

Sorting
↓ Elements are arranged
in either increasing order
or decreasing order

↓ pick an element and put that element in its correct place.

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


min = 5

0	1	2	3	4	5	6
1	4	5	8	10	7	

↑

1	4	5	7	8	10
0	1	2	3	4	<u>5</u> R

only one element that means it is in its correct place only.



1 2 4 5 6 7 8 9 10

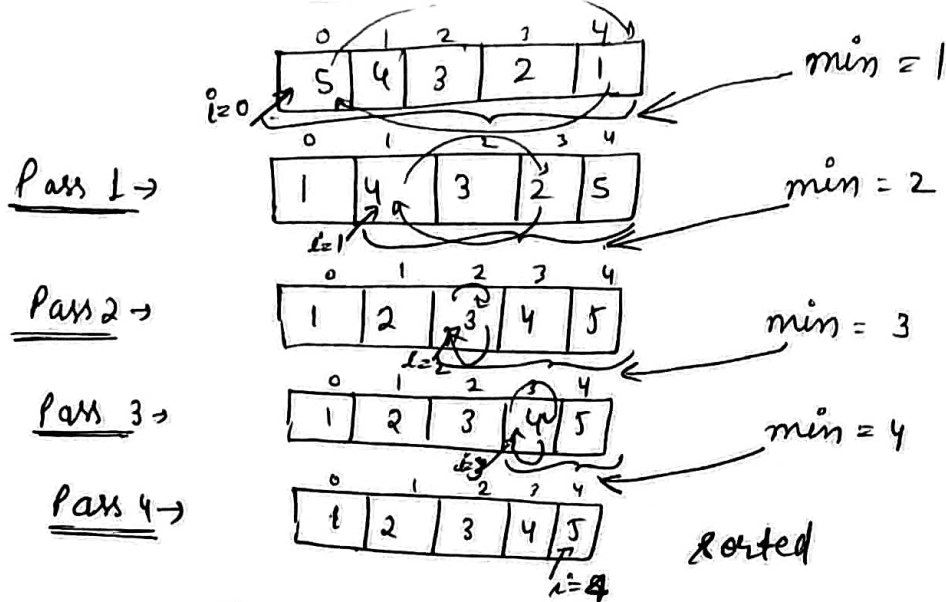
⁰	¹	²	³	⁴	⁵
1	2	4	5	7	6

Diagram 1 (Top): A 6x6 grid with numbers 1, 2, 4, 3, 7, 5 in the top row. Arrows indicate a path from the top-left cell to the top-right cell.

Diagram 2 (Bottom): A 6x6 grid with numbers 0, 1, 2, 3, 4, 5 in the top row. Arrows indicate a path from the top-left cell to the top-right cell.

1	2	4	5	6	7	↑ j
---	---	---	---	---	---	--------

sorted.

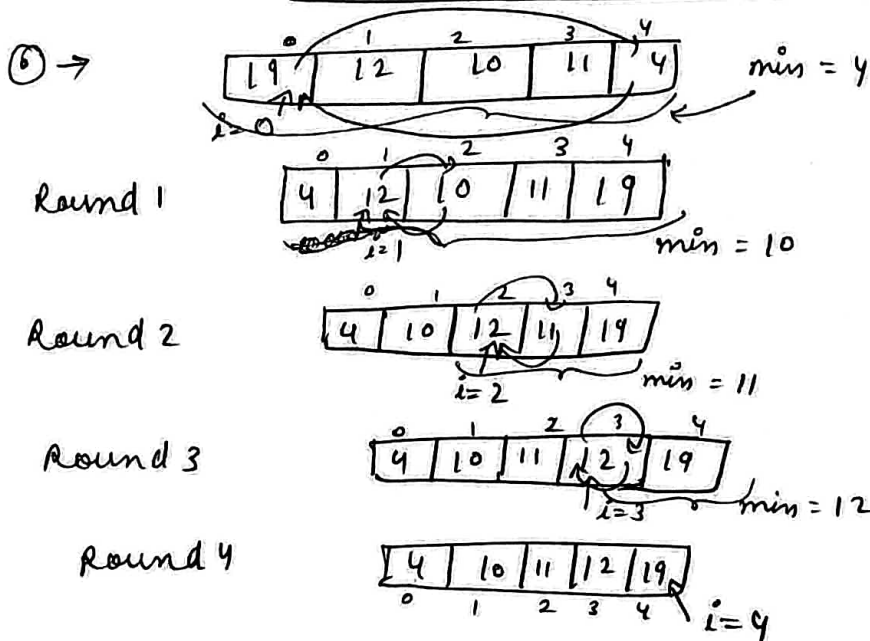


Observation

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;

