Sorting Algorithms

Bubble sort from page 15

Today’s Checklist

- Time Complexity and space complexity
- 2 Pointer approach
- Bubble sort
- Selection sort
- Insertion sort

Ques: Given an array of integers with 1 to n elements and the size of the array is n+1. One element is occurring more than once i.e duplicate number is present. Find the duplicate element.

$N = n + 1$

size of array = N

$N = n + 1 = 11$

$n = 10$

arr

0	1	2	3	4	5	6	7	8	9	10
6	1	7	3	2	5	4	8	9	9	10

$i = 0$ to $i = N - 2$

M-I :

```
for(int i = 0 ; i < N-1 ; i++){
    for (int j = i+1 ; j < N; j++){
        if (arr[i] == arr[j]){
            printf(    );
            break;
        }
    }
}
```

[N-->no of elements in array]

Efficient → In terms of space

max total no of operations
n(n+1)/2

arr

0	1	2	3	4	5	6	7	8	9 ✓	10
6	1	7	3	2	5	4	8	9	9	10

10 + 9 + 8 + 7 + 6 + 5 + 4 + 3 + 1 = 53 operations

worst case = 55

M-2

brr

0	1	2	3	4	5	6	7	8	9	10
0	1	1	1	1	1	1	1	1	1	0

Visited array

total number of operations [10]
->maximum total no of operations "size of array"

Efficient → In terms of time..

Not Efficient → In terms of space → O(n) Extra space

We are using extra space [an extra array]

M-3

	0	1	2	3	4	5	6	7	8	9 ✓	10
arr	6	1	7	3	2	5	4	8	9	9	10

sum = $\cancel{0}$ $\cancel{6}$ $\cancel{7}$ $\cancel{14}$ $\cancel{17}$ $\cancel{19}$ $\cancel{24}$ $\cancel{28}$ $\cancel{36}$ $\cancel{45}$ $\cancel{54}$ 64

Sum of numbers from 1 to 10 $\rightarrow$ $\frac{10 \times 11}{2} = 55$

$$S_n = \frac{n(n+1)}{2}$$

$64 - 55 = \boxed{9}$

Efficient in terms of time & space both

[this is called analysis of time and space complexity]

3rd gen i3

M-3

3rd gen i3

M-1

Time Complexity

Space Complexity

TLE → time limit exceeded

Q/

```
for(int i=0; i<n; i++){
    printf("Hello");
}
```

↓

n operations → $O(n)$

'Big O Notation' → time complexity

$O(n+a) \simeq O(n)$

a ↓ constant

Q/

```
for(int i=-2; i<=n; i++){
    printf("Hello");
}
```

$n+3 \rightarrow O(n+3) \sim O(n)$

Q.

```
for(int i=1; i<=3*n; i++){
    printf("Hello");
}
```

$O(3^* n) \sim O(n)$

$O(k^* n) \approx O(n)$

k → constant

Q.

```
for(int i=1; i<=n*n; i++){
    printf("Hello");
}
```

$O(n^* n) = O(n^2) \longrightarrow$ because n is a variable not constant

Q1

```
for(int i=1; i≤n; i++){
    for(int j=10; j≤n; j++){
        printf("Hello");
    }
}
```

$i=1 \rightarrow j=1$ to n

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

Q1

```
for(int i=1; i≤n; i++){
    for(int j=1; j≤i; j++){
        printf("*");
    }
    printf("\n");
}
```

star triangle

$\frac{n(n+1)}{2}$ operations

$$O\left(\frac{n(n+1)}{2}\right) = O\left(\frac{n^2}{2}+\frac{n}{2}\right)$$

$$= O\left(\underbrace{\frac{1}{2}}n^2+\underbrace{\frac{1}{2}}n\right)$$

$$\approx O(n^2+n)$$

$$\approx \boxed{O(n^2)}$$

$O(3n^3 + 2n^2 + 8n) \approx O(n^3 + n^2 + n) \approx O(n^3)$

$O(\sqrt{n} + 8) \approx O(\sqrt{n})$

$O(n^{3/2} + n + 1) \approx O(n^{3/2})$

Extra Space : 'n' size array , n^2 size array , $\frac{n}{2}$ size

$\downarrow$ $\rightarrow$ 5 size array $\rightarrow$ ✗

O(1) $\longrightarrow$ constant space or we can say no extra space used

What is Sorting :

$$\begin{tikzcd}
9 \quad 1 \quad 2 \quad 8 \quad 6 \quad 4 \arrow[d, bend right] \\
1 \quad 2 \quad 4 \quad 6 \quad 8 \quad 9
\end{tikzcd}$$

Sort → put in ascending order

Sort in decreasing order → put in descending order

*2-pointers → 'algorithms'

Ques : Given an array of integers numbers that is already sorted in non-decreasing order, find two numbers such that they add up to a specific target number.

