Linear Sorting Algorithms:

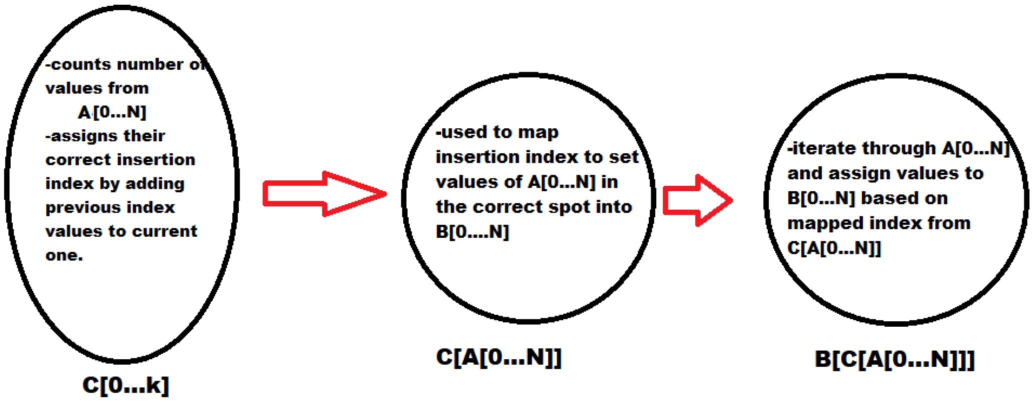
Counting Sort, Radix Sort, and Bucket Sort

Jia Siang Fung

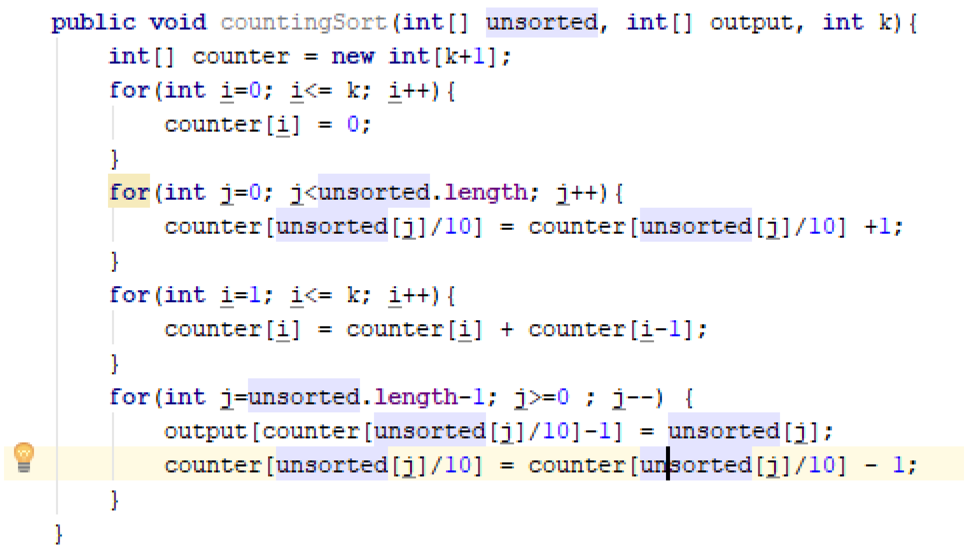
1. Design and Implementation:

1.1 Counting sort

Design:


-counts number of values from A[0…N] -assigns their correct insertion index by adding previous index values to current one.
C[0…k]
-used to map insertion index to set values of A[0…N] in the correct spot into B[0….N]
C[A[0…N]]
-iterate through A[0…N] and assign values to B[0…N] based on mapped index from C[A[0…N]]
B[C[A[0…N]]]

Implementation:

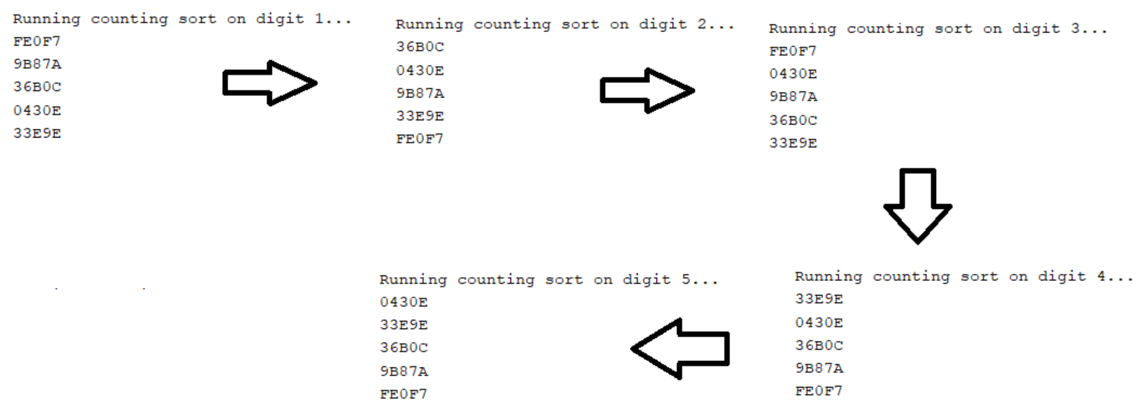


```
public void countingSort(int[] unsorted, int[] output, int k){
    int[] counter = new int[k+1];
    for(int i=0; i<= k; i++){
        counter[i] = 0;
    }
    for(int j=0; j<unsorted.length; j++){
        counter[unsorted[j]/10] = counter[unsorted[j]/10] +1;
    }
    for(int i=1; i<= k; i++){
        counter[i] = counter[i] + counter[i-1];
    }
    for(int j=unsorted.length-1; j>=0 ; j--) {
        output[counter[unsorted[j]/10]-1] = unsorted[j];
        counter[unsorted[j]/10] = counter[unsorted[j]/10] - 1;
    }
}
```

We start by initializing a new integer array ‘counter’ with size k+1 since our range will be from 0 to k. The values in counter are set to 0 since we haven’t counted any values yet. The purpose of ‘counter’ is to count the values in the input array ‘unsorted’ and keep track of the number of times they appear. counter[0…k] is then added up by the values before it to assign their correct insertion index. counter[0..k] is ready to be used to map values from unsorted[j] to its correct sorted index in ‘output’. Note that we’re using the first digit of each number in ‘unsorted’ to show the relative index of equal numbers. For example, 71 and 72 can be considered equal numbers, and would represent 7 in actual comparisons, hence why we divide by 10 when referencing the values in ‘unsorted’.

1.2 Radix Sort

Design:



```
Running counting sort on digit 1...
FE0F7
9B87A
36B0C
0430E
33E9E
```

```
Running counting sort on digit 2...
36B0C
0430E
9B87A
33E9E
FE0F7
```

```
Running counting sort on digit 3...
FE0F7
0430E
9B87A
36B0C
33E9E
```

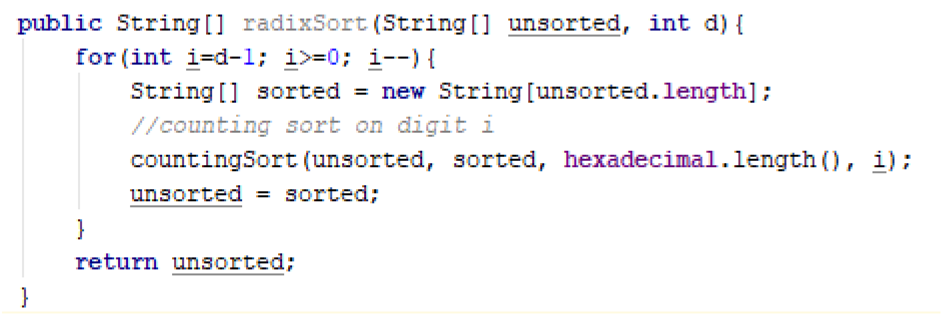
```
Running counting sort on digit 4...
33E9E
0430E
36B0C
9B87A
FE0F7
```

```
Running counting sort on digit 5...
0430E
33E9E
36B0C
9B87A
FE0F7
```

```
Sorted array: [0430E, 33E9E, 36B0C, 9B87A, FE0F7]
```

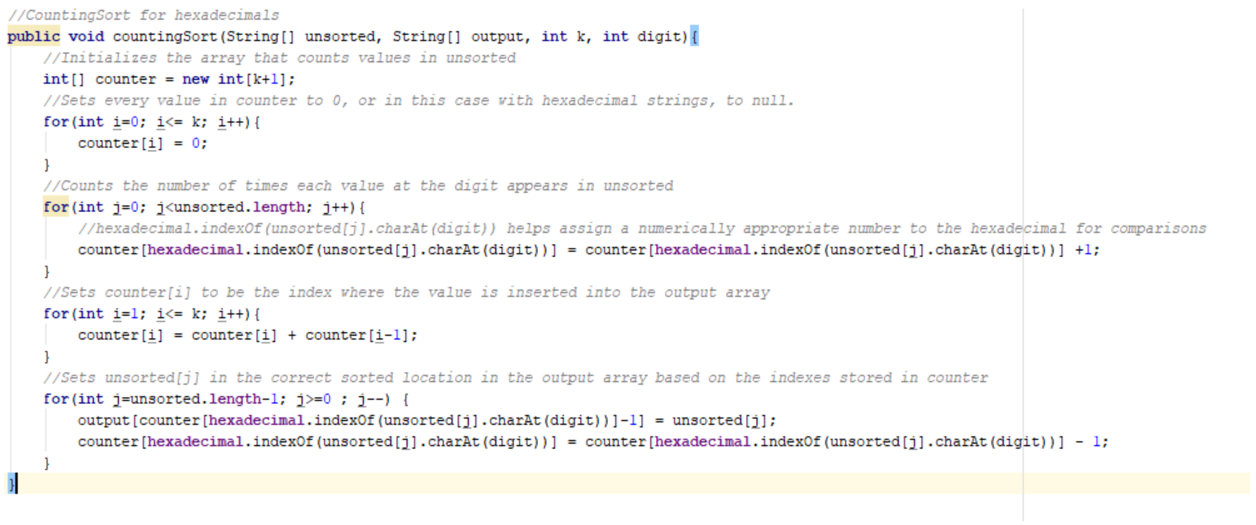
Counting sort is called on every digit starting from the 1st digit ending on the 5th digit. This ensures that the algorithm is stable and allows our algorithm to run in linear time.

Implementation:



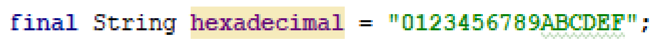
```
public String[] radixSort(String[] unsorted, int d){
    for(int i=d-1; i>=0; i--){
        String[] sorted = new String[unsorted.length];
        //counting sort on digit i
        countingSort(unsorted, sorted, hexadecimal.length(), i);
        unsorted = sorted;
    }
    return unsorted;
}
```

Radix sort calls counting sort on a digit i, going from right-most digit decrementing to left-most. Sorting this way ensures the algorithm is still stable, and still have correctly sorted values for the cases where one digit is equal to another so the next digit between the numbers have to be compared.



```java
//CountingSort for hexadecimals
public void countingSort(String[] unsorted, String[] output, int k, int digit){
    //Initializes the array that counts values in unsorted
    int[] counter = new int[k+1];
    //Sets every value in counter to 0, or in this case with hexadecimal strings, to null.
    for(int i=0; i<= k; i++){
        counter[i] = 0;
    }
    //Counts the number of times each value at the digit appears in unsorted
    for(int j=0; j<unsorted.length; j++){
        //hexadecimal.indexOf(unsorted[j].charAt(digit)) helps assign a numerically appropriate number to the hexadecimal for comparisons
        counter[hexadecimal.indexOf(unsorted[j].charAt(digit))] = counter[hexadecimal.indexOf(unsorted[j].charAt(digit))] +1;
    }
    //Sets counter[i] to be the index where the value is inserted into the output array
    for(int i=1; i<= k; i++){
        counter[i] = counter[i] + counter[i-1];
    }
    //Sets unsorted[j] in the correct sorted location in the output array based on the indexes stored in counter
    for(int j=unsorted.length-1; j>=0 ; j--) {
        output[counter[hexadecimal.indexOf(unsorted[j].charAt(digit))]-1] = unsorted[j];
        counter[hexadecimal.indexOf(unsorted[j].charAt(digit))] = counter[hexadecimal.indexOf(unsorted[j].charAt(digit))] - 1;
    }
}
```

This counting sort algorithm is tailored to only work with hexadecimals. It works by comparing the value at the digit to the corresponding index pulled from the string ‘hexadecimal’ as shown below.

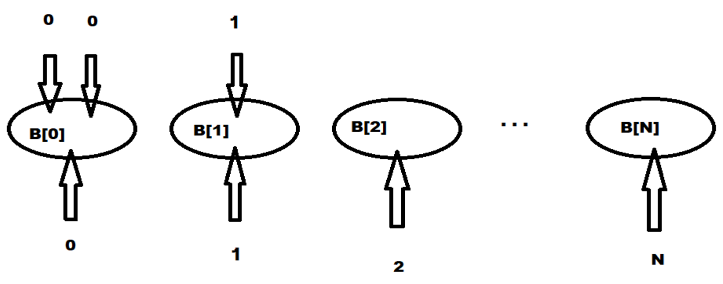


```java
final String hexadecimal = "0123456789ABCDEF";
```

We effectively tie numbers to the hexadecimals going from index 0 to 15 to be used in the comparisons for each digit of the number when running radix sort. This implementation of counting sort is very similar to the standard implementation shown in 1.1, with the major differences being the different base being used for the range (16 for hex and 10 for integers) and a digit parameter to keep track of the digit being compared when running radix sort.

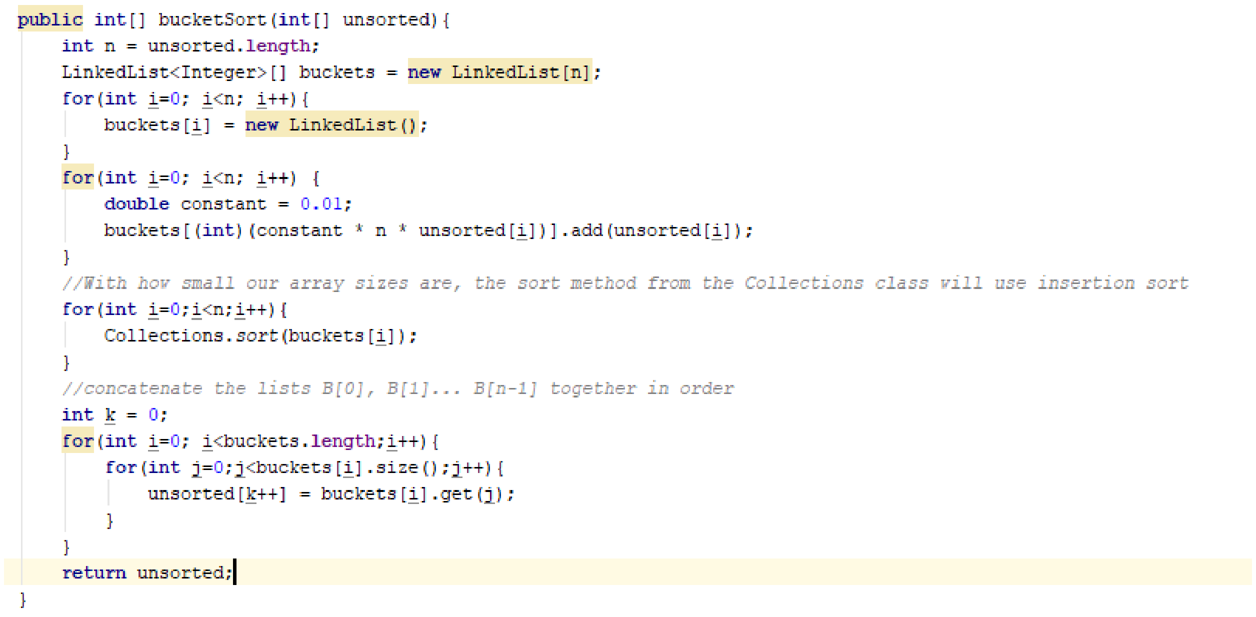
1.3 Bucket Sort

Design:



The idea behind bucket revolves around 'buckets' containing similar values from the input array, such that no one bucket is too full so we can sort each bucket efficiently using insertion sort. Note in the figure above, B[i] represents a bucket and the values going into the bucket aren't the actual values but are a way to denote that they're similar. The buckets B[0], B[1],…B[N] are concatenated at the end to form the sorted array.

