Exercise 10 - Solutions

1. In function "merge", l. 26, when S1[i] == S2[j], the "else" clause is selected, which puts the element from S2 first. Since S2 contains the latter part of the original sequence, this places the second of two elements that compare equal first.

   ⇒ The sort is not stable, but "straggling" or "anti-stable".

2. Same logic as for problem 1, just using a different underlying data structure.

3. This choice of pivot will split the sequence approximately in half at each step, so total time is $O(n \log n)$.

4.
```
def merge_no_duplicates(S1, S2, S):
    i = j = k = 0
    S = []
    while i+j < len(S1) + len(S2):
        if j == len(S2) or (i< len(S1) and S1[i] < S2[j]):
            S.append(S1[i])
            i += 1
        else if j< len(S2) and i< len(S1) and S1[i] == S2[j]:
            S.append(S1[i])
            i += 1
            j += 1
        else:
            S.append(S2[j])
            j += 1
```

5. merge sort: will need to recursive split at every level, so $O(n \log n)$

   quick sort: If elements equal to pivot are separated in their own class, which does not need recursion, then need to recurse at most once, so $O(n)$.

   If list is split into only two components, we degenerate to the case of applying quicksort with final element as pivot to a sorted list, so $O(n^2)$.

6. $O(n)$, need to look at each element once and place into one of two buckets.

7. For an easy algorithm, use insertion sort. However, even with only two keys, it's $O(n^2)$ worst case. (Consider, for example, the alternating sequence 010101...)

Getting $O(n \log n)$ behavior stable and in-place is more tricky. Here is one possibility:

- Let an "inversion" denote a maximal run of 1s followed by a maximal run of 0s.
- If m denotes the length of the inversion, it can be rectified in $O(m)$ time as the new location of each element is explicitly known. Doing this with $O(1)$ extra storage ("in place") is a bit tricky (cf. cycle sort) but can be done.
- Go through list left-to-right. Upon finding an inversion, rectify it, then move to the next inversion. As the total length of all inversions is $\leq n$, this process takes $O(n)$ time.
- Repeat this process until the list has no more inversions. As the minimum length of an inversion doubles at each step, this requires at most $O(\log n)$ repetitions.

$\Rightarrow$ Total cost is $O(n \log n)$

(Recall: Bucket sort can do this in $O(n)$ time, but not in place.)

8. This bound applies to <u>any</u> sort for sequences with distinct elements. See proof in class.

9. Store the initial index together with the key, and change the order relation: If keys compare equal, resort to comparing the initial indices, which will enforce the initial order.
This costs $O(n)$ time and $O(n)$ space, so does not change the asymptotics.

10. Any integer $i \in [0, n^2-1]$ can be written

$$i = jn + k \qquad \text{with } j, k \in 0, \dots, n-1$$

First bucket-sort on k, then perform a stable bucket-sort on j.