}

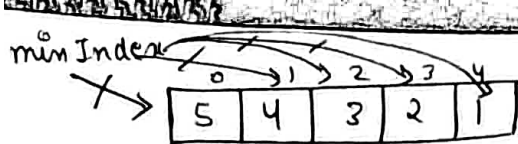
swap(arr[i], arr[minIndex]);

}

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~~

$j = i+1 \rightarrow < n$

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

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

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

Pass 1

$i = 1$

0	1	2	3	4
1	4	3	2	5

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

$j = i+1 \rightarrow < n$

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

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

Pass 2

$i = 2$

0	1	2	3	4
1	2	3	4	5

minIndex = 2

$j = i+1 \rightarrow < n$

$j = 3, j = 4$

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

0	1	2	3	4
1	2	3	4	5

$i = 3$

minIndex = 3

$j = i+1 \rightarrow < n$

$j = 4$

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

0	1	2	3	4
1	2	3	4	5

Time Complexity

$i = 0$

$j = 1$ to $n-1$

$i = 1$

$j = 2$ to $n-1$

for eg, $n = 5$

$i = 0 \rightarrow j = 1, 2, 3, 4$

$i = 1 \rightarrow j = 2, 3, 4$

$i = 2$

$j = 3, 4$

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

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

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

for ($i=0$ — $<n$) { \rightarrow n times }
 for ($j=i+1$ to $<n$) { \rightarrow n times } } In worst cases
 }
 } so $T.C = O(n \times n)$

and $S.C = O(1)$

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

② Bubble Sort \rightarrow In i^{th} round, i^{th} largest element is placed in its right position.

Round 1:

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

Round 1:

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

1	7	10	6	14	9
---	---	----	---	----	---

1	7	6	10	14	9
---	---	---	----	----	---

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

1	7	6	10	9	14
---	---	---	----	---	----

correct place

Round 2:

1	7	6	10	9	14
---	---	---	----	---	----

1	7	6	10	9	14
---	---	---	----	---	----

1	6	7	10	9	14
---	---	---	----	---	----

1	6	7	9	10	14
---	---	---	---	----	----

correct pos.

Round 3:

1	6	7	9	10	14
---	---	---	---	----	----

1	6	7	9	10	14
---	---	---	---	----	----

1	6	7	9	10	14
---	---	---	---	----	----

1	6	7	9	10	14
---	---	---	---	----	----

correct pos.

Round 4:

1	6	7	9	10	14
---	---	---	---	----	----

1	6	7	9	10	14
---	---	---	---	----	----

1	6	7	9	10	14
---	---	---	---	----	----

Round 5:

1	6	7	9	10	14
---	---	---	---	----	----

1	6	7	9	10	14
---	---	---	---	----	----

correct pos.

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

// bubble sort - code →

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

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

⑥ Can we optimize bubble sort?

