Chanh Dao Le – 986178

Assignment 4

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).

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**	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
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**

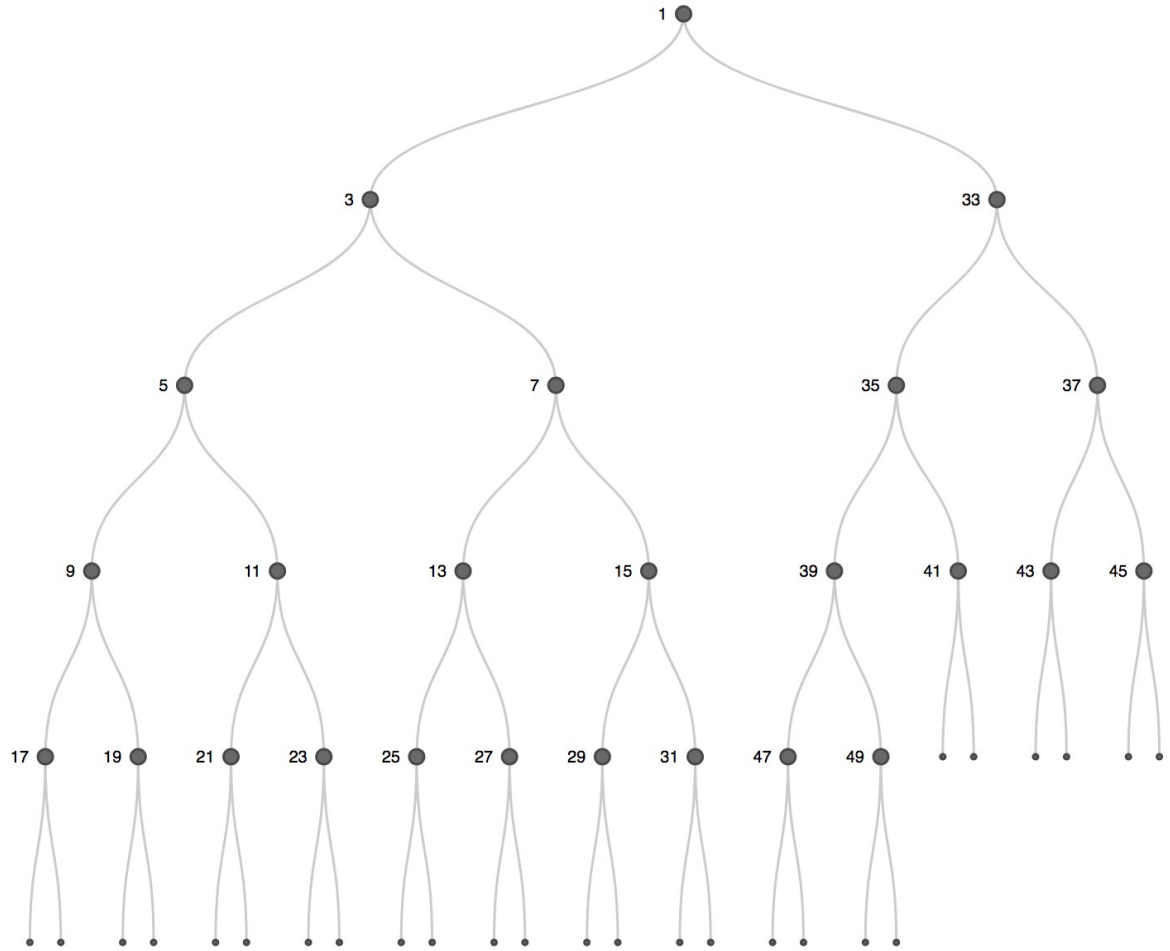
R-2.10 Give an example of a worst-case sequence with n elements for insertion-sort runs in Ω(n2) time on such a sequence.

The worst-case sequence example is one in descending order such as **9 8 7 6 5 4 3 2 1** to be sorted to ascending order.

R-2.13 Suppose a binary tree T is implemented using a vector S, as described in Section 2.3.4. If n items are stored in S in sorted order, starting with index 1, is the tree T a heap? Justify your answer.

Yes, it is a heap because: S[i] >= S[i/2] => key(v) >= key(parent(v)).

R-2-18 Draw an example of a heap whose keys are all the odd numbers from 1 to 49 (with no repeats), such that the insertion of an item with key 32 would cause up-heap bubbling to proceed all the way up to a child of the root (replacing that child's key with 32).



C-2.32 Let T be a heap storing n keys. Give an efficient algorithm for reporting all the keys in T that are smaller than or equal to a given query key x (which is not necessarily in T). For example, given the heap on Figure 2.41 and query key x=7, the algorithm should report 4, 5, 6, 7. Note that the keys do not need to be reported in sorted order. Ideally, your algorithm should run in O(k) time, where k is the number of keys reported.

```
Algorithm reportKey(T, x)
        Input: A heap T, and query value x
        Output: A list of keys smaller than or equal to x
        returnList <- new List
        If ¬T.isEmpty()
                reportKeyHelper(T, 1, x, returnList)
        return returnList
Algorithm reportKeyHelper(T, i, x, returnList)
        Input: A heap T, index of a node in the heap, the query value x, and the return list to
contain reported values
        if T[i] <= x
                returnList.add(T[i])
```

```
        left <- i * 2
        if left < T.size()
                leftList <- reportKeyHelper(T, left, x, returnList)
        right <- i * 2 + 1
        if right < T.size()
                rightList <- reportKeyHelper(T, right, x, returnList)
```

Design an algorithm, isPermutation(A, B) that takes two sequences A and B and determines whether or not they are permutations of each other, i.e., same elements but possibly occurring in a different order. Hint: A and B may contain duplicates. What is the worst-case time complexity of your algorithm? Justify your answer

```
Algorithm isPermutation(A, B)
        Input: 2 sequences to check if they are permutations of each other
        Output: true if they are permutations of each other, false otherwise
        If A.size() ¬= B.size()
                return false
        PQ <- new PriorityQueue with HeapSort implementation
        sortedA <- PQ.sort(A)
        sortedB <- PQ.sort(B)
        elementsA <- sortedA.elements()
        elementsB <- sortedB.elements()
        while elementsA.hasNext()
                elementA <- elementsA.nextObject()
                elementB <- elementsB.nextObject()
                if elementA ¬= elementB
                        return false
        return true
```

The worst-case complexity of this algorithm is O(n log n) because of the heap-sorting