Name: Georges Hatem

CS 590 Homework 7: Creativity Exercises

Due Date: March 13, 2022

Problem 8.5.14:

An efficient algorithm that we can use in this case is the merge-sort algorithm. We will add a variable (k) to the merge-sort algorithm to check how many different two equal elements in S we have. The running time of my method will be O(nlog(n)). The Algorithm for this exercise is below:

```
Algorithm merge ($S_1$, $S_2$, S):
    Input: Two arrays, $S_1$ and $S_2$, of size $n_1$ and
    $n_2$, respectively, sorted in non-decreasing
    order, and an empty array, S, of size at
    least $n_1 + n_2$
    Output: S, containing the elements from $S_1$
    and $S_2$ in sorted order and k, the number of
    different two equal elements in S.

    i $\leftarrow$ 1
    j $\leftarrow$ 1
    k $\leftarrow$ 0
```

```
while i <= n and j <= n do
    if $S_1[i]$ <= $S_2[j]$ then
        if i ← 1 and j ← 1 then
            if $S_1[i]$ ← $S_2[j]$ then
                k = k + 1
        else if $S_1[i]$ ← $S_2[j]$ and $S_1[i]$ != S[i+j-2] then
            k = k + 1
        S[i+j-1] ← $S_1[i]$
        i ← i + 1
    else
        S[i+j-1] ← $S_2[j]$
        j ← j + 1

while i <= n do
    if i != n and $S_1[i]$ ← $S_1[i+1]$ and $S_1[i]$ != S[i+j-2] then
        k = k + 1
    S[i+j-1] ← $S_1[i]$
    i ← i + 1
```

```
while j <= n do
    if j != n and $S_2[j]$ ← $S_2[j+1]$ and $S_2[j]$ != S[i+j-2] then
        k = k + 1
    S[i+j-1] ← $S_2[j]$
    j ← j + 1
```

This merge-sort algorithm above check to see if there is any two equal elements in S and if there is it increments k by 1. The Algorithm will spell how many different two equal elements there is. For example, k = 3 means that there is 3 different two equal elements. The Algorithm make sure that if an element has been repeated more than 1 time, we increment k only once since we are only counting the two equal elements. For example, if the number 5 is included 6 times in S, we only increment k one time since the number 5 has at least 2 equal elements. This way k is a representation of the number all the different two equal elements in S. We did this

by using the following equality in the if statement (S_1[i] != S[i+j-2] or S_2[j] != S[i+j-2]). This means that if for example, both S_1[i] and S_2[j] are equal to the same number (let's take for example 5), and so we increment k by 1 and included S_1[i] in S. Now, if S_1[i] contains a number 5 again. We compare S_2[j] number 5 to S_1[i] number 5, but since we have the statement S_1[i] != S[i+j-2] or S_2[j] != S[i+j-2], k is not incremented again because we are only counting it once. This way we can determine the number of different two equal elements that we have in S. The if i ← 1 and j ← 1 statement has been included because if our program goes through the statement S_1[i] != S[i+j-2] or S_2[j] != S[i+j-2] and if i ← 1 and j ← 1 then S[i+j-2] does not exist yet. This statement is not included in the last 2 while loops because in the last 2 while loops j or i has to be bigger than n (respectively), meaning that S[i+j-2] exists. As you can see from above, nothing major has been added to the merge-

sort algorithm pseudocode, meaning that the running time of the Algorithm above is O(nlog(n)).

Problem 9.5.12:

We can use the Bucket-Sort Algorithm to determine whether there are two equal numbers in S with an O(n) time. We just need to add few lines to it to make it catch the numbers of two equal numbers in S. To do that we add two variables (j and m). j is to count the number of items we have in each list B[i] when sorting them in S and if j is 2 (meaning we have at least two equal numbers of that number in B[i] list), we can go ahead and add 1 to m, meaning that the number of two equal numbers in S has increased by 1). This Algorithm will run in O(n) time. Below is the Algorithm:

Algorithm bucketSort(S):

Input: Sequence S of items with integer Keys in the range [0, N-1]

Output: Sequence S sorted in nondecreasing order of the keys and m, the number of different two equal elements in S. Let B be an array of N lists, each of which is initially empty

```
m ← 0

for each item x in S do
    let k be the key of x
    remove x from S and insert it at the
    end of bucket (list) B[k]

for i ← 0 to N-1 do
    j ← 0
    for each item x in list B[i] do
        remove x from B[i] and insert it at
        the end of S

        j ← j+1
```

```
            if j ← 2 then
                m ← m + 1
```

The program account for the different two equal elements in S. For example, if the number 5 is included 6 times in S, we only increase m by 1 when dealing with number 5. We did this by saying if j ← 2 increments m by 1. This way if j is bigger than 2 (e.g. we have the number 5 included 6 times, m will be incremented 1 time because we are calculating the number of different two equal elements in S). As you can see, few lines has been added to the Bucket-Sort Algorithm. These few lines do not affect its running time. The running time of the Algorithm is still O(n) as expected by the Exercise.