* Sorting:

It is an algorithm that allocates the elements in either ascending or descending order.

1. selection sorting.
2. Bubble sorting
3. insertion sorting.

Ex:

1	3	5	4	7	2

→ It is not in sorted order.

1	2	3	4	5	6

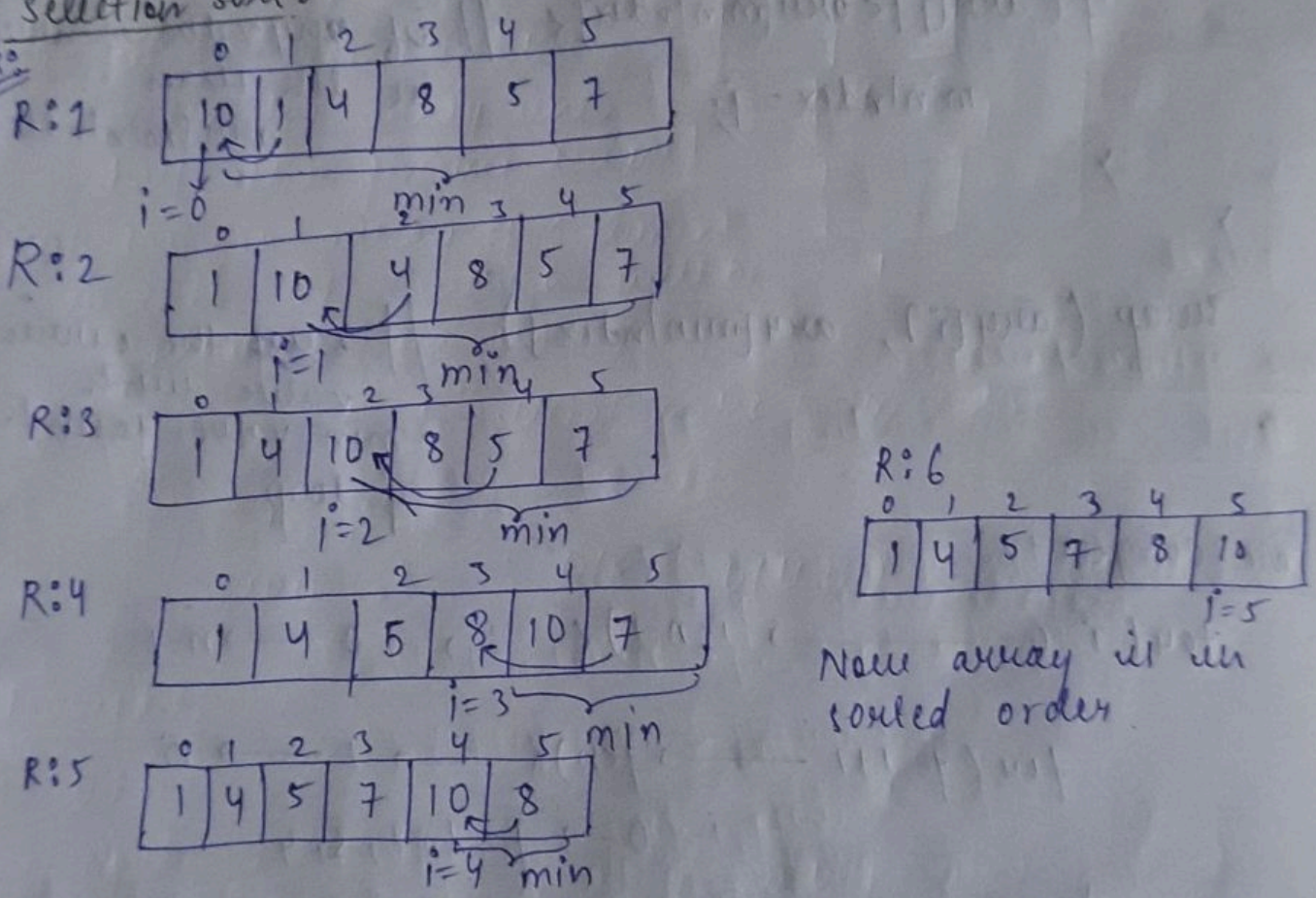
→ Inc: order

6	5	4	3	2	1

→ Dec order.