Implementation:



```
public int[] bucketSort(int[] unsorted){
    int n = unsorted.length;
    LinkedList<Integer>[] buckets = new LinkedList[n];
    for(int i=0; i<n; i++){
        buckets[i] = new LinkedList();
    }
    for(int i=0; i<n; i++) {
        double constant = 0.01;
        buckets[(int)(constant * n * unsorted[i])].add(unsorted[i]);
    }
    //With how small our array sizes are, the sort method from the Collections class will use insertion sort
    for(int i=0;i<n;i++){
        Collections.sort(buckets[i]);
    }
    //concatenate the lists B[0], B[1]... B[n-1] together in order
    int k = 0;
    for(int i=0; i<buckets.length;i++){
        for(int j=0;j<buckets[i].size();j++){
            unsorted[k++] = buckets[i].get(j);
        }
    }
    return unsorted;
}
```

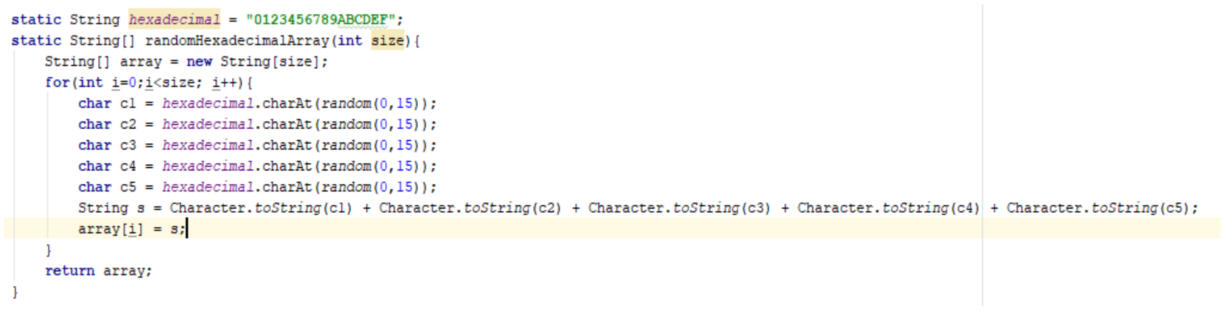
There are various ways to implement the buckets, but for my version I chose to use linked lists. The buckets array contains linked lists containing integers. Empty buckets are created and are assigned values from unsorted[i]. How choose bucket to insert the value into is by multiplying the the value at unsorted[i] by the constant 0.01. The point of the constant is to make sure that our integer at unsorted[i] is a double between 0 and 1.0. We then multiply that by the size of the unsorted array so that each bucket doesn't contain too many elements. The buckets are finally concatenated and returned.

2. List of Classes/Subroutines/Function calls:

Class:

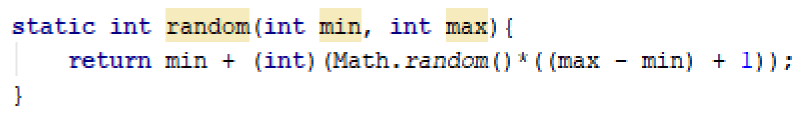
Main – runs the program that initializes a set of arrays and calls the Sorter class implementation for counting sort, radix sort, and bucket sort on them.

Main's methods:



```java
static String hexadecimal = "0123456789ABCDEF";
static String[] randomHexadecimalArray(int size){
    String[] array = new String[size];
    for(int i=0;i<size; i++){
        char c1 = hexadecimal.charAt(random(0,15));
        char c2 = hexadecimal.charAt(random(0,15));
        char c3 = hexadecimal.charAt(random(0,15));
        char c4 = hexadecimal.charAt(random(0,15));
        char c5 = hexadecimal.charAt(random(0,15));
        String s = Character.toString(c1) + Character.toString(c2) + Character.toString(c3) + Character.toString(c4) + Character.toString(c5);
        array[i] = s;
    }
    return array;
}
```

Creates an array of a given containing 5-digit hexadecimal strings. c1 – c5 represent a random hexadecimal digit created using the random method shown below. c1 – c5 are then concatenated to form a randomly generated 5-digit hex string, which is assigned to each index of the output array from 0 to size-1.

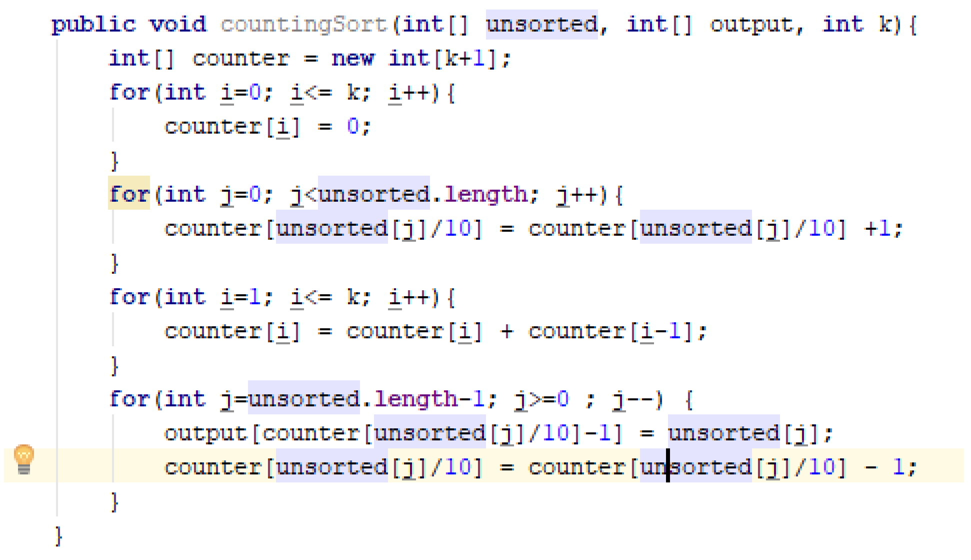


```java
static int random(int min, int max){
    return min + (int)(Math.random()*((max - min) + 1));
}
```

Returns a random integer from the integers min to max (inclusive). Math.random() returns a double from 0 to 1.0 and we can multiply that by the amount of numbers between min and max to range of output. Min is added to the result to shift the starting and end points so that the random number is from min to max as opposed to 0 to ((max-min) + 1).

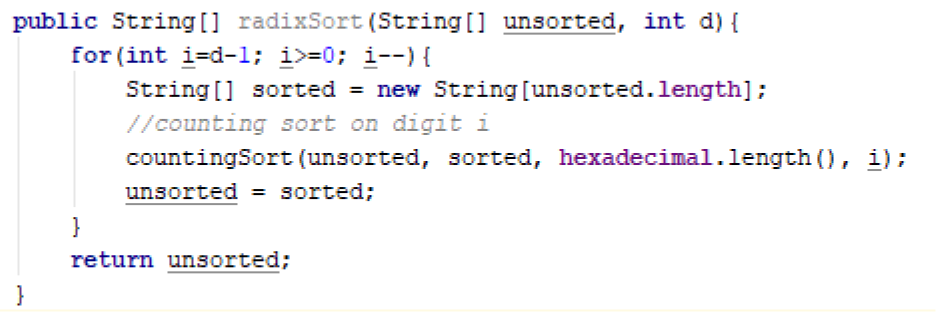
Class:
Sorter – A class representing a group of linear sorting algorithms - counting sort, radix sort, and bucket sort

Sorter's methods:



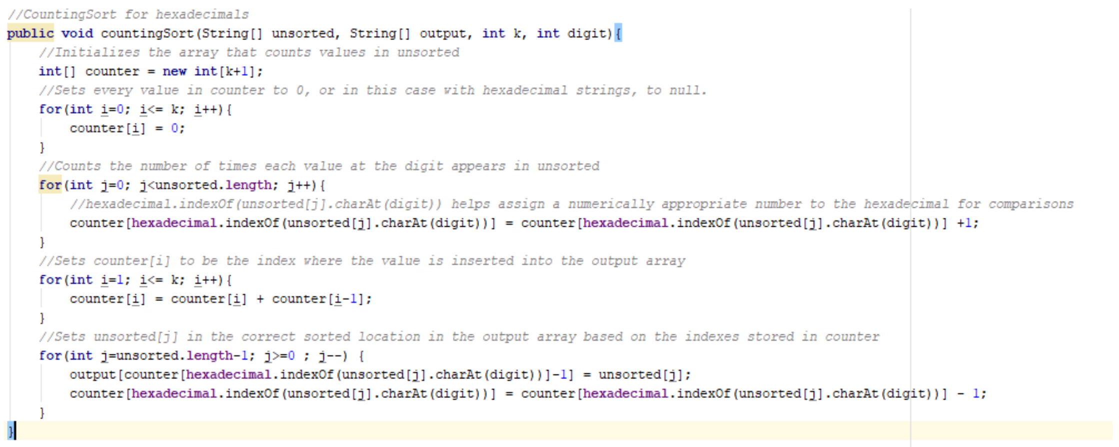
```java
public void countingSort(int[] unsorted, int[] output, int k){
    int[] counter = new int[k+1];
    for(int i=0; i<= k; i++){
        counter[i] = 0;
    }
    for(int j=0; j<unsorted.length; j++){
        counter[unsorted[j]/10] = counter[unsorted[j]/10] +1;
    }
    for(int i=1; i<= k; i++){
        counter[i] = counter[i] + counter[i-1];
    }
    for(int j=unsorted.length-1; j>=0 ; j--) {
        output[counter[unsorted[j]/10]-1] = unsorted[j];
        counter[unsorted[j]/10] = counter[unsorted[j]/10] - 1;
    }
}
```

This method runs a counting sort algorithm that sorts a given array of integers of range k and returns the sorted output in ascending order. It runs on the assumption that the first digit of every value in unsorted[j] represents the relative indexes of equal numbers, hence why we divide unsorted[j] by 10 in the comparisons and mapping. The first for loop sets every value in counter to 0 since we haven't counted anything yet. The second for loop counts the number of times each value at unsorted[j] appears. The third for loop sets counter[i] to be the index where the value is inserted into the output array. The last for loop sets the value at unsorted [j] in the correct sorted location in the output array based on the indexes stored in counter.



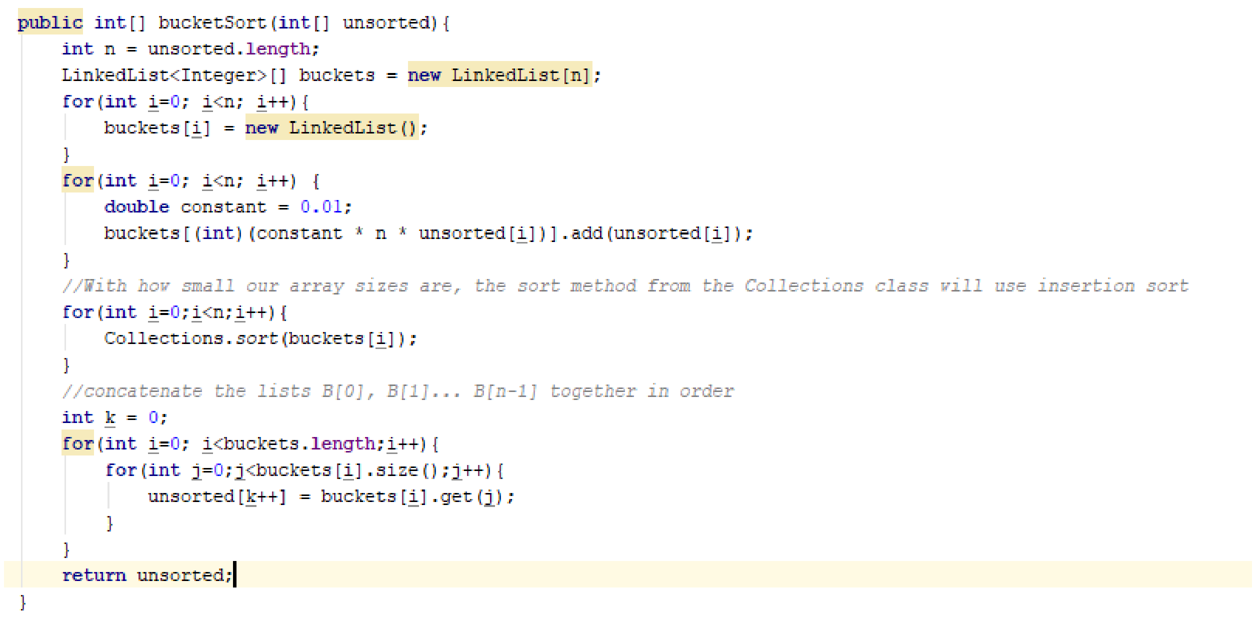
```java
public String[] radixSort(String[] unsorted, int d){
    for(int i=d-1; i>=0; i--){
        String[] sorted = new String[unsorted.length];
        //counting sort on digit i
        countingSort(unsorted, sorted, hexadecimal.length(), i);
        unsorted = sorted;
    }
    return unsorted;
}
```

This method runs a radix sort algorithm that calls the alternate counting sort method shown below on each digit starting from the first, of each value in the given array of hexadecimals and then returns the sorted array. Note that this algorithm is specialized to only work with arrays containing hexadecimal strings.



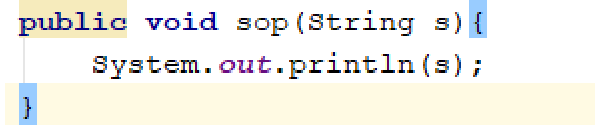
```java
//CountingSort for hexadecimals
public void countingSort(String[] unsorted, String[] output, int k, int digit){
    //Initializes the array that counts values in unsorted
    int[] counter = new int[k+1];
    //Sets every value in counter to 0, or in this case with hexadecimal strings, to null.
    for(int i=0; i<= k; i++){
        counter[i] = 0;
    }
    //Counts the number of times each value at the digit appears in unsorted
    for(int j=0; j<unsorted.length; j++){
        //hexadecimal.indexOf(unsorted[j].charAt(digit)) helps assign a numerically appropriate number to the hexadecimal for comparisons
        counter[hexadecimal.indexOf(unsorted[j].charAt(digit))] = counter[hexadecimal.indexOf(unsorted[j].charAt(digit))] +1;
    }
    //Sets counter[i] to be the index where the value is inserted into the output array
    for(int i=1; i<= k; i++){
        counter[i] = counter[i] + counter[i-1];
    }
    //Sets unsorted[j] in the correct sorted location in the output array based on the indexes stored in counter
    for(int j=unsorted.length-1; j>=0 ; j--) {
        output[counter[hexadecimal.indexOf(unsorted[j].charAt(digit))]-1] = unsorted[j];
        counter[hexadecimal.indexOf(unsorted[j].charAt(digit))] = counter[hexadecimal.indexOf(unsorted[j].charAt(digit))] - 1;
    }
}
```

This method runs a counting sort algorithm that sorts the given array of hexadecimal strings and sets the output to be the sorted array in ascending order. The implementation is very similar to the implementation of the counting sort shown above as each for loop fulfills the same purpose. The main difference between these two algorithms is that each hexadecimal digit being compared is mapped to a numerical index for comparison taken from the string 'hexadecimal' shown in 1.2.



```java
public int[] bucketSort(int[] unsorted){
    int n = unsorted.length;
    LinkedList<Integer>[] buckets = new LinkedList[n];
    for(int i=0; i<n; i++){
        buckets[i] = new LinkedList();
    }
    for(int i=0; i<n; i++) {
        double constant = 0.01;
        buckets[(int)(constant * n * unsorted[i])].add(unsorted[i]);
    }
    //With how small our array sizes are, the sort method from the Collections class will use insertion sort
    for(int i=0;i<n;i++){
        Collections.sort(buckets[i]);
    }
    //concatenate the lists B[0], B[1]... B[n-1] together in order
    int k = 0;
    for(int i=0; i<buckets.length;i++){
        for(int j=0;j<buckets[i].size();j++){
            unsorted[k++] = buckets[i].get(j);
        }
    }
    return unsorted;
}
```

This method runs a bucket sort algorithm that assigns similar values from a given array of integers into buckets that are sorted with insertion sort and then concatenated together to form a sorted array of integers that is retuned. The first for loop an empty bucket (linked list) containing integers to each index of the array ‘buckets’. The second for loop assigns similar values in ‘unsorted’ and ensures no bucket is too full. The third for loop sorts the values in each bucket using insertion sort. With how small our array sizes are, the sort method from the Collections class will use insertion sort. The final for loop concatenates the lists B[0], B[1],…, B[n-1] together in order, and then we return the sorted array.



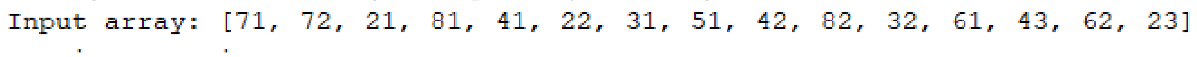
```java
public void sop(String s){
    System.out.println(s);
}
```

This method is functionally the same as System.out.println(String s). This method is purely to save time when writing print statements since I’d be writing many of them within subroutines of other methods during testing.

