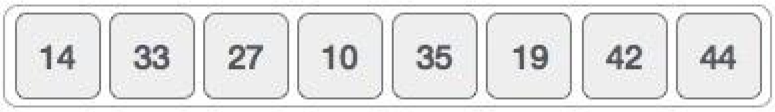
Selection Sort

Steps: (sorting in increasing order)

1. First-of-all, we will find the smallest element of the array and swap it with index 0.
2. Similarly, we will find the second smallest and swap that with the element at index 1 and so on...
3. Ultimately, we will be getting a sorted array in increasing order only.

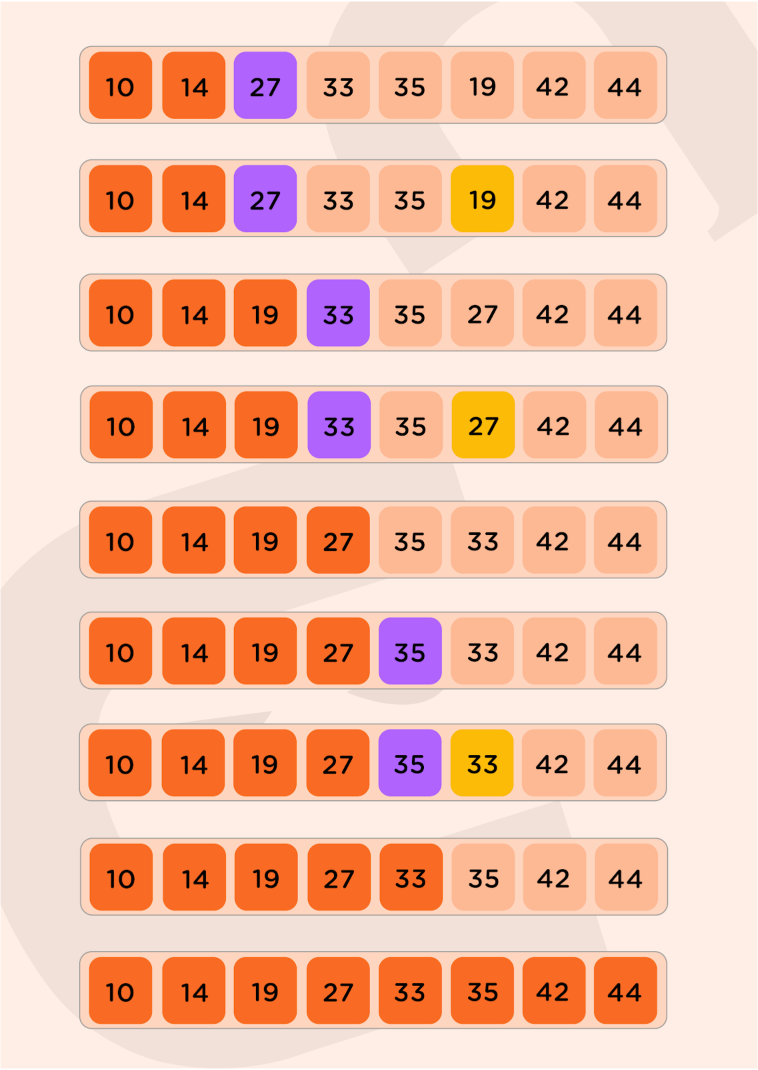
Let us look at the example for better understanding:

Consider the following depicted array as an example. You want to sort this array in increasing order.



14	33	27	10	35	19	42	44

Following is the pictorial diagram for a better explanation of how it works:



10	14	27	33	35	19	42	44
10	14	27	33	35	19	42	44
10	14	19	33	35	27	42	44
10	14	19	33	35	27	42	44
10	14	19	27	35	33	42	44
10	14	19	27	35	33	42	44
10	14	19	27	35	33	42	44
10	14	19	27	33	35	42	44
10	14	19	27	33	35	42	44

This is how we obtain the sorted array at last.

Pseudocode:

```
/*
    array of size N from 0 to N-1 is considered
*/
function selectionSort(arr, N)

        for idx = 0 to N-2
                //   Initialize minimum value to be present at idx
                minIdx = idx
                for jdx = idx+1 to N-1
                        //   If a new minimum is found, set minIdx to jdx
                        if arr[jdx] < arr[minIdx]
                                minIdx = jdx

                //   Finally swap the minimum found from idx to N-1 with idx
                swap(arr[idx], arr[minIdx])
```

Time complexity: O(N^2), in the worst case.
As to find the minimum element from the array of ‘N’ elements, we require ‘N-1’ comparisons, then after putting the minimum element in the correct position, we repeat the same for the unsorted array of the remaining ‘N-1’ elements, performing ‘N-2’ comparisons and so on.

So, total number of comparisons : N-1 + N-2 + N-3 + … + 1 = (N*(N-1))/2 and total number of exchanges(swapping): N-1
So, time complexity becomes O(N^2).

Space complexity: O(1), as no extra space is required.