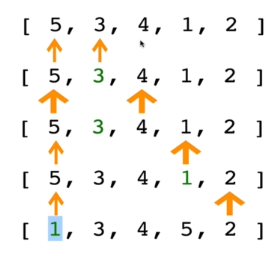
Chapter 6 : Part 2

**Sorting Algorithms: Selection Sort**

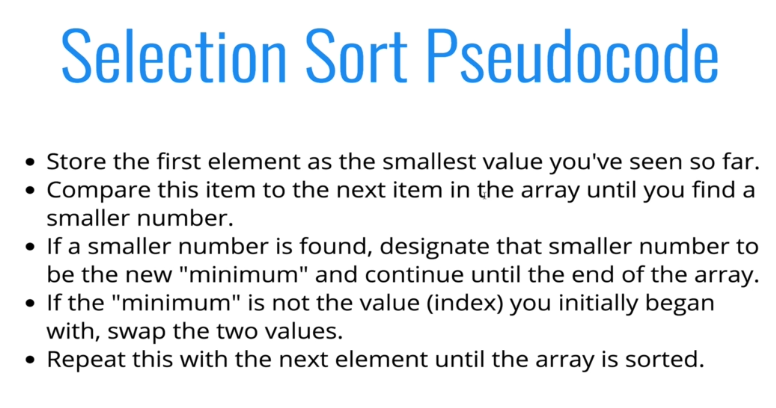
**6.2.1 Selection Sort**

Similar to bubble sort, but instead of first placing large values into sorted position, it places small values into sorted position

**6.2.2 How it works:**



**6.2.3 Pseudocode:**



* Practiced version:

**function** **selectionSort**(arr) {

**for** (**var** i=0; i **<** arr**.**length; i++) {

        // *Insted of values we store only the index of lowest value*

**var** lowest=i;

        // *lowest = arr[(arr.length)-i]*

**for** (**var** j=i+1; j **<** arr**.**length; j++) {

            // *console.log(arr, arr[j], arr[j+1]);*

**if** (arr[lowest] **>** arr[j]) {

                lowest = j;

            }

        }

        // *swapping the smallest value with the initial value arr[i]*

        console**.log**(i, lowest);

        console**.log**("#############");

        console**.log**(arr);

        console**.log**("swapping in arr");

**if** (i **!=** lowest) {

**var** temp=arr[i];

            arr[i] = arr[lowest];

            arr[lowest] = temp;

            console**.log**(arr);

            console**.log**("#############");

        }

    }

**return** arr;

}

* Instructor solution:

// *LEGACY VERSION (non ES2015 syntax)*

**function** **sselectionSort**(arr){

**for**(**var** i=0; i **<** arr**.**length; i++){

**var** lowest=i;

**for**(**var** j=i+1; j **<** arr**.**length; j++){

**if**(arr[j] **<** arr[lowest]){

                lowest = j;

            }

        }

**if**(i **!==** lowest){

            //*SWAP!*

**var** temp=arr[i];

            arr[i] = arr[lowest];

            arr[lowest] = temp;

        }

    }

**return** arr;

}

* ES2015 codes:

// *ES2015 VERSION*

**function** **selectionSort**(arr) {

**const swap** =(arr, idx1, idx2) **=>**

([arr[idx1],arr[idx2]]=[arr[idx2],arr[idx1]]);

**for** (**let** i=0; i **<** arr**.**length; i++) {

**let** lowest=i;

**for** (**let** j=i+1; j **<** arr**.**length; j++) {

**if** (arr[lowest] **>** arr[j]) {

        lowest = j;

      }

    }

**if** (i **!==** lowest) **swap**(arr, i, lowest);

  }

**return** arr;

}

**selectionSort**([0,2,34,22,10,19,17]);

* Notice: We used only index not the value.
* We shrink the window using ***i*** variable.
* ***j*** variable starts next index of ***i***, ***i+1***.
* Following finds the *index* of *smallest value* inside the *active window*

**if** (arr[lowest] **>** arr[j]) {

                lowest = j;

            }

* If ***i != lowest***: In the case ***i = lowest***, an element swaps values with itself. To avoid it we used this condition. Inside this condition swap happens.
* Time complexity: Selection sort time complexity is **O(n2)**
* Why better than bubble sort: It minimizes the number of swaps. In bubble sort the amount of swap is higher than selection sort.
* In selection sort For each value of **i**, swap happens in the outer for loop, so swap is reduced. In the case of bubbleSort, swapping happens inside the nested-for-loop.