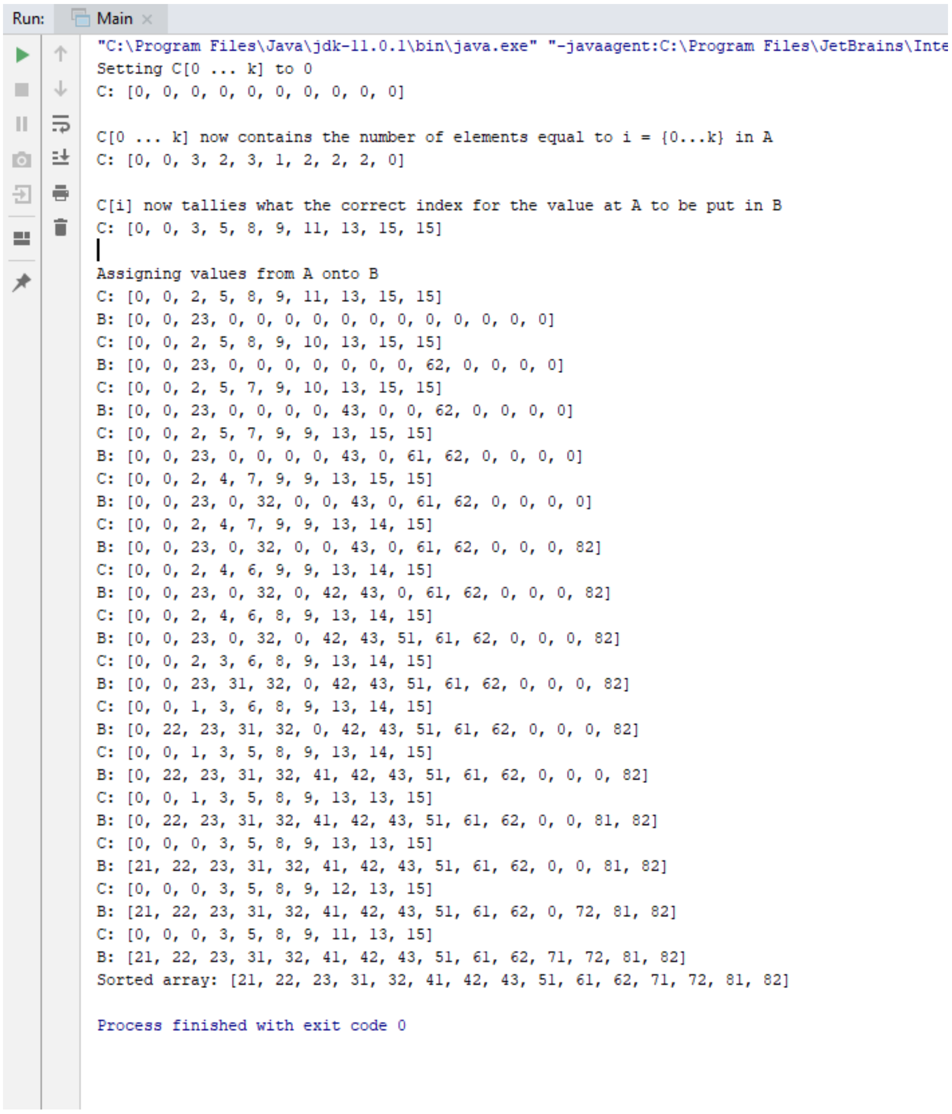
3. Self-testing Screenshots:

3.1 Counting Sort

Counting sort is called on the given input array with range of values from 0 to 9.



```
Input array: [71, 72, 21, 81, 41, 22, 31, 51, 42, 82, 32, 61, 43, 62, 23]
```

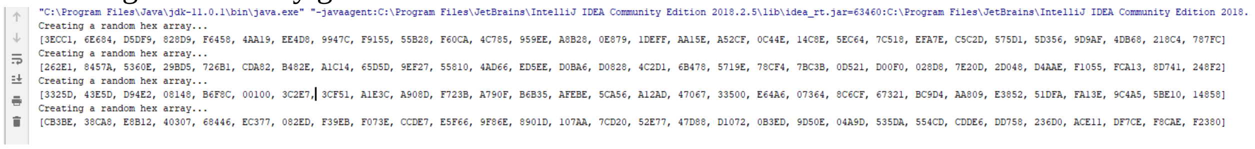


```
"C:\Program Files\Java\jdk-11.0.1\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\Inte
Setting C[0 ... k] to 0
C: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

C[0 ... k] now contains the number of elements equal to i = {0...k} in A
C: [0, 0, 3, 2, 3, 1, 2, 2, 2, 0]

C[i] now tallies what the correct index for the value at A to be put in B
C: [0, 0, 3, 5, 8, 9, 11, 13, 15, 15]

Assigning values from A onto B
C: [0, 0, 2, 5, 8, 9, 11, 13, 15, 15]
B: [0, 0, 23, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
C: [0, 0, 2, 5, 8, 9, 10, 13, 15, 15]
B: [0, 0, 23, 0, 0, 0, 0, 0, 0, 0, 62, 0, 0, 0, 0]
C: [0, 0, 2, 5, 7, 9, 10, 13, 15, 15]
B: [0, 0, 23, 0, 0, 0, 0, 43, 0, 0, 62, 0, 0, 0, 0]
C: [0, 0, 2, 5, 7, 9, 9, 13, 15, 15]
B: [0, 0, 23, 0, 0, 0, 0, 43, 0, 61, 62, 0, 0, 0, 0]
C: [0, 0, 2, 4, 7, 9, 9, 13, 15, 15]
B: [0, 0, 23, 0, 32, 0, 0, 43, 0, 61, 62, 0, 0, 0, 0]
C: [0, 0, 2, 4, 7, 9, 9, 13, 14, 15]
B: [0, 0, 23, 0, 32, 0, 0, 43, 0, 61, 62, 0, 0, 0, 82]
C: [0, 0, 2, 4, 6, 9, 9, 13, 14, 15]
B: [0, 0, 23, 0, 32, 0, 42, 43, 0, 61, 62, 0, 0, 0, 82]
C: [0, 0, 2, 4, 6, 8, 9, 13, 14, 15]
B: [0, 0, 23, 0, 32, 0, 42, 43, 51, 61, 62, 0, 0, 0, 82]
C: [0, 0, 2, 3, 6, 8, 9, 13, 14, 15]
B: [0, 0, 23, 31, 32, 0, 42, 43, 51, 61, 62, 0, 0, 0, 82]
C: [0, 0, 1, 3, 6, 8, 9, 13, 14, 15]
B: [0, 22, 23, 31, 32, 0, 42, 43, 51, 61, 62, 0, 0, 0, 82]
C: [0, 0, 1, 3, 5, 8, 9, 13, 14, 15]
B: [0, 22, 23, 31, 32, 41, 42, 43, 51, 61, 62, 0, 0, 0, 82]
C: [0, 0, 1, 3, 5, 8, 9, 13, 13, 15]
B: [0, 22, 23, 31, 32, 41, 42, 43, 51, 61, 62, 0, 0, 81, 82]
C: [0, 0, 0, 3, 5, 8, 9, 13, 13, 15]
B: [21, 22, 23, 31, 32, 41, 42, 43, 51, 61, 62, 0, 0, 81, 82]
C: [0, 0, 0, 3, 5, 8, 9, 12, 13, 15]
B: [21, 22, 23, 31, 32, 41, 42, 43, 51, 61, 62, 0, 72, 81, 82]
C: [0, 0, 0, 3, 5, 8, 9, 11, 13, 15]
B: [21, 22, 23, 31, 32, 41, 42, 43, 51, 61, 62, 71, 72, 81, 82]
Sorted array: [21, 22, 23, 31, 32, 41, 42, 43, 51, 61, 62, 71, 72, 81, 82]

Process finished with exit code 0
```

Note that the first digit showing relative indexes of equal numbers are in numerical order, showing that this algorithm is stable.

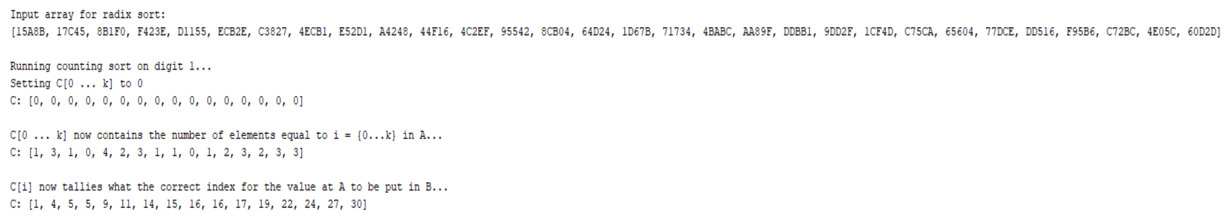
3.2 Radix Sort

Calling randomHexadecimalArray() four times with an input of size 30 to create four arrays containing 30 randomly generated hexadecimals

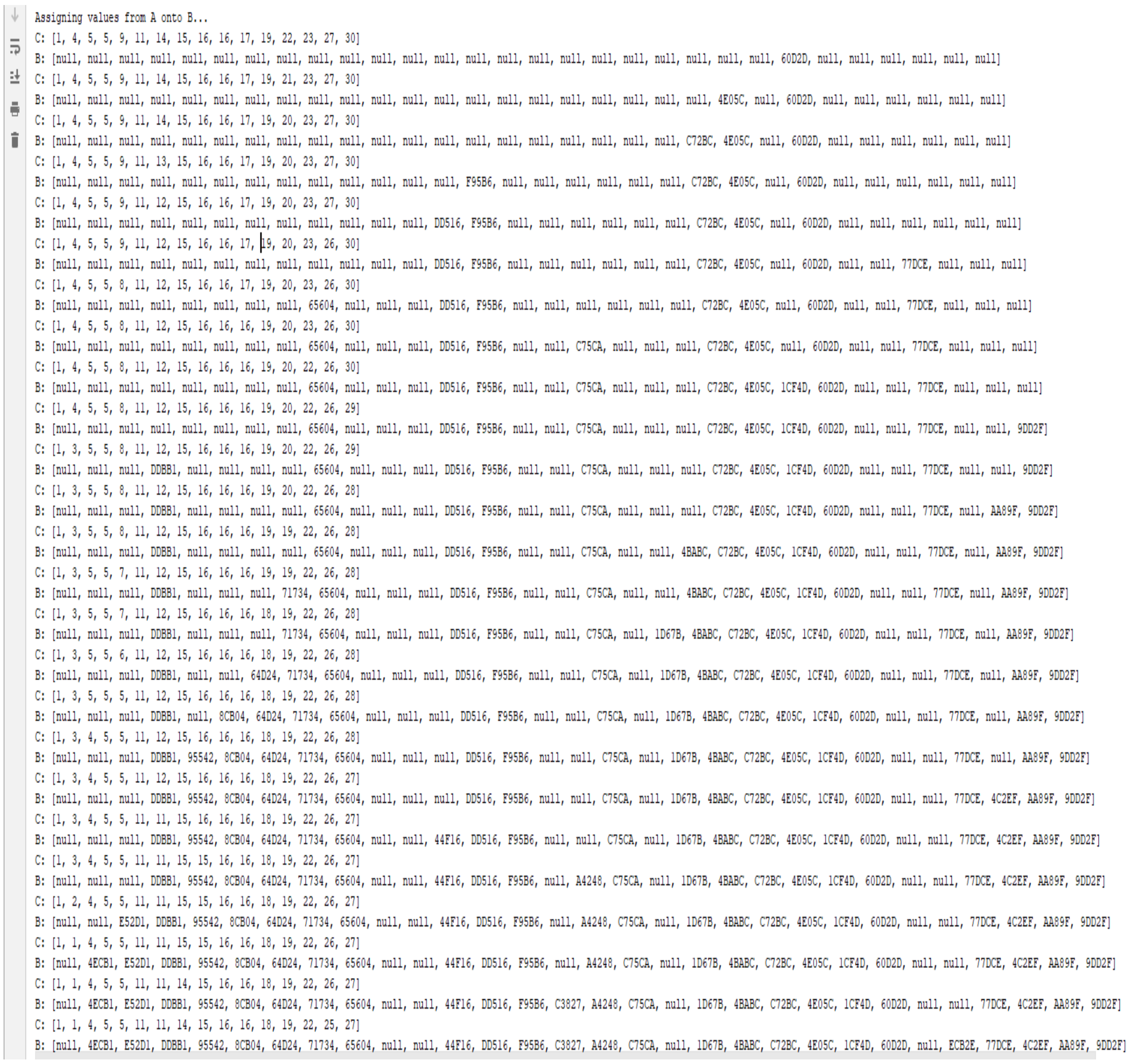


```
"C:\Program Files\Java\jdk-11.0.1\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2018.2.5\lib\idea_rt.jar=63460:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2018.
Creating a random hex array...
[3ECC1, 6E684, D5DF9, 828D9, F6458, 4AA19, EE4D8, 9947C, F9155, 55B28, F60CA, 4C785, 959EE, A8B28, 0E879, 1DEFF, AA15E, A52CF, 0C44E, 14C8E, 5EC64, 7C518, EFA7E, C5C2D, 575D1, 5D356, 9D9AF, 4DB68, 218C4, 787FC]
Creating a random hex array...
[262E1, 8457A, 5360E, 29BD5, 726B1, CDA82, B482E, A1C14, 65D5D, 9EF27, 55810, 4AD66, ED5EE, D0BA6, D0828, 4C2D1, 6B478, 5719E, 78CF4, 7BC3B, 0D521, D00F0, 028D8, 7E20D, 2D048, D4AAE, F1055, FCA13, 8D741, 248F2]
Creating a random hex array...
[3325D, 43E5D, D94E2, 08148, B6F8C, 00100, 3C2E7, 3CF51, A1E3C, A908D, F723B, A790F, B6B35, AFEBE, 5CA56, A12AD, 47067, 33500, E64A6, 07364, 8C6CF, 67321, BC9D4, AA809, E3852, 51DFA, FA13E, 9C4A5, 5BE10, 14858]
Creating a random hex array...
[CB3BE, 38CA8, E8B12, 40307, 68446, EC377, 082ED, F39EB, F073E, CCDE7, E5F66, 9F86E, 8901D, 107AA, 7CD20, 52E77, 47D88, D1072, 0B3ED, 9D50E, 04A9D, 535DA, 554CD, CDDE6, DD758, 236D0, ACE11, DF7CE, F8CAE, F2380]
```

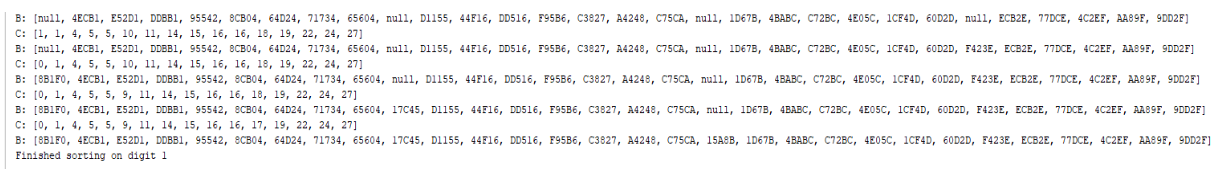
Radix sort called on the randomly generated input array below, showing every intermediate result of the array for counting ‘C’ and the output array ‘B’ for each iteration of counting sort from digits 1 to 5



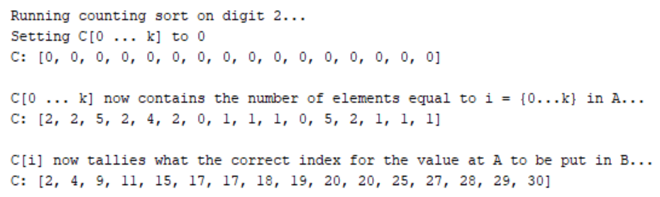
```
Input array for radix sort:
[15A8B, 17C45, 8B1F0, F423E, D1155, ECB2E, C3827, 4ECB1, E52D1, A4248, 44F16, 4C2EF, 95542, 8CB04, 64D24, 1D67B, 71734, 4BABC, AA89F, DDBB1, 9DD2F, 1CF4D, C75CA, 65604, 77DCE, DD516, F95B6, C72BC, 4E05C, 60D2D]

Running counting sort on digit 1...
Setting C[0 ... k] to 0
C: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

C[0 ... k] now contains the number of elements equal to i = {0...k} in A...
C: [1, 3, 1, 0, 4, 2, 3, 1, 1, 0, 1, 2, 3, 2, 3, 3]

C[i] now tallies what the correct index for the value at A to be put in B...
C: [1, 4, 5, 5, 9, 11, 14, 15, 16, 16, 17, 19, 22, 24, 27, 30]
```



