-> 18 - Jun - 2024 [WeekDay]

- Sorted Array | Non-decending ✓

Accending ↑ might have 2 same values

↑ $i < i+1^{th} < i+2^{nd}$

- Bubble Sort: In every step, compare adjust elems, if not correctly placed, swap them.

Worst Case Complexity: $O(n^2)$

- At each iteration, the rightmost (largest) unsorted elem gets correctly placed.

- Max passes/ iterations = len -1

- Keep track of var swapped at each inner iteration. if traversed through inner loop and still remain false, return and skip rest of iteration as it is already sorted.

- Aka Sinking sort / Exchange sort

- Outer loop i [0 to i < n-1]
  Inner loop j [1 to j < n-i]
  (check j to (j-1) ✓)

- Space Complexity : $O(1)$

No requirement of new array /space

↑

Aka inplace Sorting Algo.

- Best Case Time Complexity $O(n)$

↑

Already Sorted

- Worst Case Time Complexity $O(n^2)$

↑

Decending Array

- Complexity is how time/space grows as input increase

- It is [stable] sorting algorithm

↑

id1	id4		id2	id3
10	20	30	10	20

↓

id1	id2	id4	id3	
10	10	20	20	30

<— stable

id1 is before id2 ✓

Stable] Original order is maintain for the same values after sorting

Code for Bubble Sort]

ref ✓

```
public static ~~int[]~~ void bubbleSort (int[] arr){
    boolean swapped = false;
    for(int i=0; i< arr.length-1; i++){
        swapped = false;
        for(int j=1; j< arr.length-i; j++){
            if( arr[j] < arr[j-1] ){
                int temp = arr[j];
                arr[j] = arr[j-1];
                arr[j-1] = temp;
                swapped = true;
            }
            if(!swapped){
                break;
            }
        }
} ~~return arr;~~
```

-> Selection Sort

-> Select an elem and place it at its correct position in the array

-> Select the largest unsorted elem and swap it with its correct index position.

-> Again length-1 passes for worst case (decending array)

-> Can also take min instead max

- Space Complexity: Inplace ✓
  O(1)

- Best Case Time: $O(n^2)$ <- Has to find max every time

- Worst Case Time: $O(n^2)$

iter i -> n-1
iter i-1 -> n-2
...
iter last -> 0

Total Comparisions: $1+2+\dots+(n-1)$

$$= \frac{(n-1)(n-1+1)}{2}$$

$$= \frac{n(n-1)}{2}$$

$$= \frac{n^2 - \overset{\text{less dominating term}}{n^2}}{\underset{\text{<- const}}{2}}$$

$\approx(n^2)$ ✓

- Not Stable Sorting Algo

- Well performance on small lists