int target = 8

int arr[ ] =

0	1	2	3	4	5	6	7
1	2	3	4	5	8	9	10

way 1 :
worst way

```
for(int i = 0; i< n-1; i++){
    for(int j=i+1; j<n; j++){
        if( arr[i] + arr[j] = =target){
            //found
}
```

→ $O(n^2)$

```
int target = 8
int arr[ ] =
```

0	1	2	3	4	5	6	7
1	2	3	4	5	8	9	10
~~i~~	~~i~~	i		j	~~j~~	~~j~~	~~j~~

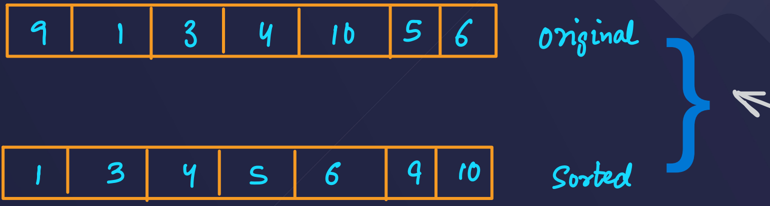
→ max → 2*n comparisons

way 2 :
two pointers way

```
int i = 0;
int j = n-1;

if(arr[i] + arr[j] == target) {
    //found
}

if (arr[i] + arr[j] > target) {
    j--;  // to decrease
}
if(arr[i] + arr[j] < target) {
    i++;  // to increase
}
```

```
int i = 0;                    O(n)
int j = n-1;

while ( i < j ){
    if( arr[i] + arr[j] = = target){
            printf(      )
            break;
    }
    else if ( arr[i] + arr[j] > target) j--;

    else  i++;
}
```

Bubble Sort :

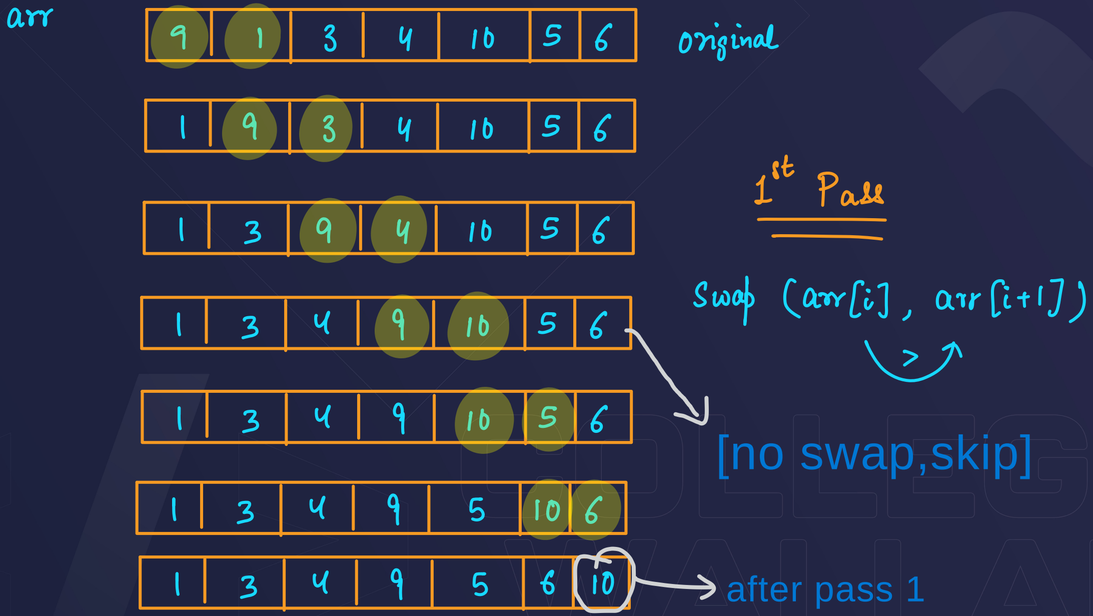


9	1	3	4	10	5	6
1	3	4	5	6	9	10

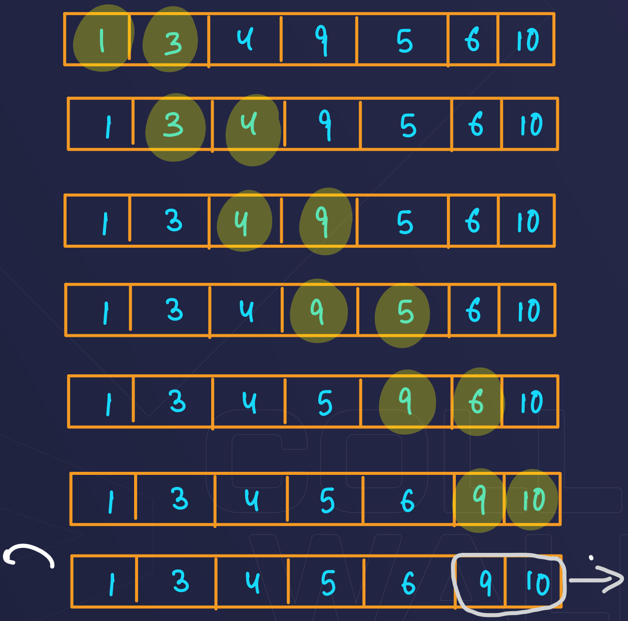
1) Technique
2) Explanation
3) Optimization
4) Complexities