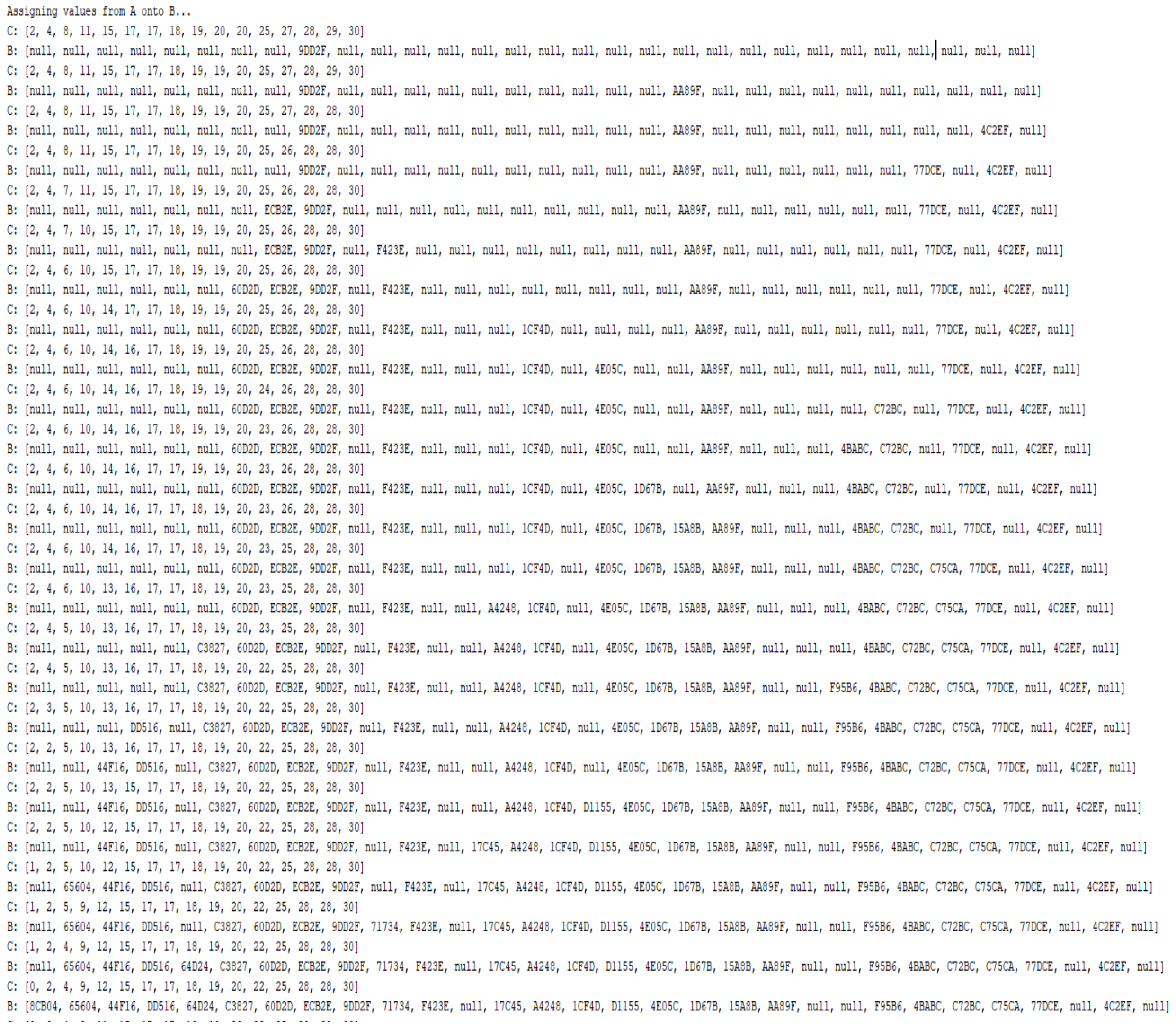
```
Assigning values from A onto B...
C: [1, 4, 5, 5, 9, 11, 14, 15, 16, 16, 17, 19, 22, 23, 27, 30]
B: [null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, 60D2D, null, null, null, null, null, null]
C: [1, 4, 5, 5, 9, 11, 14, 15, 16, 16, 17, 19, 21, 23, 27, 30]
B: [null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, 4E05C, null, 60D2D, null, null, null, null, null, null]
C: [1, 4, 5, 5, 9, 11, 14, 15, 16, 16, 17, 19, 20, 23, 27, 30]
B: [null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, C72BC, 4E05C, null, 60D2D, null, null, null, null, null, null]
C: [1, 4, 5, 5, 9, 11, 13, 15, 16, 16, 17, 19, 20, 23, 27, 30]
B: [null, null, null, null, null, null, null, null, null, null, null, null, null, F95B6, null, null, null, null, null, null, C72BC, 4E05C, null, 60D2D, null, null, null, null, null, null]
C: [1, 4, 5, 5, 9, 11, 12, 15, 16, 16, 17, 19, 20, 23, 27, 30]
B: [null, null, null, null, null, null, null, null, null, null, null, null, DD516, F95B6, null, null, null, null, null, null, C72BC, 4E05C, null, 60D2D, null, null, null, null, null, null]
C: [1, 4, 5, 5, 9, 11, 12, 15, 16, 16, 17, 19, 20, 23, 26, 30]
B: [null, null, null, null, null, null, null, null, null, null, null, null, DD516, F95B6, null, null, null, null, null, null, C72BC, 4E05C, null, 60D2D, null, null, 77DCE, null, null, null]
C: [1, 4, 5, 5, 8, 11, 12, 15, 16, 16, 17, 19, 20, 23, 26, 30]
B: [null, null, null, null, null, null, null, null, 65604, null, null, null, DD516, F95B6, null, null, null, null, null, null, C72BC, 4E05C, null, 60D2D, null, null, 77DCE, null, null, null]
C: [1, 4, 5, 5, 8, 11, 12, 15, 16, 16, 16, 19, 20, 23, 26, 30]
B: [null, null, null, null, null, null, null, null, 65604, null, null, null, DD516, F95B6, null, null, C75CA, null, null, null, C72BC, 4E05C, null, 60D2D, null, null, 77DCE, null, null, null]
C: [1, 4, 5, 5, 8, 11, 12, 15, 16, 16, 16, 19, 20, 22, 26, 30]
B: [null, null, null, null, null, null, null, null, 65604, null, null, null, DD516, F95B6, null, null, C75CA, null, null, null, C72BC, 4E05C, 1CF4D, 60D2D, null, null, 77DCE, null, null, null]
C: [1, 4, 5, 5, 8, 11, 12, 15, 16, 16, 16, 19, 20, 22, 26, 29]
B: [null, null, null, null, null, null, null, null, 65604, null, null, null, DD516, F95B6, null, null, C75CA, null, null, null, C72BC, 4E05C, 1CF4D, 60D2D, null, null, 77DCE, null, null, 9DD2F]
C: [1, 3, 5, 5, 8, 11, 12, 15, 16, 16, 16, 19, 20, 22, 26, 29]
B: [null, null, null, DDBB1, null, null, null, null, 65604, null, null, null, DD516, F95B6, null, null, C75CA, null, null, null, C72BC, 4E05C, 1CF4D, 60D2D, null, null, 77DCE, null, null, 9DD2F]
C: [1, 3, 5, 5, 8, 11, 12, 15, 16, 16, 16, 19, 20, 22, 26, 28]
B: [null, null, null, DDBB1, null, null, null, null, 65604, null, null, null, DD516, F95B6, null, null, C75CA, null, null, null, C72BC, 4E05C, 1CF4D, 60D2D, null, null, 77DCE, null, AA89F, 9DD2F]
C: [1, 3, 5, 5, 8, 11, 12, 15, 16, 16, 16, 19, 19, 22, 26, 28]
B: [null, null, null, DDBB1, null, null, null, null, 65604, null, null, null, DD516, F95B6, null, null, C75CA, null, null, 4BABC, C72BC, 4E05C, 1CF4D, 60D2D, null, null, 77DCE, null, AA89F, 9DD2F]
C: [1, 3, 5, 5, 7, 11, 12, 15, 16, 16, 16, 19, 19, 22, 26, 28]
B: [null, null, null, DDBB1, null, null, null, 71734, 65604, null, null, null, DD516, F95B6, null, null, C75CA, null, null, 4BABC, C72BC, 4E05C, 1CF4D, 60D2D, null, null, 77DCE, null, AA89F, 9DD2F]
C: [1, 3, 5, 5, 7, 11, 12, 15, 16, 16, 16, 18, 19, 22, 26, 28]
B: [null, null, null, DDBB1, null, null, null, 71734, 65604, null, null, null, DD516, F95B6, null, null, C75CA, null, 1D67B, 4BABC, C72BC, 4E05C, 1CF4D, 60D2D, null, null, 77DCE, null, AA89F, 9DD2F]
C: [1, 3, 5, 5, 6, 11, 12, 15, 16, 16, 16, 18, 19, 22, 26, 28]
B: [null, null, null, DDBB1, null, null, 64D24, 71734, 65604, null, null, null, DD516, F95B6, null, null, C75CA, null, 1D67B, 4BABC, C72BC, 4E05C, 1CF4D, 60D2D, null, null, 77DCE, null, AA89F, 9DD2F]
C: [1, 3, 5, 5, 5, 11, 12, 15, 16, 16, 16, 18, 19, 22, 26, 28]
B: [null, null, null, DDBB1, null, 8CB04, 64D24, 71734, 65604, null, null, null, DD516, F95B6, null, null, C75CA, null, 1D67B, 4BABC, C72BC, 4E05C, 1CF4D, 60D2D, null, null, 77DCE, null, AA89F, 9DD2F]
C: [1, 3, 4, 5, 5, 11, 12, 15, 16, 16, 16, 18, 19, 22, 26, 28]
B: [null, null, null, DDBB1, 95542, 8CB04, 64D24, 71734, 65604, null, null, null, DD516, F95B6, null, null, C75CA, null, 1D67B, 4BABC, C72BC, 4E05C, 1CF4D, 60D2D, null, null, 77DCE, null, AA89F, 9DD2F]
C: [1, 3, 4, 5, 5, 11, 12, 15, 16, 16, 16, 18, 19, 22, 26, 27]
B: [null, null, null, DDBB1, 95542, 8CB04, 64D24, 71734, 65604, null, null, null, DD516, F95B6, null, null, C75CA, null, 1D67B, 4BABC, C72BC, 4E05C, 1CF4D, 60D2D, null, null, 77DCE, 4C2EF, AA89F, 9DD2F]
C: [1, 3, 4, 5, 5, 11, 11, 15, 16, 16, 16, 18, 19, 22, 26, 27]
B: [null, null, null, DDBB1, 95542, 8CB04, 64D24, 71734, 65604, null, null, 44F16, DD516, F95B6, null, null, C75CA, null, 1D67B, 4BABC, C72BC, 4E05C, 1CF4D, 60D2D, null, null, 77DCE, 4C2EF, AA89F, 9DD2F]
C: [1, 3, 4, 5, 5, 11, 11, 15, 15, 16, 16, 18, 19, 22, 26, 27]
B: [null, null, null, DDBB1, 95542, 8CB04, 64D24, 71734, 65604, null, null, 44F16, DD516, F95B6, null, A4248, C75CA, null, 1D67B, 4BABC, C72BC, 4E05C, 1CF4D, 60D2D, null, null, 77DCE, 4C2EF, AA89F, 9DD2F]
C: [1, 2, 4, 5, 5, 11, 11, 15, 15, 16, 16, 18, 19, 22, 26, 27]
B: [null, null, E52D1, DDBB1, 95542, 8CB04, 64D24, 71734, 65604, null, null, 44F16, DD516, F95B6, null, A4248, C75CA, null, 1D67B, 4BABC, C72BC, 4E05C, 1CF4D, 60D2D, null, null, 77DCE, 4C2EF, AA89F, 9DD2F]
C: [1, 1, 4, 5, 5, 11, 11, 15, 15, 16, 16, 18, 19, 22, 26, 27]
B: [null, 4ECB1, E52D1, DDBB1, 95542, 8CB04, 64D24, 71734, 65604, null, null, 44F16, DD516, F95B6, null, A4248, C75CA, null, 1D67B, 4BABC, C72BC, 4E05C, 1CF4D, 60D2D, null, null, 77DCE, 4C2EF, AA89F, 9DD2F]
C: [1, 1, 4, 5, 5, 11, 11, 14, 15, 16, 16, 18, 19, 22, 26, 27]
B: [null, 4ECB1, E52D1, DDBB1, 95542, 8CB04, 64D24, 71734, 65604, null, null, 44F16, DD516, F95B6, C3827, A4248, C75CA, null, 1D67B, 4BABC, C72BC, 4E05C, 1CF4D, 60D2D, null, null, 77DCE, 4C2EF, AA89F, 9DD2F]
C: [1, 1, 4, 5, 5, 11, 11, 14, 15, 16, 16, 18, 19, 22, 25, 27]
B: [null, 4ECB1, E52D1, DDBB1, 95542, 8CB04, 64D24, 71734, 65604, null, null, 44F16, DD516, F95B6, C3827, A4248, C75CA, null, 1D67B, 4BABC, C72BC, 4E05C, 1CF4D, 60D2D, null, ECB2E, 77DCE, 4C2EF, AA89F, 9DD2F]
```



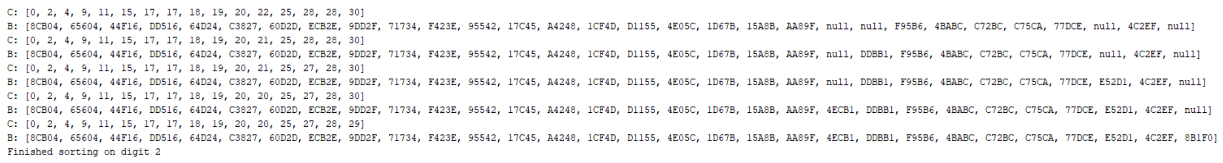
```
B: [null, 4ECB1, E52D1, DDBB1, 95542, 8CB04, 64D24, 71734, 65604, null, D1155, 44F16, DD516, F95B6, C3827, A4248, C75CA, null, 1D67B, 4BABC, C72BC, 4E05C, 1CF4D, 60D2D, null, ECB2E, 77DCE, 4C2EF, AA89F, 9DD2F]
C: [1, 1, 4, 5, 5, 10, 11, 14, 15, 16, 16, 18, 19, 22, 24, 27]
B: [null, 4ECB1, E52D1, DDBB1, 95542, 8CB04, 64D24, 71734, 65604, null, D1155, 44F16, DD516, F95B6, C3827, A4248, C75CA, null, 1D67B, 4BABC, C72BC, 4E05C, 1CF4D, 60D2D, F423E, ECB2E, 77DCE, 4C2EF, AA89F, 9DD2F]
C: [0, 1, 4, 5, 5, 10, 11, 14, 15, 16, 16, 18, 19, 22, 24, 27]
B: [8B1F0, 4ECB1, E52D1, DDBB1, 95542, 8CB04, 64D24, 71734, 65604, null, D1155, 44F16, DD516, F95B6, C3827, A4248, C75CA, null, 1D67B, 4BABC, C72BC, 4E05C, 1CF4D, 60D2D, F423E, ECB2E, 77DCE, 4C2EF, AA89F, 9DD2F]
C: [0, 1, 4, 5, 5, 9, 11, 14, 15, 16, 16, 18, 19, 22, 24, 27]
B: [8B1F0, 4ECB1, E52D1, DDBB1, 95542, 8CB04, 64D24, 71734, 65604, 17C45, D1155, 44F16, DD516, F95B6, C3827, A4248, C75CA, null, 1D67B, 4BABC, C72BC, 4E05C, 1CF4D, 60D2D, F423E, ECB2E, 77DCE, 4C2EF, AA89F, 9DD2F]
C: [0, 1, 4, 5, 5, 9, 11, 14, 15, 16, 16, 17, 19, 22, 24, 27]
B: [8B1F0, 4ECB1, E52D1, DDBB1, 95542, 8CB04, 64D24, 71734, 65604, 17C45, D1155, 44F16, DD516, F95B6, C3827, A4248, C75CA, 15A8B, 1D67B, 4BABC, C72BC, 4E05C, 1CF4D, 60D2D, F423E, ECB2E, 77DCE, 4C2EF, AA89F, 9DD2F]
Finished sorting on digit 1
```



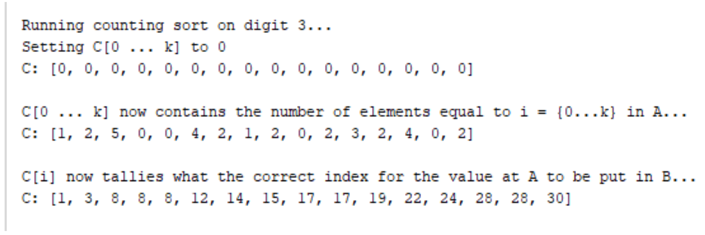
```
Running counting sort on digit 2...
Setting C[0 ... k] to 0
C: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

C[0 ... k] now contains the number of elements equal to i = {0...k} in A...
C: [2, 2, 5, 2, 4, 2, 0, 1, 1, 1, 0, 5, 2, 1, 1, 1]

C[i] now tallies what the correct index for the value at A to be put in B...
C: [2, 4, 9, 11, 15, 17, 17, 18, 19, 20, 20, 25, 27, 28, 29, 30]
```



