Utility: Bubble Sort

- It is a comparison sort method.
- In every step we compare adjacent elements.

3, 1, 5, 4, 2

1st pass:

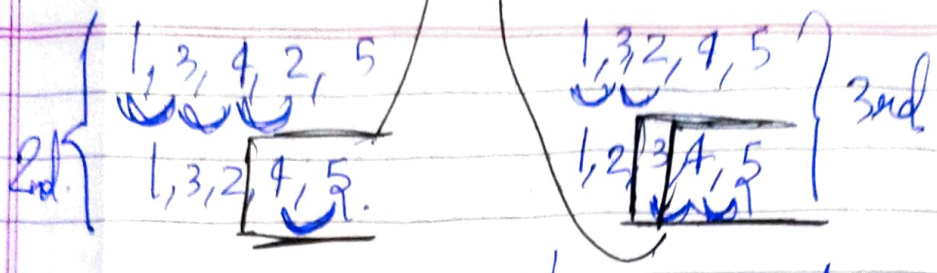
1, 3, 5, 4, 2

1, 3, 4, 5, 2

1, 3, 4, 2, 5

Largest element at the end.

We don't need to compare this part.



Bubble Sort / Sinking Sort / Exchange Sort

→ Outer loop → (runs n times)
Inner loop → (runs n-2 times).

Space complexity = O(1) // const. // No extra space required i.e. copying the array etc. not required.

↓
aka
Inplace sorting algorithms

Time Complexity → Best Case : O(N) ⇒ the array is sorted
Worst Case : O(N²) ⇒ the array is reverse sorted

Total comparisions in each pass

$$= (N-1) + (N-2) + (N-3) + (N-4)$$
$$= 4N - (1+2+3+4+5)$$
$$= 4N - \frac{N(N+1)}{2}$$
$$= 4N - \frac{N^2+N}{2}$$
$$= O\left(\frac{7N - N^2}{2}\right)$$
$$= O(N^2)$$

Stable Sort → When the original order for elements with equal value is maintained, it is known as stable sort, otherwise unstable sort.

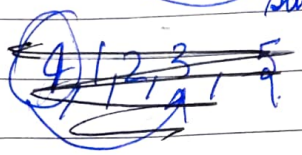
→ Bubble ~~Sort~~ Sort is stable sort

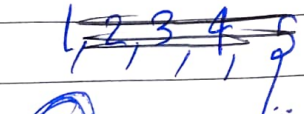
Selection Sort

→ We select a particular element and put it at its right position.

→ 4 5 1, 2, 3 — n-1 comparisions for finding the largest element.

Swap





→ 4, 3, 1, 2, 5 — n-2

→ 2 3, 1, 4, 5 — n-3

→ 2, 1, 3, 4, 5 — n-4

→ 1, 2, 3, 4, 5

n-i-1

1

Total comparisions → $0+1+2+3+\ldots+(n-1)$

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

Time complexity = $O(n^2)$ ← { Worst Case, Best Case

→ Not stable
→ Performs well on small arrays / lists.

Insertion Sort

0 1 2 3 4

5, 3, 4, 1, 2 — we sort parts by parts.
— finally first index
— then till second index
— then till third index
— — — — —

4, 3, 5, 1, 2

insert the next index element (1) at the correct position
exit left.

(Outer loop)
will run till $(n-2)$

$i \le n-2$ $i \ge 0$

0 1
1 2

i j
5, 3, 4, 1, 2

3, 5, 4, 1, 2

3, 4, 5, 1, 2

since (3<4) we break loop

3, 4, 5, 1, 2

3, 4, 1, 5, 2

3, 1, 4, 5, 2

1, 3, 4, 5, 2

2. }

1, 3, 4, 5, 2

1, 3, 4, 2, 5

1, 3, 2, 4, 5

1, 2, 3, 4, 5

Break loop

So i increments until j decrements

Complexity worst Case →

No. of Comparisons at every step = $O\left(\frac{(N-1)(N-1+1)}{2}\right)$

$1+2+3 \ldots \rightarrow (N-1)$

$= O(N^2)$.

Best Case (Array is already sorted).

O(N) Linear.

features →

- Adaptive – Steps get reduced if array is already sorted.
  - No. of swaps reduced compared to bubble sort.
- Stable.
- Used for smaller values of N.
- Works well when array is partially sorted.
- Takes part in hybrid sorting algorithm.

Cyclic Sort

Whenever the problem statement has numbers given from range 1 to N → use cyclic sort

After sorting →

0	1	2	3	4
1	2	3	4	5

index = value − 1

0	1	2	3	4
3	5	2	1	4

swap

2 5 3 1 4

5 2 3 1 4

4 2 3 1 5

1 2 3 4 5 loop ends aur.

4 swaps + 5 comparisons

= N + 1 + N

= 2N + 1

= O(N) ← Time complexity for worst Case