**Sorting Algorithms-I**

**VisualAlog:** [*https://visualgo.net/en/sorting*](https://visualgo.net/en/sorting)

**SortingAnimationSite:** [*https://www.toptal.com/developers/sorting-algorithms*](https://www.toptal.com/developers/sorting-algorithms)

**(Bubble, Selection & Insertion Sort)**

**What is sorting?**

Sorting is the process of rearranging the items in a collection (e.g. an array) so that the items are in some kind of order.

**Examples**

Sorting numbers from smallest to largest

Sorting names alphabetically

Sorting movies based on release year

Sorting movies based on revenue

Here we’re going to learn the following sorting algorithms:-

Start with Elementary Sorting Algorithms:  
Implement **bubble** sort  
Implement **selection** sort  
Implement **insertion** sort

Then,

**Merge** Sort

**Quick** Sort

**Radix** Sort

**JavaScript has a sort method...**

...but it doesn't always work the way you expect.

[ "Steele", "Colt", "Data Structures", "Algorithms" ].sort();

// [ "Algorithms", "Colt", "Data Structures", "Steele" ]

[ 6, 4, 15, 10 ].sort();

// [ 10, 15, 4, 6 ]...but it doesn't always work the way you expect.

* The built-in sort method accepts an optional **comparator function**
* You can use this comparator function to tell JavaScript how you want it to sort
* The comparator function looks at pairs of elements (a and b), determines their sort order based on the return value
* If it returns a negative number, a should come before b
* If it returns a positive number, a should come after b,
* If it returns 0, a and b are the same as far as the sort is concerned.

**#1\_Example:**

function numberCompare(num1, num2) {

return num1 - num2;

}

[ 6, 4, 15, 10 ].sort(numberCompare);

Output:  
[ 4, 6, 10, 15 ]

**#2\_Example:**

function compareByLen(str1, str2) {

return str1.length - str2.length;

}

[ "Steele", "Colt", "Data Structures", "Algorithms" ]

.sort(compareByLen);

Output:  
[ "Colt", "Steele", "Algorithms", "Data Structures" ]

**Bubble Sort**

A sorting algorithm where the largest values bubble up to the top!

[ 5, 3, 4, 1, 2 ]

[ 3, 5, 4, 1, 2 ]

[ 3, 4, 5, 1, 2 ]

[ 3, 4, 1, 5, 2 ]

[ 3, 4, 1, 2, 5 ]  
 5 is now in its sorted position!

**Pseudo Code**

1. Start looping from the end of the array towards the beginning with a variable called i.
2. Start an inner loop with a variable called j from the beginning until i - 1
3. If arr[i] is greater than arr[j], swap those two values!
4. Return the sorted array

**Example:**

**1st way:**

function bubbleSort(arr){

for(let i=arr.length; i>0; i--){

let noSwap = true;

for(let j=0; j<i-1; j++){

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

if(arr[j]>arr[j+1]){

let temp = arr[j+1];

arr[j+1] = arr[j];

arr[j]= temp;

noSwap = false;

}

If(noSwap) break;

}

// console.log("One Scann Complete throughout the Array");

}

return arr;

}

bubbleSort([ 5, 3, 4, 1, 2 ]);  
bubbleSort([37,45,29,8,12,88,-3]);  
bubbleSort([1,2,3,4,5,6,7,8]);

**Output:**[1, 2, 3, 4, 5]  
[-3, 8, 12, 29, 37, 45, 88]  
 [1,2,3,4,5,6,7,8]

**2nd way:**

function bubbleSort(arr){

let noSwap = true;

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

for(let j=i+1; j<arr.length; j++){  
 //console.log(arr, arr[i], arr[j]); *//for checking purpose.*

if(arr[i]>arr[j]){  
 let temp = arr[i];  
 arr[i] = arr[j];  
 arr[j]= temp;

noSwap = false;

}

If(noSwap) break;  
 }

//console.log("One Scann Complete throughout the Array");  
 }

return arr;

}

bubbleSort([ 5, 3, 4, 1, 2 ]);  
bubbleSort([37,45,29,8,12,88,-3]);  
bubbleSort([1,2,3,4,5,6,7,8]);

**Output:**[1, 2, 3, 4, 5]  
[-3, 8, 12, 29, 37, 45, 88]  
[1,2,3,4,5,6,7,8]

**Note:**

* Here, The reason behind using noSwap concept in the program is that, As for an unsorted array, program will loop over the array for every element which leads to Big-O of O(n2). But if the array is already sorted, then it will loop over the array only at once (length of array), which leads to Big-O of O(n)
* Bubble sort performs very well for nearly sorted data in the array.

**Selection Sort**

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

[ 5, 3, 4, 1, 2 ]

[ 5, 3, 4, 1, 2 ]

[ 5, 3, 4, 1, 2 ]

[ 5, 3, 4, 1, 2 ]

[ 1, 3, 4, 5, 2 ]

So, here we first loop through an array and while looping we’re going to compare and select only those element which is smaller among all. When loop reached at the end of the arry, we just swap the last selected minimum element with the first element of the array.