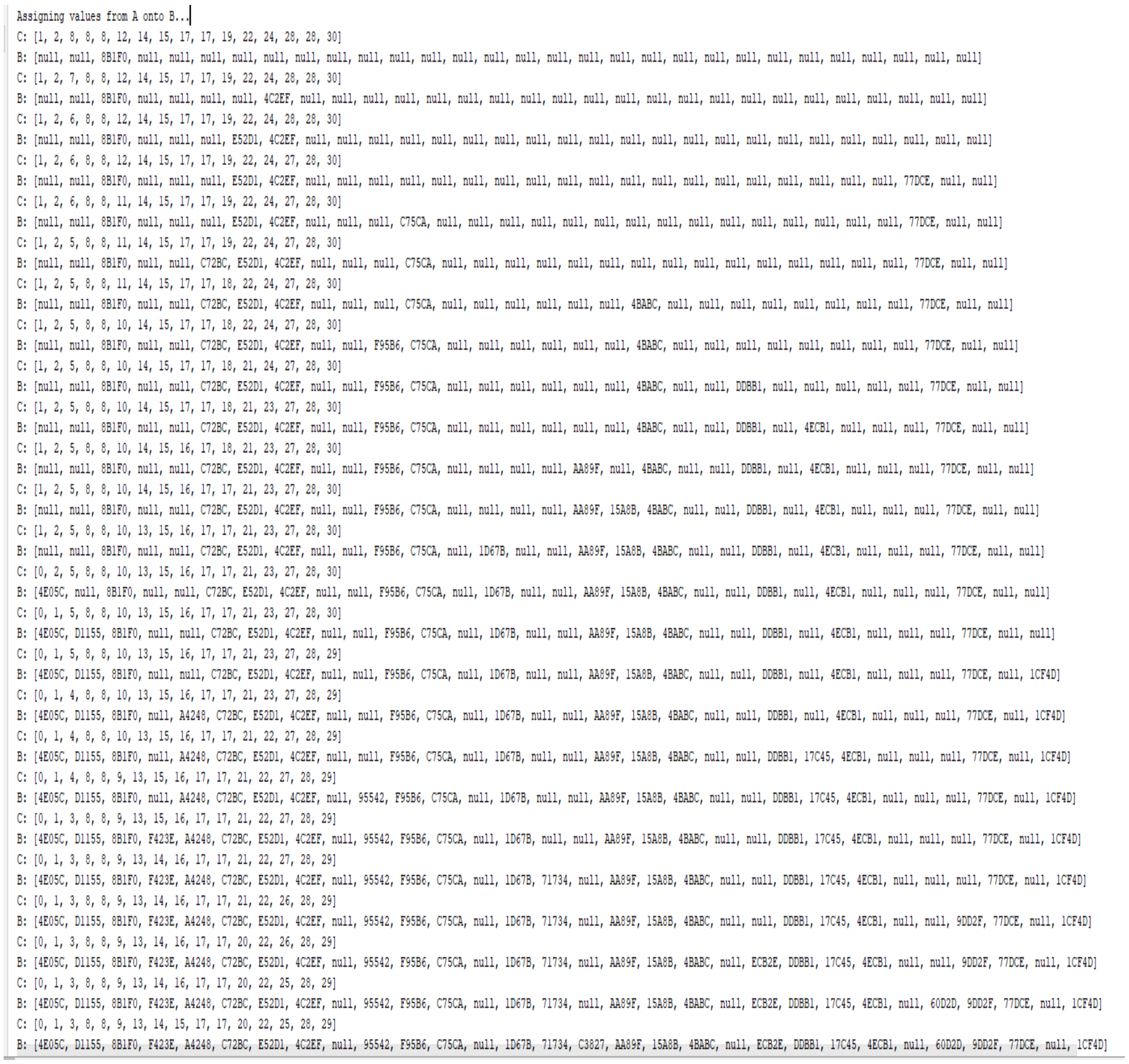
```
Assigning values from A onto B...
C: [2, 4, 8, 11, 15, 17, 17, 18, 19, 20, 20, 25, 27, 28, 29, 30]
B: [null, null, null, null, null, null, null, null, 9DD2F, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null]
C: [2, 4, 8, 11, 15, 17, 17, 18, 19, 19, 20, 25, 27, 28, 29, 30]
B: [null, null, null, null, null, null, null, null, 9DD2F, null, null, null, null, null, null, null, null, null, null, AA89F, null, null, null, null, null, null, null, null, null, null]
C: [2, 4, 8, 11, 15, 17, 17, 18, 19, 19, 20, 25, 27, 28, 28, 30]
B: [null, null, null, null, null, null, null, null, 9DD2F, null, null, null, null, null, null, null, null, null, null, AA89F, null, null, null, null, null, null, null, null, 4C2EF, null]
C: [2, 4, 8, 11, 15, 17, 17, 18, 19, 19, 20, 25, 26, 28, 28, 30]
B: [null, null, null, null, null, null, null, null, 9DD2F, null, null, null, null, null, null, null, null, null, null, AA89F, null, null, null, null, null, null, 77DCE, null, 4C2EF, null]
C: [2, 4, 7, 11, 15, 17, 17, 18, 19, 19, 20, 25, 26, 28, 28, 30]
B: [null, null, null, null, null, null, null, ECB2E, 9DD2F, null, null, null, null, null, null, null, null, null, null, AA89F, null, null, null, null, null, null, 77DCE, null, 4C2EF, null]
C: [2, 4, 7, 10, 15, 17, 17, 18, 19, 19, 20, 25, 26, 28, 28, 30]
B: [null, null, null, null, null, null, null, ECB2E, 9DD2F, null, F423E, null, null, null, null, null, null, null, null, AA89F, null, null, null, null, null, null, 77DCE, null, 4C2EF, null]
C: [2, 4, 6, 10, 15, 17, 17, 18, 19, 19, 20, 25, 26, 28, 28, 30]
B: [null, null, null, null, null, null, 60D2D, ECB2E, 9DD2F, null, F423E, null, null, null, null, null, null, null, null, AA89F, null, null, null, null, null, null, 77DCE, null, 4C2EF, null]
C: [2, 4, 6, 10, 14, 17, 17, 18, 19, 19, 20, 25, 26, 28, 28, 30]
B: [null, null, null, null, null, null, 60D2D, ECB2E, 9DD2F, null, F423E, null, null, null, 1CF4D, null, null, null, null, AA89F, null, null, null, null, null, null, 77DCE, null, 4C2EF, null]
C: [2, 4, 6, 10, 14, 16, 17, 18, 19, 19, 20, 25, 26, 28, 28, 30]
B: [null, null, null, null, null, null, 60D2D, ECB2E, 9DD2F, null, F423E, null, null, null, 1CF4D, null, 4E05C, null, null, AA89F, null, null, null, null, null, null, 77DCE, null, 4C2EF, null]
C: [2, 4, 6, 10, 14, 16, 17, 18, 19, 19, 20, 24, 26, 28, 28, 30]
B: [null, null, null, null, null, null, 60D2D, ECB2E, 9DD2F, null, F423E, null, null, null, 1CF4D, null, 4E05C, null, null, AA89F, null, null, null, null, C72BC, null, 77DCE, null, 4C2EF, null]
C: [2, 4, 6, 10, 14, 16, 17, 18, 19, 19, 20, 23, 26, 28, 28, 30]
B: [null, null, null, null, null, null, 60D2D, ECB2E, 9DD2F, null, F423E, null, null, null, 1CF4D, null, 4E05C, null, null, AA89F, null, null, null, 4BABC, C72BC, null, 77DCE, null, 4C2EF, null]
C: [2, 4, 6, 10, 14, 16, 17, 17, 19, 19, 20, 23, 26, 28, 28, 30]
B: [null, null, null, null, null, null, 60D2D, ECB2E, 9DD2F, null, F423E, null, null, null, 1CF4D, null, 4E05C, 1D67B, null, AA89F, null, null, null, 4BABC, C72BC, null, 77DCE, null, 4C2EF, null]
C: [2, 4, 6, 10, 14, 16, 17, 17, 18, 19, 20, 23, 26, 28, 28, 30]
B: [null, null, null, null, null, null, 60D2D, ECB2E, 9DD2F, null, F423E, null, null, null, 1CF4D, null, 4E05C, 1D67B, 15A8B, AA89F, null, null, null, 4BABC, C72BC, null, 77DCE, null, 4C2EF, null]
C: [2, 4, 6, 10, 14, 16, 17, 17, 18, 19, 20, 23, 25, 28, 28, 30]
B: [null, null, null, null, null, null, 60D2D, ECB2E, 9DD2F, null, F423E, null, null, null, 1CF4D, null, 4E05C, 1D67B, 15A8B, AA89F, null, null, null, 4BABC, C72BC, C75CA, 77DCE, null, 4C2EF, null]
C: [2, 4, 6, 10, 13, 16, 17, 17, 18, 19, 20, 23, 25, 28, 28, 30]
B: [null, null, null, null, null, null, 60D2D, ECB2E, 9DD2F, null, F423E, null, null, A4248, 1CF4D, null, 4E05C, 1D67B, 15A8B, AA89F, null, null, null, 4BABC, C72BC, C75CA, 77DCE, null, 4C2EF, null]
C: [2, 4, 5, 10, 13, 16, 17, 17, 18, 19, 20, 23, 25, 28, 28, 30]
B: [null, null, null, null, null, C3827, 60D2D, ECB2E, 9DD2F, null, F423E, null, null, A4248, 1CF4D, null, 4E05C, 1D67B, 15A8B, AA89F, null, null, null, 4BABC, C72BC, C75CA, 77DCE, null, 4C2EF, null]
C: [2, 4, 5, 10, 13, 16, 17, 17, 18, 19, 20, 22, 25, 28, 28, 30]
B: [null, null, null, null, null, C3827, 60D2D, ECB2E, 9DD2F, null, F423E, null, null, A4248, 1CF4D, null, 4E05C, 1D67B, 15A8B, AA89F, null, null, F95B6, 4BABC, C72BC, C75CA, 77DCE, null, 4C2EF, null]
C: [2, 3, 5, 10, 13, 16, 17, 17, 18, 19, 20, 22, 25, 28, 28, 30]
B: [null, null, null, DD516, null, C3827, 60D2D, ECB2E, 9DD2F, null, F423E, null, null, A4248, 1CF4D, null, 4E05C, 1D67B, 15A8B, AA89F, null, null, F95B6, 4BABC, C72BC, C75CA, 77DCE, null, 4C2EF, null]
C: [2, 2, 5, 10, 13, 16, 17, 17, 18, 19, 20, 22, 25, 28, 28, 30]
B: [null, null, 44F16, DD516, null, C3827, 60D2D, ECB2E, 9DD2F, null, F423E, null, null, A4248, 1CF4D, null, 4E05C, 1D67B, 15A8B, AA89F, null, null, F95B6, 4BABC, C72BC, C75CA, 77DCE, null, 4C2EF, null]
C: [2, 2, 5, 10, 13, 15, 17, 17, 18, 19, 20, 22, 25, 28, 28, 30]
B: [null, null, 44F16, DD516, null, C3827, 60D2D, ECB2E, 9DD2F, null, F423E, null, null, A4248, 1CF4D, D1155, 4E05C, 1D67B, 15A8B, AA89F, null, null, F95B6, 4BABC, C72BC, C75CA, 77DCE, null, 4C2EF, null]
C: [2, 2, 5, 10, 12, 15, 17, 17, 18, 19, 20, 22, 25, 28, 28, 30]
B: [null, null, 44F16, DD516, null, C3827, 60D2D, ECB2E, 9DD2F, null, F423E, null, 17C45, A4248, 1CF4D, D1155, 4E05C, 1D67B, 15A8B, AA89F, null, null, F95B6, 4BABC, C72BC, C75CA, 77DCE, null, 4C2EF, null]
C: [1, 2, 5, 10, 12, 15, 17, 17, 18, 19, 20, 22, 25, 28, 28, 30]
B: [null, 65604, 44F16, DD516, null, C3827, 60D2D, ECB2E, 9DD2F, null, F423E, null, 17C45, A4248, 1CF4D, D1155, 4E05C, 1D67B, 15A8B, AA89F, null, null, F95B6, 4BABC, C72BC, C75CA, 77DCE, null, 4C2EF, null]
C: [1, 2, 5, 9, 12, 15, 17, 17, 18, 19, 20, 22, 25, 28, 28, 30]
B: [null, 65604, 44F16, DD516, null, C3827, 60D2D, ECB2E, 9DD2F, 71734, F423E, null, 17C45, A4248, 1CF4D, D1155, 4E05C, 1D67B, 15A8B, AA89F, null, null, F95B6, 4BABC, C72BC, C75CA, 77DCE, null, 4C2EF, null]
C: [1, 2, 4, 9, 12, 15, 17, 17, 18, 19, 20, 22, 25, 28, 28, 30]
B: [null, 65604, 44F16, DD516, 64D24, C3827, 60D2D, ECB2E, 9DD2F, 71734, F423E, null, 17C45, A4248, 1CF4D, D1155, 4E05C, 1D67B, 15A8B, AA89F, null, null, F95B6, 4BABC, C72BC, C75CA, 77DCE, null, 4C2EF, null]
C: [0, 2, 4, 9, 12, 15, 17, 17, 18, 19, 20, 22, 25, 28, 28, 30]
B: [8CB04, 65604, 44F16, DD516, 64D24, C3827, 60D2D, ECB2E, 9DD2F, 71734, F423E, null, 17C45, A4248, 1CF4D, D1155, 4E05C, 1D67B, 15A8B, AA89F, null, null, F95B6, 4BABC, C72BC, C75CA, 77DCE, null, 4C2EF, null]
```



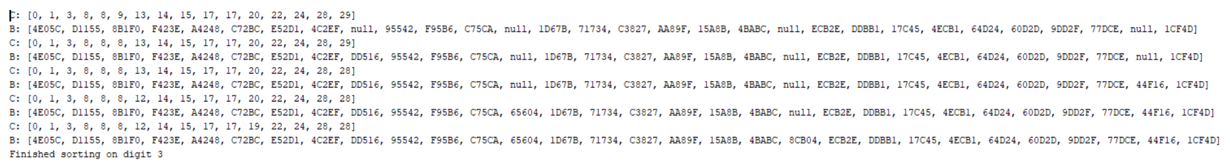
```
C: [0, 2, 4, 9, 11, 15, 17, 17, 18, 19, 20, 22, 25, 28, 28, 30]
B: [8CB04, 65604, 44F16, DD516, 64D24, C3827, 60D2D, ECB2E, 9DD2F, 71734, F423E, 95542, 17C45, A4248, 1CF4D, D1155, 4E05C, 1D67B, 15A8B, AA89F, null, null, F95B6, 4BABC, C72BC, C75CA, 77DCE, null, 4C2EF, null]
C: [0, 2, 4, 9, 11, 15, 17, 17, 18, 19, 20, 21, 25, 28, 28, 30]
B: [8CB04, 65604, 44F16, DD516, 64D24, C3827, 60D2D, ECB2E, 9DD2F, 71734, F423E, 95542, 17C45, A4248, 1CF4D, D1155, 4E05C, 1D67B, 15A8B, AA89F, null, DDBB1, F95B6, 4BABC, C72BC, C75CA, 77DCE, null, 4C2EF, null]
C: [0, 2, 4, 9, 11, 15, 17, 17, 18, 19, 20, 21, 25, 27, 28, 30]
B: [8CB04, 65604, 44F16, DD516, 64D24, C3827, 60D2D, ECB2E, 9DD2F, 71734, F423E, 95542, 17C45, A4248, 1CF4D, D1155, 4E05C, 1D67B, 15A8B, AA89F, null, DDBB1, F95B6, 4BABC, C72BC, C75CA, 77DCE, E52D1, 4C2EF, null]
C: [0, 2, 4, 9, 11, 15, 17, 17, 18, 19, 20, 20, 25, 27, 28, 30]
B: [8CB04, 65604, 44F16, DD516, 64D24, C3827, 60D2D, ECB2E, 9DD2F, 71734, F423E, 95542, 17C45, A4248, 1CF4D, D1155, 4E05C, 1D67B, 15A8B, AA89F, 4ECB1, DDBB1, F95B6, 4BABC, C72BC, C75CA, 77DCE, E52D1, 4C2EF, null]
C: [0, 2, 4, 9, 11, 15, 17, 17, 18, 19, 20, 20, 25, 27, 28, 29]
B: [8CB04, 65604, 44F16, DD516, 64D24, C3827, 60D2D, ECB2E, 9DD2F, 71734, F423E, 95542, 17C45, A4248, 1CF4D, D1155, 4E05C, 1D67B, 15A8B, AA89F, 4ECB1, DDBB1, F95B6, 4BABC, C72BC, C75CA, 77DCE, E52D1, 4C2EF, 8B1F0]
Finished sorting on digit 2
```



```
Running counting sort on digit 3...
Setting C[0 ... k] to 0
C: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

C[0 ... k] now contains the number of elements equal to i = {0...k} in A...
C: [1, 2, 5, 0, 0, 4, 2, 1, 2, 0, 2, 3, 2, 4, 0, 2]

C[i] now tallies what the correct index for the value at A to be put in B...
C: [1, 3, 8, 8, 8, 12, 14, 15, 17, 17, 19, 22, 24, 28, 28, 30]
```

