**Social Security Numbers Sorting**

**Programming Assignment 2**

April 28, 2019

# 

# 

# **By: Arielle Shnayder**

CS146-07

San Jose State University

Table of Contents

Design and Implementation………………...…….……………..…………………………...2-3

Methods…………………………………………………………………………….………...4-5

1. Main……………………………………………..……………………..……………...4
2. QuickSort…………………………………………..………………………………….4
3. Partition…………………………………………..……………………………….…..4
4. Bucket Sort…………………………………………..………………………….…….4
5. Insertion Sort…………………………………………..………………………….…..4
6. Radix Sort…………………………………………..…………………………….…...5
7. Counting Sort…………………………………………..…………………………...…5
8. Output Stats…………………………………………..……………………………......5

Screenshots………………………………………………………………………………….6-7

Procedure to Unzip……………………………………………………………………...…….8

Encountered Problems…………………………………………………………………..….8-9

Lessons Learned…………………………………………………………………….………..9

# **Design and Implementation**

For the Social Security Number Sorting project I used an array to store all the nine digit numbers. First I used the PrintWriter to generate my four txt files; “Random\_SSN.txt”, “Quick\_SSN.txt”, “Bucket\_SSN.txt”, and “Radix\_SSN.txt”. I defined three integer arrays with the size of 300 to store the randomized SSN numbers. I used a for loop to generate three random integers one with three digits, one with two digits, and lastly four digits and stored those in the three arrays. I also entered the “-” in the correct places for a real SSN and pushed these nine digit integers in the “Random\_SSN.txt” file.

Next, I started with calling QuickSort and passing the first array and its length. Then in the for loop (300 times), I create a string of the integer so that I could add the “-” in between and then I print it to the “Quick\_SSN.txt” in sorted order. Lastly, I call OutputStats with array A to print the results of how many people are from which states. Then, I do the same thing as above for Bucket Sort with array B and the same for Radix Sort with array C. The output prints the statistics three times using all three arrays by calling outputStats method.

