**Chapter 12**

**R-12.4 Is our array-based implementation of merge-sort given in Section 12.2.2 stable? Explain why or why not**

**def merge sort(S):**

**n = len(S)**

**if n < 2:**

**return # list is already sorted**

**6 # divide**

**7 mid = n // 2**

**8 S1 = S[0:mid] # copy of first half**

**9 S2 = S[mid:n] # copy of second half**

**10 # conquer (with recursion)**

**11 merge sort(S1) # sort copy of first half**

**12 merge sort(S2) # sort copy of second half**

**13 # merge results**

**14 merge(S1, S2, S) #**

The sort is not stable, the S2 portion is not configured correctly and part of it should be in S1 then the sort would be stable.

**R-12.19 Suppose S is a sequence of n values, each equal to 0 or 1. How long will it take to sort S with the merge-sort algorithm? What about quick-sort?**

The longest the sort could take is On^2. Regularly it takes Onlogn, so the sequence of the elements does not matter. If the array is already sorted quicksort would work with a small movement of the last element.

**C-12.26 Describe and analyze an efficient method for removing all duplicates from a collection A of n elements.**

**def removeDuplicate(array):**

**#ideas from https://paste.ee/p/8h2fu**

**array.sort()**

**x = [array[0]]**

**y = array[0]**

**for i in range(1, len(array)):**

**if array[i] != y:**

**x.append(array[i])**

**y = array[i]**

**return x**

**if \_\_name\_\_ == '\_\_main\_\_':**

**print(removeDuplicate([5,5,5,7,8,9,9,9,10,11,12]))**

**[5, 7, 8, 9, 10, 11, 12]**