**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, 15, 22, 26, 44, 29, 25).  
(3, 9, 10, 13, 15, 22, 25, 44, 29, 26).  
(3, 9, 10, 13, 15, 22, 25, 26, 29, 44).  
(3, 9, 10, 13, 15, 22, 25, 26, 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).  
(15,22, 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, , 25, 26, 29, 44).

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

**Using the pseudo-code in today’s notes implement insertionSort, ShellSort, and heapSort  
in JavaScript. Insert a counter in each of the algorithms to count the number of key  
comparisons and swaps for heapSort. Similarly, insert a counter for the number of key  
comparisons and shifts in insertionSort and ShellSort. Run several tests of small,  
medium, and large arrays to compare the algorithms; use the ArraySort.js file that imports  
your HW07-ArraySorter.js file and runs tests on your sort algorithms. What is your  
conclusion about running times?**

heapify(arr) {

        // your code goes here

        let last = arr.length - 1;

        let next = last;

        while (next > 0) {

            this.\_downheap(arr, this.\_parent(next), last);

            next = next - 2;

        }

    }

    \_downheap(arr, index, last) {

        // your code goes here

        let property = false;

        while (!(property)) {

            let maxIndex = this.\_indexOfMax(arr, index, last);

            if (maxIndex != index) {

                this.\_swapElements(arr, maxIndex, index);

                index = maxIndex;

            } else {

                property = true;

            }

        }

    }

indexOfMax(arr, index, last) {

        // your code goes here

        let largest = index;

        let left = 2 \* index + 1;

        let right = left + 1;

        if ((left <= last) && (arr[left] > arr[largest])) {

            largest = left;

            this.\_compCount++;

        }

        if ((right <= last) && (arr[right] > arr[largest])) {

            largest = right

            this.\_compCount++;

        }

        return largest;

    }

heapSort(arr) {

        this.\_compCount = 0;

        this.\_swapCount = 0;

        this.\_heapify(arr);

        console.log("key comparisons to build the Heap " + this.\_compCount);

        // your code goes here

        let end = arr.length - 1;

        while (end > 0) {

            this.\_swapElements(arr, 0, end);

            end = end - 1;

            this.\_downheap(arr, 0, end)

        }

    }

insertionSort(arr) {

        // your code goes here

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

            let j = i;

            let temp = arr[i]

            this.\_compCount += 1

            while ((0 < j) && (temp < arr[j - 1])) {

                arr[j] = arr[j - 1];

                this.\_shiftCount += 1;

                j = j - 1;

            }

            arr[j] = temp;

        }

    }

**Level 2:  
2. Use one of these sorts from 1 above to implement in JavaScript another version of  
isPermutation(A,B). First start with the pseudo-code, then translate into JavaScript.  
Submit both the pseudo-code and the JavaScript program file with test cases.**

Algorithm isPermutation(A, B)

Input := Array A and Array B

Output := if both array have same elements then true;

If A.length = 0 \/ B.length = 0 then return false

heapSort(A)

heapsort(B)

return helperPermutation(A, B)

Algorithm helperPermutation(A, B)

If A.length != B.length then return false

for i:=0 to arr.length-1 do

if A.[i] != B[i] then return false

return true