PA2 Linear Sorting Algorithms Report

JianBin Wu

Design and Implementation

To implement counting sort, I used an ArrayList to store IntegerToSort objects, which contains the testing integers and the index of the testing integers. Then I sort the testing array using counting sort and put the ArrayList in sorted order. To implement radix sort, I used an String array to store the 30 hex numbers that were generated randomly using generateHexNumList() in the RadixSort class. After that, I converted each digit into ASCII code and sorted each digit based on the order of ASCII code. The radix sort will sort from the least significant digit to the first digit.To implement the Bucket Sort, I used an ArrayList to store the buckets and I used LinkedList for each bucket. Since the testing integers range from 0-9, I had created 10 buckets in this case. After placing the integers into specified buckets, I use insertion sort to sort each bucket and concatenate the LinkedList into an array. I also did the same thing as counting sort, used an ArrayList to store the testing integers and the index of the testing integers. Then, put the ArrayList in sorted order based on the sorting result.

Classes, Subroutines and Function calls

IntegerToSort Class - Use to prove and explain if Counting Sort and Bucket Sort is a stable algorithm.

- IntegerToSort(int integer, int subscript)- Construct an Object with a integer and subscript
- getInteger() - Get the integer
- getSubscript()- Get the subscript

CountingSort Class - Implementation of Counting Sort, following the pseudo codes from the book Introduction To Algorithms Third Edition

- countingSort(int[] A, int[] B, int k) - Use Counting Sort to sort an array

BucketSort Class - Implementation of Bucket Sort, following the pseudo codes from the book Introduction To Algorithms Third Edition

- bucketSort(int[] A) - Use BucketSort to sort an array

RadixSort Class - Implementation of Radix Sort, following the pseudo codes from the book Introduction To Algorithms Third Edition

- radixSort(String[] A, int d) - Radix Sort, sort element with ASCII code
- countingSort(int[] A, int[] B, int k) - Counting Sort for Radix Sort
- generateHexNumList() - Generate 30 random Hex number

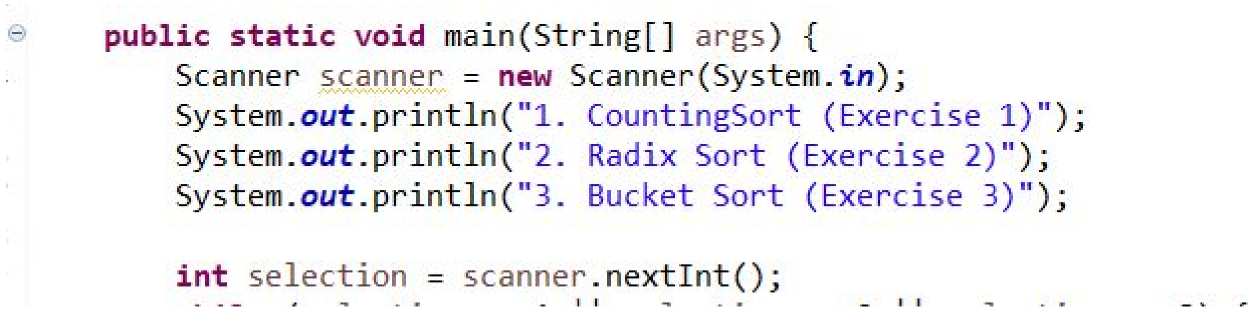
Tester Class - A tester to test out the implementation of the sorting algorithms

main () - Main method to run the program countingSort () - Use counting sort from the **CountingSort** class to sort an ArrayList

bucketSort () - Use Bucket Sort from **BucketSort** class to sort an ArrayList

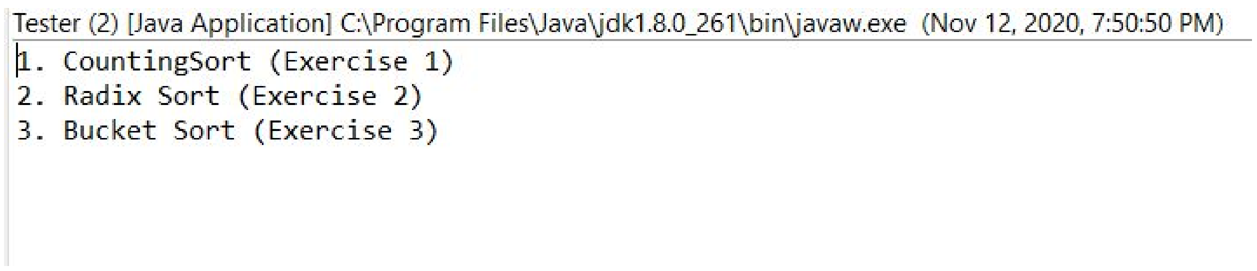
Self-testing Screenshots

The program will begin with asking users to selection Counting Sort, Radix Sort or Bucket Sort



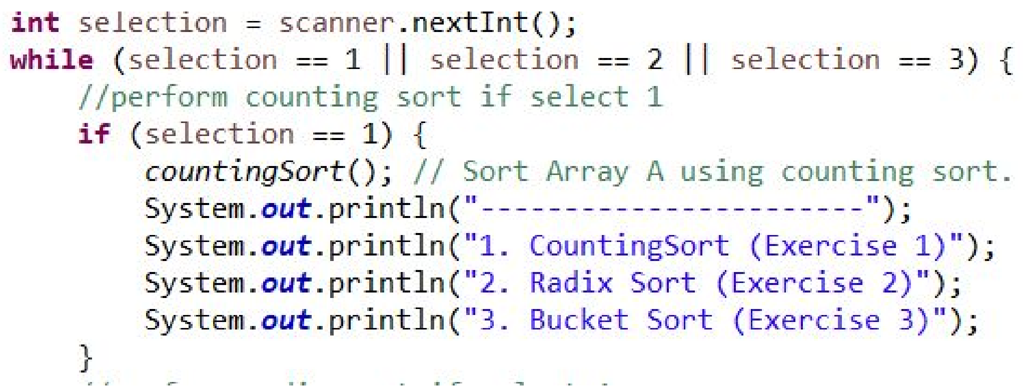
```
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.println("1. CountingSort (Exercise 1)");
        System.out.println("2. Radix Sort (Exercise 2)");
        System.out.println("3. Bucket Sort (Exercise 3)");

        int selection = scanner.nextInt();
```

Output:



```
Tester (2) [Java Application] C:\Program Files\Java\jdk1.8.0_261\bin\javaw.exe  (Nov 12, 2020, 7:50:50 PM)
1. CountingSort (Exercise 1)
2. Radix Sort (Exercise 2)
3. Bucket Sort (Exercise 3)
```

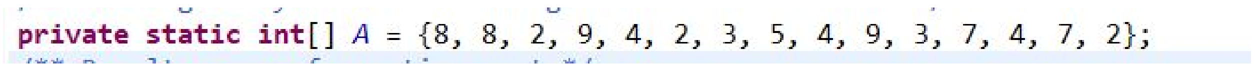
Then, the CountingSort() method will be called and it will perform the counting process if the user enters 1.



