Interviewer: Write a Selection sort.

Me: Ok I write it.

Interviewer: How can you improve this algorithm.

Me: I could inch it a little closer to O(n) in a best case scenario, by only doing the nested loop only if the current parent loops value is not equal to the previous. Additionally if the item type is an integer I could check for the current value being equal to the previous or previous minus one. This works because Selection Sort will always move the highest one to highest point + n. If the current one being checked is already equal to the highest number in the list there is no need to check or swap. Additionally saving the highest overall value will let you break out of the inner loop faster.

Stable Sort: Keeps the order of equal valued elements the same.

UnStable: The original order of equal valued elements will not be the same.

Selection Sort is unstable.

Selection sort algorithm

Always check the current element to the rest of the elements in the list. There is no point in comparing the current element to itself. Search in the inner loop until all results are exhausted or you find one equal to the current highest value. After the loop the nextHighest index should be swapped with the current index of the parent loop. In JavaScript, my function can take a function that converts the current value to a comparable value.

JavaScirpt’s Array.sort will sort ascending by default. JavaScript does the sort in place.

Remember length – 1 is not needed because j’s initialization will fall out of the loop. Length – 1 breaks the sort. Only length – 1 when inside of the loop

JavaScript:

var sort;

// Modifies in place

function selectionSort(arr, getValue){

if(typeof getValue != “function”)

getValue = undefined;

/// Only sort the arr if the array is greater than 1, because there is no point otherwise

if(arr && arr.length > 1){

getValue = getValue || function(arr){ return arr;};

// Start at the beginning of the list and compare it to every other item in the list

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

var highestIndex = i;

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

if(getValue(arr[highestIndex]) < getValue(arr[j])){

highestIndex = j;

}

}

// Do the swap only if the highest index changed

if(highestIndex != i){

var highValue = arr[highestIndex];

arr[highestIndex] = arr[i];

arr[i] = highValue;

}

}

}

}

// Lets modify this so that it will be stable and be able to leave the loop early

// If equal to the highest over value there is no need to search

function selectionSortModified(arr, getValue){

if(typeof getValue != “function”)

getValue = undefined;

/// Only sort the arr if the array is greater than 1, because there is no point otherwise

if(arr && arr.length > 1){

var overallHighest;

getValue = getValue || function(arr){ return arr;};

// Start at the beginning of the list and compare it to every other item in the list

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

var highestIndex = i;

Var highestValue = getValue(arr[highestIndex]);

// If the highest value is equal to the overall highest then no need

// to continue searching. The highest value is also found from the

// start

for(var j = i + 1;j< arr.length && highestValue !== overallHighest; j++){

if(highestValue < getValue(arr[j])){

highestIndex = j;

highestValue = getValue(arr[j]);

}

}

// Do the swap only if the highest index changed

if(highestIndex != i){

var highValue = arr[highestIndex];

arr[highestIndex] = arr[i];

arr[i] = highValue;

}

}

// The overall highest is the highest value found the on first full iteration

if(i == 0){

overallHighest = highestValue;

}

}

}