24/Feb/2023

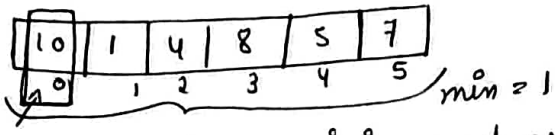
Sorting

Elements are arranged in either increasing order or decreasing order

① Selection Sort →

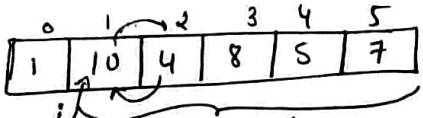
pick an element and put that element in it's correct place.

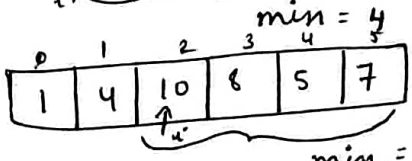
arr →

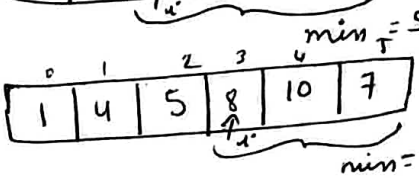


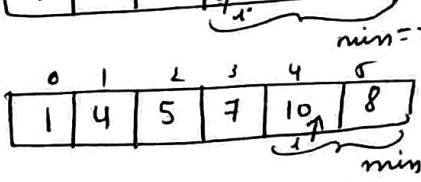
We will place minimum of the array in 0^{th} index.

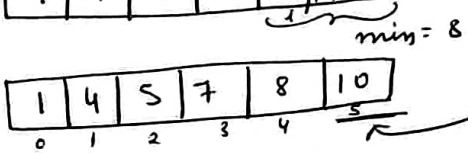
1 is the minimum, so we will swap it with the element at 0^{th} index.



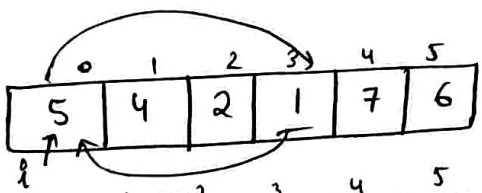






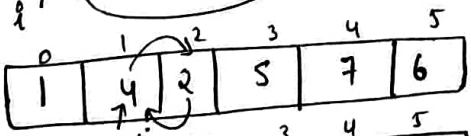


only one element left. If an array has only one element that means it is in it's correct place only.



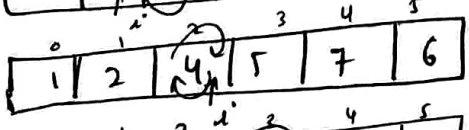
min = 1, swap 5 and 1

Round 1/Pass 1 →



min = 2, swap 4 and 2

Round 2 →

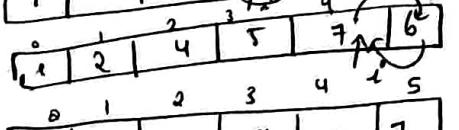


min = 4, swap 4 and 4

Round 3 →

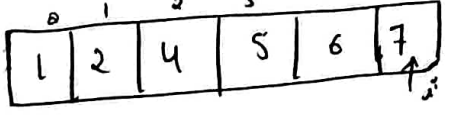
min = 5, swap 5 and 5

Round 4 →

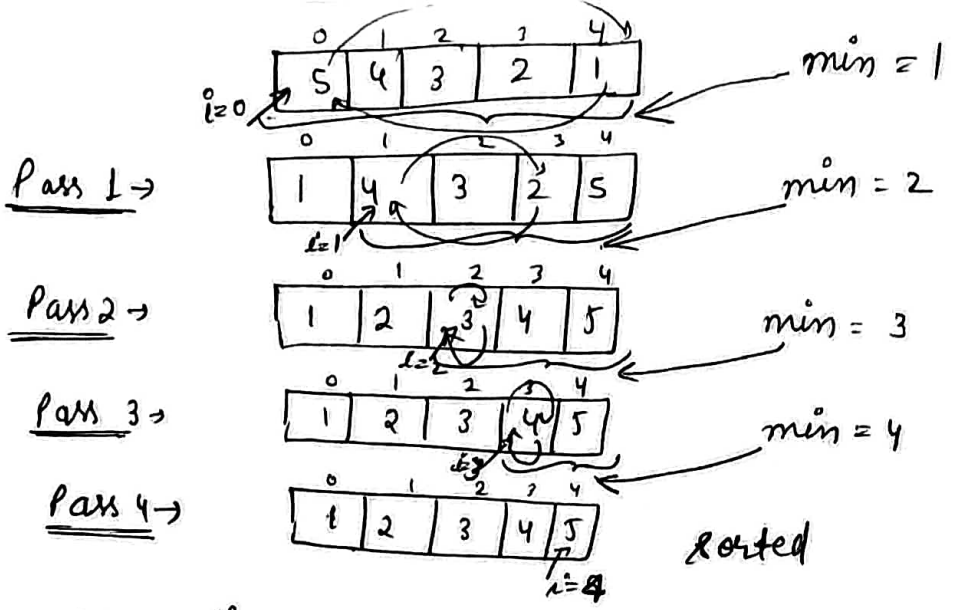


min = 6, swap 6 and 7

Round 5 →



sorted.