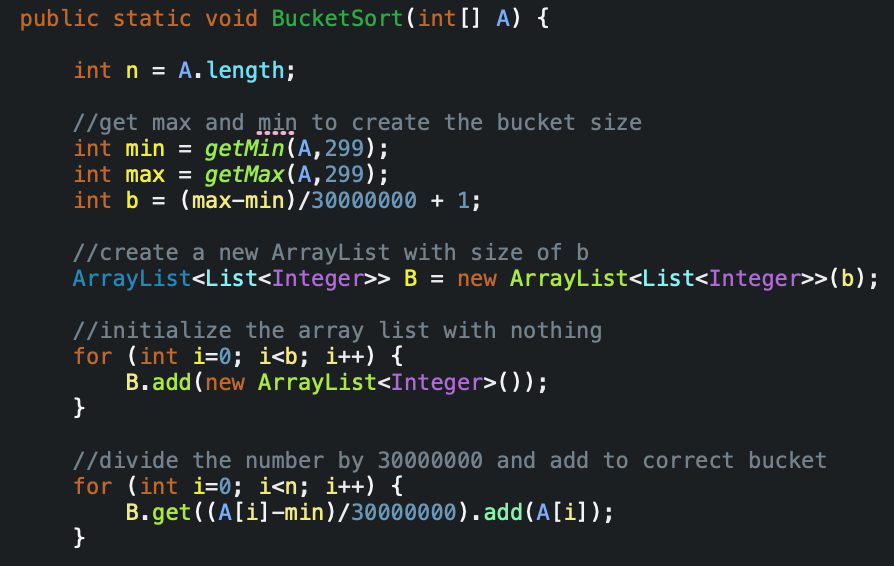
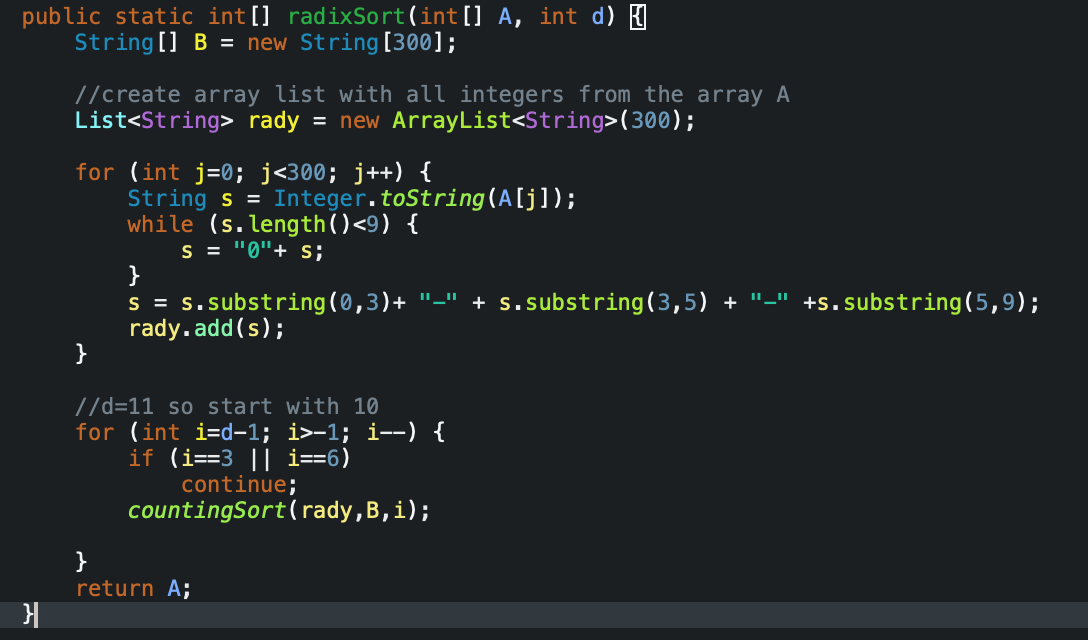
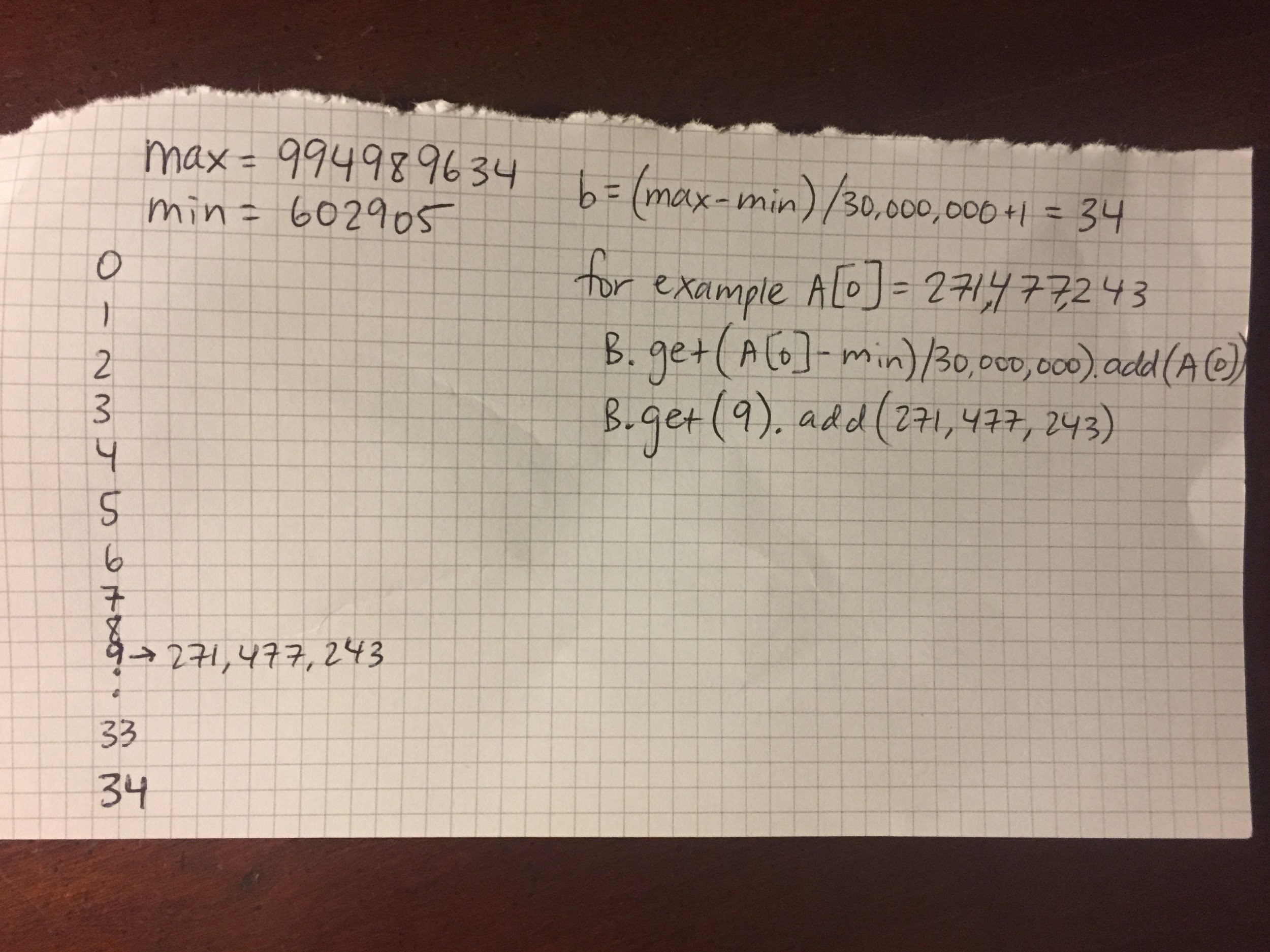
After my main method, I define all my sorting methods such as QuickSort, Partition, BucketSort, insertionSort, radixSort, countingSort, getMax. Lastly I have outputStats which prints out the statistics that has a for loop and many if statements. For

