going till less than (length - i).

Pass-3 (for i = 2)

$\overset{0}{1}, \overset{1}{3}, \overset{2}{2}, \overset{3}{4}, \overset{4}{5}$

» j = 1
Check , if [j] < [j-1] ⇒ Here, $\boxed{3 < 1}$ ⇒ No ⇒ No swapping

1 , 3 , 2 , 4 , 5

» j = 2
check , if [j] < [j-1] ⇒ Here, $\boxed{2 < 3}$ ⇒ Yes ⇒ $\boxed{\text{swap}}$

$\boxed{1 , 2 , 3 , 4 , 5}$

Now, finally, we got the sorted array.

* Space Complexity = $O(1)$ // constant

[No extra space is required. i.e, copying the array, etc not required]

>>> This is also known as Inplace Sorting Algorithm.
[means, we don't have to create new/extra array].

* Time Complexity

Best case = $O(N)$ // when array is sorted

Worst case = $O(N^2)$ // when array is sorted in opposite
[For ex:- you are given an array sorted in descending order and we want to sort in ascending order]

Here,
N = no. of comparisons

NOTE : As the size of array is growing, the no. of comparisons is also growing.

Explaination of Best case :

Ex → 1, 2, 3, 4, 5 ← sorted array

In this case, loop runs for only one time (i.e for $i=0$)

NOTE : when j never swaps for a value of i, it means array is sorted. Hence, we can end the program.

In this case, i ran only one time and j ran (N-1) time. we ignore constant.

So, $\boxed{\text{no. of comparison} = N}$

Explaination of Worst case :

Ex :- 5, 4, 3, 2, 1 ← descending order

for $i = 0$, j ran $(N-1)$ times.
for $i = 1$, j ran $(N-2)$ times.
for $i = 2$, j ran $(N-3)$ times
for $i = 3$, j ran $(N-4)$ times.

So, Total Comparison $= (N-1) + (N-2) + (N-3) + (N-4)$

$$= 4N - (1+2+3+4)$$

$$= 4N - \left[N * \frac{(N+1)}{2}\right]$$

$$= 4N - \left(\frac{N^2+N}{2}\right) = \frac{8N - N^2 - N}{2}$$

$$= O\left(\frac{\cancel{7}N - N^2}{\cancel{2}}\right) = O(N^2)$$

* Stable Sorting Algorithm :

In this, order should be same, when value is same.

Ex → 10 20 20 30 10 ← original array

↓ sort

10 10 20 20 30 ← sorted array

In original array, black ball of 10 was before red ball of 10. And in the sorted array, this order is maintained. This is stable.

* Unstable Sorting Algorithm : In this, order are different, when value is same.

Ex → 10 20 20 30 10 ← original array

↓ after sorting

10 10 20 20 30 ← sorted array

Here, the order is not maintained