1.sorting means arranging in ascending order (arranging from small to big number)

2.main principle of bubble sort is swap 2 numbers


arr
9 1 3 4 10 5 6
original
1 9 3 4 10 5 6
1 3 9 4 10 5 6
1 3 4 9 10 5 6
1 3 4 9 10 5 6
1 3 4 9 5 10 6
1 3 4 9 5 6 10
1st Pass
Swap (arr[i], arr[i+1])
>
[no swap,skip]
after pass 1

Pass-2



Sorted already

3rd pass

1	3	4	5	6	9	10

[worst case]

$\boxed{n = 5}$

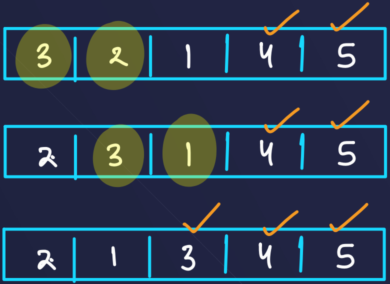
1st Pass

5	4	3	2	1
4	5	3	2	1
4	3	5	2	1
4	3	2	5	1
4	3	2	1	5 ✓

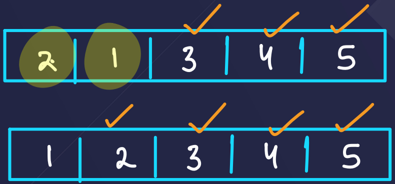
2nd Pass

4	3	2	1	5 ✓
3	4	2	1	5 ✓
3	2	4	1	5 ✓
3	2	1	4 ✓	5 ✓

3rd Pass



4th Pass



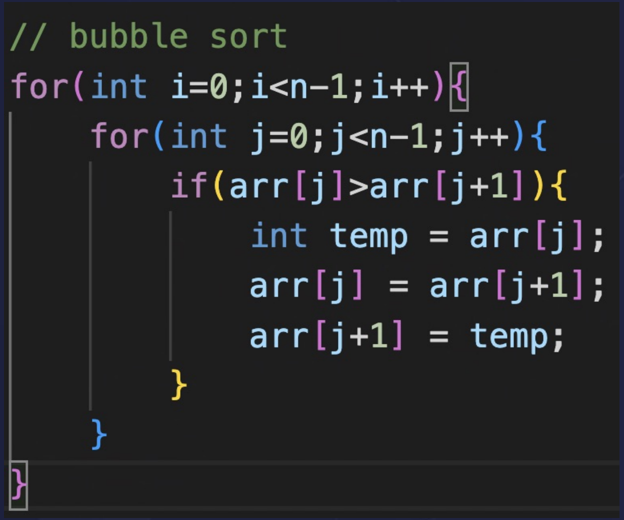
- 'n' elements in the array → 'n-1' passes
- After every pass, we need to apply bubble sort on the unsorted elements only & we do not need to check the 'largest' [1 step will reduce after each pass]

Coding implementation of bubble sort

Nested Loops

↳ Outer loop will stand for no. of passes

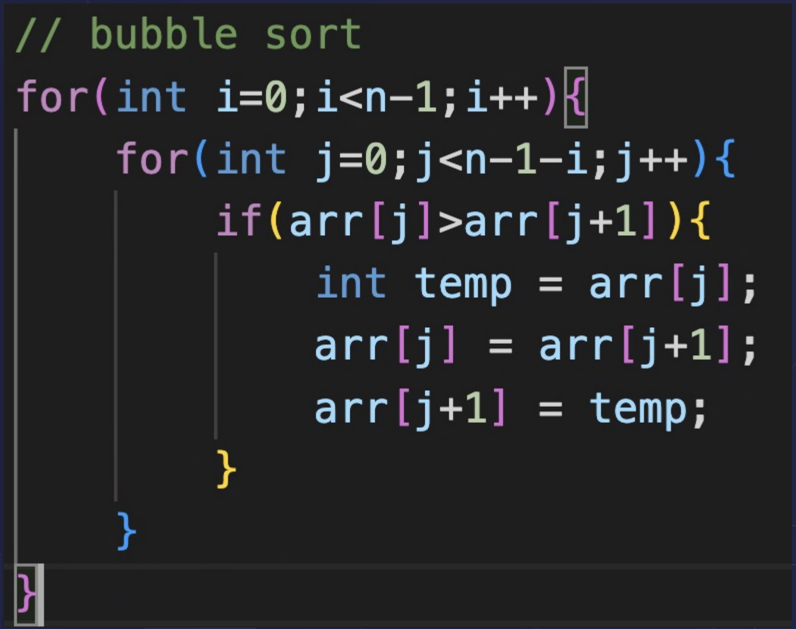
↳ Inner loop will do the swapping



```
// bubble sort
for(int i=0;i<n-1;i++){
    for(int j=0;j<n-1;j++){
        if(arr[j]>arr[j+1]){
            int temp = arr[j];
            arr[j] = arr[j+1];
            arr[j+1] = temp;
        }
    }
}
```