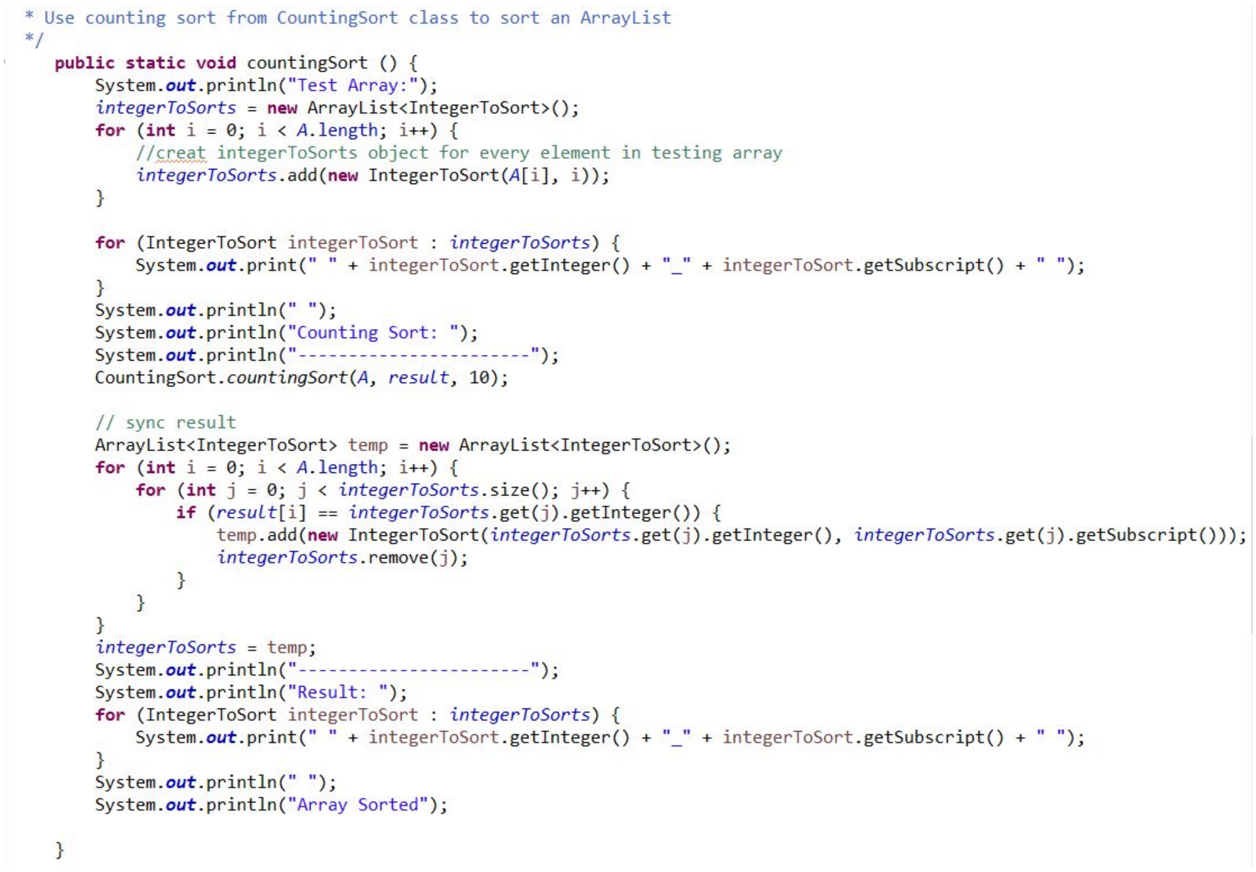
```
int selection = scanner.nextInt();
while (selection == 1 || selection == 2 || selection == 3) {
    //perform counting sort if select 1
    if (selection == 1) {
        countingSort(); // Sort Array A using counting sort.
        System.out.println("----------------------");
        System.out.println("1. CountingSort (Exercise 1)");
        System.out.println("2. Radix Sort (Exercise 2)");
        System.out.println("3. Bucket Sort (Exercise 3)");
    }
```

The CountingSort method will begin with creating a new ArrayList of IntegerToSort objects. Then, store every element in the testing array to the array and assign index to each element. Next, print out every element with the assigned index. After that, call the countingSort() in the CountingSort class to sort the testing array. After the array has been sorted, sync the elements in the array and the ArrayList, So the ArrayList is updated to sorted order. At last, print out the result and corresponding index to prove the algorithm is a stable algorithm. This algorithm is stable because it sorts in order and the indexes are in order as well.

The testing array will be



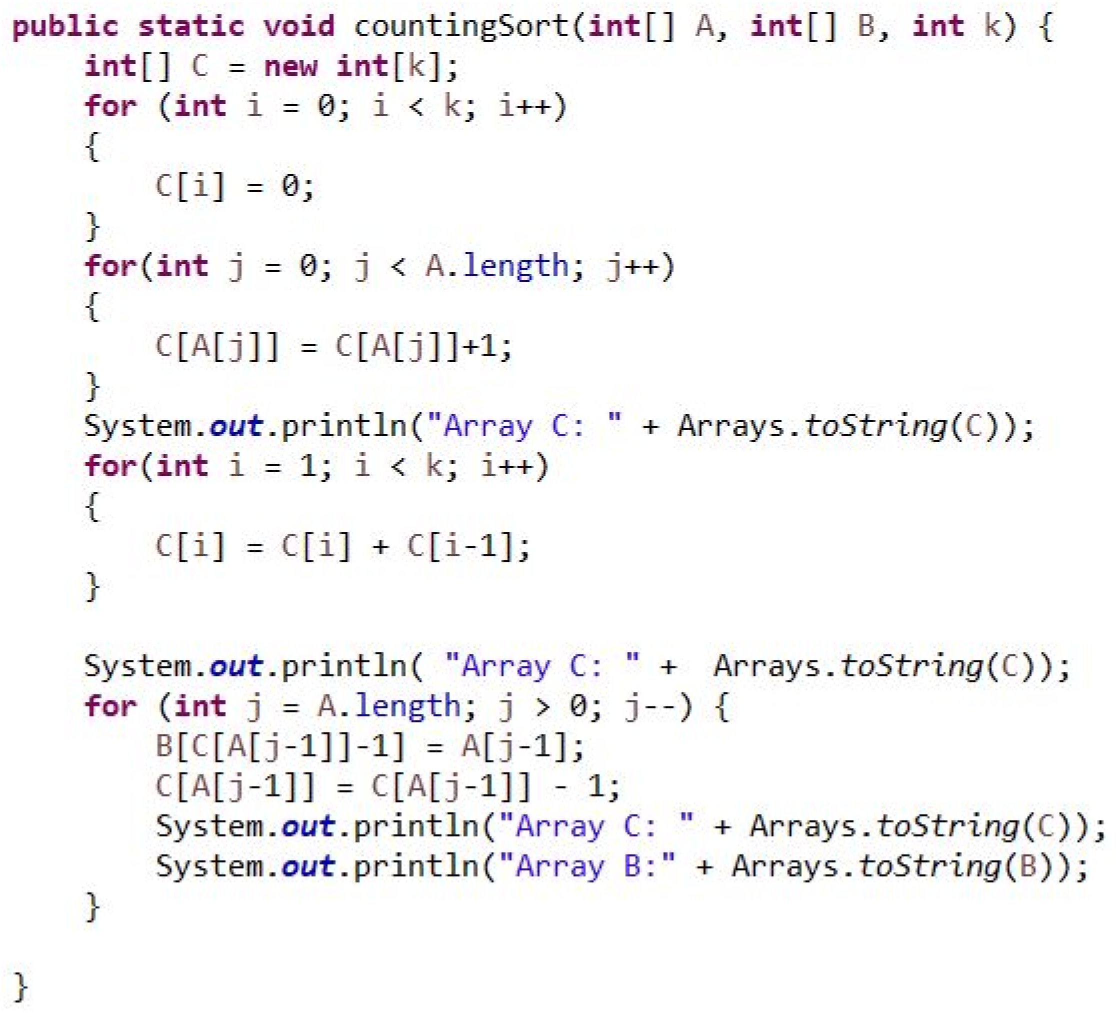
```
private static int[] A = {8, 8, 2, 9, 4, 2, 3, 5, 4, 9, 3, 7, 4, 7, 2};
```



```
 * Use counting sort from CountingSort class to sort an ArrayList
 */
    public static void countingSort () {
        System.out.println("Test Array:");
        integerToSorts = new ArrayList<IntegerToSort>();
        for (int i = 0; i < A.length; i++) {
            //creat integerToSorts object for every element in testing array
            integerToSorts.add(new IntegerToSort(A[i], i));
        }

        for (IntegerToSort integerToSort : integerToSorts) {
            System.out.print(" " + integerToSort.getInteger() + "_" + integerToSort.getSubscript() + " ");
        }
        System.out.println(" ");
        System.out.println("Counting Sort: ");
        System.out.println("-----------------------");
        CountingSort.countingSort(A, result, 10);

        // sync result
        ArrayList<IntegerToSort> temp = new ArrayList<IntegerToSort>();
        for (int i = 0; i < A.length; i++) {
            for (int j = 0; j < integerToSorts.size(); j++) {
                if (result[i] == integerToSorts.get(j).getInteger()) {
                    temp.add(new IntegerToSort(integerToSorts.get(j).getInteger(), integerToSorts.get(j).getSubscript()));
                    integerToSorts.remove(j);
                }
            }
        }
        integerToSorts = temp;
        System.out.println("-----------------------");
        System.out.println("Result: ");
        for (IntegerToSort integerToSort : integerToSorts) {
            System.out.print(" " + integerToSort.getInteger() + "_" + integerToSort.getSubscript() + " ");
        }
        System.out.println(" ");
        System.out.println("Array Sorted");

    }
```