Yes, if no swap happened in any round, we will break ~~the~~ and come out of loop.

```
for (int round = 1; round < n; round++) {
    bool swapped = false;
    for (int j = 0; j < n - round; j++) {
        if (arr[j] > arr[j+1]) {
            swapped = true;
            swap(arr[j], arr[j+1]);
        }
    }
    if (!swapped) 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,

1	2	3	4	5
---	---	---	---	---

comparisons

$1 > 2 \rightarrow F$
 $2 > 3 \rightarrow F$
 $3 > 4 \rightarrow F$
 $4 > 5 \rightarrow F$

} 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 \rightarrow Used in small size arrays.

Bubble sort \rightarrow Used when we have to place i^{th} largest element in its correct place.

Bubble sort is also called Sinking sort.

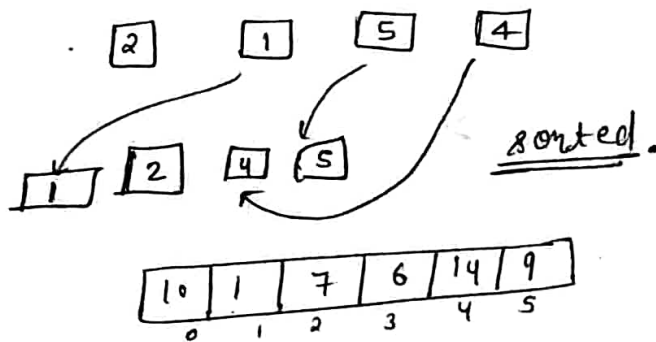
② Stable or Unstable algo.

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

Unstable algo \rightarrow Algos which do not preserve order of elements.

⑥ Find out selection sort and bubble sort are stable or unstable.

③ Insertion sort \rightarrow Pick elements one by one and insert at their correct position.



1 10

Steps
~~1~~ $1 > 10 \rightarrow$ left of 10
 $7 > 10 \rightarrow$ left of 10

$7 > 1 \rightarrow$ right of 1

and so on.

① fetch the number
val = 1

② comparison
 $1 < 10$

③ shift
shift 10

④ copy
empty space \neq copy.

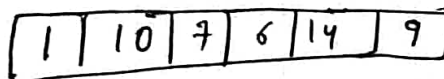
Round 1 - $i = 1$

val = 1

$1 < 10 \Rightarrow$ 1 को 10
के पहले रखना है।

shift 10

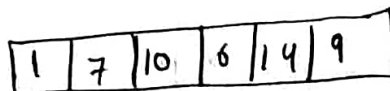
copy 1.



Round 2 - $i = 2$

val = 7

$10 > 7$
 $7 > 1$



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-

$i = 5$

$val = 9$

$14 > 9$, $10 > 9$, $9 \geq 7$

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
$j=0$	$i=1$				

R1 → $i = 1$

$val = 1$

$10 > 1$

↳ shifting

$arr[j+1] = val$
 $arr[-1+1] = 1 \Rightarrow arr[0] = 1$

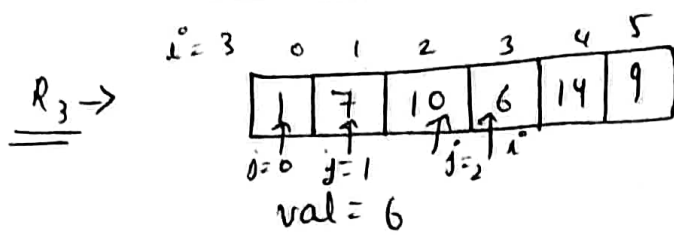
$10 > 7 \rightarrow$ shifting

$1 < 7 \rightarrow$ break

$arr[j+1] = val$

R2 →

1	10	7	6	14	9
$j=0$	$j=1$	$i=2$			



$10 > 6 \rightarrow \text{shift}$

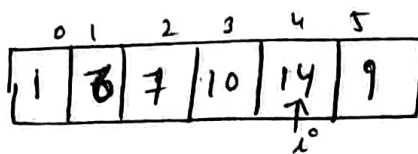
$7 > 6 \rightarrow \text{shift}$

$1 < 6 \rightarrow \text{break}$

$arr[j+1] = \text{value}$

$arr[0+1] = 6$

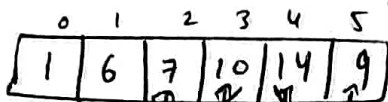
$arr[1] = 6$



$R_4 \rightarrow i=4$

$val = 14$

$14 > 10 \rightarrow \text{break}$



$R_5 \rightarrow i=5$

$val = 9$

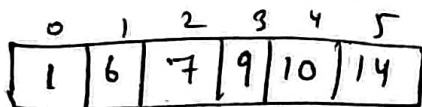
$14 > 9 \rightarrow \text{shift}$

$10 > 9 \rightarrow \text{shift}$

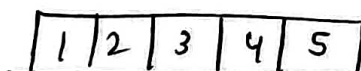
$7 < 9 \rightarrow \text{break}$

$arr[j+1] = \text{val}$

$arr[2+1] = 9 \Rightarrow arr[3] = 9$



Best case



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