* Selection Sort

select an element & put it on its correct index.

[get the greatest element of the array & put it on its correct index]

Ex. 4 (0), 5 (1), 1 (2), 2 (3), 3 (4) — swap 5 & 3

4 (0), 3 (1), 1 (2), 2 (3), 5 (4) — swap 4 & 2

2 (0), 3 (1), 1 (2), 4 (3), 5 (4) — swap 3 & 1

2 (0), 1 (1), 3 (2), 4 (3), 5 (4) — swap 2 & 1

1, 2, 3, 4, 5 sorted

→ Here we have selected maximum element & put it on right index, we can also do vice versa i.e. select minimum & put it on right index

Complexity:

Total comparisons =>

4, 5, 1, 2, 3 — (n-1)

4, 3, 1, 2, 5 — (n-2)

2, 3, 1, 4, 5 — (n-3)

2, 1, 3, 4, 5 — 1

1, 2, 3, 4, 5 — 0

already at correct position ignore in future steps.

$= 0 + 1 + 2 + 3 + \cdots (n-1) = \frac{n(n-1)}{2} = \left(\frac{n^2-n}{2}\right)$

neglect less dominating & const. elements

worst case → $O(n^2)$

Best case → $O(n^2)$

⇒ It is not stable sorting algorithm.

⇒ It performs well on small lists.

Ex.

```
static void selectionSort(int[] arr){
    for(int i=0; i<arr.length; i++){
        int last = arr.length-i-1;
        int maxIndex = getMax(arr, 0, last);
        swap(arr, ~~start~~, ~~end~~ last);
                  maxIndex
    }
}
static void swap(int[] arr, int start, int last)
{
    int temp = arr[start];
    arr[start] = arr[last];
    arr[last] = temp;
}
static ~~vo~~int getMax(int[] arr, int start, int end)
{   int Max = start;
    for(int i=start; i<=end; i++){
        if(arr[Max] < arr[i]){
            Max = i;
        }
    }
    return Max;
}
```