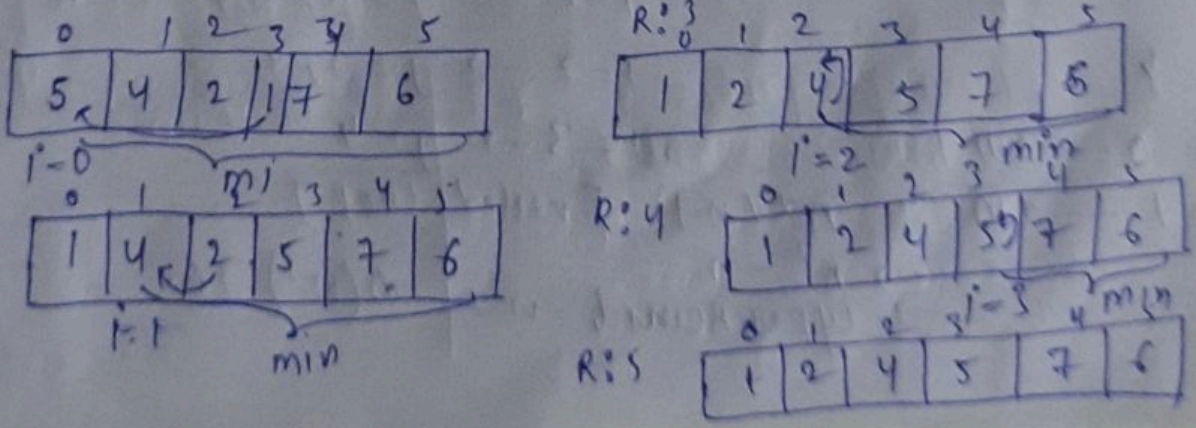
1. selection sort:

Ex:

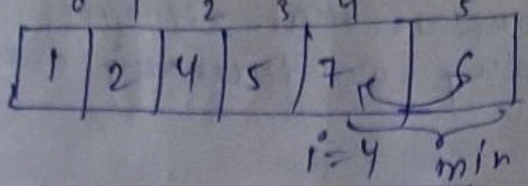


Now array is in sorted order.

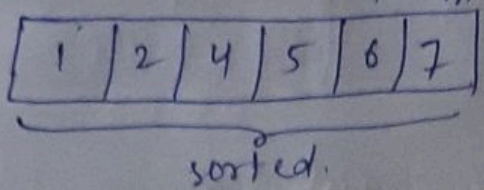
Ex: 2:



R: 5



R: 6




```
for(int i=0; i<n-1; i++){ // for every iteration we need to find the minimum (n-1) round.
    int minIndex = i; // min value is consider present at ith index
    for(j=i+1; j<n; j++){ // to find the minimum value from i+1 to n.
        if(arr[j]<arr[minIndex]){ // if any value of arr[i] is less than arr[minIndex]
            minIndex = j;
        }
    }
    swap(arr[i], arr[minIndex]); // swap the current value with min value found in j loop.
}
```

Time complexity:

for(i=0 ⟶ i<n-1){ ⟶ 0 ⟶ n-1

for(j= i+1 ⟶ j<n){ ⟶ 1 ⟶ n

i = 0 ⟶ j = 1, 2, 3, 4

i = 1 ⟶ j = 2, 3, 4

i = 2 ⟶ j = 3, 4

i = 3 ⟶ j = 4

i = 0 ⟶ n-1

j = 1 ⟶ n

n*(n-1) [nested loop]

