Sorting Algorithms:-

* Bubble Sort:

Used to sort arrays.

How?

We compare adjacent elements in every step.

And swap accordingly

WHY?

After every pass through Array, the largest element will be at right end.

It succeeds the same way with each pass.

* It is also known as SINKING / EXCHANGE Sort

Eg: 1, 3, 4, 2, 5 ⇒ 1, 3, 2, 4, 5 ⇒ 1, 2, 3, 4, 5
(i: 0, 1, 2, 3, 4)

which means for every i^{th} pass,

we check for $<= j - i$ elements

(or) $j - i - 1$

to save time as w.k.t after each pass largest is at end.

* Space complexity: O(1) // constant, No extra space.

Time Complexity :-

Best Case : $O(N)$ → Array already sorted

Worst Case : $O(N^2)$ → Array is sorted in reverse

- As size of array grows, no. of comparisons grow hence more time.

→ Best Case is when 'j' is never swaped.

It'll run for 1 pass only.

What if Stable and unstable Sorting Algorithm?

→ when the order of values is same even after sorting is called stable..

Eg: Each element represents balls of colour.

Suppose

$\frac{10}{B}$ $\frac{10}{R}$ $\frac{30}{B}$ $\frac{20}{B}$ $\frac{10}{G}$

⇒ After Sorting

$\frac{10}{B}$ $\frac{10}{R}$ $\frac{10}{G}$ $\frac{20}{B}$ $\frac{30}{B}$

Here, the order of value remains same after sorting, also.

⇒ $\frac{10}{B}$ $\frac{10}{G}$ $\frac{10}{R}$ $\frac{20}{B}$ $\frac{30}{B}$

. Here, the values are sorted but order

is broken or not maintained,
Hence, it is unstable.

---

* Insertion Sort :-

- for every index (element) you are at, put that element at its correct index of left hand side.

Working :-

for each pass, the suceeding index is sorted

Eg : 5, 3, 4, 1, 2

for (i=0; i<n-1; ++)
↓ prevent out of bound

After 1st pass :- 0, 1 indexes are sorted

⇒ 3, 5, 4, 1, 2 i = 0 ; j = 1

↳ always (j--) & always 7 0 //

After 2nd pass : 0, 1, 2 indexes are sorted

3, 4, 5, 1, 2 i = 1 ; j = 2

After 3rd pass : 0, 1, 2, 3 indexes are sorted

1, 3, 4, 5, 2 i = 2 ; j = 3

After 4th pass : 0, 1, 2, 3, 4 (Array) is sorted

1, 2, 3, 4, 5 i = 3 ; j = 4

* Array is sorted for every pass.

How the Algorithm works?

when i=0, j will be equal to (i+1) //.

where, 'i' stands for each pass.

So, for every pass, 'j' is checked to its left and sorted.

- So, at all times, 'j' must be > 0.

  and 'i' $\leq n-2$ (At last, to prevent out of bound)

- 'j' will break the loop when $! <$ left side as the left side is already sorted.

  So, it won't be smaller than other elements on left.

* Complexity of insertion Sort:

  worst case :- $O(N^2)$
  ↳ No. of elements

  Total Sum of 'N' = $\left(\frac{N^2 - N}{2}\right) \Rightarrow (N^2)O$ //

  Best case : When Array is already Sorted //

  $O(N)$ //

  It is linear, which is the speciality of this sort.

* Critical to know why use insertion Sort?

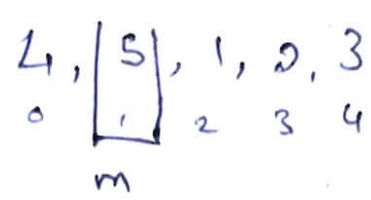
- Adaptive ⇒ steps are reduced if array sorted, swaps are reduced.
- It is stable and used for smaller values of 'N'.

- It works well when array is partially sorted.
→ takes part in hybrid sorting algorithms.
→ All sorting algos till here are not good for large data
- Always check for edge cases.

* Selection Sort

- The Algorithm where max element is found for each pass and swaped/placed in its correct index.

Eg:



⇒ 1st pass

4, 3, 1, 2, 5 ⇒ 2nd pass

2, 3, 1, 4, 5 ⇒ 3rd pass

2, 1, 3, 4, 5 ⇒ 4th pass

1, 2, 3, 4, 5 = Sorted Array

- (OR)

Selection Sort can work the same way with minimum value instead of maximum

Time Complexity: $\frac{N(N-1)}{2} \Rightarrow \frac{N^2-N}{2} \Rightarrow O(N^2)$

Worst Case & Best Case are the same ⇒ $O(N^2)$

- It not a stable algorithm.
- It performs well on small list/Array.