Sorting algorithms

14 June 2023 19:00

Bubble Sort: → Puts largest element at the end (sorted position)

arr = { 5, 3, 2, 4, 1 }

index: 0 1 2 3 4

{ if arr[0] > arr[1]
swap (arr[0], arr[1])

3 5 > 2 4 1

[3, 2, 5, 4, 1]

2, 3

if (arr[i] > arr[i+1])
swap (arr[i], arr[i+1])

3, 2, 4, 5, 1

3, 2, 4, 1, 5

[0, n-1)

1st time → largest element

comparison is done upto 4th element

that i → [0, 3]

[0, n-1) → open, not including

[0, n-2] → close, including

→ 3, 2, 4, 1, 5

2nd time → 2nd largest.

3, 2, 4, 1, 5

2, 3, 4, 1, 5

2, 3, 1, 4, 5

largest at 4th index

a[i] a[i+1]

3 > 2

3 < 4

4 > 1

2, 5, 1, 1, 2 4 > 1

[0, 2], [0, n-2)

(i) ki range [0, n-3]

3rd time

2, 1, 3, 4, 5 [0, 2] [0, n-4]

4th time

| 1, 2, 3, 4, 5 |

i → [0] , i+1 → 1

Array is sorted.

5 = n

< 4

Bubble sort.

```
for (int i=0; i<n-1; i++)
{ for (int j=0; j<n-i-1; j++)
    { if (arr[j] > arr[j+1]) → true
        swap(arr[j], arr[j+1]);
    }
}
```

i=0 n-1

i=1 n-2

i=2 n-3

→ j=0; j<n-i-1; i=3 n-4

i		
0	0 to	n-1)
1	0 to	n-2)
2	0 to	n-3)
3	0 to	n-4)

4 times

i = 4 → outer loop se bahar

---

my code will run

n-1 + n-2 + n-3 + n-4 .....

...... 1

1 + 2 + 3 + ... + n-1

$\sum n-1$ natural nos. (n-1) (n)

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

(10)

$\frac{10 \times 9}{2} = 45$

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

$\approx \frac{n^2}{2} - \frac{n}{2}$

(10) → (100)

$= O(n^2)$

$\frac{100 \times 99}{2} = 5050, 5450$

---

arr = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10] →

$\frac{10 \times 9}{2} = 45$

Optimising Bubble Sort

$i \in [0, 8]$

if (arr[i] > arr[i+1])
swap ( ) ⊗

5 0 0 × 4 9 9 → [499, 500] ✓

Before optimising for best case: $\frac{n(n-1)}{2} = O(n^2)$

1000 → $\frac{1000 \times 999}{2}$

After optimising for best case: $O(n)$ ④

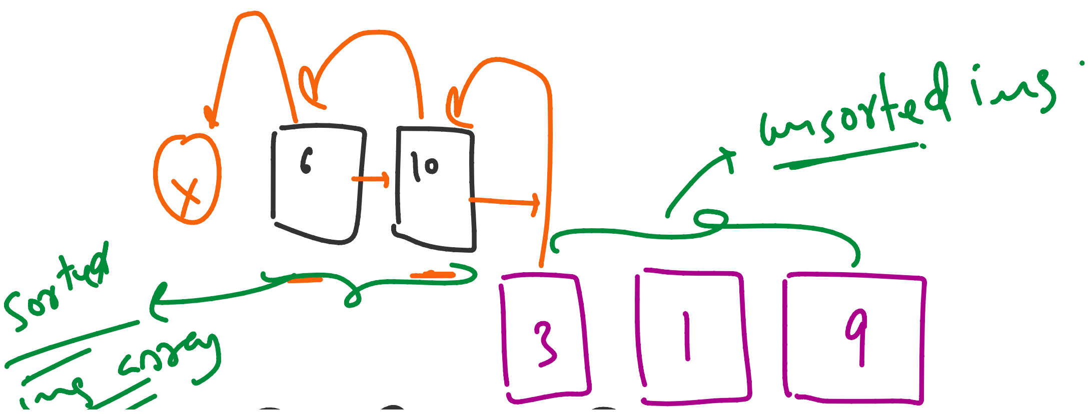
→ [1000]