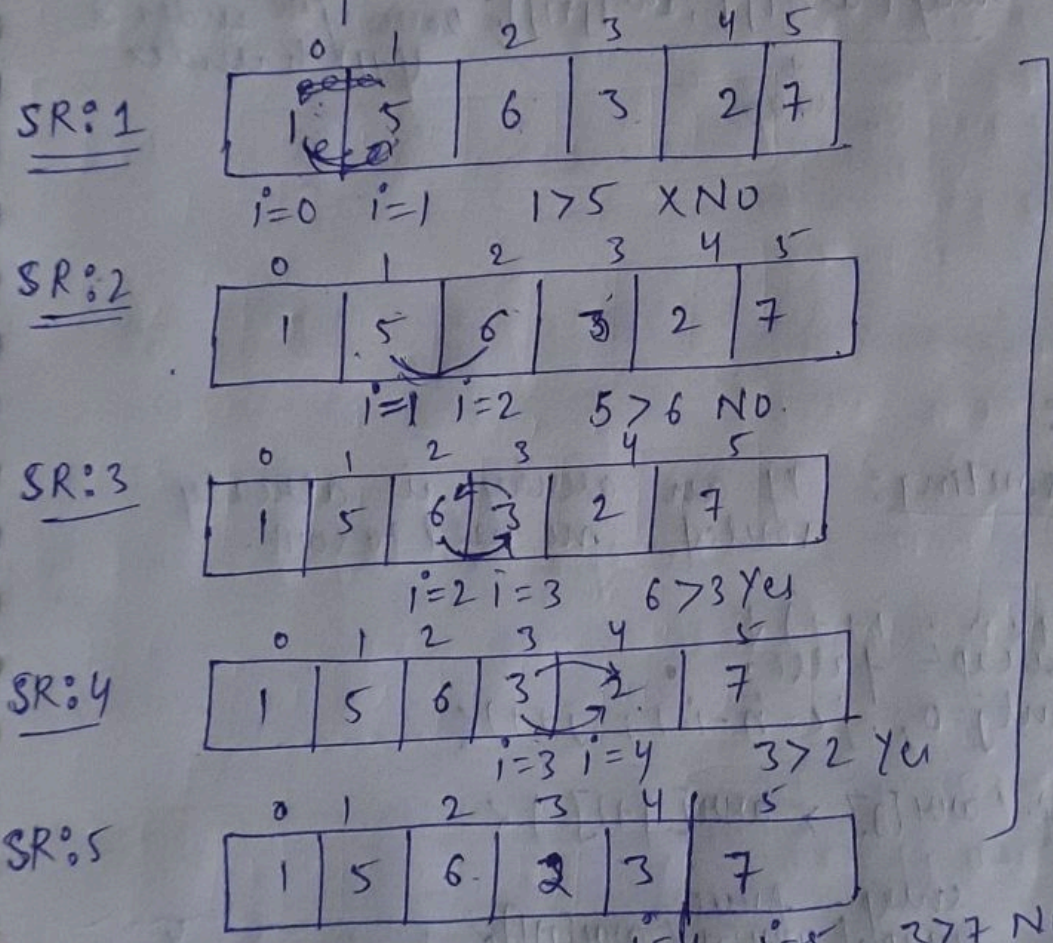
⟶ $n^2 - n$

⟶ upper Bound n^2 $O(n^2)$ TC.

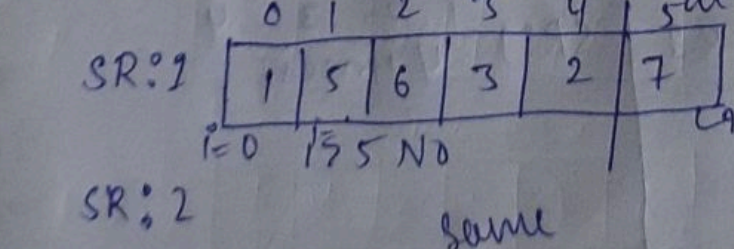
Space complexity:
S.C = O(1) as we don't create any extra space inside the function.