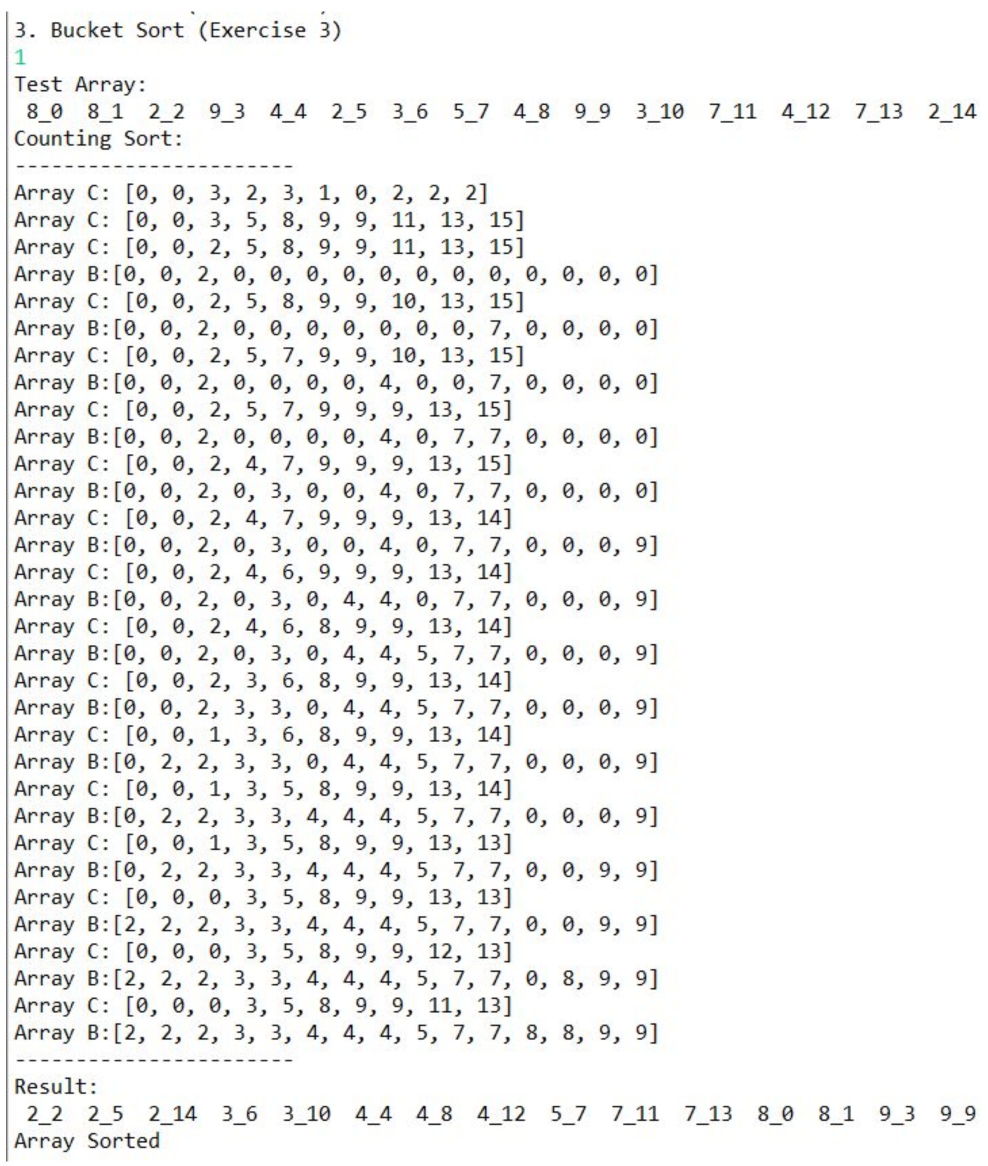
countingSort() in CountingSort class:

I add lines between the code to print out intermediate steps. The counting sort algorithm is follow by the pseudo codes from the book Introduction To Algorithms Third Edition