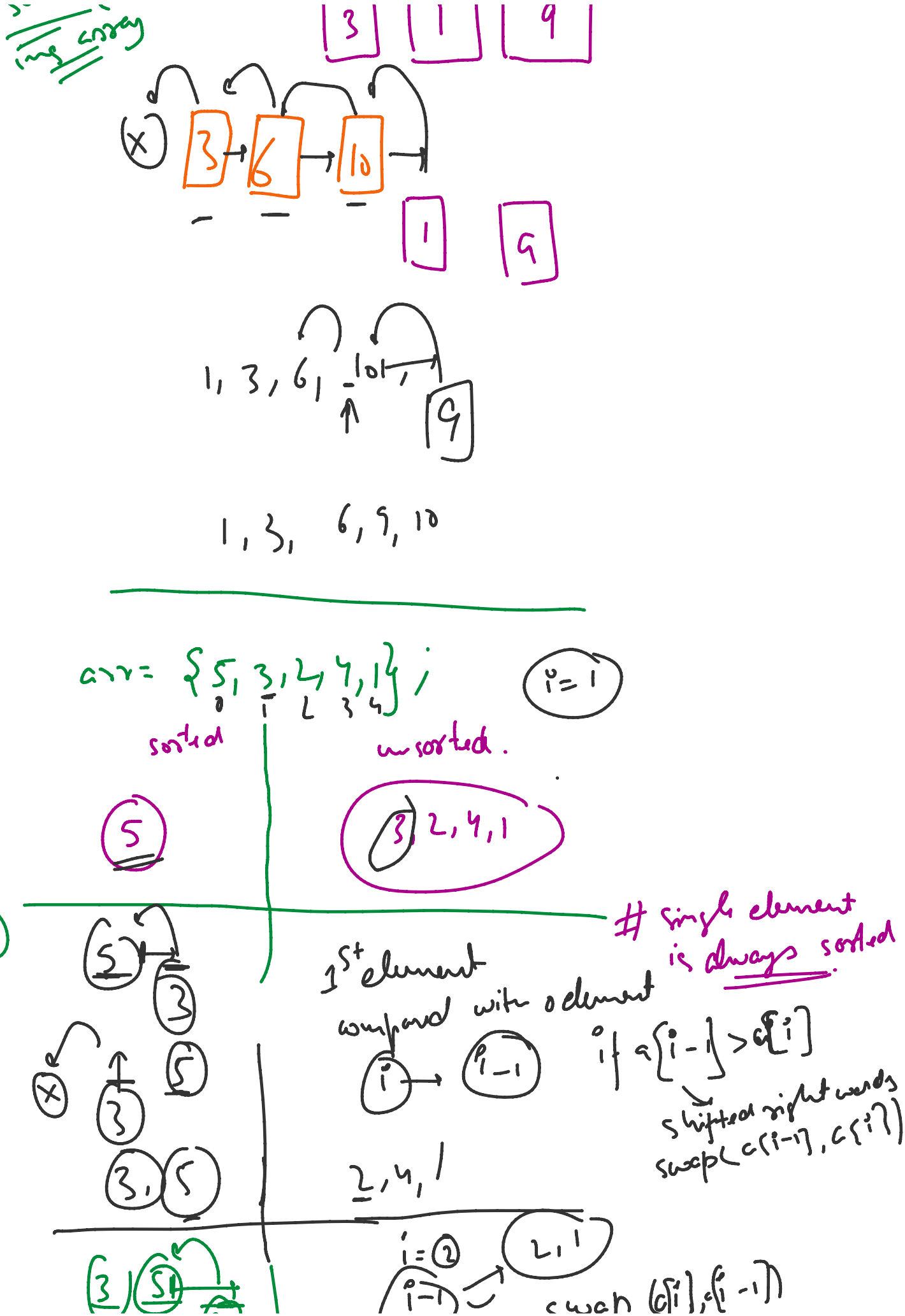
best case means → already sorted array.

