timothy allec

**Name, SID, Date** .....

# In Class Assignment 15: Counting Sort

**Benjamin Sanders, MS** November 25, 2020

## 1 Introduction

You may work in groups of up to two or three students. Write all student names at the top of all pages for this assignment. Turn in all work to Blackboard on or before the deadline to receive credit.

You may use additional libraries and online resources, if you get them approved in writing, over email, from the instructor first. If you have received approval from the instructor, write the approved libraries and any references in the space below.

import java.util.Vector;

.....
.....
.....
.....

## 2 Assignment Description

### 2.1 Big Picture

Counting Sort is faster than any other sorting method we have explored so far.

### 2.2 Algorithm Implementation

Implement the following algorithm in Java, using the Vector data structure for any 1-D array, 2-D array, or linear algebra purposes.

```
COUNTING-SORT(A, B, k)
 1  let C[0..k] be a new array
 2  for i = 0 to k
 3      C[i] = 0
 4  for j = 1 to A.length
 5      C[A[j]] = C[A[j]] + 1
 6  // C[i] now contains the number of elements equal to i.
 7  for i = 1 to k
 8      C[i] = C[i] + C[i − 1]
 9  // C[i] now contains the number of elements less than or equal to i.
10  for j = A.length downto 1
11      B[C[A[j]]] = A[j]
12      C[A[j]] = C[A[j]] − 1
```

```
import java.util.Vector;

public class CountingSort{
   public static void counting_sort(Vector<Integer> data, Vector<Integer> sortedData,
Integer max){
      Vector<Integer> count = new Vector<Integer>();
      for (int i = 0; i <= max; i++){
         count.add(0);
      }
      for (int i = 0; i < data.size(); i++){
         count.set(data.elementAt(i), count.elementAt(data.elementAt(i)) + 1);
      }
      for (int i = 1; i <= max; i++){
         count.set(i, count.elementAt(i) + count.elementAt(i - 1));
      }
      for (int i = data.size() - 1; i >= 0; i--){
         sortedData.add(0);
      }
      for (int i = data.size() - 1; i >= 0; i--){
         sortedData.set(count.elementAt(data.elementAt(i)) - 1, data.elementAt(i));
         count.set(data.elementAt(i), count.elementAt(data.elementAt(i)) - 1);
      }
   }
   public static void main (String[] args){
      Vector<Integer> myData = new Vector<Integer>();
      myData.add(11);
      myData.add(7);
      myData.add(12);
      myData.add(18);
      myData.add(21);
      myData.add(15);
      myData.add(14);
      myData.add(4);

      System.out.println("Original Data: " + myData);

      Integer max = myData.elementAt(0);
      for (int i = 1; i < myData.size(); i++){
         if (myData.elementAt(i) > max){
            max = myData.elementAt(i);
         }
      }
      Vector <Integer> mySortedData = new Vector<Integer>();
      counting_sort(myData, mySortedData, max);
      System.out.println("Sorted Data: " + mySortedData);

   }
}
```

$A$ is the input array of integers. $B$ is the output array. $k$ is the largest element in $A$, which should be identified prior to calling COUNTING-SORT.

### 2.3 Time Complexity Analysis

Where $n$ is the number of data points in $A$, analyze the time complexity of the given algorithm with respect to $n$. Write the result of your analysis in big-$O$ notation, i.e. $O(n^2)$ in the space below.

O(n+k)

.....

**Name, SID, Date** .....

### 2.4 Space Complexity Analysis

Where $n$ is the number of data points in $A$, analyze the space complexity of the given algorithm with respect to $n$. Write the result of your analysis in big-$O$ notation, i.e. $O(n \cdot log(n))$ in the space below.

O(n+k)

### 2.5 New Algorithm Implementation

Implement the given algorithm in Java, and now generalize it to work for generic data. Your implementation needs to be able to sort collections of user-defined objects, given that the following operators have been defined for the object:

- maximal index corresponding to maximal value: `k` in the integer case as input to the given algorithm. For example, if you were using the capital letters of the English alphabet, you may choose Z, being 25 (if A is 0) to represent the maximal value, and 25 to be the index corresponding to it.
- integer index corresponding to the value of the user-defined object. This is similar to maximal index, but this is the general case. This is used in lines 5, 11, and 12 of the given algorithm.
- default value: `0` in the integer case in line 3 of the given algorithm. Use a default constructor here.
- assignment =
- increment ++
- decrement −−
- addition +

```
import java.util.*;

public class CountingSortOptimization {
    public static <T extends Comparable<? super T>> void counting_sort(Vector<T> data, Vector<T> sortedData, T max) {
        HashMap<T, Integer> count = new HashMap<>();

        // Initialize count map
        for (T element : data) {
            count.put(element, 0);
        }
        // Count occurrences of each element
        for (T element : data) {
            count.put(element, count.get(element) + 1);
        }
        // Calculate cumulative counts
        ArrayList<T> keys = new ArrayList<>(count.keySet());
        Collections.sort(keys);
        for (int i = 1; i < keys.size(); i++) {
            count.put(keys.get(i), count.get(keys.get(i)) + count.get(keys.get(i - 1)));
        }
        // Build the sorted output array
        for (int i = data.size() - 1; i >= 0; i--) {
            sortedData.add(null); // Adjust size to avoid IndexOutOfBoundsException
        }
        for (int i = data.size() - 1; i >= 0; i--) {
            sortedData.set(count.get(data.get(i)) - 1, data.get(i));
            count.put(data.get(i), count.get(data.get(i)) - 1);
        }
    }
    public static void main(String[] args) {
        Vector<String> myData = new Vector<>();
        myData.add("banana");
        myData.add("apple");
        myData.add("orange");
        myData.add("grape");
        myData.add("kiwi");
        myData.add("pear");
        myData.add("strawberry");
        myData.add("blueberry");

        System.out.println("Original Data: " + myData);

        String max = Collections.max(myData);
        Vector<String> mySortedData = new Vector<>();
        counting_sort(myData, mySortedData, max);
        System.out.println("Sorted Data: " + mySortedData);
    }
}
```

## 3 What to Turn In

*Turn in one PDF or Word document on Blackboard, containing the following items.*

1. All pages scanned or photographed of the In Class Assignment completed document.
2. Any additional pages you used to complete the assignment.
3. All code created for the assignment, along with test cases.
4. One statement indicating which parts of your implementation(s) are working, and which parts are not.
5. Screenshots demonstrating the code working, if it is working.