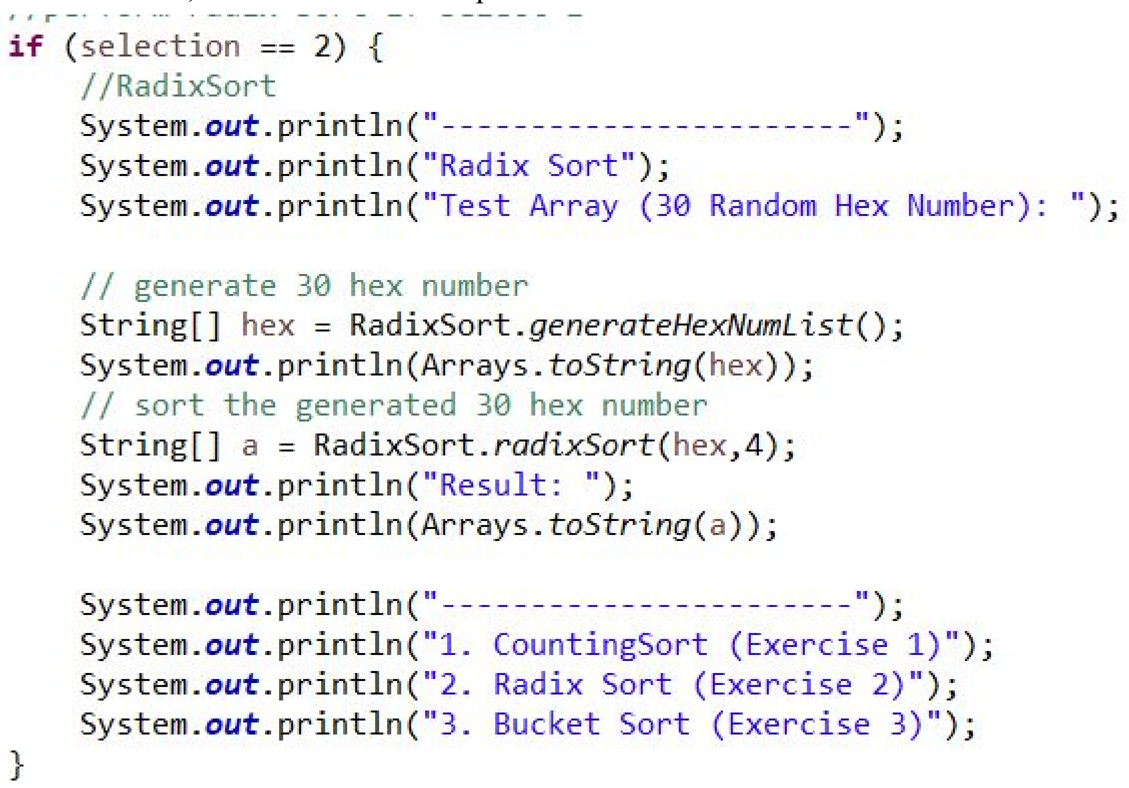
```java
public static void countingSort(int[] A, int[] B, int k) {
    int[] C = new int[k];
    for (int i = 0; i < k; i++)
    {
        C[i] = 0;
    }
    for(int j = 0; j < A.length; j++)
    {
        C[A[j]] = C[A[j]]+1;
    }
    System.out.println("Array C: " + Arrays.toString(C));
    for(int i = 1; i < k; i++)
    {
        C[i] = C[i] + C[i-1];
    }

    System.out.println( "Array C: " +  Arrays.toString(C));
    for (int j = A.length; j > 0; j--) {
        B[C[A[j-1]]-1] = A[j-1];
        C[A[j-1]] = C[A[j-1]] - 1;
        System.out.println("Array C: " + Arrays.toString(C));
        System.out.println("Array B:" + Arrays.toString(B));
    }

}
```

Output:



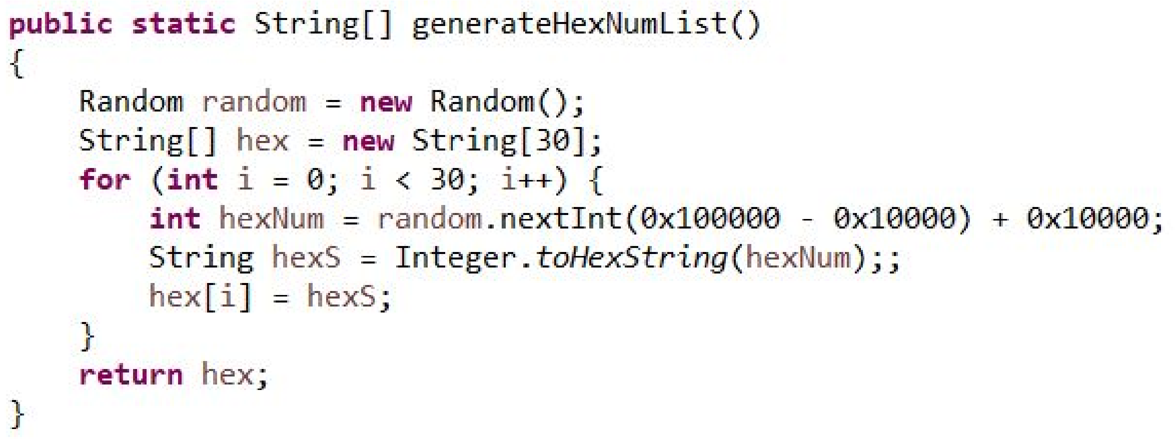
```
3. Bucket Sort (Exercise 3)
1
Test Array:
 8_0  8_1  2_2  9_3  4_4  2_5  3_6  5_7  4_8  9_9  3_10  7_11  4_12  7_13  2_14
Counting Sort:
-----------------------
Array C: [0, 0, 3, 2, 3, 1, 0, 2, 2, 2]
Array C: [0, 0, 3, 5, 8, 9, 9, 11, 13, 15]
Array C: [0, 0, 2, 5, 8, 9, 9, 11, 13, 15]
Array B:[0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
Array C: [0, 0, 2, 5, 8, 9, 9, 10, 13, 15]
Array B:[0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 7, 0, 0, 0, 0]
Array C: [0, 0, 2, 5, 7, 9, 9, 10, 13, 15]
Array B:[0, 0, 2, 0, 0, 0, 0, 4, 0, 0, 7, 0, 0, 0, 0]
Array C: [0, 0, 2, 5, 7, 9, 9, 9, 13, 15]
Array B:[0, 0, 2, 0, 0, 0, 0, 4, 0, 7, 7, 0, 0, 0, 0]
Array C: [0, 0, 2, 4, 7, 9, 9, 9, 13, 15]
Array B:[0, 0, 2, 0, 3, 0, 0, 4, 0, 7, 7, 0, 0, 0, 0]
Array C: [0, 0, 2, 4, 7, 9, 9, 9, 13, 14]
Array B:[0, 0, 2, 0, 3, 0, 0, 4, 0, 7, 7, 0, 0, 0, 9]
Array C: [0, 0, 2, 4, 6, 9, 9, 9, 13, 14]
Array B:[0, 0, 2, 0, 3, 0, 4, 4, 0, 7, 7, 0, 0, 0, 9]
Array C: [0, 0, 2, 4, 6, 8, 9, 9, 13, 14]
Array B:[0, 0, 2, 0, 3, 0, 4, 4, 5, 7, 7, 0, 0, 0, 9]
Array C: [0, 0, 2, 3, 6, 8, 9, 9, 13, 14]
Array B:[0, 0, 2, 3, 3, 0, 4, 4, 5, 7, 7, 0, 0, 0, 9]
Array C: [0, 0, 1, 3, 6, 8, 9, 9, 13, 14]
Array B:[0, 2, 2, 3, 3, 0, 4, 4, 5, 7, 7, 0, 0, 0, 9]
Array C: [0, 0, 1, 3, 5, 8, 9, 9, 13, 14]
Array B:[0, 2, 2, 3, 3, 4, 4, 4, 5, 7, 7, 0, 0, 0, 9]
Array C: [0, 0, 1, 3, 5, 8, 9, 9, 13, 13]
Array B:[0, 2, 2, 3, 3, 4, 4, 4, 5, 7, 7, 0, 0, 9, 9]
Array C: [0, 0, 0, 3, 5, 8, 9, 9, 13, 13]
Array B:[2, 2, 2, 3, 3, 4, 4, 4, 5, 7, 7, 0, 0, 9, 9]
Array C: [0, 0, 0, 3, 5, 8, 9, 9, 12, 13]
Array B:[2, 2, 2, 3, 3, 4, 4, 4, 5, 7, 7, 0, 8, 9, 9]
Array C: [0, 0, 0, 3, 5, 8, 9, 9, 11, 13]
Array B:[2, 2, 2, 3, 3, 4, 4, 4, 5, 7, 7, 8, 8, 9, 9]
-----------------------
Result:
 2_2  2_5  2_14  3_6  3_10  4_4  4_8  4_12  5_7  7_11  7_13  8_0  8_1  9_3  9_9
Array Sorted
```