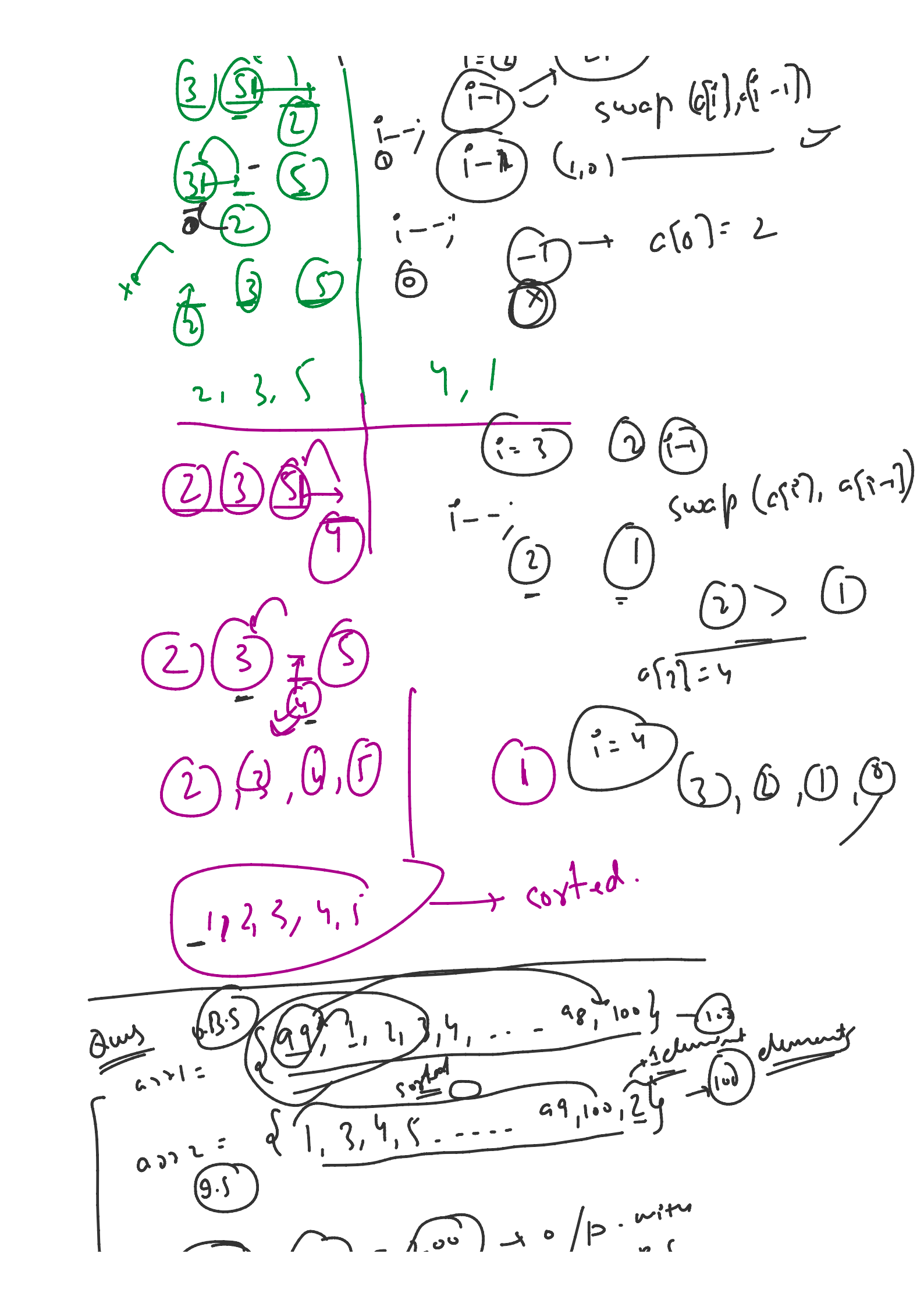
4th method → In built swap function use karna ka

swap (_ , _) ;

Insertion Sort: It is similar to the way we sort playing cards in our hands one by one

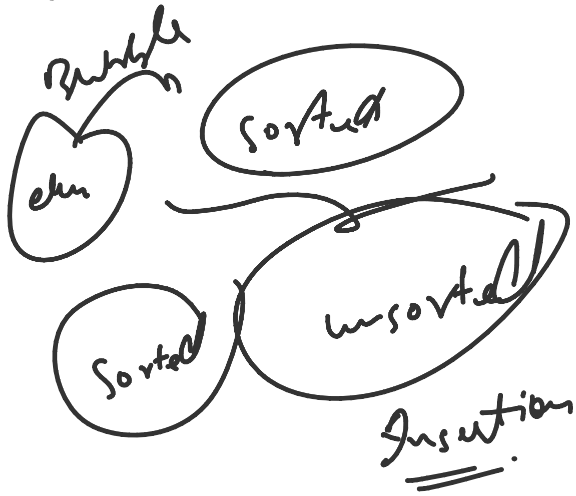



3
1
9
3
6
10
1
9
1, 3, 6, 10
9
1, 3, 6, 9, 10
arr = {5, 3, 2, 4, 1};
0 1 2 3 4
i = 1
sorted
unsorted.
5
3, 2, 4, 1
for (3)
# single element is always sorted
1st element
compared with o element
if a[i-1] > a[i]
shifted right words
swap(a[i-1], a[i])
2, 4, 1
i = 2
2, 1
i-1
swap(a[i], a[i-1])


swap (a[i], a[i-1])
i--;
a[0] = 2
2, 3, 5
4, 1
swap (a[i], a[i-1])
2 > 1
a[1] = 4
sorted.
Ans
element
element