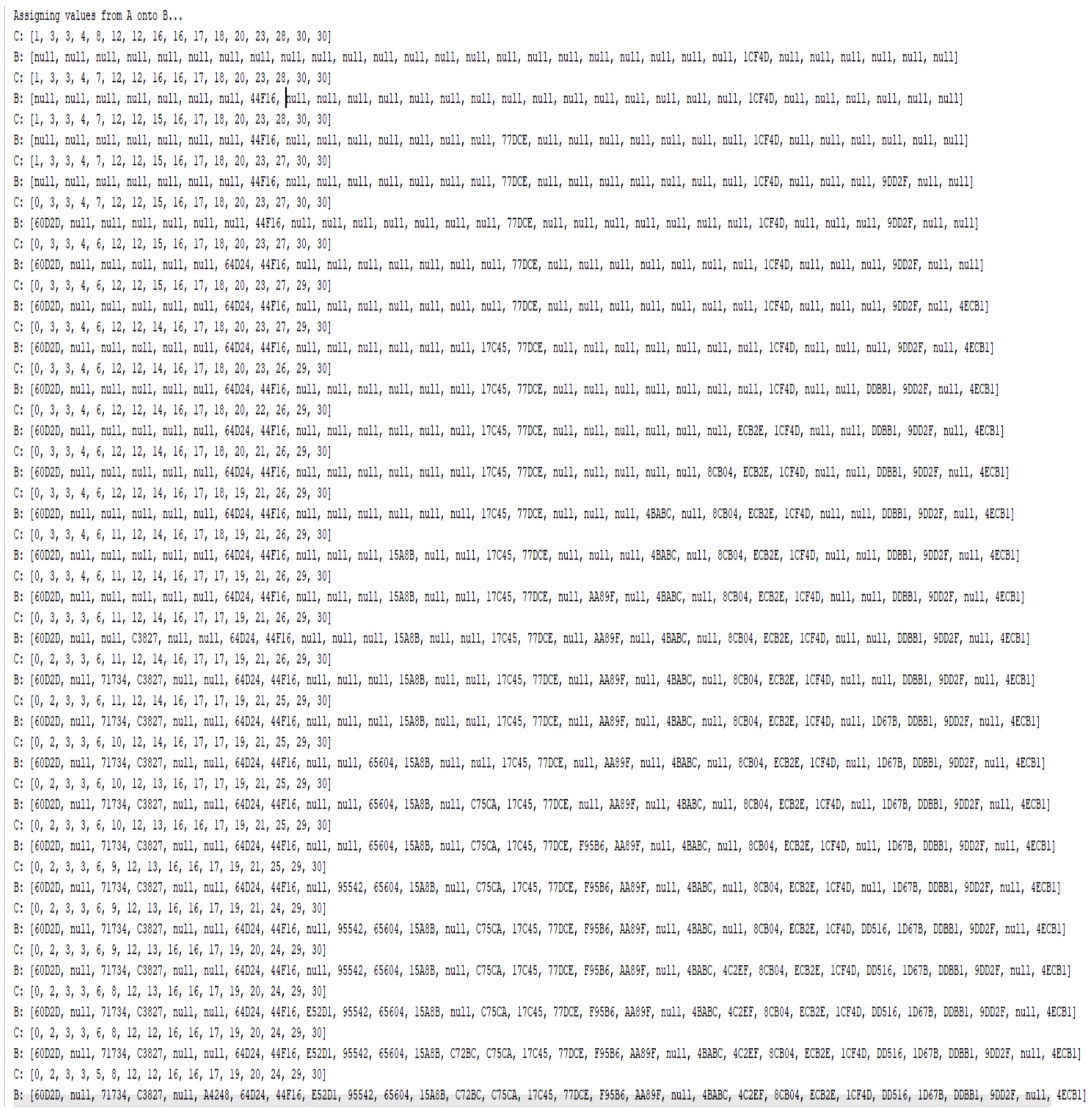


```
Assigning values from A onto B...
C: [1, 2, 8, 8, 8, 12, 14, 15, 17, 17, 19, 22, 24, 28, 28, 30]
B: [null, null, 8B1F0, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null]
C: [1, 2, 7, 8, 8, 12, 14, 15, 17, 17, 19, 22, 24, 28, 28, 30]
B: [null, null, 8B1F0, null, null, null, null, 4C2EF, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null]
C: [1, 2, 6, 8, 8, 12, 14, 15, 17, 17, 19, 22, 24, 28, 28, 30]
B: [null, null, 8B1F0, null, null, null, E52D1, 4C2EF, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null]
C: [1, 2, 6, 8, 8, 12, 14, 15, 17, 17, 19, 22, 24, 27, 28, 30]
B: [null, null, 8B1F0, null, null, null, E52D1, 4C2EF, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, 77DCE, null, null]
C: [1, 2, 6, 8, 8, 11, 14, 15, 17, 17, 19, 22, 24, 27, 28, 30]
B: [null, null, 8B1F0, null, null, null, E52D1, 4C2EF, null, null, null, C75CA, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, 77DCE, null, null]
C: [1, 2, 5, 8, 8, 11, 14, 15, 17, 17, 19, 22, 24, 27, 28, 30]
B: [null, null, 8B1F0, null, null, C72BC, E52D1, 4C2EF, null, null, null, C75CA, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, 77DCE, null, null]
C: [1, 2, 5, 8, 8, 11, 14, 15, 17, 17, 18, 22, 24, 27, 28, 30]
B: [null, null, 8B1F0, null, null, C72BC, E52D1, 4C2EF, null, null, null, C75CA, null, null, null, null, null, null, 4BABC, null, null, null, null, null, null, null, null, 77DCE, null, null]
C: [1, 2, 5, 8, 8, 10, 14, 15, 17, 17, 18, 22, 24, 27, 28, 30]
B: [null, null, 8B1F0, null, null, C72BC, E52D1, 4C2EF, null, null, F95B6, C75CA, null, null, null, null, null, null, 4BABC, null, null, null, null, null, null, null, null, 77DCE, null, null]
C: [1, 2, 5, 8, 8, 10, 14, 15, 17, 17, 18, 21, 24, 27, 28, 30]
B: [null, null, 8B1F0, null, null, C72BC, E52D1, 4C2EF, null, null, F95B6, C75CA, null, null, null, null, null, null, 4BABC, null, null, DDBB1, null, null, null, null, null, 77DCE, null, null]
C: [1, 2, 5, 8, 8, 10, 14, 15, 17, 17, 18, 21, 23, 27, 28, 30]
B: [null, null, 8B1F0, null, null, C72BC, E52D1, 4C2EF, null, null, F95B6, C75CA, null, null, null, null, null, null, 4BABC, null, null, DDBB1, null, 4ECB1, null, null, null, 77DCE, null, null]
C: [1, 2, 5, 8, 8, 10, 14, 15, 16, 17, 18, 21, 23, 27, 28, 30]
B: [null, null, 8B1F0, null, null, C72BC, E52D1, 4C2EF, null, null, F95B6, C75CA, null, null, null, null, AA89F, null, 4BABC, null, null, DDBB1, null, 4ECB1, null, null, null, 77DCE, null, null]
C: [1, 2, 5, 8, 8, 10, 14, 15, 16, 17, 17, 21, 23, 27, 28, 30]
B: [null, null, 8B1F0, null, null, C72BC, E52D1, 4C2EF, null, null, F95B6, C75CA, null, null, null, null, AA89F, 15A8B, 4BABC, null, null, DDBB1, null, 4ECB1, null, null, null, 77DCE, null, null]
C: [1, 2, 5, 8, 8, 10, 13, 15, 16, 17, 17, 21, 23, 27, 28, 30]
B: [null, null, 8B1F0, null, null, C72BC, E52D1, 4C2EF, null, null, F95B6, C75CA, null, 1D67B, null, null, AA89F, 15A8B, 4BABC, null, null, DDBB1, null, 4ECB1, null, null, null, 77DCE, null, null]
C: [0, 2, 5, 8, 8, 10, 13, 15, 16, 17, 17, 21, 23, 27, 28, 30]
B: [4E05C, null, 8B1F0, null, null, C72BC, E52D1, 4C2EF, null, null, F95B6, C75CA, null, 1D67B, null, null, AA89F, 15A8B, 4BABC, null, null, DDBB1, null, 4ECB1, null, null, null, 77DCE, null, null]
C: [0, 1, 5, 8, 8, 10, 13, 15, 16, 17, 17, 21, 23, 27, 28, 30]
B: [4E05C, D1155, 8B1F0, null, null, C72BC, E52D1, 4C2EF, null, null, F95B6, C75CA, null, 1D67B, null, null, AA89F, 15A8B, 4BABC, null, null, DDBB1, null, 4ECB1, null, null, null, 77DCE, null, null]
C: [0, 1, 5, 8, 8, 10, 13, 15, 16, 17, 17, 21, 23, 27, 28, 29]
B: [4E05C, D1155, 8B1F0, null, null, C72BC, E52D1, 4C2EF, null, null, F95B6, C75CA, null, 1D67B, null, null, AA89F, 15A8B, 4BABC, null, null, DDBB1, null, 4ECB1, null, null, null, 77DCE, null, 1CF4D]
C: [0, 1, 4, 8, 8, 10, 13, 15, 16, 17, 17, 21, 23, 27, 28, 29]
B: [4E05C, D1155, 8B1F0, null, A4248, C72BC, E52D1, 4C2EF, null, null, F95B6, C75CA, null, 1D67B, null, null, AA89F, 15A8B, 4BABC, null, null, DDBB1, null, 4ECB1, null, null, null, 77DCE, null, 1CF4D]
C: [0, 1, 4, 8, 8, 10, 13, 15, 16, 17, 17, 21, 22, 27, 28, 29]
B: [4E05C, D1155, 8B1F0, null, A4248, C72BC, E52D1, 4C2EF, null, null, F95B6, C75CA, null, 1D67B, null, null, AA89F, 15A8B, 4BABC, null, null, DDBB1, 17C45, 4ECB1, null, null, null, 77DCE, null, 1CF4D]
C: [0, 1, 4, 8, 8, 9, 13, 15, 16, 17, 17, 21, 22, 27, 28, 29]
B: [4E05C, D1155, 8B1F0, null, A4248, C72BC, E52D1, 4C2EF, null, 95542, F95B6, C75CA, null, 1D67B, null, null, AA89F, 15A8B, 4BABC, null, null, DDBB1, 17C45, 4ECB1, null, null, null, 77DCE, null, 1CF4D]
C: [0, 1, 3, 8, 8, 9, 13, 15, 16, 17, 17, 21, 22, 27, 28, 29]
B: [4E05C, D1155, 8B1F0, F423E, A4248, C72BC, E52D1, 4C2EF, null, 95542, F95B6, C75CA, null, 1D67B, null, null, AA89F, 15A8B, 4BABC, null, null, DDBB1, 17C45, 4ECB1, null, null, null, 77DCE, null, 1CF4D]
C: [0, 1, 3, 8, 8, 9, 13, 14, 16, 17, 17, 21, 22, 27, 28, 29]
B: [4E05C, D1155, 8B1F0, F423E, A4248, C72BC, E52D1, 4C2EF, null, 95542, F95B6, C75CA, null, 1D67B, 71734, null, AA89F, 15A8B, 4BABC, null, null, DDBB1, 17C45, 4ECB1, null, null, null, 77DCE, null, 1CF4D]
C: [0, 1, 3, 8, 8, 9, 13, 14, 16, 17, 17, 21, 22, 26, 28, 29]
B: [4E05C, D1155, 8B1F0, F423E, A4248, C72BC, E52D1, 4C2EF, null, 95542, F95B6, C75CA, null, 1D67B, 71734, null, AA89F, 15A8B, 4BABC, null, null, DDBB1, 17C45, 4ECB1, null, null, 9DD2F, 77DCE, null, 1CF4D]
C: [0, 1, 3, 8, 8, 9, 13, 14, 16, 17, 17, 20, 22, 26, 28, 29]
B: [4E05C, D1155, 8B1F0, F423E, A4248, C72BC, E52D1, 4C2EF, null, 95542, F95B6, C75CA, null, 1D67B, 71734, null, AA89F, 15A8B, 4BABC, null, ECB2E, DDBB1, 17C45, 4ECB1, null, null, 9DD2F, 77DCE, null, 1CF4D]
C: [0, 1, 3, 8, 8, 9, 13, 14, 16, 17, 17, 20, 22, 25, 28, 29]
B: [4E05C, D1155, 8B1F0, F423E, A4248, C72BC, E52D1, 4C2EF, null, 95542, F95B6, C75CA, null, 1D67B, 71734, null, AA89F, 15A8B, 4BABC, null, ECB2E, DDBB1, 17C45, 4ECB1, null, 60D2D, 9DD2F, 77DCE, null, 1CF4D]
C: [0, 1, 3, 8, 8, 9, 13, 14, 15, 17, 17, 20, 22, 25, 28, 29]
B: [4E05C, D1155, 8B1F0, F423E, A4248, C72BC, E52D1, 4C2EF, null, 95542, F95B6, C75CA, null, 1D67B, 71734, C3827, AA89F, 15A8B, 4BABC, null, ECB2E, DDBB1, 17C45, 4ECB1, null, 60D2D, 9DD2F, 77DCE, null, 1CF4D]
```

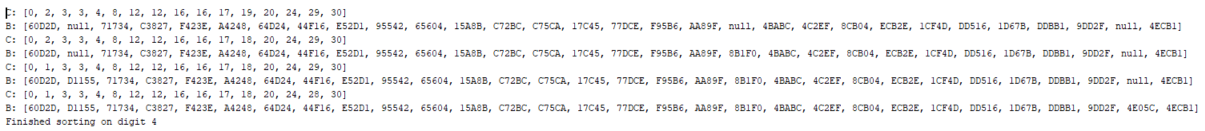


```
C: [0, 1, 3, 8, 8, 9, 13, 14, 15, 17, 17, 20, 22, 24, 28, 29]
B: [4E05C, D1155, 8B1F0, F423E, A4248, C72BC, E52D1, 4C2EF, null, 95542, F95B6, C75CA, null, 1D67B, 71734, C3827, AA89F, 15A8B, 4BABC, null, ECB2E, DDBB1, 17C45, 4ECB1, 64D24, 60D2D, 9DD2F, 77DCE, null, 1CF4D]
C: [0, 1, 3, 8, 8, 8, 13, 14, 15, 17, 17, 20, 22, 24, 28, 29]
B: [4E05C, D1155, 8B1F0, F423E, A4248, C72BC, E52D1, 4C2EF, DD516, 95542, F95B6, C75CA, null, 1D67B, 71734, C3827, AA89F, 15A8B, 4BABC, null, ECB2E, DDBB1, 17C45, 4ECB1, 64D24, 60D2D, 9DD2F, 77DCE, null, 1CF4D]
C: [0, 1, 3, 8, 8, 8, 13, 14, 15, 17, 17, 20, 22, 24, 28, 28]
B: [4E05C, D1155, 8B1F0, F423E, A4248, C72BC, E52D1, 4C2EF, DD516, 95542, F95B6, C75CA, null, 1D67B, 71734, C3827, AA89F, 15A8B, 4BABC, null, ECB2E, DDBB1, 17C45, 4ECB1, 64D24, 60D2D, 9DD2F, 77DCE, 44F16, 1CF4D]
C: [0, 1, 3, 8, 8, 8, 12, 14, 15, 17, 17, 20, 22, 24, 28, 28]
B: [4E05C, D1155, 8B1F0, F423E, A4248, C72BC, E52D1, 4C2EF, DD516, 95542, F95B6, C75CA, 65604, 1D67B, 71734, C3827, AA89F, 15A8B, 4BABC, null, ECB2E, DDBB1, 17C45, 4ECB1, 64D24, 60D2D, 9DD2F, 77DCE, 44F16, 1CF4D]
C: [0, 1, 3, 8, 8, 8, 12, 14, 15, 17, 17, 19, 22, 24, 28, 28]
B: [4E05C, D1155, 8B1F0, F423E, A4248, C72BC, E52D1, 4C2EF, DD516, 95542, F95B6, C75CA, 65604, 1D67B, 71734, C3827, AA89F, 15A8B, 4BABC, 8CB04, ECB2E, DDBB1, 17C45, 4ECB1, 64D24, 60D2D, 9DD2F, 77DCE, 44F16, 1CF4D]
Finished sorting on digit 3
```

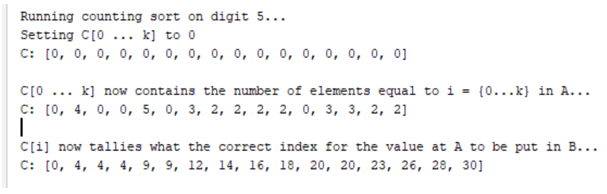
```
Running counting sort on digit 4...
Setting C[0 ... k] to 0
C: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

C[0 ... k] now contains the number of elements equal to i = {0...k} in A...
C: [1, 2, 0, 1, 4, 4, 0, 4, 0, 1, 1, 2, 4, 4, 2, 0]

C[i] now tallies what the correct index for the value at A to be put in B...
C: [1, 3, 3, 4, 8, 12, 12, 16, 16, 17, 18, 20, 24, 28, 30, 30]
```