If user selects 2, the radix sort will be performed.



```
if (selection == 2) {
    //RadixSort
    System.out.println("----------------------");
    System.out.println("Radix Sort");
    System.out.println("Test Array (30 Random Hex Number): ");

    // generate 30 hex number
    String[] hex = RadixSort.generateHexNumList();
    System.out.println(Arrays.toString(hex));
    // sort the generated 30 hex number
    String[] a = RadixSort.radixSort(hex,4);
    System.out.println("Result: ");
    System.out.println(Arrays.toString(a));

    System.out.println("----------------------");
    System.out.println("1. CountingSort (Exercise 1)");
    System.out.println("2. Radix Sort (Exercise 2)");
    System.out.println("3. Bucket Sort (Exercise 3)");
}
```

First it will call generteHexNumList() method in RadixSort class to generate 30 random 5 digit hex numbers and store them in a String array.



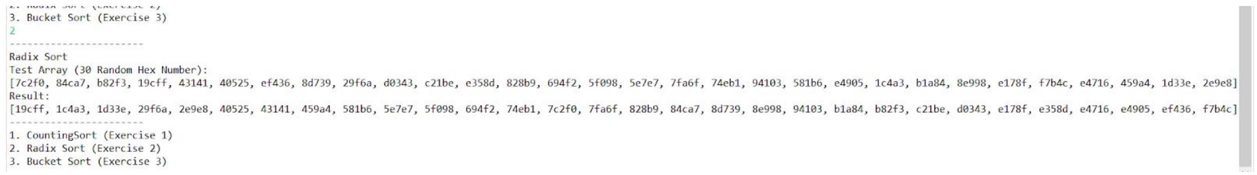
```
public static String[] generateHexNumList()
{
    Random random = new Random();
    String[] hex = new String[30];
    for (int i = 0; i < 30; i++) {
        int hexNum = random.nextInt(0x100000 - 0x10000) + 0x10000;
        String hexS = Integer.toHexString(hexNum);;
        hex[i] = hexS;
    }
    return hex;
}
```

Next, radixSort() method from RadixSort class will be called to sort the array using radix sort

```java
 */
public static String[] radixSort(String[] A, int d) {

    for(int i = d ; i >= 0 ; i--){
        // convert digits in to ASCII code
        int[] asciiToSort = new int[30];
        for(int j = 0 ; j <  A.length; j++) {
            int k = A[j].charAt(i);
            asciiToSort[j] = k;
        }
        int[] hexResult = new int[30];
        // sort digits using counting sort and store result in array hexResult
        countingSort(asciiToSort,hexResult,103);

        // sync array A with the sorting result (base on array hexResult)
        String[] temp = new String[30];
        int counter = 0;
        for(int k = 0; k < hexResult.length; k++) {
            for (int o = 0; o < A.length; o++) {
                if(A[o] != null){
                   if(hexResult[k] == (int)A[o].charAt(i)){
                        temp[counter] = A[o];
                        counter++;
                        A[o] = null;
                    }
                }
            }

        }

        //set A to temp array in order to sort the next digit
        A = temp;
    }

    return A;
```

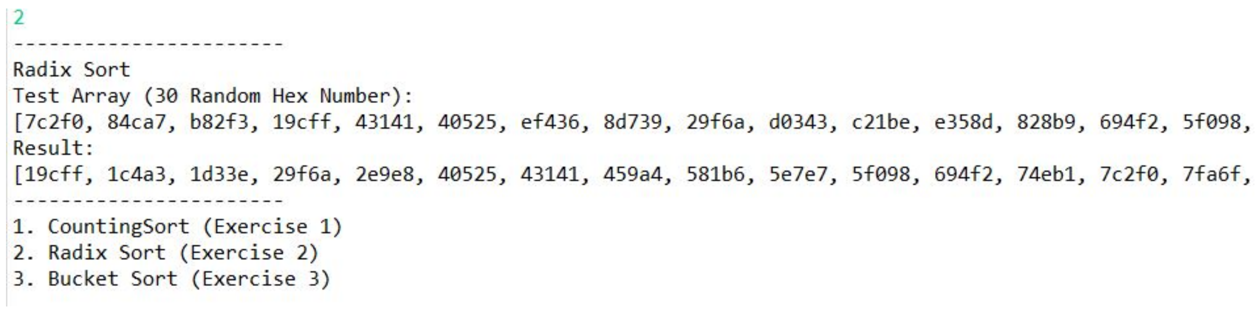
First, it begins with the last digit of every 5 digit hex number. It converts every digit into ASCII code and stores them into an int Array called asciiToSort. Then, use counting sort to sort the int Array called hexResult. Then, update the original input array with the hexResult array. Using a loop to sort every digit, first will be the least significant digit, which is the last digit. Then, will be the fourth digit… so on and so forth. The result printed after the sorting process.

Output:

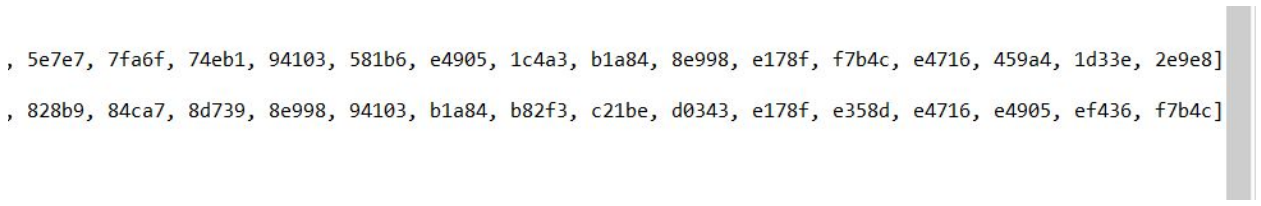


```
3. Bucket Sort (Exercise 3)
2
-----------------------
Radix Sort
Test Array (30 Random Hex Number):
[7c2f0, 84ca7, b82f3, 19cff, 43141, 40525, ef436, 8d739, 29f6a, d0343, c21be, e358d, 828b9, 694f2, 5f098, 5e7e7, 7fa6f, 74eb1, 94103, 581b6, e4905, 1c4a3, b1a84, 8e998, e178f, f7b4c, e4716, 459a4, 1d33e, 2e9e8]
Result:
[19cff, 1c4a3, 1d33e, 29f6a, 2e9e8, 40525, 43141, 459a4, 581b6, 5e7e7, 5f098, 694f2, 74eb1, 7c2f0, 7fa6f, 828b9, 84ca7, 8d739, 8e998, 94103, b1a84, b82f3, c21be, d0343, e178f, e358d, e4716, e4905, ef436, f7b4c]
-----------------------
1. CountingSort (Exercise 1)
2. Radix Sort (Exercise 2)
3. Bucket Sort (Exercise 3)
```

To have a better view, I will cut it in half. The top array will be the test array and the bottom array will be the result array.