Bucket sort, I used max 34 buckets. I divide the integer by 30,000,000 and then use

Insertion sort to sort the elements in the same bucket. For radix sort I define a digit

by using the substring to get the last digit first then I pass the last digit for every element to counting sort.

# 



# 

# 

# 



# **Methods**

Main:

1. Use PrintWriter to create all txt files when java project is ran.
2. Creating random integer
3. Adding the “-” to the random integer after the first three digits then after the next two digits
4. Call quicksort, bucketsort, radix
5. Print the sorted arrays to corresponding txt files

QuickSort(int[] A, int p, int r):

1. Use pseudo code from textbook
2. Calls Partition and then calls itself recursively
3. Ends up sorting the array

Partition(int[] A, int p, int r):

1. Checks each number with pivot and if it is less than the pivot it pushes them to the left

of all the numbers larger the pivot.

2. Swaps the pivot with the first number that is larger than it so pivot is in between the

smaller numbers and larger numbers than it. Returns i+1 which is the index where pivot is.

Bucket Sort(int[] A):

1. Define int b the number of buckets. I chose to use (max-min)/30000000+1 = about 33
2. Create an ArrayList and initialize each element to 0 size of b
3. Then add A[i] to the arraylist of (A[i]-min/30000000) adds to the correct bucket.
4. Then it goes through the for loop to check to see if more than one integer is in the

same bucket and then uses insertion sort to sort the numbers in each bucket.

Insertion Sort(int[] b):

1. Pseudo code from book
2. Uses a for loop to check if the next element is smaller than the previous and moves it

to the correct spot.

Radix Sort(int[] A, int d):

1. Create an array of strings B
2. Create an array list and add all elements from array A to it
3. Add the dashes to the numbers
4. Call counting sort passing the empty array B and index 10 till 0

Counting Sort(int[] A, int n, int k):

1. Creates a new array C that is from 0-9 and initializes them to 0
2. The for loop checks if the value of an input is i then we increment C[i]
3. Then we place the element from array A to its correct position in array B.

GetMax(int A[], int n):

1. Finds max integer from the array passed into it.

GetMin(int A[], int n):

1. Finds the minimum integer in the array passed.

Output Stats(int[] A):

1. Looks at each number in the array and checks whether the first three integers are from

which section.