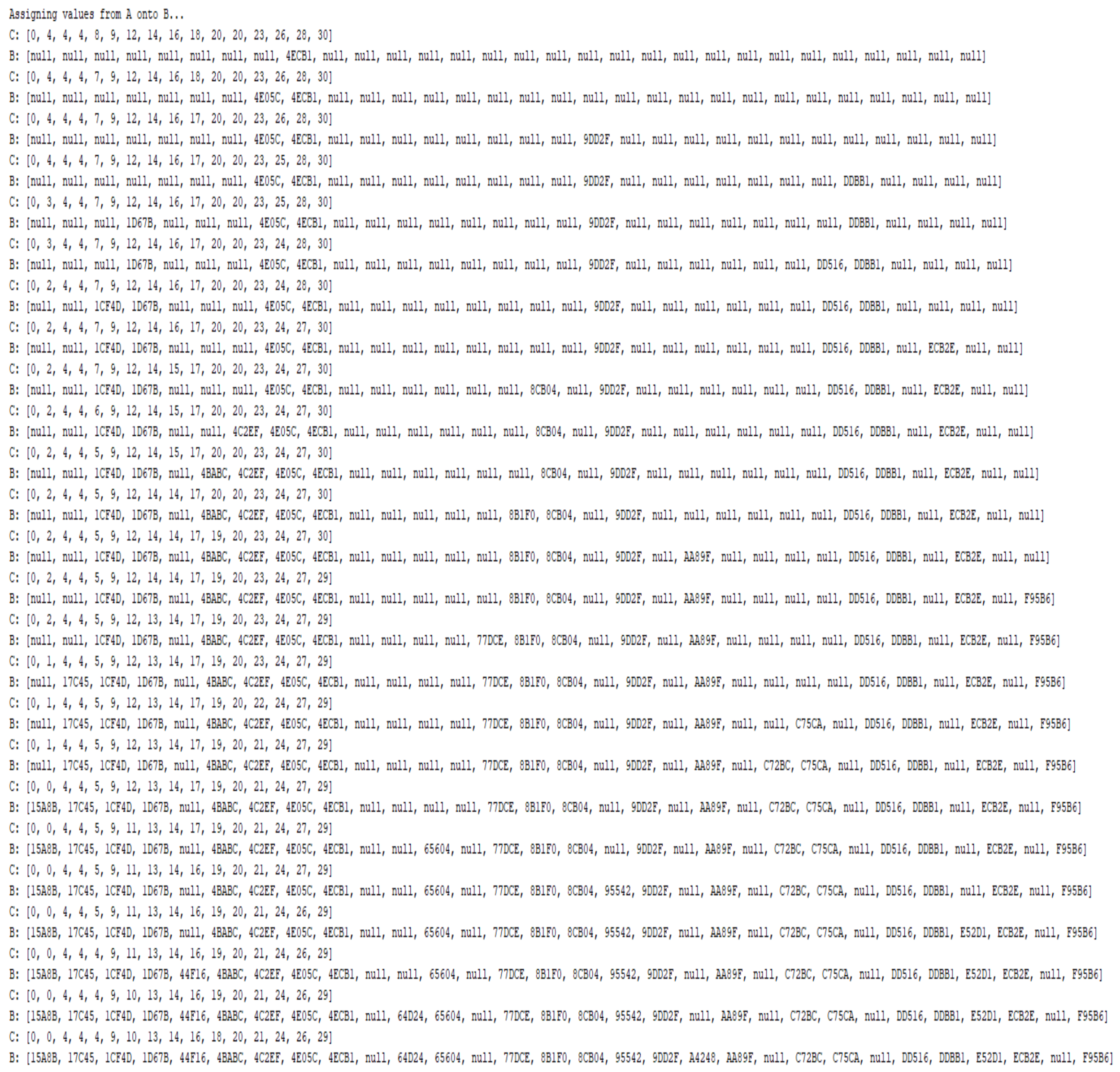
```
Assigning values from A onto B...
C: [1, 3, 3, 4, 8, 12, 12, 16, 16, 17, 18, 20, 23, 28, 30, 30]
B: [null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, 1CF4D, null, null, null, null, null, null]
C: [1, 3, 3, 4, 7, 12, 12, 16, 16, 17, 18, 20, 23, 28, 30, 30]
B: [null, null, null, null, null, null, null, 44F16, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, 1CF4D, null, null, null, null, null, null]
C: [1, 3, 3, 4, 7, 12, 12, 15, 16, 17, 18, 20, 23, 28, 30, 30]
B: [null, null, null, null, null, null, null, 44F16, null, null, null, null, null, null, null, 77DCE, null, null, null, null, null, null, null, 1CF4D, null, null, null, null, null, null]
C: [1, 3, 3, 4, 7, 12, 12, 15, 16, 17, 18, 20, 23, 27, 30, 30]
B: [null, null, null, null, null, null, null, 44F16, null, null, null, null, null, null, null, 77DCE, null, null, null, null, null, null, null, 1CF4D, null, null, null, 9DD2F, null, null]
C: [0, 3, 3, 4, 7, 12, 12, 15, 16, 17, 18, 20, 23, 27, 30, 30]
B: [60D2D, null, null, null, null, null, null, 44F16, null, null, null, null, null, null, null, 77DCE, null, null, null, null, null, null, null, 1CF4D, null, null, null, 9DD2F, null, null]
C: [0, 3, 3, 4, 6, 12, 12, 15, 16, 17, 18, 20, 23, 27, 30, 30]
B: [60D2D, null, null, null, null, null, 64D24, 44F16, null, null, null, null, null, null, null, 77DCE, null, null, null, null, null, null, null, 1CF4D, null, null, null, 9DD2F, null, null]
C: [0, 3, 3, 4, 6, 12, 12, 15, 16, 17, 18, 20, 23, 27, 29, 30]
B: [60D2D, null, null, null, null, null, 64D24, 44F16, null, null, null, null, null, null, null, 77DCE, null, null, null, null, null, null, null, 1CF4D, null, null, null, 9DD2F, null, 4ECB1]
C: [0, 3, 3, 4, 6, 12, 12, 14, 16, 17, 18, 20, 23, 27, 29, 30]
B: [60D2D, null, null, null, null, null, 64D24, 44F16, null, null, null, null, null, null, 17C45, 77DCE, null, null, null, null, null, null, null, 1CF4D, null, null, null, 9DD2F, null, 4ECB1]
C: [0, 3, 3, 4, 6, 12, 12, 14, 16, 17, 18, 20, 23, 26, 29, 30]
B: [60D2D, null, null, null, null, null, 64D24, 44F16, null, null, null, null, null, null, 17C45, 77DCE, null, null, null, null, null, null, null, 1CF4D, null, null, DDBB1, 9DD2F, null, 4ECB1]
C: [0, 3, 3, 4, 6, 12, 12, 14, 16, 17, 18, 20, 22, 26, 29, 30]
B: [60D2D, null, null, null, null, null, 64D24, 44F16, null, null, null, null, null, null, 17C45, 77DCE, null, null, null, null, null, null, ECB2E, 1CF4D, null, null, DDBB1, 9DD2F, null, 4ECB1]
C: [0, 3, 3, 4, 6, 12, 12, 14, 16, 17, 18, 20, 21, 26, 29, 30]
B: [60D2D, null, null, null, null, null, 64D24, 44F16, null, null, null, null, null, null, 17C45, 77DCE, null, null, null, null, null, 8CB04, ECB2E, 1CF4D, null, null, DDBB1, 9DD2F, null, 4ECB1]
C: [0, 3, 3, 4, 6, 12, 12, 14, 16, 17, 18, 19, 21, 26, 29, 30]
B: [60D2D, null, null, null, null, null, 64D24, 44F16, null, null, null, null, null, null, 17C45, 77DCE, null, null, null, 4BABC, null, 8CB04, ECB2E, 1CF4D, null, null, DDBB1, 9DD2F, null, 4ECB1]
C: [0, 3, 3, 4, 6, 11, 12, 14, 16, 17, 18, 19, 21, 26, 29, 30]
B: [60D2D, null, null, null, null, null, 64D24, 44F16, null, null, null, 15A8B, null, null, 17C45, 77DCE, null, null, null, 4BABC, null, 8CB04, ECB2E, 1CF4D, null, null, DDBB1, 9DD2F, null, 4ECB1]
C: [0, 3, 3, 4, 6, 11, 12, 14, 16, 17, 17, 19, 21, 26, 29, 30]
B: [60D2D, null, null, null, null, null, 64D24, 44F16, null, null, null, 15A8B, null, null, 17C45, 77DCE, null, AA89F, null, 4BABC, null, 8CB04, ECB2E, 1CF4D, null, null, DDBB1, 9DD2F, null, 4ECB1]
C: [0, 3, 3, 3, 6, 11, 12, 14, 16, 17, 17, 19, 21, 26, 29, 30]
B: [60D2D, null, null, C3827, null, null, 64D24, 44F16, null, null, null, 15A8B, null, null, 17C45, 77DCE, null, AA89F, null, 4BABC, null, 8CB04, ECB2E, 1CF4D, null, null, DDBB1, 9DD2F, null, 4ECB1]
C: [0, 2, 3, 3, 6, 11, 12, 14, 16, 17, 17, 19, 21, 26, 29, 30]
B: [60D2D, null, 71734, C3827, null, null, 64D24, 44F16, null, null, null, 15A8B, null, null, 17C45, 77DCE, null, AA89F, null, 4BABC, null, 8CB04, ECB2E, 1CF4D, null, null, DDBB1, 9DD2F, null, 4ECB1]
C: [0, 2, 3, 3, 6, 11, 12, 14, 16, 17, 17, 19, 21, 25, 29, 30]
B: [60D2D, null, 71734, C3827, null, null, 64D24, 44F16, null, null, null, 15A8B, null, null, 17C45, 77DCE, null, AA89F, null, 4BABC, null, 8CB04, ECB2E, 1CF4D, null, 1D67B, DDBB1, 9DD2F, null, 4ECB1]
C: [0, 2, 3, 3, 6, 10, 12, 14, 16, 17, 17, 19, 21, 25, 29, 30]
B: [60D2D, null, 71734, C3827, null, null, 64D24, 44F16, null, null, 65604, 15A8B, null, null, 17C45, 77DCE, null, AA89F, null, 4BABC, null, 8CB04, ECB2E, 1CF4D, null, 1D67B, DDBB1, 9DD2F, null, 4ECB1]
C: [0, 2, 3, 3, 6, 10, 12, 13, 16, 17, 17, 19, 21, 25, 29, 30]
B: [60D2D, null, 71734, C3827, null, null, 64D24, 44F16, null, null, 65604, 15A8B, null, C75CA, 17C45, 77DCE, null, AA89F, null, 4BABC, null, 8CB04, ECB2E, 1CF4D, null, 1D67B, DDBB1, 9DD2F, null, 4ECB1]
C: [0, 2, 3, 3, 6, 10, 12, 13, 16, 16, 17, 19, 21, 25, 29, 30]
B: [60D2D, null, 71734, C3827, null, null, 64D24, 44F16, null, null, 65604, 15A8B, null, C75CA, 17C45, 77DCE, F95B6, AA89F, null, 4BABC, null, 8CB04, ECB2E, 1CF4D, null, 1D67B, DDBB1, 9DD2F, null, 4ECB1]
C: [0, 2, 3, 3, 6, 9, 12, 13, 16, 16, 17, 19, 21, 25, 29, 30]
B: [60D2D, null, 71734, C3827, null, null, 64D24, 44F16, null, 95542, 65604, 15A8B, null, C75CA, 17C45, 77DCE, F95B6, AA89F, null, 4BABC, null, 8CB04, ECB2E, 1CF4D, null, 1D67B, DDBB1, 9DD2F, null, 4ECB1]
C: [0, 2, 3, 3, 6, 9, 12, 13, 16, 16, 17, 19, 21, 24, 29, 30]
B: [60D2D, null, 71734, C3827, null, null, 64D24, 44F16, null, 95542, 65604, 15A8B, null, C75CA, 17C45, 77DCE, F95B6, AA89F, null, 4BABC, null, 8CB04, ECB2E, 1CF4D, DD516, 1D67B, DDBB1, 9DD2F, null, 4ECB1]
C: [0, 2, 3, 3, 6, 9, 12, 13, 16, 16, 17, 19, 20, 24, 29, 30]
B: [60D2D, null, 71734, C3827, null, null, 64D24, 44F16, null, 95542, 65604, 15A8B, null, C75CA, 17C45, 77DCE, F95B6, AA89F, null, 4BABC, 4C2EF, 8CB04, ECB2E, 1CF4D, DD516, 1D67B, DDBB1, 9DD2F, null, 4ECB1]
C: [0, 2, 3, 3, 6, 8, 12, 13, 16, 16, 17, 19, 20, 24, 29, 30]
B: [60D2D, null, 71734, C3827, null, null, 64D24, 44F16, E52D1, 95542, 65604, 15A8B, null, C75CA, 17C45, 77DCE, F95B6, AA89F, null, 4BABC, 4C2EF, 8CB04, ECB2E, 1CF4D, DD516, 1D67B, DDBB1, 9DD2F, null, 4ECB1]
C: [0, 2, 3, 3, 6, 8, 12, 12, 16, 16, 17, 19, 20, 24, 29, 30]
B: [60D2D, null, 71734, C3827, null, null, 64D24, 44F16, E52D1, 95542, 65604, 15A8B, C72BC, C75CA, 17C45, 77DCE, F95B6, AA89F, null, 4BABC, 4C2EF, 8CB04, ECB2E, 1CF4D, DD516, 1D67B, DDBB1, 9DD2F, null, 4ECB1]
C: [0, 2, 3, 3, 5, 8, 12, 12, 16, 16, 17, 19, 20, 24, 29, 30]
B: [60D2D, null, 71734, C3827, null, A4248, 64D24, 44F16, E52D1, 95542, 65604, 15A8B, C72BC, C75CA, 17C45, 77DCE, F95B6, AA89F, null, 4BABC, 4C2EF, 8CB04, ECB2E, 1CF4D, DD516, 1D67B, DDBB1, 9DD2F, null, 4ECB1]
```



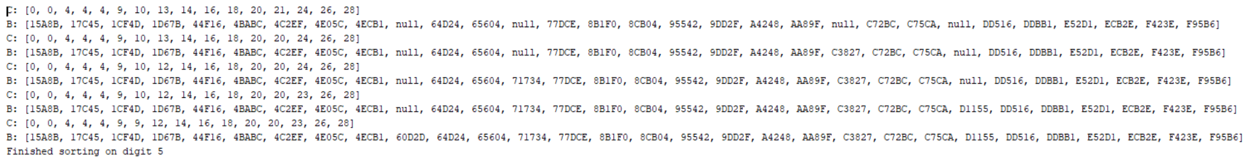
```
C: [0, 2, 3, 3, 4, 8, 12, 12, 16, 16, 17, 19, 20, 24, 29, 30]
B: [60D2D, null, 71734, C3827, F423E, A4248, 64D24, 44F16, E52D1, 95542, 65604, 15A8B, C72BC, C75CA, 17C45, 77DCE, F95B6, AA89F, null, 4BABC, 4C2EF, 8CB04, ECB2E, 1CF4D, DD516, 1D67B, DDBB1, 9DD2F, null, 4ECB1]
C: [0, 2, 3, 3, 4, 8, 12, 12, 16, 16, 17, 18, 20, 24, 29, 30]
B: [60D2D, null, 71734, C3827, F423E, A4248, 64D24, 44F16, E52D1, 95542, 65604, 15A8B, C72BC, C75CA, 17C45, 77DCE, F95B6, AA89F, 8B1F0, 4BABC, 4C2EF, 8CB04, ECB2E, 1CF4D, DD516, 1D67B, DDBB1, 9DD2F, null, 4ECB1]
C: [0, 1, 3, 3, 4, 8, 12, 12, 16, 16, 17, 18, 20, 24, 29, 30]
B: [60D2D, D1155, 71734, C3827, F423E, A4248, 64D24, 44F16, E52D1, 95542, 65604, 15A8B, C72BC, C75CA, 17C45, 77DCE, F95B6, AA89F, 8B1F0, 4BABC, 4C2EF, 8CB04, ECB2E, 1CF4D, DD516, 1D67B, DDBB1, 9DD2F, null, 4ECB1]
C: [0, 1, 3, 3, 4, 8, 12, 12, 16, 16, 17, 18, 20, 24, 28, 30]
B: [60D2D, D1155, 71734, C3827, F423E, A4248, 64D24, 44F16, E52D1, 95542, 65604, 15A8B, C72BC, C75CA, 17C45, 77DCE, F95B6, AA89F, 8B1F0, 4BABC, 4C2EF, 8CB04, ECB2E, 1CF4D, DD516, 1D67B, DDBB1, 9DD2F, 4E05C, 4ECB1]
Finished sorting on digit 4
```



```
Running counting sort on digit 5...
Setting C[0 ... k] to 0
C: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

C[0 ... k] now contains the number of elements equal to i = {0...k} in A...
C: [0, 4, 0, 0, 5, 0, 3, 2, 2, 2, 2, 0, 3, 3, 2, 2]

C[i] now tallies what the correct index for the value at A to be put in B...
C: [0, 4, 4, 4, 9, 9, 12, 14, 16, 18, 20, 20, 23, 26, 28, 30]
```