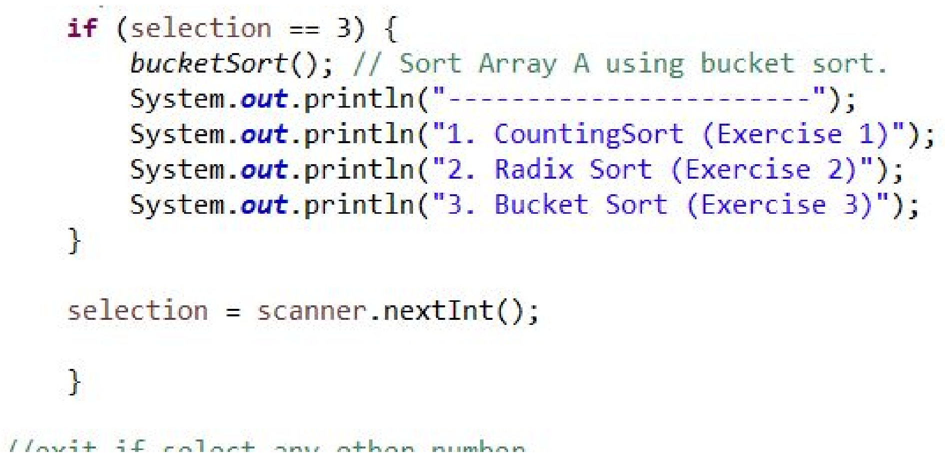


```
2
-----------------------
Radix Sort
Test Array (30 Random Hex Number):
[7c2f0, 84ca7, b82f3, 19cff, 43141, 40525, ef436, 8d739, 29f6a, d0343, c21be, e358d, 828b9, 694f2, 5f098,
Result:
[19cff, 1c4a3, 1d33e, 29f6a, 2e9e8, 40525, 43141, 459a4, 581b6, 5e7e7, 5f098, 694f2, 74eb1, 7c2f0, 7fa6f,
-----------------------
1. CountingSort (Exercise 1)
2. Radix Sort (Exercise 2)
3. Bucket Sort (Exercise 3)
```



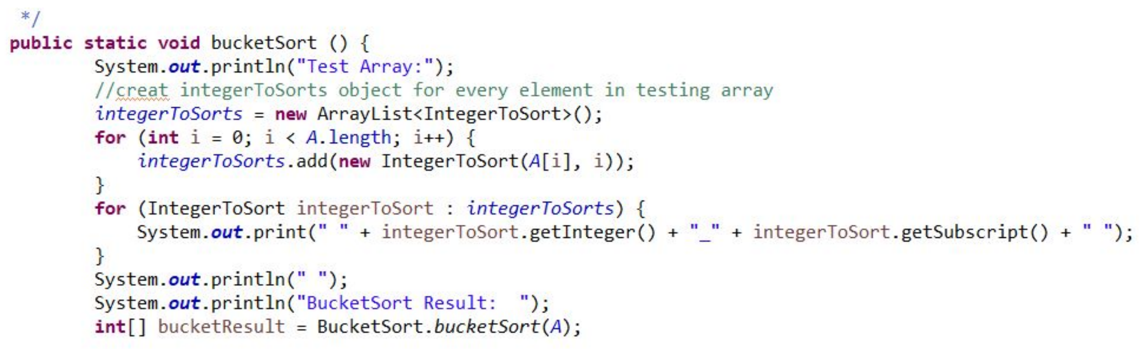
```
, 5e7e7, 7fa6f, 74eb1, 94103, 581b6, e4905, 1c4a3, b1a84, 8e998, e178f, f7b4c, e4716, 459a4, 1d33e, 2e9e8]

, 828b9, 84ca7, 8d739, 8e998, 94103, b1a84, b82f3, c21be, d0343, e178f, e358d, e4716, e4905, ef436, f7b4c]
```

If the user selects 3, Bucket sort will be performed.



```
if (selection == 3) {
    bucketSort(); // Sort Array A using bucket sort.
    System.out.println("-----------------------");
    System.out.println("1. CountingSort (Exercise 1)");
    System.out.println("2. Radix Sort (Exercise 2)");
    System.out.println("3. Bucket Sort (Exercise 3)");
}

selection = scanner.nextInt();

}

//exit if select any other number
```

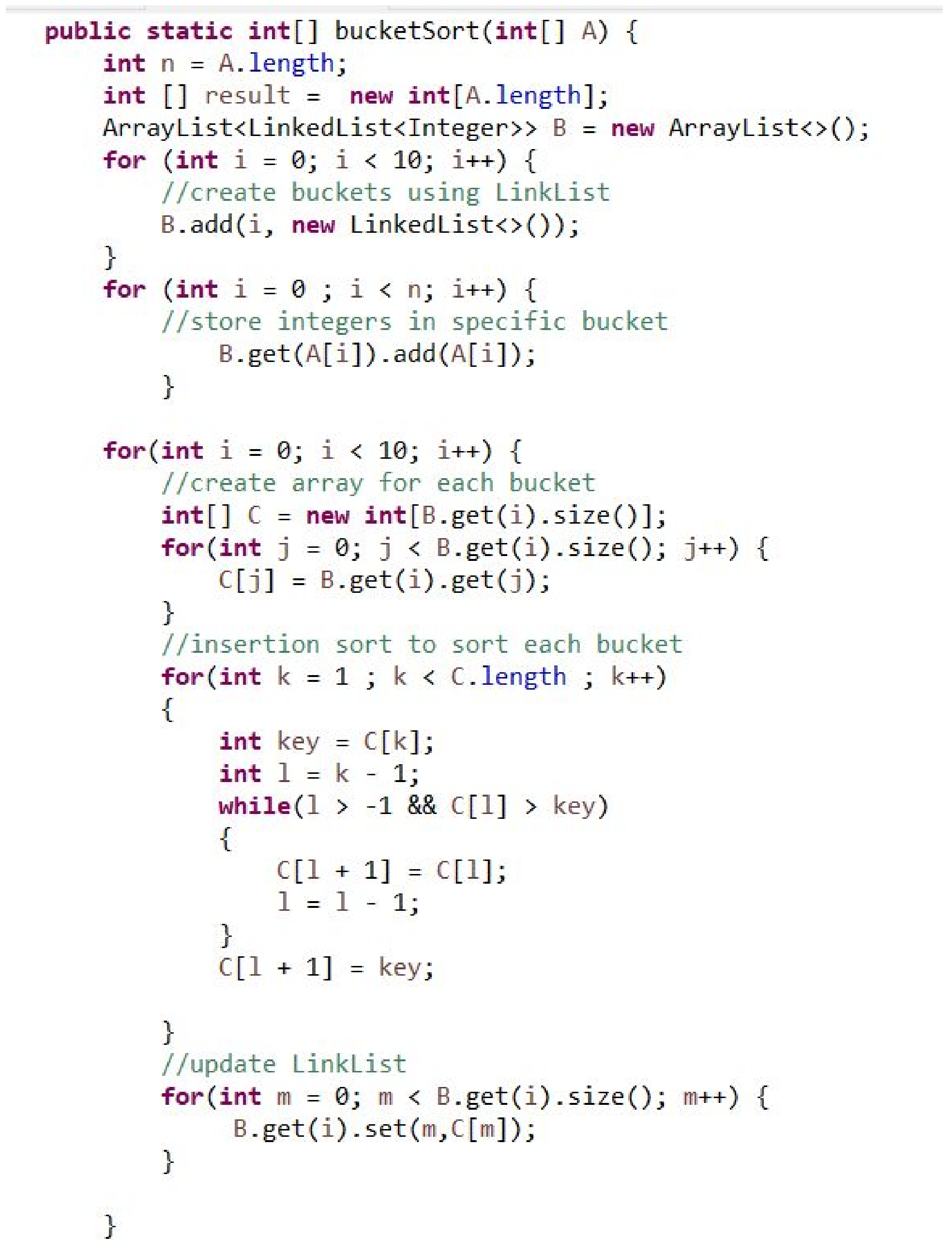
The bucketSort method in Tester class will be called



