

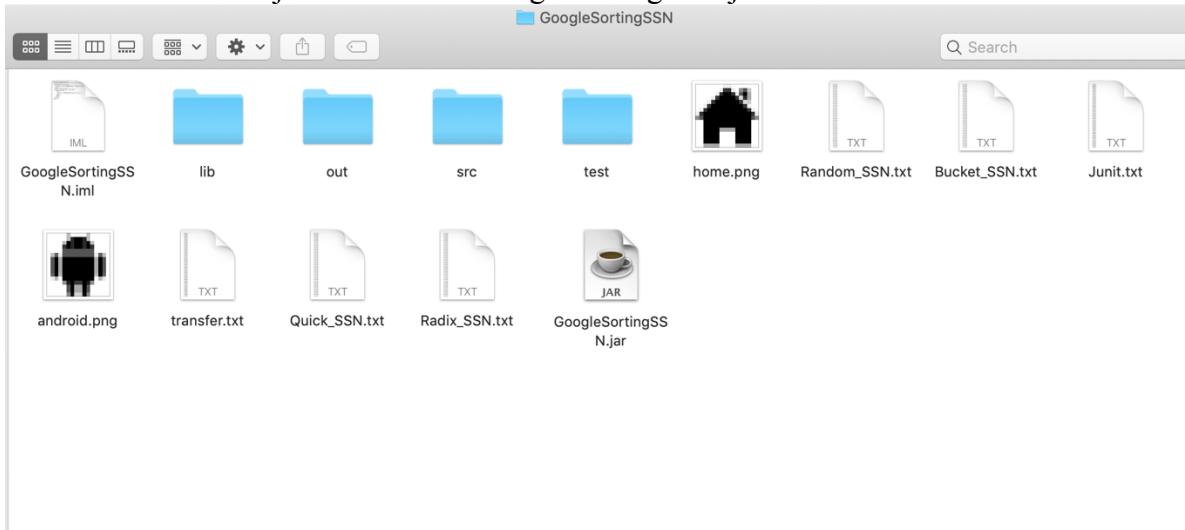
Programming Assignment 2: Sorting Social Security Numbers

Yuxiao (Tom) Zheng
CS146-Section7

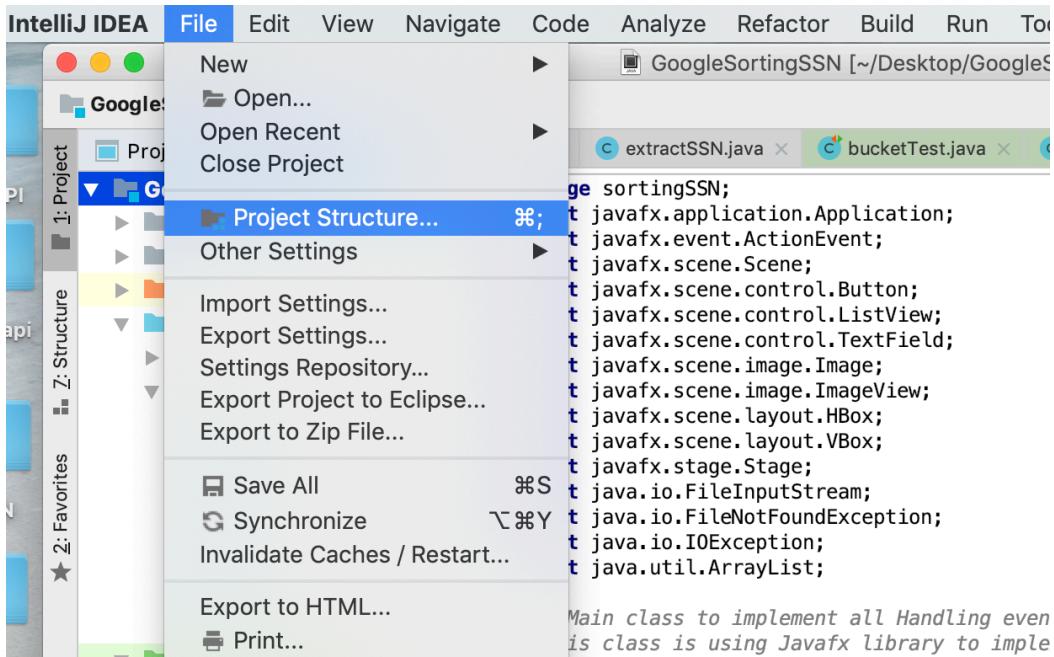
1. Unzip and Installation (P.1-5)
2. Programming Design(P.6-14)
3. Implementation(P.15-19)
4. classes/subroutines/**bucketSort radixSort**
explanation (P.20-31)
5. Self-testing (P.32-35)
6. Problems encountered during
implementation(P.36-39)
7. Leasson Learned(P. 40)
8. Running Time Big-**O** (Junit -Test)
(P. 41-46)