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

Time complexity



```
// bubble sort
for(int i=0;i<n-1;i++){
    for(int j=0;j<n-1-i;j++){
        if(arr[j]>arr[j+1]){
            int temp = arr[j];
            arr[j] = arr[j+1];
            arr[j+1] = temp;
        }
    }
}
```

Outer Loop → $0 \le i \le n-2$ → $n-1$ baar chalega

Inner Loop

$i=0$ → $n-1$ baar

$i=1$ → $n-2$ baar

$i=2$ → $n-3$ baar

$i=3$ → $n-4$ baar

no Ops = $n-1 + n-2 + n-3 + n-4 + \ldots\ldots 2 + 1$

$$= \frac{(n-1)^{*} n}{2} \rightarrow O\left(\frac{n^2}{2} - \frac{n}{2}\right) \approx O\left(\frac{n^2}{2}\right) \approx O(n^2)$$

Maximum no of swaps in worst case in Bubble Sort

$\downarrow$

descending

If size of array is 'n'

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

How to optimize the bubble sort in the case of nearly sorted arrays?

$\downarrow$

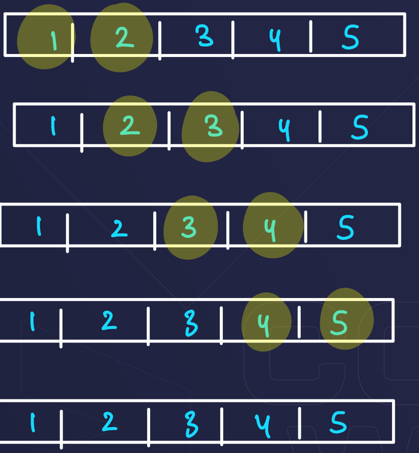
Check if array after every pass is already sorted or not.

$\downarrow$

with the help of a checkmark.

Time Complexity of Bubble Sort in best case :

flag = true;