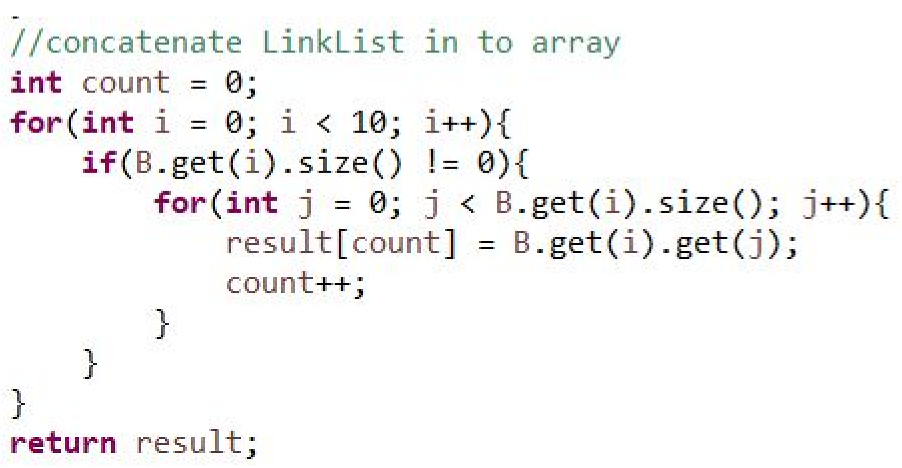
```
 */
public static void bucketSort () {
        System.out.println("Test Array:");
        //creat integerToSorts object for every element in testing array
        integerToSorts = new ArrayList<IntegerToSort>();
        for (int i = 0; i < A.length; i++) {
            integerToSorts.add(new IntegerToSort(A[i], i));
        }
        for (IntegerToSort integerToSort : integerToSorts) {
            System.out.print(" " + integerToSort.getInteger() + "_" + integerToSort.getSubscript() + " ");
        }
        System.out.println(" ");
        System.out.println("BucketSort Result:  ");
        int[] bucketResult = BucketSort.bucketSort(A);
```

Same as counting sort, it begins with creating a new ArrayList of IntegerToSort objects. Then, store every element in the testing array to the array and assign index to each element. Next, print out every element with the assigned index. After that, call the buckerSort() in the BucketSort class to sort the testing array.

The buckerSort() in the BucketSort class will begin with creating an ArrayList that has LinkedList elements inside of it. Then, create 10 buckets using LinkedList since the testing array is range from 0-9. Then it will assign the integers to the specified bucket. After that, I create an int array for LinkedList in the ArrayList to perform insertion sort. Then, I update the LinkedList based on the insertion result. Finally, concatenate the LinkedList into an array called result.

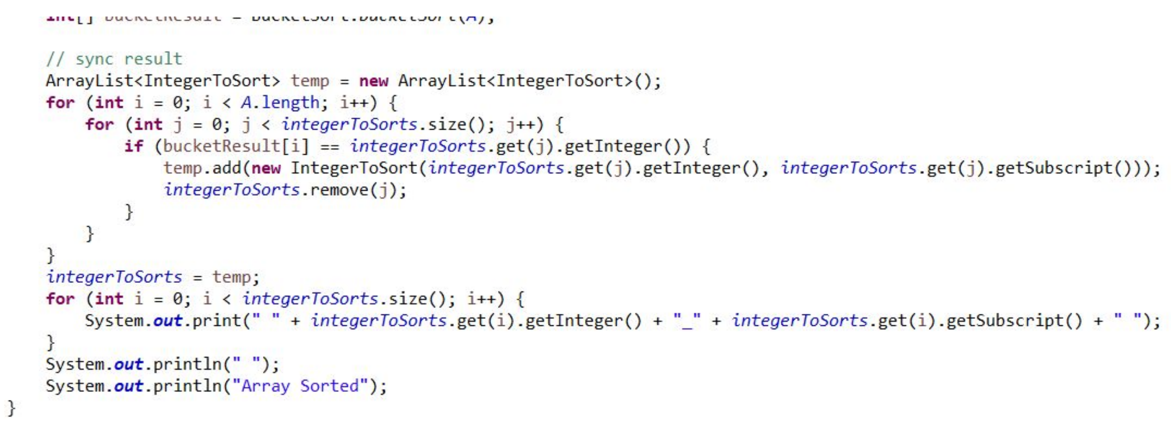


```java
public static int[] bucketSort(int[] A) {
    int n = A.length;
    int [] result =  new int[A.length];
    ArrayList<LinkedList<Integer>> B = new ArrayList<>();
    for (int i = 0; i < 10; i++) {
        //create buckets using LinkList
        B.add(i, new LinkedList<>());
    }
    for (int i = 0 ; i < n; i++) {
        //store integers in specific bucket
            B.get(A[i]).add(A[i]);
        }

    for(int i = 0; i < 10; i++) {
        //create array for each bucket
        int[] C = new int[B.get(i).size()];
        for(int j = 0; j < B.get(i).size(); j++) {
            C[j] = B.get(i).get(j);
        }
        //insertion sort to sort each bucket
        for(int k = 1 ; k < C.length ; k++)
        {
            int key = C[k];
            int l = k - 1;
            while(l > -1 && C[l] > key)
            {
                C[l + 1] = C[l];
                l = l - 1;
            }
            C[l + 1] = key;

        }
        //update LinkList
        for(int m = 0; m < B.get(i).size(); m++) {
             B.get(i).set(m,C[m]);
        }

    }
```