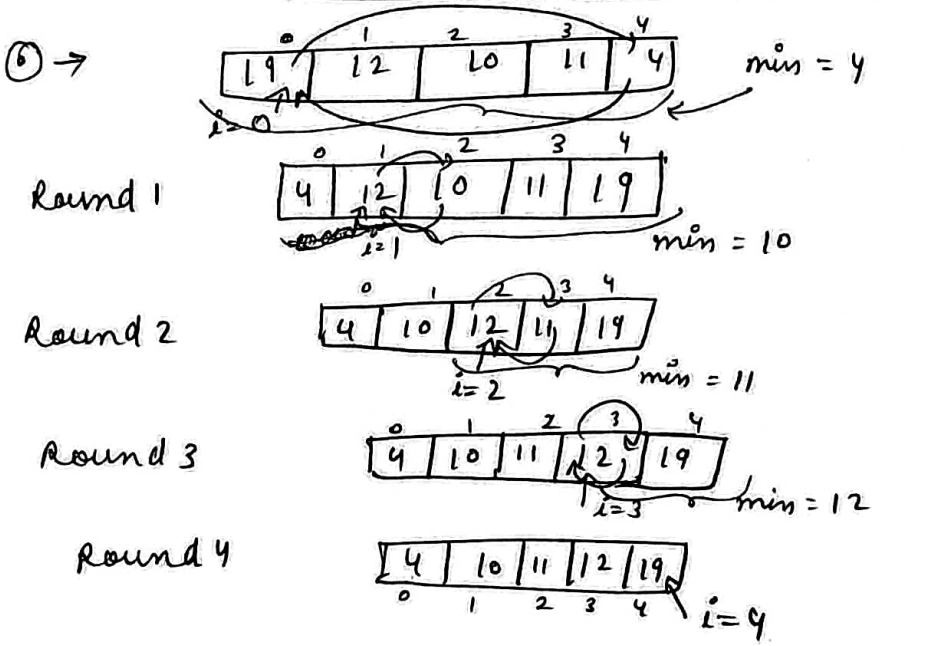


Observation

5 elements → 4 rounds

6 elements → 5 rounds

⇒ n elements → (n-1) rounds

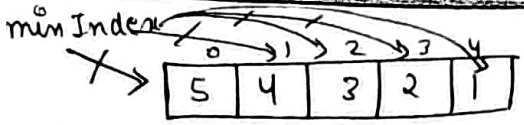



```
int main(){
    vector<int> arr{5,4,3,2,1};
    int n = arr.size();
    // Selection Sort
    for(int i=0; i<n-1; i++){
        int minIndex = i;
        for(int j=i+1; j<n; j++){
            if(arr[j] < arr[minIndex])
                minIndex = j;
        }
        swap(arr[i], arr[minIndex]);
    }
```

Outer loop → no. of rounds (n-1)

we will find minimum till second last element.

inner loop → index of minimum element in range i to n.



$i = 0$

minIndex = ~~0~~ ~~1~~ ~~2~~, ~~3~~ 4

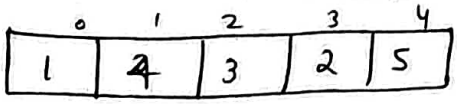
$j = i+1 \longrightarrow < n$

$j = 1$, ~~2~~ $j = 2$, ~~3~~ $j = 3$, $j = 4$

$i = 0$, ~~minIndex~~ minIndex = 4

swap (arr[0], arr[4])

Pass 1



$i = 1$

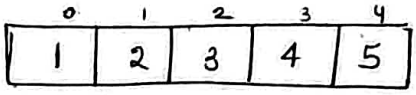
minIndex = ~~1~~ ~~2~~ 3

$j = i+1 \longrightarrow < n$

$j = 2$, $j = 3$, $j = 4$

$i = 1$, minIndex = 3 swap (arr[1], arr[3])

Pass 2



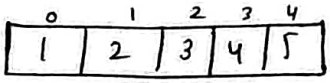
$i = 2$

minIndex = 2

$j = i+1$ to $< n$

$j = 3$, $j = 4$,

$i = 2$, minIndex = 2 swap (arr[2], arr[2])



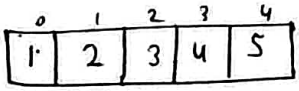
$i = 3$

minIndex = 3

$j = i+1$ to $< n$

$j = 4$, ~~5~~

$i = 3$, minIndex = 3 swap (arr[3], arr[3])



Time complexity

$i = 0$
$j = 1$ to $n-1$

$i = 1$
$j = 2$ to $n-1$

for eg, $n = 5$

i.e. $i = 0$ ↳ $j = 1, 2, 3, 4$

$i = 1$ ↳ $j = 2, 3, 4$

$i = 2$
$j = 3, 4$ ....

