Assignment 7

R-2.8 Illustrate the performance of the selection-sort algorithm on the following input sequence (22, 15, 26, 44, 10, 3, 9, 13, 29, 25).
(3, 15, 26, 44, 10, 22, 9, 13, 29, 25).
(3, 9, 26, 44, 10, 22, 15, 13, 29, 25).
(3, 9, 10, 44, 26, 22, 15, 13, 29, 25).
(3, 9, 10, 13, 26, 22, 15, 44, 29, 25).
(3, 9, 10, 13, 26, 22, 15, 44, 29, 25).
(3, 9, 10, 13, 15, 22, 26, 44, 29, 25).
(3, 9, 10, 13, 15, 22, 26, 44, 29, 25).
(3, 9, 10, 13, 15, 22, 26, 25, 29, 44).
(3, 9, 10, 13, 15, 22, 26, 25, 29, 44).
(3, 9, 10, 13, 15, 22, 26, 25, 29, 44).

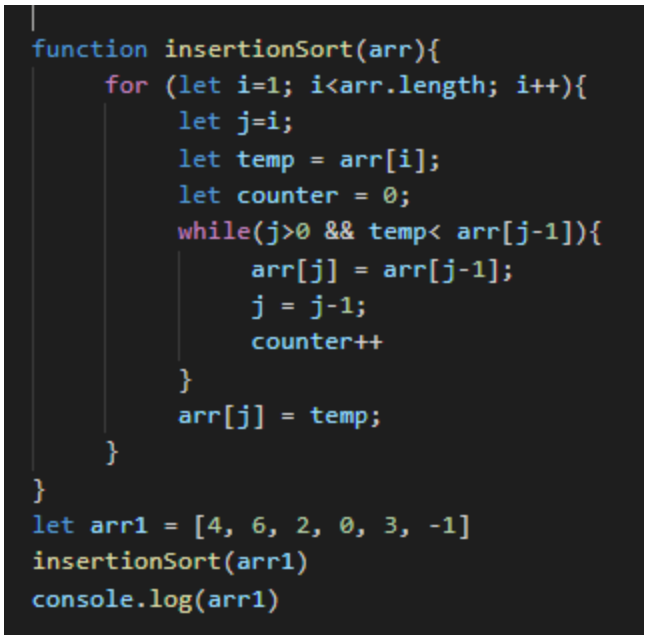
R-2.9 Illustrate the performance of the insertion-sort algorithm on the input sequence of the previous problem.
(22, 15, 26, 44, 10, 3, 9, 13, 29, 25).
(15, **22, 26, 44**, 10, 3, 9, 13, 29, 25).
(10, 15, 22, 26, 44, 3, 9, 13, 29, 25).
(3, 10, 15, 22, 26, 44, 9, 13, 29, 25).
(3, 9, 10, 15, 22, 26, 44, 13, 29, 25).
(3, 9, 10, 13, 15, 22, 26, 44, 29, 25).
(3, 9, 10, 13, 15, 22, 26, 29, 44, 25).
(3, 9, 10, 13, 15, 22, 26, 25, 29, 44).

R-2.10 Give an example of a worst-case sequence with n elements for insertion-sort runs in Ω(n2) time on such a sequence.

1. Suppose a binary tree T is implemented as an array S, as described in today's notes. If n items are stored in S in sorted order, assuming the root is at index 0, is the tree T a heap? Justify your answer.

   Yes, because heap is a sorted binary tree. And the children are greater than parent;

2. Using the pseudo-code in today's notes implement selectionSort, insertionSort, and heapSort in JavaScript. Insert a counter in each of the algorithms to count the number of key comparisons and swaps for selectionSort and heapSort . Similarly, insert a counter for the number of key comparisons and shifts in insertionSort. Run several tests of small, medium, and large arrays to compare the algorithms. What is your conclusion about running times?

3. 

```
function insertionSort(arr){
     for (let i=1; i<arr.length; i++){
          let j=i;
          let temp = arr[i];
          let counter = 0;
          while(j>0 && temp< arr[j-1]){
               arr[j] = arr[j-1];
               j = j-1;
               counter++
          }
          arr[j] = temp;
     }
}
let arr1 = [4, 6, 2, 0, 3, -1]
insertionSort(arr1)
console.log(arr1)
```

```
function selectionSorting(arr){
    let last = arr.length-1;
    let counter= 0;
    for(let i=0; i<last; i++){
        let nextMin = findNextMinIndex(arr, i, last);
        SwapElements(arr, i, nextMin);
        counter++
    }

}

function findNextMinIndex(arr, bottom, top){
    let minIndex = bottom;
    let min = arr[bottom];
    for(let i = bottom+1; i<= top; i++){
        if(arr[i]<min){
            min = arr[i]
        minIndex = i;
        }
    }
    return minIndex
}

function SwapElements(arr, i, nextMin){
    let temp = arr[nextMin];
    arr[nextMin]= arr[i];
    arr[i] = temp;
}
let arr = [4, 6, 2, 0, 3, -1]
console.log(arr)
selectionSorting(arr);
console.log(arr)
```