1. unzip and install

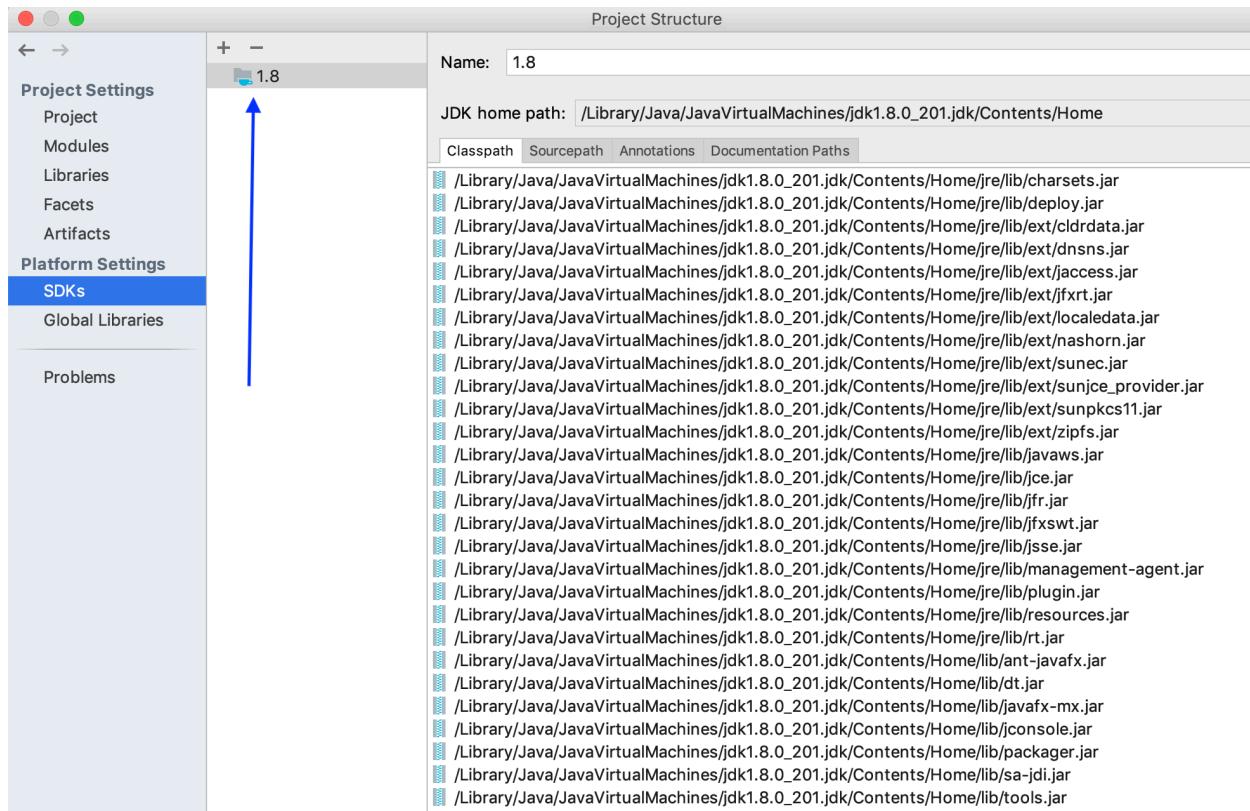
- Unzip the zip folder, there are three part, a **SRC** folder that contains all of the .java files, Quick_SSN.txt, Radix_SSN.txt, Bucket_SSN.txt, Random_SSN.txt, home.png, android.png, transfer.txt, Junit.txt, and a **TEST** folder contains Junit test .java files: bucketTest.java, quickTest.java, and radixTest.java
a Programming Report which is Microsoft Word Document which you are reading it now and a runnable .jar file named “GoogleSortingSSN.jar”.