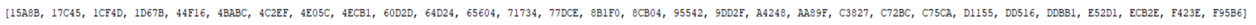


```
Assigning values from A onto B...
C: [0, 4, 4, 4, 8, 9, 12, 14, 16, 18, 20, 20, 23, 26, 28, 30]
B: [null, null, null, null, null, null, null, null, 4ECB1, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null]
C: [0, 4, 4, 4, 7, 9, 12, 14, 16, 18, 20, 20, 23, 26, 28, 30]
B: [null, null, null, null, null, null, null, 4E05C, 4ECB1, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null]
C: [0, 4, 4, 4, 7, 9, 12, 14, 16, 17, 20, 20, 23, 26, 28, 30]
B: [null, null, null, null, null, null, null, 4E05C, 4ECB1, null, null, null, null, null, null, null, null, 9DD2F, null, null, null, null, null, null, null, null, null, null, null, null]
C: [0, 4, 4, 4, 7, 9, 12, 14, 16, 17, 20, 20, 23, 25, 28, 30]
B: [null, null, null, null, null, null, null, 4E05C, 4ECB1, null, null, null, null, null, null, null, null, 9DD2F, null, null, null, null, null, null, null, DDBB1, null, null, null, null]
C: [0, 3, 4, 4, 7, 9, 12, 14, 16, 17, 20, 20, 23, 25, 28, 30]
B: [null, null, null, 1D67B, null, null, null, 4E05C, 4ECB1, null, null, null, null, null, null, null, null, 9DD2F, null, null, null, null, null, null, null, DDBB1, null, null, null, null]
C: [0, 3, 4, 4, 7, 9, 12, 14, 16, 17, 20, 20, 23, 24, 28, 30]
B: [null, null, null, 1D67B, null, null, null, 4E05C, 4ECB1, null, null, null, null, null, null, null, null, 9DD2F, null, null, null, null, null, null, DD516, DDBB1, null, null, null, null]
C: [0, 2, 4, 4, 7, 9, 12, 14, 16, 17, 20, 20, 23, 24, 28, 30]
B: [null, null, 1CF4D, 1D67B, null, null, null, 4E05C, 4ECB1, null, null, null, null, null, null, null, null, 9DD2F, null, null, null, null, null, null, DD516, DDBB1, null, null, null, null]
C: [0, 2, 4, 4, 7, 9, 12, 14, 16, 17, 20, 20, 23, 24, 27, 30]
B: [null, null, 1CF4D, 1D67B, null, null, null, 4E05C, 4ECB1, null, null, null, null, null, null, null, null, 9DD2F, null, null, null, null, null, null, DD516, DDBB1, null, ECB2E, null, null]
C: [0, 2, 4, 4, 7, 9, 12, 14, 15, 17, 20, 20, 23, 24, 27, 30]
B: [null, null, 1CF4D, 1D67B, null, null, null, 4E05C, 4ECB1, null, null, null, null, null, null, 8CB04, null, 9DD2F, null, null, null, null, null, null, DD516, DDBB1, null, ECB2E, null, null]
C: [0, 2, 4, 4, 6, 9, 12, 14, 15, 17, 20, 20, 23, 24, 27, 30]
B: [null, null, 1CF4D, 1D67B, null, null, 4C2EF, 4E05C, 4ECB1, null, null, null, null, null, null, 8CB04, null, 9DD2F, null, null, null, null, null, null, DD516, DDBB1, null, ECB2E, null, null]
C: [0, 2, 4, 4, 5, 9, 12, 14, 15, 17, 20, 20, 23, 24, 27, 30]
B: [null, null, 1CF4D, 1D67B, null, 4BABC, 4C2EF, 4E05C, 4ECB1, null, null, null, null, null, null, 8CB04, null, 9DD2F, null, null, null, null, null, null, DD516, DDBB1, null, ECB2E, null, null]
C: [0, 2, 4, 4, 5, 9, 12, 14, 14, 17, 20, 20, 23, 24, 27, 30]
B: [null, null, 1CF4D, 1D67B, null, 4BABC, 4C2EF, 4E05C, 4ECB1, null, null, null, null, null, 8B1F0, 8CB04, null, 9DD2F, null, null, null, null, null, null, DD516, DDBB1, null, ECB2E, null, null]
C: [0, 2, 4, 4, 5, 9, 12, 14, 14, 17, 19, 20, 23, 24, 27, 30]
B: [null, null, 1CF4D, 1D67B, null, 4BABC, 4C2EF, 4E05C, 4ECB1, null, null, null, null, null, 8B1F0, 8CB04, null, 9DD2F, null, AA89F, null, null, null, null, DD516, DDBB1, null, ECB2E, null, null]
C: [0, 2, 4, 4, 5, 9, 12, 14, 14, 17, 19, 20, 23, 24, 27, 29]
B: [null, null, 1CF4D, 1D67B, null, 4BABC, 4C2EF, 4E05C, 4ECB1, null, null, null, null, null, 8B1F0, 8CB04, null, 9DD2F, null, AA89F, null, null, null, null, DD516, DDBB1, null, ECB2E, null, F95B6]
C: [0, 2, 4, 4, 5, 9, 12, 13, 14, 17, 19, 20, 23, 24, 27, 29]
B: [null, null, 1CF4D, 1D67B, null, 4BABC, 4C2EF, 4E05C, 4ECB1, null, null, null, null, 77DCE, 8B1F0, 8CB04, null, 9DD2F, null, AA89F, null, null, null, null, DD516, DDBB1, null, ECB2E, null, F95B6]
C: [0, 1, 4, 4, 5, 9, 12, 13, 14, 17, 19, 20, 23, 24, 27, 29]
B: [null, 17C45, 1CF4D, 1D67B, null, 4BABC, 4C2EF, 4E05C, 4ECB1, null, null, null, null, 77DCE, 8B1F0, 8CB04, null, 9DD2F, null, AA89F, null, null, null, null, DD516, DDBB1, null, ECB2E, null, F95B6]
C: [0, 1, 4, 4, 5, 9, 12, 13, 14, 17, 19, 20, 22, 24, 27, 29]
B: [null, 17C45, 1CF4D, 1D67B, null, 4BABC, 4C2EF, 4E05C, 4ECB1, null, null, null, null, 77DCE, 8B1F0, 8CB04, null, 9DD2F, null, AA89F, null, null, C75CA, null, DD516, DDBB1, null, ECB2E, null, F95B6]
C: [0, 1, 4, 4, 5, 9, 12, 13, 14, 17, 19, 20, 21, 24, 27, 29]
B: [null, 17C45, 1CF4D, 1D67B, null, 4BABC, 4C2EF, 4E05C, 4ECB1, null, null, null, null, 77DCE, 8B1F0, 8CB04, null, 9DD2F, null, AA89F, null, C72BC, C75CA, null, DD516, DDBB1, null, ECB2E, null, F95B6]
C: [0, 0, 4, 4, 5, 9, 12, 13, 14, 17, 19, 20, 21, 24, 27, 29]
B: [15A8B, 17C45, 1CF4D, 1D67B, null, 4BABC, 4C2EF, 4E05C, 4ECB1, null, null, null, null, 77DCE, 8B1F0, 8CB04, null, 9DD2F, null, AA89F, null, C72BC, C75CA, null, DD516, DDBB1, null, ECB2E, null, F95B6]
C: [0, 0, 4, 4, 5, 9, 11, 13, 14, 17, 19, 20, 21, 24, 27, 29]
B: [15A8B, 17C45, 1CF4D, 1D67B, null, 4BABC, 4C2EF, 4E05C, 4ECB1, null, null, 65604, null, 77DCE, 8B1F0, 8CB04, null, 9DD2F, null, AA89F, null, C72BC, C75CA, null, DD516, DDBB1, null, ECB2E, null, F95B6]
C: [0, 0, 4, 4, 5, 9, 11, 13, 14, 16, 19, 20, 21, 24, 27, 29]
B: [15A8B, 17C45, 1CF4D, 1D67B, null, 4BABC, 4C2EF, 4E05C, 4ECB1, null, null, 65604, null, 77DCE, 8B1F0, 8CB04, 95542, 9DD2F, null, AA89F, null, C72BC, C75CA, null, DD516, DDBB1, null, ECB2E, null, F95B6]
C: [0, 0, 4, 4, 5, 9, 11, 13, 14, 16, 19, 20, 21, 24, 26, 29]
B: [15A8B, 17C45, 1CF4D, 1D67B, null, 4BABC, 4C2EF, 4E05C, 4ECB1, null, null, 65604, null, 77DCE, 8B1F0, 8CB04, 95542, 9DD2F, null, AA89F, null, C72BC, C75CA, null, DD516, DDBB1, E52D1, ECB2E, null, F95B6]
C: [0, 0, 4, 4, 4, 9, 11, 13, 14, 16, 19, 20, 21, 24, 26, 29]
B: [15A8B, 17C45, 1CF4D, 1D67B, 44F16, 4BABC, 4C2EF, 4E05C, 4ECB1, null, null, 65604, null, 77DCE, 8B1F0, 8CB04, 95542, 9DD2F, null, AA89F, null, C72BC, C75CA, null, DD516, DDBB1, E52D1, ECB2E, null, F95B6]
C: [0, 0, 4, 4, 4, 9, 10, 13, 14, 16, 19, 20, 21, 24, 26, 29]
B: [15A8B, 17C45, 1CF4D, 1D67B, 44F16, 4BABC, 4C2EF, 4E05C, 4ECB1, null, 64D24, 65604, null, 77DCE, 8B1F0, 8CB04, 95542, 9DD2F, null, AA89F, null, C72BC, C75CA, null, DD516, DDBB1, E52D1, ECB2E, null, F95B6]
C: [0, 0, 4, 4, 4, 9, 10, 13, 14, 16, 18, 20, 21, 24, 26, 29]
B: [15A8B, 17C45, 1CF4D, 1D67B, 44F16, 4BABC, 4C2EF, 4E05C, 4ECB1, null, 64D24, 65604, null, 77DCE, 8B1F0, 8CB04, 95542, 9DD2F, A4248, AA89F, null, C72BC, C75CA, null, DD516, DDBB1, E52D1, ECB2E, null, F95B6]
```



```
C: [0, 0, 4, 4, 4, 9, 10, 13, 14, 16, 18, 20, 21, 24, 26, 28]
B: [15A8B, 17C45, 1CF4D, 1D67B, 44F16, 4BABC, 4C2EF, 4E05C, 4ECB1, null, 64D24, 65604, null, 77DCE, 8B1F0, 8CB04, 95542, 9DD2F, A4248, AA89F, null, C72BC, C75CA, null, DD516, DDBB1, E52D1, ECB2E, F423E, F95B6]
C: [0, 0, 4, 4, 4, 9, 10, 13, 14, 16, 18, 20, 20, 24, 26, 28]
B: [15A8B, 17C45, 1CF4D, 1D67B, 44F16, 4BABC, 4C2EF, 4E05C, 4ECB1, null, 64D24, 65604, null, 77DCE, 8B1F0, 8CB04, 95542, 9DD2F, A4248, AA89F, C3827, C72BC, C75CA, null, DD516, DDBB1, E52D1, ECB2E, F423E, F95B6]
C: [0, 0, 4, 4, 4, 9, 10, 12, 14, 16, 18, 20, 20, 24, 26, 28]
B: [15A8B, 17C45, 1CF4D, 1D67B, 44F16, 4BABC, 4C2EF, 4E05C, 4ECB1, null, 64D24, 65604, 71734, 77DCE, 8B1F0, 8CB04, 95542, 9DD2F, A4248, AA89F, C3827, C72BC, C75CA, null, DD516, DDBB1, E52D1, ECB2E, F423E, F95B6]
C: [0, 0, 4, 4, 4, 9, 10, 12, 14, 16, 18, 20, 20, 23, 26, 28]
B: [15A8B, 17C45, 1CF4D, 1D67B, 44F16, 4BABC, 4C2EF, 4E05C, 4ECB1, null, 64D24, 65604, 71734, 77DCE, 8B1F0, 8CB04, 95542, 9DD2F, A4248, AA89F, C3827, C72BC, C75CA, D1155, DD516, DDBB1, E52D1, ECB2E, F423E, F95B6]
C: [0, 0, 4, 4, 4, 9, 9, 12, 14, 16, 18, 20, 20, 23, 26, 28]
B: [15A8B, 17C45, 1CF4D, 1D67B, 44F16, 4BABC, 4C2EF, 4E05C, 4ECB1, 60D2D, 64D24, 65604, 71734, 77DCE, 8B1F0, 8CB04, 95542, 9DD2F, A4248, AA89F, C3827, C72BC, C75CA, D1155, DD516, DDBB1, E52D1, ECB2E, F423E, F95B6]
Finished sorting on digit 5
```

The radix sort operation finishes and the resulting array is shown below.

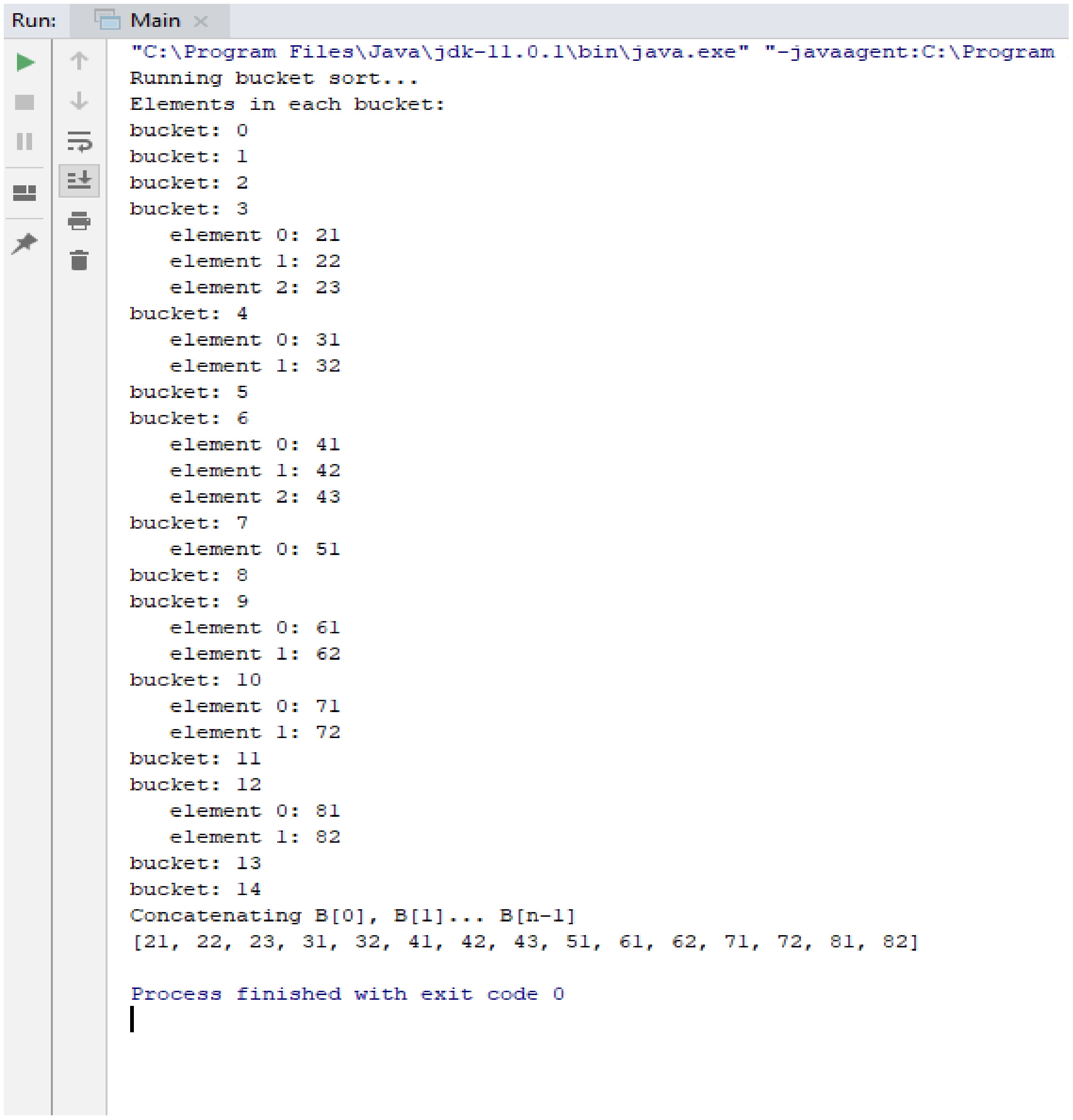


```
[15A8B, 17C45, 1CF4D, 1D67B, 44F16, 4BABC, 4C2EF, 4E05C, 4ECB1, 60D2D, 64D24, 65604, 71734, 77DCE, 8B1F0, 8CB04, 95542, 9DD2F, A4248, AA89F, C3827, C72BC, C75CA, D1155, DD516, DDBB1, E52D1, ECB2E, F423E, F95B6]
```

3.3 Bucket Sort

Calling bucket sort on the given array below

```
Input array: [71, 72, 21, 81, 41, 22, 31, 51, 42, 82, 32, 61, 43, 62, 23]
```



```
Run:    Main
"C:\Program Files\Java\jdk-11.0.1\bin\java.exe" "-javaagent:C:\Program
Running bucket sort...
Elements in each bucket:
bucket: 0
bucket: 1
bucket: 2
bucket: 3
   element 0: 21
   element 1: 22
   element 2: 23
bucket: 4
   element 0: 31
   element 1: 32
bucket: 5
bucket: 6
   element 0: 41
   element 1: 42
   element 2: 43
bucket: 7
   element 0: 51
bucket: 8
bucket: 9
   element 0: 61
   element 1: 62
bucket: 10
   element 0: 71
   element 1: 72
bucket: 11
bucket: 12
   element 0: 81
   element 1: 82
bucket: 13
bucket: 14
Concatenating B[0], B[1]... B[n-1]
[21, 22, 23, 31, 32, 41, 42, 43, 51, 61, 62, 71, 72, 81, 82]

Process finished with exit code 0
```

Note that the first digit showing relative indexes of equal numbers are in numerical order, showing that this algorithm is stable.

4. Setup Procedure (Windows):

1) Download the .zip file named ‘PA2-Fung’ containing PA2.jar, PA2_Report.pdf, and the java class files for the code implementation off the site I uploaded it to.

2) Extract the .zip file using a program such as 7Zip.

3) To run the program, open the command line – if you don't know where it is, type 'cmd' in the Windows search bar.

4) Type in "java -jar " and enter the file directory of PA2.jar, for example "C:\Users\File_Path\PA2.jar"

5) (optional) Since there are lots of line being output, the command line may omit some of the earlier lines. If this happens just type out the same thing as you did earlier, but add " > output.txt" to create a text file in the same directory as your jar file with all of the output.

5. Problems Encountered:

A problem I encountered in this assignment was my random number generator for hexadecimal. My method of generating a random hexadecimal number revolved around generating a random hexadecimal for each of the five digits for each number. The goal was to concatenate every digit to form the number. However, I came across an issue - concatenating letters with numbers results in just numbers. After testing concatenation with different primitive types, I came to the conclusion of setting each digit to be of type character and using its builtin toString(char c) method to turn it into a string for concatenation of the 5 digits.

Initially, I wasn't sure how to have subscripts to show the relative indexes of equal numbers, so I just had the relative number be the first digit. For example, 7_1 would be 71 and 7_2 would be 72. In the actual comparisons, I would divide the values in the array by 10 so that the numbers in these two cases would still be 7 since we take the floor of the result. This worked in showing that the algorithms were stable, as we can see when we run the program.

I had a bug in bucket sort where the constant used to decide which bucket to insert the value into was causing all the values inserted to just go into the first bucket. It took a bit of testing to figure out that setting 1/100 to a double would just return 0 since the numbers being divided are integers and the floor would be taken after the division. This gives us 0, which is still 0 after being turned into a double. I fixed this issue by simply setting the constant 1/100 to 0.01.

6. Lessons Learned:

I've gotten a lot better at detecting bugs due to a combination of the work done on this and the last programming homework assignment. More specifically, my bug testing has gotten better since I've started using the debugging tools in my IDE as well as writing more informative and legible print statements when tracking every subroutine in each sorting algorithm.

I'm not sure if this is worth listing, but I've gotten better at reading documentation for Java's built-in libraries to see if my ideas for manipulating data are even feasible for a certain object type.

A skill I’ve learned is hexadecimal generation. From my testing, there were complications creating random hexadecimal digits when trying to concatenate them to form a 5-digit hex number. Experimenting with various primitive types and converting them to another type gave me knowledge of what worked and what didn’t for concatenation between integers and numbers.

I’d say the skill I’ve improved the most working on this programming assignment was writing professional reports. Comparing what I’ve written for this and the last report, there is a huge difference in quality. Looking back, I felt as if a lot of what I wrote in the PA1 report could’ve been heavily simplified with just pictures of the code. I have no idea why I explained what the code did for my implementation portion when I could’ve literally pasted a screenshot of the code.