Sorting + Time Complexity 2

Agenda :

- Optimized bubble sort, Best case, worst case
- Selection sort
- Insertion sort

★ Optimised Bubble sort :

li = [1, 2, 5, 3, 4]

I^st pass => [1, 2, 3, 4, 5]

II^nd pass => no swap

III^rd pass => no swap

★ Time complexity

→ worst case : $O(n^2)$

→ best case : $O(n)$

★ Selection Sort :

$li = [\overset{i}{2}, 8, 5, 3, 9, 4, \underset{j}{\overset{\text{current min}}{1}}]$

current_min = min element of list.
Keep on iterating & find current_min in list

After find current min swap it

I^st pass ⇒ $[1 \,\vdots\, \overset{i}{8}, 5, 3, 9, 4, \underset{\text{current\_min}}{\overset{j}{2}}]$

II^nd pass ⇒ $[1, 2 \,\vdots\, \overset{i}{5}, \underset{c}{3}, 9, 4, 8]$

III^rd pass ⇒ $[1, 2, \underset{c}{3} \,\vdots\, \overset{i}{5}, \underset{j}{9}, 4, 8]$

pseudo code :

```
for i in range (n-1) :
        current_min = i

    for j in range (i+1, n) :

        if li[current_min] > li[j] :
            current_min = j

    if current_min ! = i :
        li[current_min], li[i] = li[i], li[current_min]
return li
```

★ Time Complexity :

T(C) ⇒ $O(n^2)$

This is the best case & worst case time complexity.

There is no scope for optimization.

★ Insertion Sort :

li = [4, 5 , 1, 3, 2]

unsorted list

sorted ↶

4 ┆, 5 , 1, 3, 2

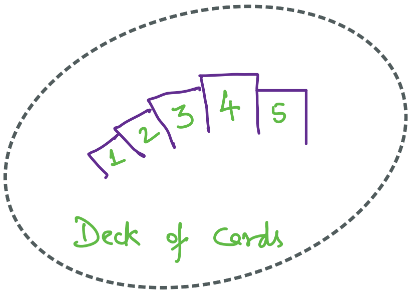
Pick one element in unsorted array & put it in its correct position in sorted array

index_to_insert

4 ┆ 5 1 3 2

j

virtual boundry



index_to_insert

4 5 ┆ 1 3 2

j

4 1 5 ┆ 3 2

j

index_to_insert

index_to_insert

1 4 5 ┆ 3 2

j

index_to_insert

1 3 4 5 ┆ 2

j

1 2 3 4 5 # Sorted Array.

Pseudo Code :

```
for i in range(1, n):

    index_to_insert = i
     j = i - 1

    while j >= 0 :          # move left

        if a[j] < a[index_to_insert]
                break

         a[j], a[index_to_insert] = a[a_t_i], a[j]

         index_to_insert = j
          j -= 1

 return a
```

✯ Time Complexity :

Worst Case = $O(n^2)$

Best Case = $O(n)$