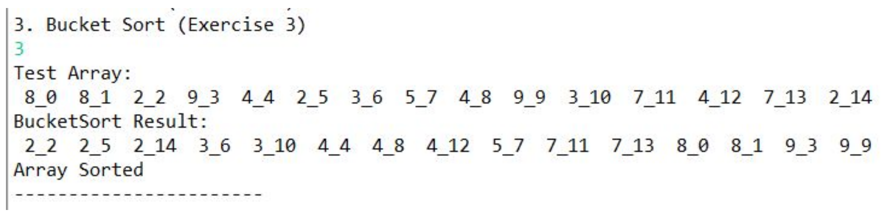
```
//concatenate LinkList in to array
int count = 0;
for(int i = 0; i < 10; i++){
    if(B.get(i).size() != 0){
        for(int j = 0; j < B.get(i).size(); j++){
            result[count] = B.get(i).get(j);
            count++;
        }
    }
}
return result;
```

After the bucket sort process. Sync the elements in the array and the ArrayList based on the result in the result array. So the ArrayList is updated to sorted order. At last, print out the result and corresponding index to prove the algorithm is a stable algorithm. This algorithm is stable because it sorts in order and the indexes are in order as well.



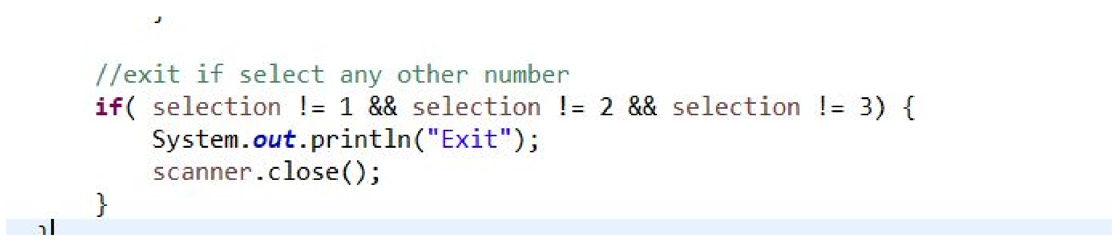
```
        int[] bucketResult = BucketSort.bucketSort(A);

        // sync result
        ArrayList<IntegerToSort> temp = new ArrayList<IntegerToSort>();
        for (int i = 0; i < A.length; i++) {
            for (int j = 0; j < integerToSorts.size(); j++) {
                if (bucketResult[i] == integerToSorts.get(j).getInteger()) {
                    temp.add(new IntegerToSort(integerToSorts.get(j).getInteger(), integerToSorts.get(j).getSubscript()));
                    integerToSorts.remove(j);
                }
            }
        }
        integerToSorts = temp;
        for (int i = 0; i < integerToSorts.size(); i++) {
            System.out.print(" " + integerToSorts.get(i).getInteger() + "_" + integerToSorts.get(i).getSubscript() + " ");
        }
        System.out.println(" ");
        System.out.println("Array Sorted");
    }
```

Output:

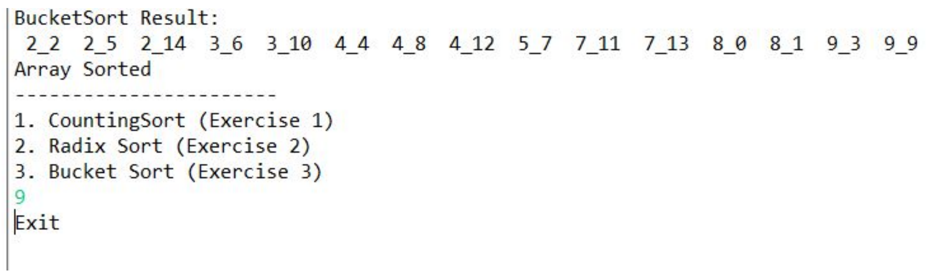


```
3. Bucket Sort (Exercise 3)
3
Test Array:
 8_0  8_1  2_2  9_3  4_4  2_5  3_6  5_7  4_8  9_9  3_10  7_11  4_12  7_13  2_14
BucketSort Result:
 2_2  2_5  2_14  3_6  3_10  4_4  4_8  4_12  5_7  7_11  7_13  8_0  8_1  9_3  9_9
Array Sorted
-----------------------
```

If the user presses any other number, the scanner will be closed and the program will exit, I will use 9 as an example.


```
        //exit if select any other number
        if( selection != 1 && selection != 2 && selection != 3) {
            System.out.println("Exit");
            scanner.close();
        }
```

Output:


```
BucketSort Result:
 2_2  2_5  2_14  3_6  3_10  4_4  4_8  4_12  5_7  7_11  7_13  8_0  8_1  9_3  9_9
Array Sorted
-----------------------
1. CountingSort (Exercise 1)
2. Radix Sort (Exercise 2)
3. Bucket Sort (Exercise 3)
9
Exit
```

Procedure to unzip files, install applications and run code.

Window OS:

1. Unzip PA2-Wu.zip on the preferred directory.
2. Run cmd (Command Prompt) on the computer.
3. Use command cd in cmd to change the directory to where PA-2.jar located.
4. Use command java -jar PA-2.jar in cmd to run the program.

Problems encountered during the implementation

One of the problems that I encountered during the implementation is out of bound exceptions while implementing the soring methods. Since the textbook began with index 1 and I started with index 0. So, it will be different. To encounter it , I use the debugger in the IDE to figure out the issues by going through every step and solving them.

Another problem that I encountered will be implementing the bucket sort algorithm. I found out that I am not able to store an array inside an array for bucket purpose. So I switch to use LinkedList for the buckets and sort the LinkedLists into an ArrayList. I am also not sure how to use insertion sort to sort LinkedList, so I sort the LinkedList elements into an array and use insertion sort to sort the array.

For radix sort, I was having issues with how to sort the hex number because it contains letters and numbers. Then, professor Wu gave us a hint that we can use ASCII code to perform the sorting process. So, I convert each character in each hex number into ASCII code and use counting sort to sort them. Another problem I encounter during the implementation of radix sort is updating the original array after the first sorting. Some elements will be null after updating. So

I used IDE to figure out and solve it by adding a if statement to see if a specific element is null. It works as if a specific element is null, skip it and go to the next element.

Lessons Learned

Throughout this program assignment, I learned more about how to implement pseudo codes. I learned that there are various ways to implement pseudo codes. For this assignment, the hardest part for me is debugging and thinking about how to implement these sorting algorithms. I learned that the debugging tool in the IDE is a useful tool when I am facing bugs. I can track every step of the program using the debugging tool and figure out what the issues are. Figuring out how to write the code is also making this assignment harder for me. I was stuck on what should be used as buckets for bucket sort and also stuck on how to sort letters and numbers. However, I had solved them during the development process. I learned that I can use LinkedList as buckets and use ascii codes to sort letters and numbers. I had spent a lot of time on this assignment. This is a long process and I am getting better on it. It is just a part of my way to be a software engineer.