Chanh Dao Le – 986178

Assignment 6

R-4.14 Which, if any, of the following algorithms, bubble-sort, heap-sort, insertion sort, merge-sort, and quick-sort, are stable? Briefly justify your answer.

- Bubble sort is stable because when sorting it compares the adjacent items and swaps them if they are in wrong order. Thus, if there are two identical items, their orders remain unchanged.
- Heap sort is unstable because when sorting heap sort first peaks the largest element and put it in the last of the list. Therefore, the element which is picked first stays last and the element which is picked second stays second last in the sorted list. Therefore, the order of the same items can be changed.
- Insertion sort is stable because it does not change the relative order of elements with equal keys.
- Merge sort is stable because it preserves the input order of equal elements in the sorted output.
- Quick sort is not stable because it exchanges the non-adjacent elements. Therefore, it can't maintain the relative order of the equal elements.

R-4.16 Is the bucket-sort algorithm in-place? Why or why not?

Bucket sort is not in-place because elements are put into buckets which may require as much space as the input array in some cases.

C-4.13 Suppose we are given two sequences A and B of n elements, possibly containing duplicates, on which a total order relation is defined. Describe an efficient algorithm for determining if A and B contain the same set of elements (possibly in different orders). What is the running time of this method?

```
Algorithm isSameElements(A, B)
        Input: Sequence A and B to compare
        Output: Whether or not A and B contain same elements
        A <- MergeSort(A, C)
        B <- MergeSort(A, C)
        for i <- 0 to n – 1 do
                a <- A.removeFirst()
                b <- B.removeFirst()
                if a ¬= b then
                        return false
        return true
```

The running time is O(nlogn)

R-5.4 Characterize each of the following recurrence equations using the master method (assuming that T(n) = c for n < d, for constant c > 0 and d ≥ 1).

- $T(n) = 2T(n/2) + \log n$
  $a = 2, b = 2, \log_b a = 1$
  Case 1: $f(n) = \log n$ is $O(n^{1-\varepsilon}) = O(\sqrt{n})$ with $\varepsilon = 1/2$ -> $T(n)$ is $\theta(n)$
- $T(n) = 8T(n/2) + n^2$
  $a = 8, b = 2, \log_b a = 3$
  Case 1: $f(n) = n^2$ is $O(n^{3-\varepsilon}) = O(n^2)$ with $\varepsilon = 1$ -> $T(n)$ is $\theta(n^3)$

- $T(n) = 16T(n/2) + (n\log n)^4$
  $a = 16, b = 2, \log_b a = 4$
  Case 2: $f(n) = (n\log n)^4$ is $\theta(n^4 \log^k n) = (n^4 \log^4 n)$ with $k = 4$ -> $T(n)$ is $\theta(n^4 \log^5 n)$
- $T(n) = 7T(n/3) + n$
  $a = 7, b = 3, \log_b a = 1.xxx...$
  Case 1: $f(n) = n$ is $O(n^{1.xxx... - \varepsilon}) = O(n)$ with $\varepsilon = 0.xxx...$ -> $T(n)$ is $\theta(n^{1.xxx...})$
- $T(n) = 9\ T(n/3) + n^3 \log n$
  $a = 9, b = 3, \log_b a = 2$
  Case 3: $f(n) = n^3 \log n$ is is $\Omega(n^{2+\varepsilon}) = \Omega\ (n^3)$ with $\varepsilon = 1$ -> $T(n)$ is $\theta(n^3 \log n)$
  Also, $9(n/3)^3 \log(n/3) = (1/3)(n^3 \log(n/3)) < (1/3)(n^3 \log n)$