$n \rightarrow n-1$

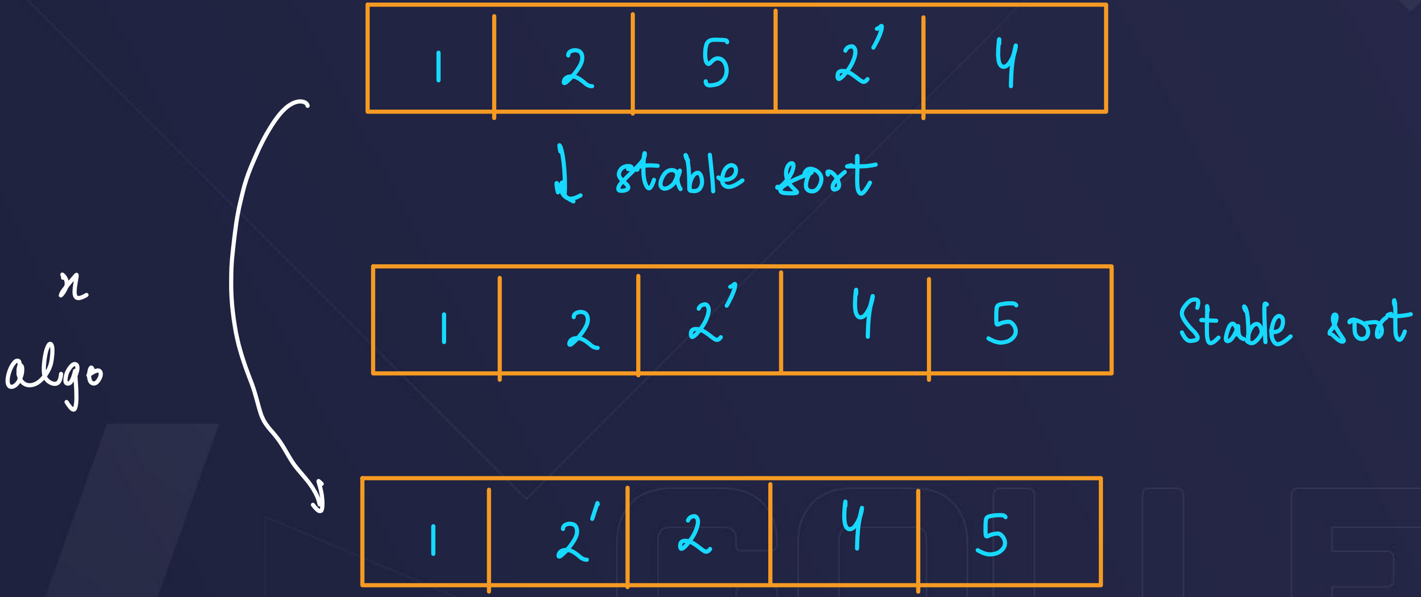
$\downarrow$

$O(n-1) \approx O(n)$

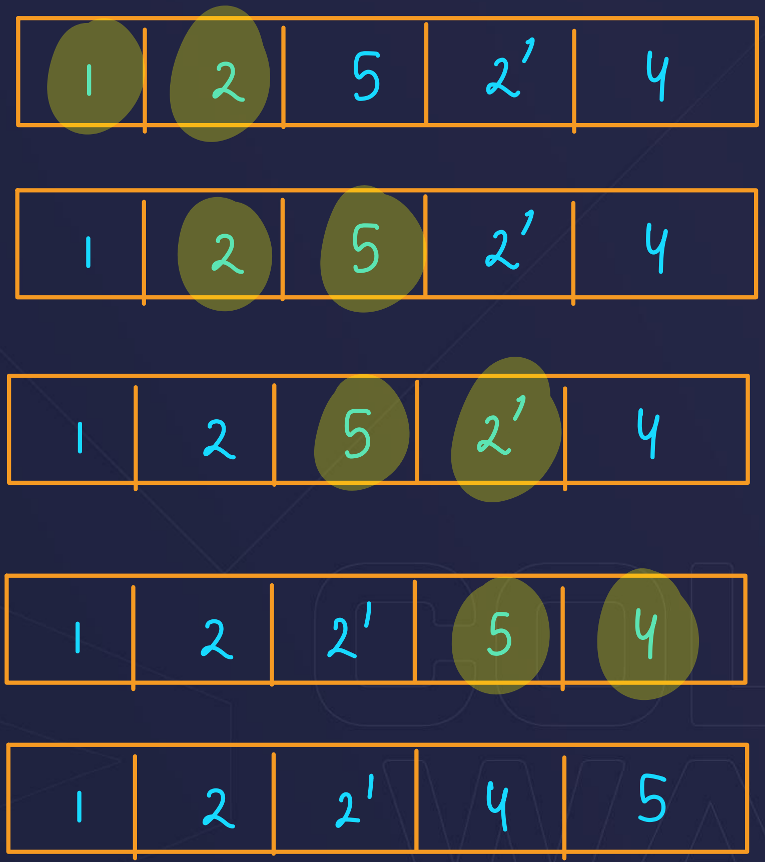
flag = true,

	Time Complexity	Space Complexity
Best Case	$O(n)$	$O(1)$
Avg. Case	$O(n^2)$	$O(1)$
Worst Case	$O(n^2)$	$O(1)$

Is Bubble Sort Stable? Yes!



Is Bubble Sort Stable?



Ques : What is the best case time and space complexity of bubble sort:

a) O(1) & O(1)
[✓ b)] O(n) & O(1)
c) O(n) & O(n)
d) O(logn) & O(1)

Ques : Given an array of 6 elements, what is the max number of swaps we need to sort the array:

a) 21

[✓ b) 15]

c) 10

d) 28

6 5 4 3 2 1

1

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

$5 + 4 + 3 + 2 + 1 = 15$

Bubble Sort → Unsorted array → ascending order

3 2 5 1 4 → 5 4 3 2 1

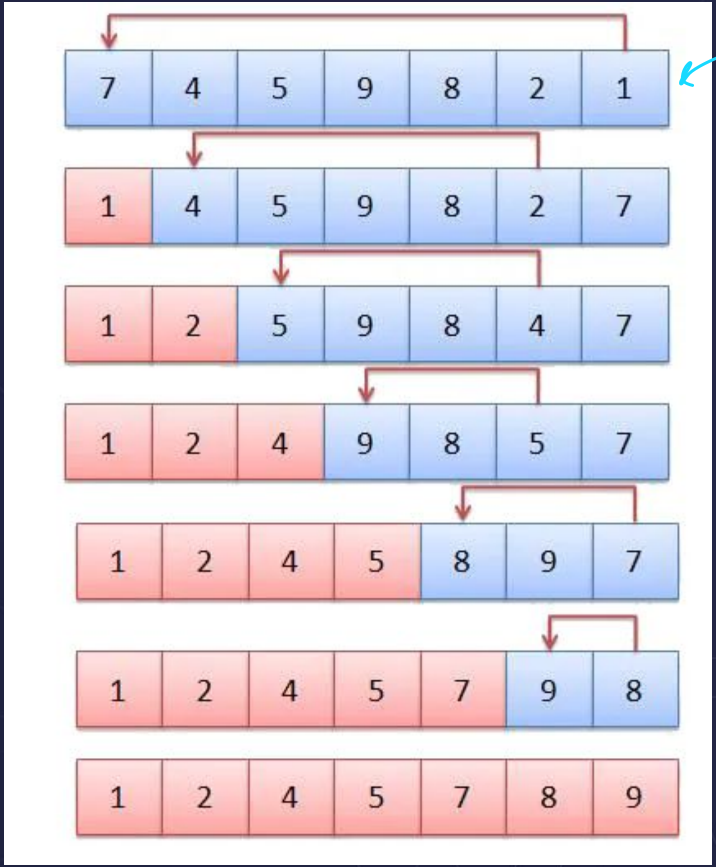
Q. Sort in descending order.