100 + 100 = 200 → o/p with O.B.S

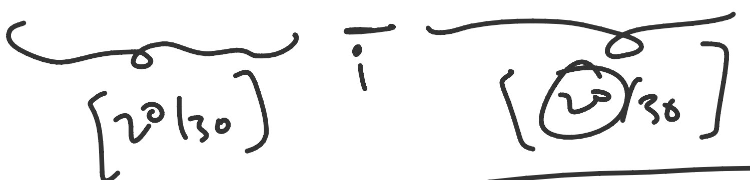
1 + 2 + 3 + 4 . . . . . . 100, 49 = very high



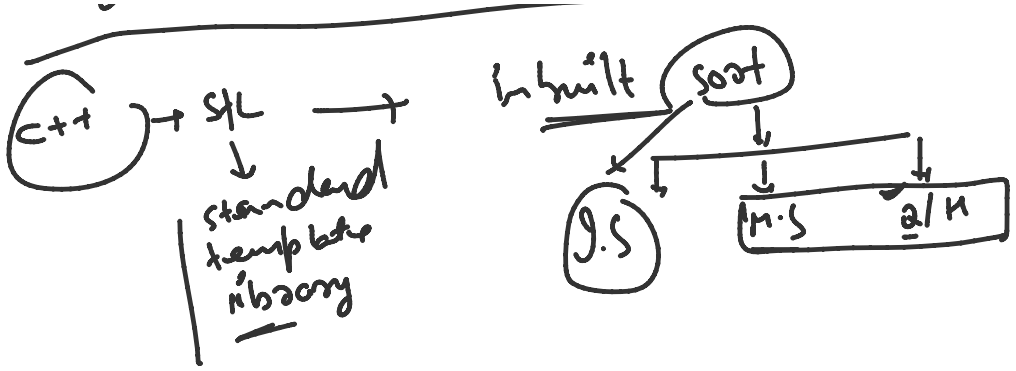
Insertion sort will take least time in case of almost sorted / sorted array.

\# If you are given an array and each element is k positions away from its sorted posn. then insertion sort is fastest (0 <= k <= 20, 25, 30)

array size = 10^6



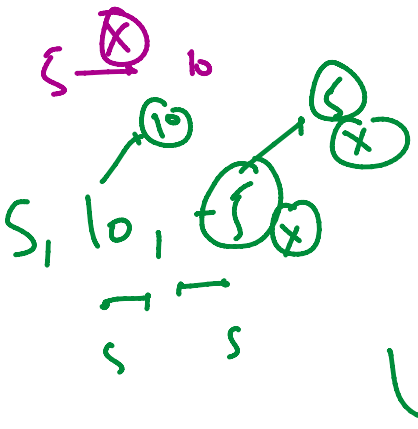
inbuilt sort



Insertion sort (most of the time) is fastest among these 3 sorting algo.

selection sort is always the slowest among these 3 algo.

ast :- [5, 10, -5]



+ive → ast moving rightwards

-ive → ast moving leftwards

ast moving in same dir, never collide

[5, 10]

i/p. [5, 10, -5]

o/p = [5, 10]

How many solution

1.

How many collision

ast1 = [1, 2, 3, 4, 5, ... 10] → 0

ast2 = [10, 9, 8 . . . . 1] → 0

ast3 = [-1, -2, -3, ... -10] 0

ast4 = [-1, -2, -3 . . . . . -9, 10] (0)

ast5 = [1, 2, 3, ... 9, -10] (9)

(9)

-ive	-ive	collision X
ith	i+1st	
-ive	+ive	
+	+	✓

[+ -]

ast1 = [1, 2, 3, 4, 5, 6, 7, 8, 9, -10]

X X X X X X X X X [-10]

ast2 = [1, 2, 8, 9, 7, 6, 5, 4, 3, 2, -9]

X X X X X X X X

[1, 2, 8]

[1, 2, 8, 9]

=8

[1, 2, 8, 9]