1/ Sorting Algorithm Property

Exercise 8.3-2

* The sorting algorithms are stable: Insertion Sort, Merge Sort

The sorting algorithms are not stable: Heap Sort, Quick Sort

* Simple Scheme that makes sorting algorithm stable:
* Store the original index of each element
* Use that index as a secondary way of sorting elements with equal primary value.
* To implement this, the comparison function or operator (for example <) would be implemented so A < B returns TRUE if A.*originalIndex* is less than or equal to B.*originalIndex*. The otherwise induction returns FALSE.
* This requires one additional *originalIndex* to be stored per element. There are n elements, hence Ꝋ(n*lg*n) extra space is required.

2/ Counting Sort, Radix Sort, Bucket Sort

1. Counting Sort: A = <6, 0, 2, 6, 0, 8>
2. Radix Sort: PAT, CAT, CART, FAT, FIX

P A T P A T C A R T

C A T C A T C A T

C A R T => F A T => F A T

F A T F I X F I X

F I X C A R T P A T

C A T

F A T

=> F I X

P A T

C A R T

1. Bucket Sort: A = <0.67, 0.82, 0.12, 0.46, 0.88, 0.61>

A has 6 elements

= = 4

= = 4

= = 0

= = 2

= = 5

= = 3

* Answer is <0.12, 0.46, 0.61, 0.67, 0.82, 0.88>

4/ Sorting: Exercise 8.4-2 p204

* Why the worst-case running time for bucket sort is Ꝋ(n2)?
* Bucket Sort executes like Insertion Sort does, where all elements are inserted into the sorted array. This cause the array will be re-arranged n2 times. So the worst-case running time for Bucket Sort is Ꝋ(n2).
* Solving:
* To fix the worst-case running time to be Ꝋ(n*lg*n), we should call Merge Sort algorithm at each sorting step due to Merge Sort has worst-case running time is Ꝋ(n*lg*n).