$1 + 2 + 3 + 4 + \cdots + (n-1) = \frac{n(n-1)}{2}$

$T.C = O\left(\frac{n^2}{2}\right)$

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

```
for ( i=0 ——— <n){        → n times
    for ( j= i+1 to  <n){  → n times
    — —
    }
}
```

In worst cases

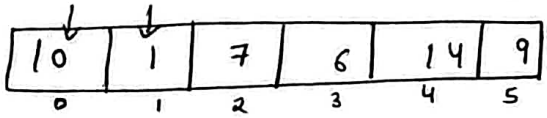
so $T.C = O(n \times n)$

$$\boxed{T.C = O(n^2)}$$

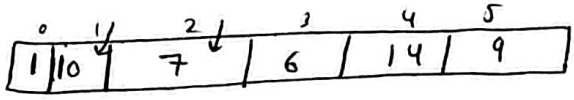
and $\boxed{S.C = O(1)}$

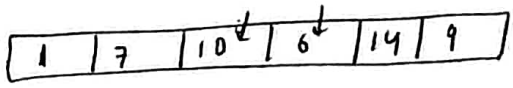
② **Bubble Sort** → In i^{th} round, i^{th} largest element is placed in it's right position.

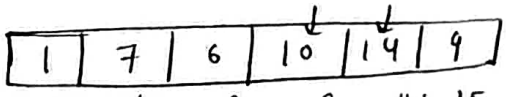
Round 1:

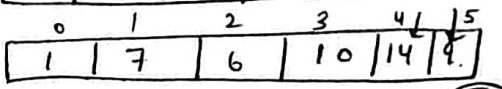


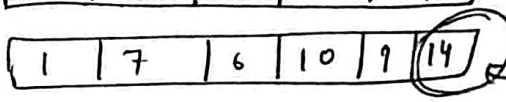
Round 1:



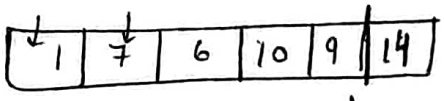


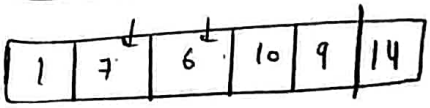


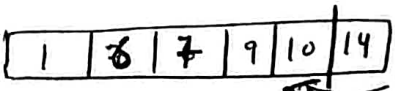




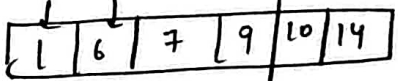
Round 2-

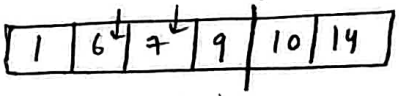


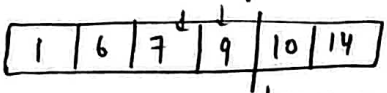


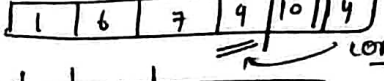


Round 3-

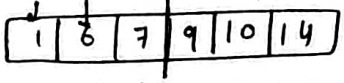


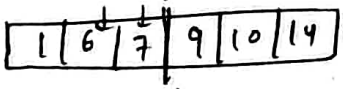


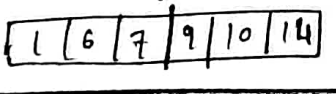




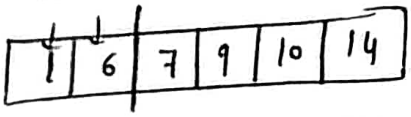
Round 4 →

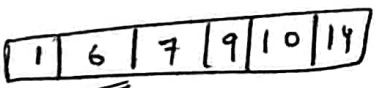






Round 5 :-





$$\boxed{T.C = O(n^2)}$$

// bubble sort - code →

```
for (int round = 1 ; round < n ; round ++){ ⟶ n times
                = 0          n-1
    for (int j = 0 ; j < n - round ; j++){ ⟶ n times
                       n-round-1
        if (arr[j] > arr[j+1])
            swap (arr[j], arr[j+1]);
    }
}
```

T.C = O(n²)

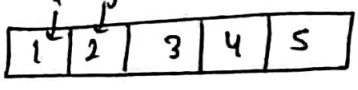
⑥ Can we optimize bubble sort?