PW SKILLS

Selection sort

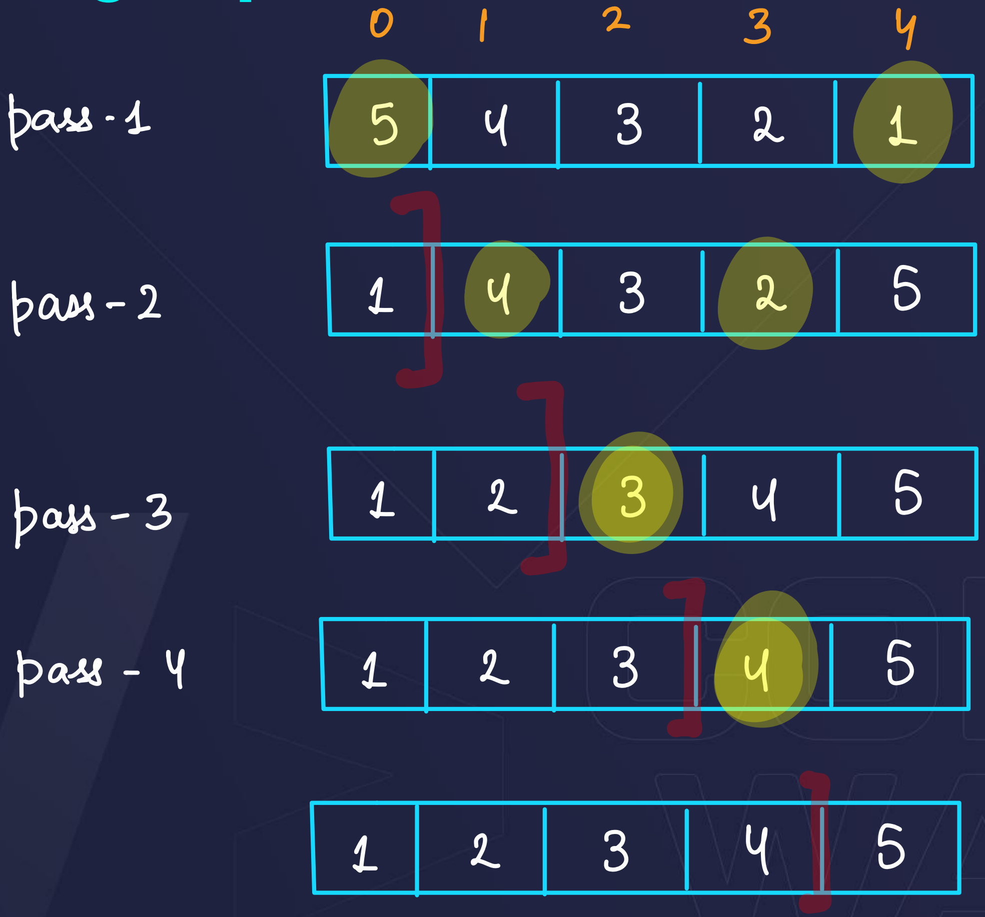
red → sorted part

blue → unsorted part



min^m element

Coding implementation of selection sort

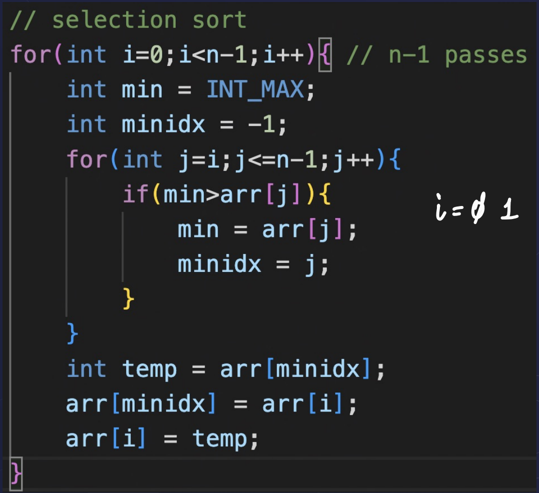


Observations :

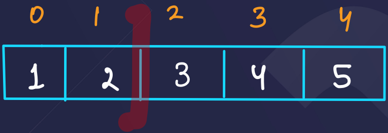
- For 'n' elements we need 'n-1' passes.
- In each pass we find out the min^m element in the unsorted part.
- After every pass the unsorted array reduces by 1 length.