*Note: The sc doesn't depends upon the inputs we take in main function.
It depends upon the extra space we created in user defined function

(2) Bubble sort: For every round the ith ~~item~~ index will be compared with (i+1)th index if i index is greater than (i+1)th index then moves to right



Now 7 max element is present at last



R:1 → 4 times
R:2 → 3 times
R:3 → 2 times
R:4 → 1 time
R:5 → 0 time
→ loop runs

outer loop — inner loop

```
for(int i=1; i<n; i++){ // outer loop for no. of rounds-1 & it will run 4 times
    for(int j=0; j<=n-i-1; j++){ // inner loop to check how many time comparison done jth index & (j+1)th index
        if(arr[j]>arr[j+1]){
            swap(arr[j],arr[j+1]); // swap
        }
    }
}
print
```

More optimized way:

Best case in sorting: If an array is already sorted no need to sort.

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

1 2 3 4 5

R1 [(R:1) 4 times

Time complexity:

We check jth value greater than (j+1)th value mean swap done.

T.C for( ) {

for( ) {

}

$i=1 \longrightarrow i<n$

$j=0 \longrightarrow j<n-i-1$

$n(n-i-1)$

n^2-ni-n

$n^2-n(i+1)$

upper Bound

$T.C = O(n^2)$

For Best case when array is sorted

→ for( ) {

bool swap = false;

for( ) {

if (arr[j] > arr[j+1]) { → if this loop doesn't comes to true

}

* } if (swap == false) { // means it run for 4 times if swap == false break mean already sorted No need to move to next Round by iterating i.

break;

}

T.C = O(n) for Best case

S.C = O(1)

* **Worst case:** reverse case.

0	1	2	3	4
5	4	3	2	1

4 5 ④

4 5 3 ③

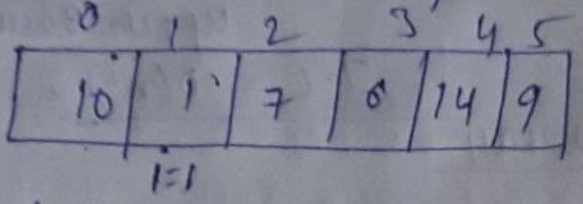
4 3 2 5 ②

4 3 2 1 5 ①

→ $\frac{n(n-1)}{2}$ $O(n^2)$

* We use selection sort when we have small array. We select minimum element & place at right place
* Bubble sort is used when ith largest elements needed to be shift in right. (maximum element to last place)

* Insertion Sort: Inserting the elements means picking the each elements & inserting at right place.



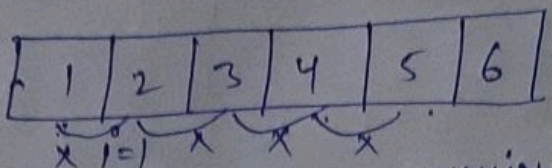
We will use insertion sort when it is partially sorted.

Step: A : fetch value
Step: B : Compare with the previous value
Step: C : shift the previous value if it is greater.
Step: D : copy the ~~element~~ value to previous value

→ arr[0] will not be considered as first ~~element~~ value as it will start from arr[1] as first element is considered to be sorted.

```
for (int i=0; i<n ; i++){ // outer loop for rounds
    int value = arr[i]; // to store initial value.
    int j= i-1;
    for( ; j>=0; j--){
        if (arr[j] > value){
            arr[j+1] = arr[j]
        }
        else {
            break;
        }
    }
    arr[j+1] = value;
}
```

* The Best case: when the array is already in sorted order.



As we see every previous value is less than current element. then it will check this times

T.C = O(n)
S.C = O(1)

* Worst case: for(i=1 ; i<n; i++){ → 1 ⟶ n

for (; j>=0; j--){ j = i-1 ⟶ 0

0 ⟶ 0

n ⟶ 0

(n-1) ⟶ 0

(n-2) ⟶ 0

}

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

T.C = $O(n^2)$

S.C = O(1)

$O(\frac{n^2}{2})$

$O(n^2)$