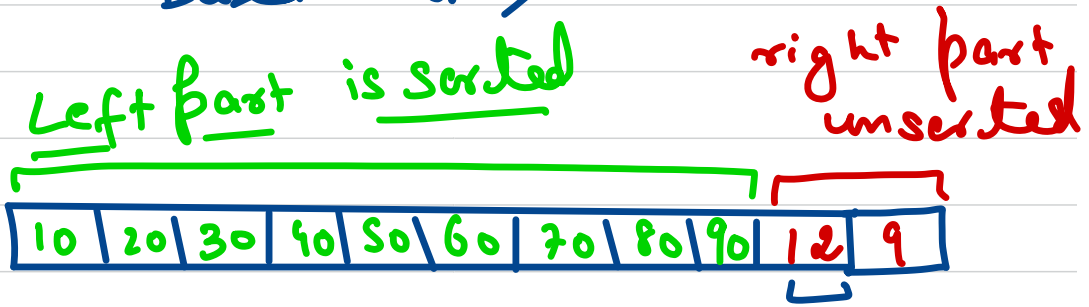
Insertion Sort (Comparison Based Sort)

el = 12

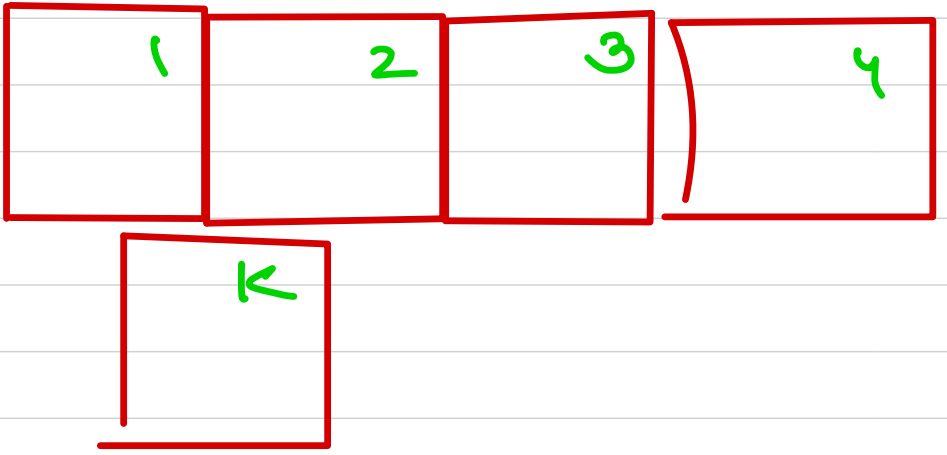
* Situation



i) Left part of the array is sorted.

ii) Right part has only one element unsorted.

Procedure → Pick the first unsorted element and insert it to its correct position.


1
2
3
4
K

Search for correct pos

Linear

13 17 22 25 31 18

13 17 18 22 25 31

el = 22

To the left of 18, find the first element lesser than 18. The correct pos of 18 is to the immediate right of it.

Binary Search

$O(n(\log n + n)) \Rightarrow O(n^2)$

shifting

finding correct place to insert

n = 5

i = 4

el = 3

j = 3 2 1 0

```
58  void insertion_sort(std::vector<int> &arr) {
59      int n = arr.size();
60      int i;
61      for(i = 1; i < n; i++) {
62          int el = arr[i]; // this is the eleme
63          int j = i - 1;
64          while(j >= 0 and arr[j] > el) {
65              arr[j+1] = arr[j];
66              j -= 1;
67          }
68          arr[j+1] = el;
69      }
70  }
```

Time Complexity →

$O(n^2)$ Worst

$\Theta(n^2)$ Avg

$\Omega(n)$ Best

1 2 3 4 5

2 3 4 5 1 → almost sorted

5 4 3 2 1 ①

4 5 3 2 1 ①

3 4 5 2 1 ②

2 3 4 5 1 ③

⋮

$(1+2+3 \cdots\cdots n-1)$

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

$\frac{n^2-n}{2}$ →

Space Complexity → O(1) → Because we are not using any extra ds

Is insertion sort inplace ??

Yes

Application

→ In case of almost sorted arrays it performs well.

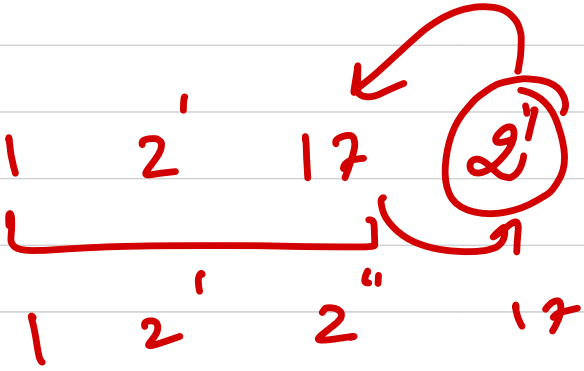
→ It is used as a subroutine in internal sorting functions like Tim Sort / Introsort

↓

heapsort / quicksort / insertion sort

Q. Is insertion sort stable ??

Yes



No. of comparisons → $O(n^2)$

no of shifts → $O(n^2)$