Dry Run :

i = 0 to 3



```
// selection sort
for(int i=0;i<n-1;i++){ // n-1 passes
    int min = INT_MAX;
    int minidx = -1;
    for(int j=i;j<=n-1;j++){
        if(min>arr[j]){
            min = arr[j];
            minidx = j;
        }
    }
    int temp = arr[minidx];
    arr[minidx] = arr[i];
    arr[i] = temp;
}
```



i = ~~0~~ 1

min = ~~Int Max~~ ~~5~~ ~~4~~ ~~3~~ ~~2~~ ~~1~~ ~~Int Max~~ 2

minidx = ~~-1~~ ~~0~~ ~~1~~ ~~2~~ ~~3~~ ~~4~~ ~~-1~~ 3

Time complexity

Worst Case $\rightarrow$ $O(n^2)$

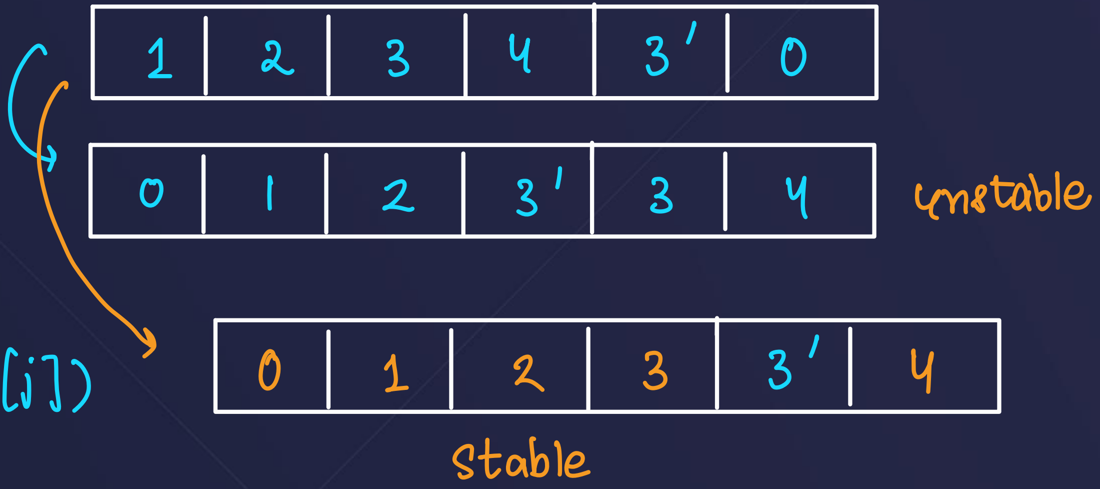
no of ope $\rightarrow$ $n + n-1 + n-2 + \dots 3 + 2 \boxed{+ 1} = \frac{n(n+1)}{2} \sim \frac{n^2}{2} + \frac{n}{2}$

$\rightarrow \frac{n^2}{2} \sim n^2$

Avg. Case $\rightarrow$ $O(n^2)$

Best Case $\rightarrow$ $O(n^2)$

Is selection sort stable ? No.



```
if (min $\boxed{\geq}$ arr[j])
{
    min = arr[j]
}
```

Ques : What will the array look like after the first iteration of selection sort [2,3,1,6,4] ?

a) [1,2,3,6,4]
b) [1,3,2,4,6]
[✓ c)] [1,3,2,6,4]
d) [2,3,1,4,6]

Ques : Which of the following is an advantage of selection sort over bubble sort:

a) It has a worst case complexity which is better than that of bubble sort.

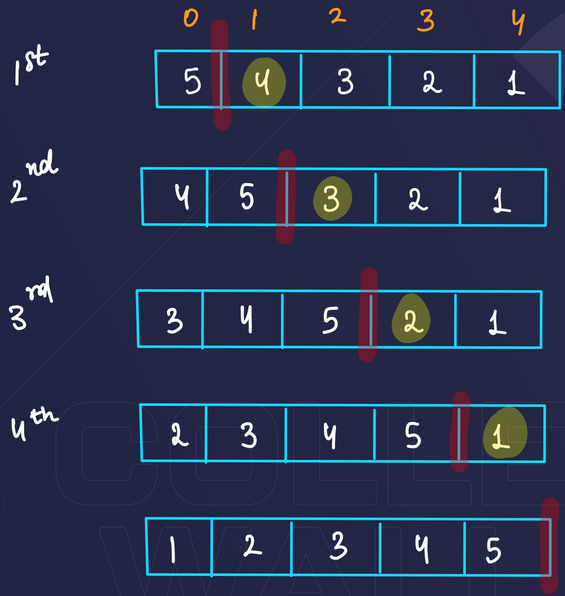
[✓ b)] It takes O(N) swaps while the other techniques take O(N^2) swaps.

c) The cost of swapping is an issue.

