Selection Sort

Selection sort is a simple and efficient sorting algorithm that works by repeatedly selecting the smallest (or largest) element from the unsorted portion of the list and moving it to the sorted portion of the list.

Selection sort is generally used when -

- A small array is to be sorted
- Swapping cost doesn't matter
- It is compulsory to check all elements

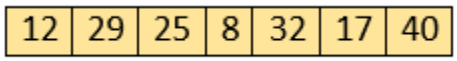
Now, let's see the algorithm of selection sort.

Working of Selection sort Algorithm

Now, let's see the working of the Selection sort Algorithm.

To understand the working of the Selection sort algorithm, let's take an unsorted array. It will be easier to understand the Selection sort via an example.

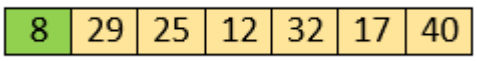
Let the elements of array are -



Now, for the first position in the sorted array, the entire array is to be scanned sequentially.

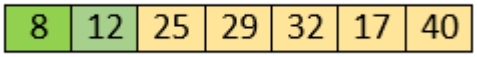
At present, **12** is stored at the first position, after searching the entire array, it is found that **8** is the smallest value.

So, swap 12 with 8. After the first iteration, 8 will appear at the first position in the sorted array.

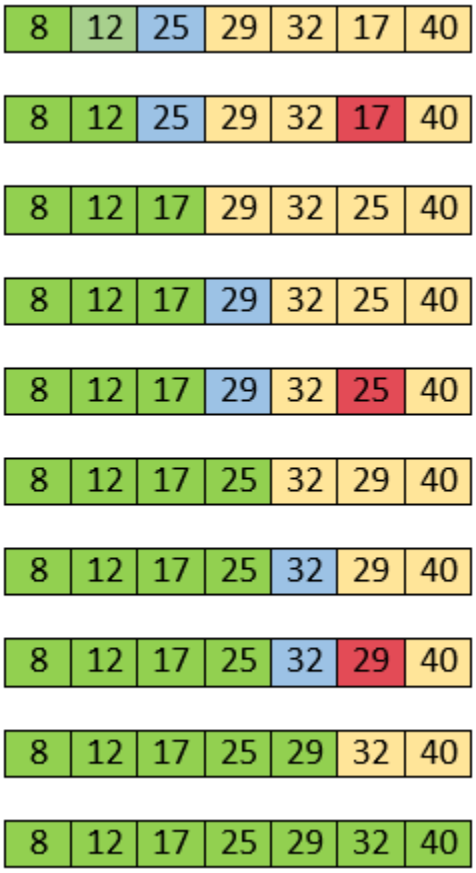


For the second position, where 29 is stored presently, we again sequentially scan the rest of the items of unsorted array. After scanning, we find that 12 is the second lowest element in the array that should be appeared at second position.

Now, swap 29 with 12. After the second iteration, 12 will appear at the second position in the sorted array. So, after two iterations, the two smallest values are placed at the beginning in a sorted way.



The same process is applied to the rest of the array elements. Now, we are showing a pictorial representation of the entire sorting process.



Now, the array is completely sorted.

Selection sort complexity

Now, let's see the time complexity of selection sort in best case, average case, and in worst case. We will also see the space complexity of the selection sort.

1. Time Complexity

Case	Time Complexity
Best Case	$O(n^2)$
Average Case	$O(n^2)$
Worst Case	$O(n^2)$

- **Best Case Complexity -** It occurs when there is no sorting required, i.e. the array is already sorted. The best-case time complexity of selection sort is **$O(n^2)$**.
- **Average Case Complexity -** It occurs when the array elements are in jumbled order that is not properly ascending and not properly descending. The average case time complexity of selection sort is **$O(n^2)$**.
- **Worst Case Complexity -** It occurs when the array elements are required to be sorted in reverse order. That means suppose you have to sort the array elements in ascending order, but its elements are in descending order. The worst-case time complexity of selection sort is **$O(n^2)$**.

2. Space Complexity

Space Complexity	O(1)
Stable	YES

- The space complexity of selection sort is O(1). It is because, in selection sort, an extra variable is required for swapping

Selection Sort Algorithm

```java
package Sorting;

import java.util.Arrays;
public class SelectionSort {
    public static void main(String[] args) {
        int[] arr = {3,1,5,1,2};
        sort(arr);
    }

    public static void sort (int[] arr) {
        for (int i = 0; i < arr.length; i++) {

            int last = arr.length-1-i;
            int maxInd = getMaxIndex(arr, 0, last);

            swap(arr, maxInd, last);
        }
        System.out.println(Arrays.toString(arr));
    }

    public static void swap(int[] arr, int max, int last) {
        int temp = arr[max];
        arr[max] = arr[last];
        arr[last] = temp;
    }

    public static int getMaxIndex(int[] arr, int first, int last) {
        int max = first;
        for (int i = first; i <= last; i++) {
            if (arr[max] < arr[i])  {
                max = i;
            }
        }
        return max;
    }

}
```

s