Yes, if no swap happend in any round, ~~we will~~ that means our array is sorted and we will break ~~the~~ and come out of loop.

```
for (int round = 1 - - - - ){
    bool swapped = false;
    for (int j = - - - - ){
        if (arr[j] > arr[j+1])
            swapped = true;
            swap (arr[j], arr[j+1])
    }
    if (swapped == false)
        break;
}
```

This will reduce the T.C from $O(n^2)$ to $O(n)$ in best case. when the given array is already sorted.

eg:



comparisons

1 > 2 → F
2 > 3 → F
3 > 4 → F
4 > 5 → F

} only four comparisons

n elements → (n-1) comparisons.

So T.C = O(n).

Average and worst time complexity = $O(n^2)$

S.C = O(1)

Selection Sort → Used in small size anays.

Bubble sort → Used when we have to place i^{th} largest element in it's correct place.

Bubble sort is also called Sinking sort.

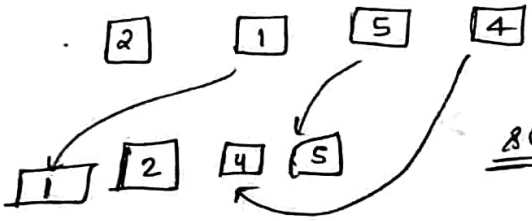
⑥ **Stable or Unstable algo.**

Stable algo → Algos which preserve the order of elements are stable.

Unstable algo → Algos which do not preserve order of elements.

④ Find out selection sort and bubble sort are stable or Unstable.

③ **Insertion sort** → Pick elements one by one and insert at their correct position.



sorted.

10	1	7	6	14	9
0	1	2	3	4	5

1 10

~~1>10~~ → F → left of 10

7>10 → f → left of 10

7>1 → right of 1

and so on.

Steps

① fetch the number
val = 1

② comparison
1 < 10

③ shift
swift 10

④ Copy
empty space में copy.

Round 1 - i = 1

val = 1

1<10 ⇒ 1 को 10 से पहले रखना है।

swift 10

copy 1.

1	10	7	6	14	9

Round 2 - i = 2

val = 7 10 > 7

7 > 1

1	7	10	6	14	9

Round 3- i = 3 val = 6 10>6 , 7>6 , 1<6

1	6	7	10	14	9
0	1	2	3	4	5

Round 4- i = 4

val = 14

14>10

1	6	7	10	14	9
0	1	2	3	4	5

Round 5- i = 5

val = 9

14>9 , 10>9, 9≥7

1	6	7	9	10	14

```
//insertion sort code

for (int round =1; round <n; round ++){
        // Step 1 → fetch
        int val = arr[round];
        // Step 2 → Compare  int j;
        for (~~int~~ j= round -1; j>=0; j--){
                if (arr[j]> val)

                        arr[j+1]= arr[j];

                else

                        break;

        }

}
```

10	1	7	6	14	9

j=-1 j=0 i=1

R1 → i = 1

val = 1

10>1

↳ shifting

arr[j+1] = val

arr[-1+1] = 1 ⇒ arr[0] = 1

R2 →

1	10	7	6	14	9

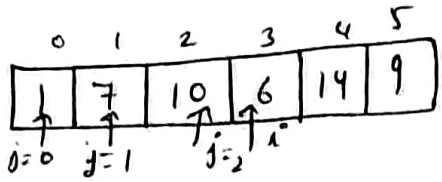
j=0 j=1 i=2

10>7 → shifting

1<7 → break

arr[j+1] = val

$R_3 \rightarrow$ $i = 3$



val = 6

10 > 6 → shift

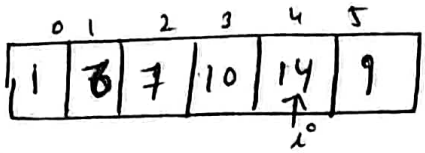
7 > 6 → shift

1 < 6 → break

arr[j+1] = value

arr[0+1] = 6

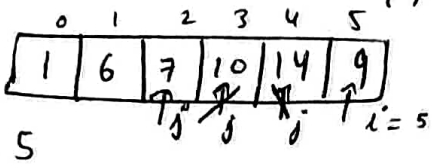
arr[1] = 6



$R_4 \rightarrow$ $i = 4$

val = 14

14 > 10 → break.



$R_5 \rightarrow$ $i = 5$

val = 9

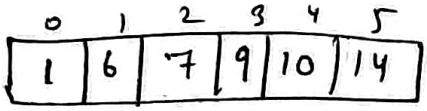
14 > 9 → shift

10 > 9 → shift

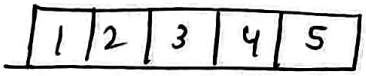
7 < 9 → break.

arr[j+1] = val

arr[2+1] = 9 ⇒ arr[3] = 9



Best Case



T.C = O(n) (Best case)