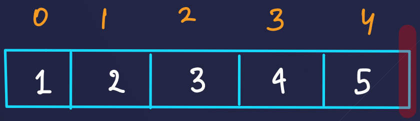
d) All of these.

Insertion Sort :

- Swapping from End till the element finds its position.
- Swap happens only when the element is smaller than its left element



Best Case for Insertion Sort :

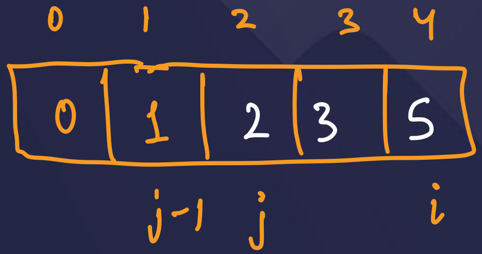


$n \rightarrow n-1$ operations $\rightarrow O(n)$

Coding implementation of insertion sort

- No. of passes → n-1 passes

```
for(int i=1; i<=n-1 ; i++){
    int j = i;
    while ( j>=1 && arr[j] < arr[j-1] ){
        swap (arr[j], arr[j-1]);
        j--;
    }
}
```



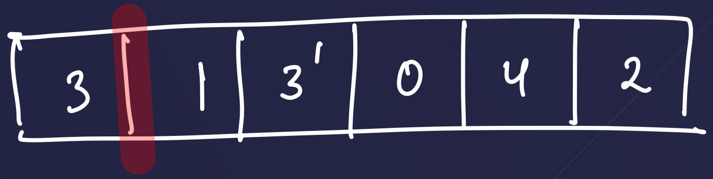
Time complexity

Worst Case - $O(n^2)$

Avg. Case - $O(n^2)$

Best Case - $O(n)$

Is Insertion Sort Stable? → yes !!



Ques : Which of the following examples represent the worst case input for an insertion sort?

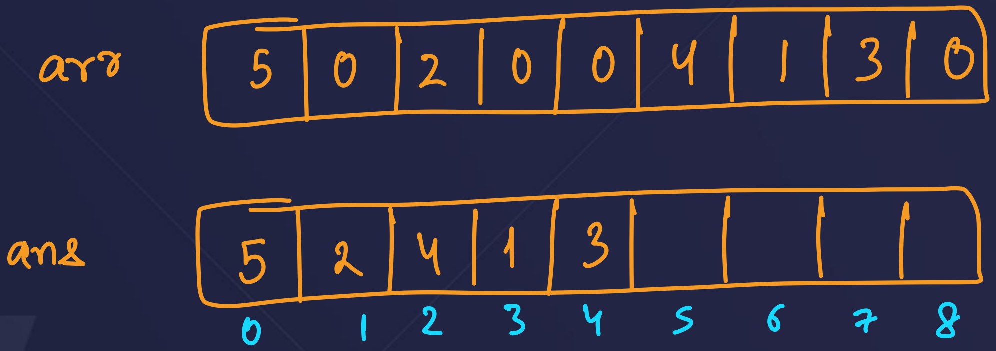
a) array in sorted order
b) large array
c) normal unsorted array
[✓ d) array sorted in reverse order]

Ques : How many passes would be required during insertion sort to sort an array of 5 elements?

a) 1
b) Depends on order of elements
[✓ c) 4]
d) 5

* **Ques** : Given an integer array arr, move all 0's to the end of it while maintaining the relative order of the non-zero elements.

idx = $\cancel{0}\,\cancel{1}\,\cancel{2}\,\cancel{3}\,\cancel{4}\,5$



↪ T.C = $O(n)$

S.C. = $O(n)$

Ques : Given an integer array arr, move all 0's to the end of it while maintaining the relative order of the non-zero elements.

Note that you must do this in-place without making a copy of the array.

Hint : Bubble Sort , Sort mat socho

arr

5	2	4	1	3	0	0	0	0

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

Homework

~~Ques~~ : Given an integer array and an integer k where k <= size of array, We need to return the kth smallest element of the array.

5 2 1 3 4

$n=5$

$K=3$

$O(K)$

Ques : Given an array of digits (values are from 0 to 9), the task is to find the minimum possible sum of two numbers formed from digits of the array. Please note that all digits of the given array must be used to form the two numbers.

Step-1 → Sort

5	3	1	2	~~4~~ 5

1 2 3 5 5

1 2 5 3 5

1 2 3 4 5 min no.

1 2 3 5 4 sec. min no.

Homework : If last two index elements are same then ?