2. Prints the statistics from Northeast, south, northwest, middle, and west coast states.

# 

# 

# 

# 

# 

# 

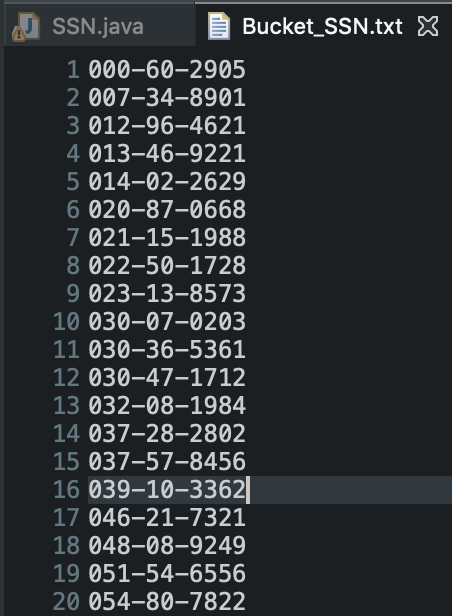
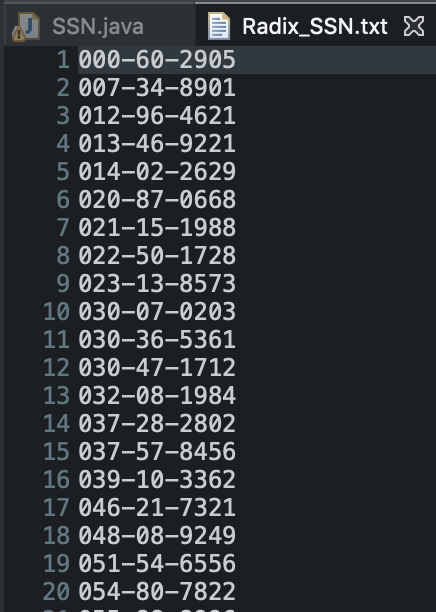
# 

# **Screenshots**

# 

# 

# 



# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# **Procedure to Unzip**

1. Unzip folder
2. Open terminal
3. Change to the directory of the project
4. Type “ls” to make sure PA2.jar is inside that directory
5. Type “java -jar PA2.jar”
6. Follow instructions from the project

# 

# **Encountered Problems**

I encountered many problems throughout the process of building this SSN generator.

The first problem I encountered was generating a nine digit integer, the two middle numbers

were sometimes 00 and it was supposed to be between 01 and 99. To fix this problem I created

three random integers and made each number generate in the correct range. Then I added all the

numbers to create a nine digit number. The second problem I encountered was adding the dash

between the first three numbers and the next two digits because it was an integer and the dash

is a string.

Another major issue I struggled with is reading the numbers to an array. At first I tried

reading each number by using scanner from the Random\_SSN.txt file but then I had to get

rid of all the dashes. I figured out that I did not need to read anything from the txt file and

that I could create my array at the beginning before I put the dash into it. This helped a lot

because I was able to quickly create three arrays with the same random integers in all three.

For quicksort, I had an easy time sorting the array and pushing the integers with the

dash to the Quick\_SSN.txt. However, I struggled with Radix and Bucket Sort. For bucket sort

I had a hard time creating buckets because I did not want to make a ton of buckets. I figured

out that you can just divide each number by the same digit, I chose 30,000,000, and enter the

digit corresponding to each digit.

# **Lessons Learned**

While building this project, I learned many things about programming. First thing I

learned was how all the sorting algorithms work more in depth because I had to change

some things for most of the sorting algorithms. I understood how radix sort uses counting

Sort for the stable sort. It was very challenging thinking of ways to get the buckets working

as well as getting the radix sort to pass only one digit from each element before passing it

to counting sort. I learned how to use a PrintWriter and how to make a txt file because I had

never learned that in my cs46b class. I also improved on coding less lines by creating one

method that I call several times.

I did try my best to try out different code to see if something would work and that

led me to understanding more about the overall code I had. This project was very interesting

to me because I saw the progress I was making in each step and it was something I had not

done in the past. I now have a better understanding about how sorting algorithms work in

real life including the social security number sorter.