For Ex: Let say for the first looping over an array, at first iteration, we got (5&3)🡪3 minimum, then (3&4)🡪3 minimum, then (4&1)🡪 1minimum, then (1&2)🡪still 1 minimum. Now swap, this last got minimum num(1) with first num (5). Result of first looping: [ 1, 3, 4, 5, 2 ]

**Pseudo Code:**

1. Store the first element as the smallest value you've seen so far.
2. Compare this item to the next item in the array until you find a smaller number.
3. If a smaller number is found, designate that smaller number to be the new "minimum" and continue until the end of the array.
4. If the "minimum" is not the value (index) you initially began with, swap the two values.
5. Repeat this with the next element until the array is sorted.

**Example:**

**1st way:**

function selectionSort(arr){

for(let i=0; i<arr.length; i++){  
 let tempMin = arr[i];

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

//console.log(arr, arr[i], arr[j]);

if(tempMin>arr[j]){

tempMin = arr[j];  
   
 }

}

let temp = arr[i];  
 arr[arr.indexOf(tempMin)] = temp;  
 arr[i] = tempMin;

}

return arr;

}

selectionSort([1,2,3,5,4]);  
selectionSort([32,27,10,2,-8,50]);  
selectionSort([0,2,10,17,19,22,34]);

**Output:**[1, 2, 3, 4, 5]  
[-8, 2, 10, 27, 32, 50]  
[0, 2, 10, 17, 19, 22, 34]

**2nd way:**

**// LEGACY VERSION (non ES2015 syntax)**

function selectionSort(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!

console.log(i, lowest);

var temp = arr[i];

arr[i] = arr[lowest];

arr[lowest] = temp;

}

}

return arr;

}

selectionSort([1,2,3,5,4]);  
selectionSort([32,27,10,2,-8,50]);  
selectionSort([0,2,10,17,19,22,34]);

**Output:**[1, 2, 3, 4, 5]  
[-8, 2, 10, 27, 32, 50]  
[0, 2, 10, 17, 19, 22, 34]

**3rd way:**

**// 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([1,2,3,5,4]);  
selectionSort([32,27,10,2,-8,50]);  
selectionSort([0,2,10,17,19,22,34]);

**Output:**[1, 2, 3, 4, 5]  
[-8, 2, 10, 27, 32, 50]  
[0, 2, 10, 17, 19, 22, 34]

**Note:**

* Selection sort doesn’t performs well, even the array is sorted. Because it still gonna go through each time to find the lowest item, so it iterate all the loop.
* But selection sort is easy to understand.

**Insertion Sort**

Builds up the sort by gradually creating a larger left half which is always sorted

[ 5, 3, 4, 1, 2 ]

[ 3, 5, 4, 1, 2 ]

[ 3, 4, 5, 1, 2 ]

[ 1, 3, 4, 5, 2 ]

[ 1, 2, 3, 4, 5 ]

**Pseudo Code:**

1. Start by picking the second element in the array
2. Now compare the second element with the one before it and swap if necessary.
3. Continue to the next element and if it is in the incorrect order, iterate through the sorted portion (i.e. the left side) to place the element in the correct place.
4. Repeat until the array is sorted.

**1st way:**

function insertionSort(arr){

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

let currentVal = arr[i];

for(var j=i-1; j>=0 && arr[j]>currentVal; j--){

arr[j+1] = arr[j];

}

arr[j+1] = currentVal; //here we defined j with var instead of let.

}

return arr;

}

insertionSort([2,1,9,76,4]);  
insertionSort ([1,2,3,5,4]);  
insertionSort([32,27,10,2,-8,50]);  
insertionSort ([0,2,10,17,19,22,34]);

**Output:**[1, 2, 4, 9, 76]  
[1, 2, 3, 4, 5]  
[-8, 2, 10, 27, 32, 50]  
[0, 2, 10, 17, 19, 22, 34]

**Note:**

* Bubble sort performs very well, if the data is almost sorted.

**Big O of Sorting Algorithms**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Algorithm** | **Time Complexity (Best)** | **Time Complexity (Average)** | **Time Complexity (Worst)** | **Space Complexity** |
| Bubble Sort | O(*n*) | O(*n*2) | O(*n*2) | O(1) |
| Insertion Sort | O(*n*) | O(*n*2) | O(*n*2) | O(1) |
| Selection Sort | O(*n*2) | O(*n*2) | O(*n*2) | O(1) |

**Conclusion**

* Sorting is fundamental!
* Bubble sort, selection sort, and insertion sort are all roughly equivalent
* All have average time complexities that are quadratic
* We can do better...but we need more complex algorithms.



**SortingAnimationSite:** [*https://www.toptal.com/developers/sorting-algorithms*](https://www.toptal.com/developers/sorting-algorithms)