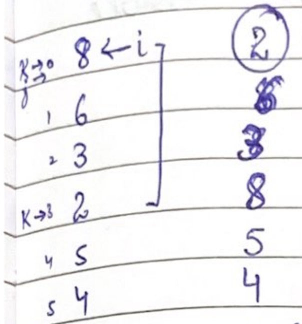
3) Selection Sort :→

A	8	6	3	2	5	4

→ We select a Position to find out an element for that Position (smallest among remaining)

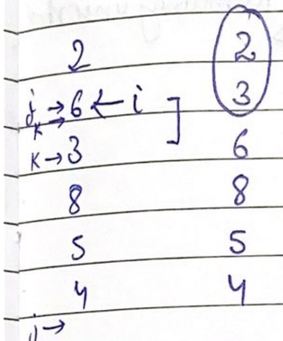


I^st Pass

5 comparisons.
1 swap.

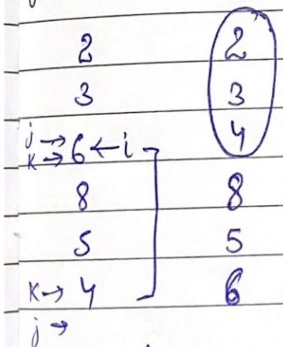
j→ → Move j to next element + check if element at j is smaller than element at K. Bring K to ~~position~~ position of j if it is smaller.

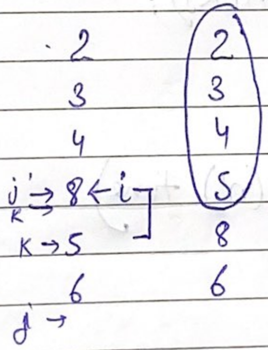
→ At last K will point to smallest element.

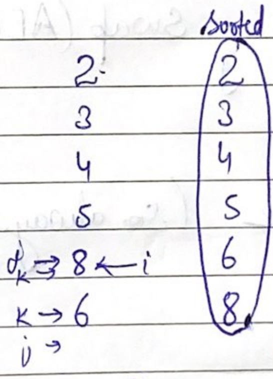


2^nd Pass

4 comparisons
1 swap.







3^rd Pass

4^th Pass

5^th Pass.

3 comparisons
1 swap

2comparisons
1 swap

1comparison
1 swap.

1. Min swaps Performed
- Intermediate results give K smallest elements } Imp

Total comparisons → 1+2+3+...(n-1) . $O(n^2)$ Time Complexity

★★★ Total Swaps → For n Passes (n-1) swaps
⇒ $O(n)$ swaps

→ This sorts the elements in minimum swaps Possible.

★★★ K Passes ⟶ K smallest elements

Algo/Pseudocode →

```
void selectionSort(int A[], int n)
{   int i, j, k;
    for(i=0; i<n-1; i++)  // For the Passes
    {
        for(j=k=i; j<n; j++)  // to check Remaining unsorted elements
        {
            if(A[j] < A[k])
            {
                k = j;
            }
        }
        Swap(A[i], A[k]);
}
```

- Not Adaptive (so always $O(n^2)$ time)

- Not Stable.