- Now the above are the all of required files including icon .png
- Still need to import jdk 8, because we using javaFX library



1. Click File → Project Structure
2. Choose SDKs → click “+”
3. Pick the jdk 8 to import



- Now, the whole program interface look like the following:

```

1 package sortingSSN;
2 import javafx.application.Application;
3 import javafx.event.ActionEvent;
4 import javafx.scene.Scene;
5 import javafx.scene.control.Button;
6 import javafx.scene.control.ListView;
7 import javafx.scene.image.Image;
8 import javafx.scene.image.ImageView;
9 import javafx.scene.layout.HBox;
10 import javafx.scene.layout.VBox;
11 import javafx.stage.Stage;
12 import java.io.FileInputStream;
13 import java.io.FileNotFoundException;
14 import java.io.IOException;
15 import java.util.ArrayList;
16 import java.util.List;
17 /**
18 * A Main class to implement all Handling events for Sorting Social Security Numbers
19 * This class is using JavaFX library to implement User Interface
20 * The goal is to let all users to feel more intuitive than display on console
21 */
22 public class Main extends Application {
23
24     private ArrayList<String> homelist = new ArrayList<>();
25     private ArrayList<String> quicklist = new ArrayList<>();
26     private ArrayList<String> bucketlist = new ArrayList<>();
27     private ArrayList<String> radixlist = new ArrayList<>();
28     private Button homebtn; //generate 300 random SSN numbers
29     private Button quickbtn;
30     private Button bucketbtn;
31     private static TextField inputText;
32
33     /**
34      * @param primaryStage the primary stage
35      * @throws FileNotFoundException if the reading or writing file doesn't find, it will throw the exception
36      */
37     public void start(Stage primaryStage) throws FileNotFoundException {
38         inputText = new TextField("Input Social Security Number: ");
39         inputText.setPrefSize(200, 30);
40
41         //added a home button icon into the button
42         FileInputStream input = new FileInputStream( name: "home.png");
43         Image image = new Image(input);
44         ImageView imageView = new ImageView(image);
45         homebtn = new Button( text: "Search", imageView);
46         homebtn.setPrefSize( 200, 30);
47

```

- If you have seen this program main interface, that means you have installed successfully! Now you can test all function and Junit test!
- Note: the left directory path should look like the following:

The screenshot shows the IntelliJ IDEA interface with the following details:

- Project Bar:** GoogleSortingSSN > src > sortingSSN > Main
- Project View (1: Project):**
 - GoogleSortingSSN (~D)
 - .idea
 - lib
 - out
 - src
 - META-INF
 - sortingSSN
 - META-INF
 - bucket
 - extractSSN
 - Main
 - quick
 - radix
 - ssnCrawler
 - test
 - sortingSSN
 - bucketTest
 - quickTest
 - radixTest
 - External Libraries
 - Scratches and Consoles
- Code Editor (Main.java):**

```

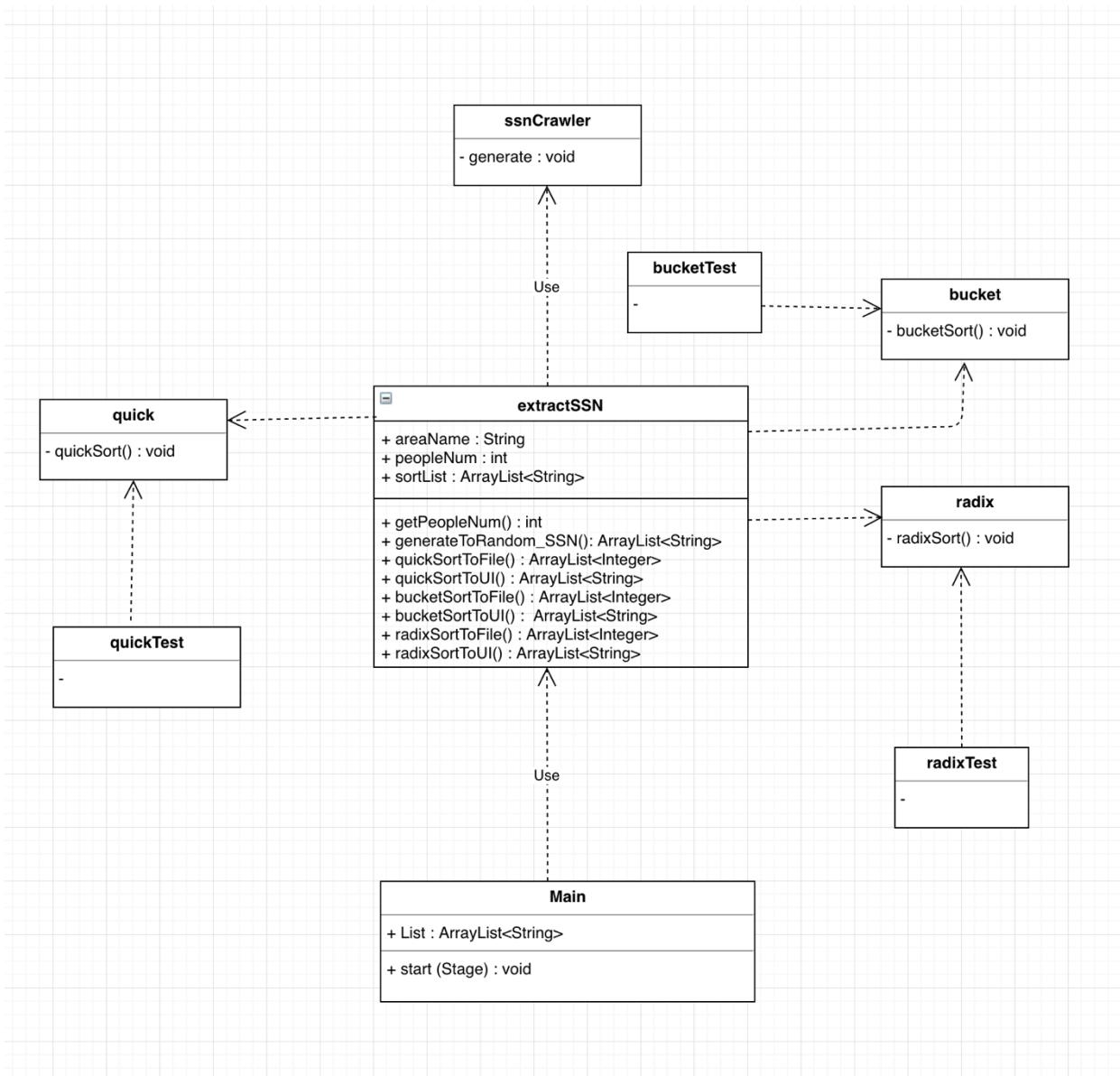
1 package sortingSSN;
2 import javafx.appli
3 import javafx.event
4 import javafx.scene
5 import javafx.scene
6 import javafx.scene
7 import javafx.scene
8 import javafx.scene
9 import javafx.scene
10 import javafx.scene
11 import javafx.scene
12 import javafx.stage
13 import java.io.File
14 import java.io.File
15 import java.io.IOEx
16 import java.util.Ar
17 /**
18 * A Main class to
19 * This class is us
20 * The goal is to l
21 */
22 public class Main {
23
24     private ArrayList<
25     private ArrayList<
26     private ArrayList<
27     private ArrayList<
28     private Button
29     private Button
30     private Button
31     private Button
32     private static
33
34     /**
35      * @param prime
36      * @throws File
37      */
38     public void sta
39         inputText =
40         inputText.s
41
42     //added a t

```

- **Warning:** Don't move the **.txt file** and **.png file** to the other folder, or you won't see the icon display on the button in the **User Interface**, or you won't see the **SSN** numbers into the **.txt** files.
- **Note:** the further installation for **Junit Test**, you will see it in **Part 8 : Running Time Big-O (Junit -Test)**

2. Programming Design (deeply encapsulation)

- UML diagram:



- Create a **ssnCrawler** class which has a method named [void generate()], this method will generate 300 random SSN numbers.
- Create a class **extractSSN** which has some necessary attributes:

```

/* Area name */
private String necs = "[Northeast Coast States] : ";
private String scs = "[South Coast States] : ";
private String ms = "[Middle States] : ";
private String ms = "[Middle States] : ";
private String nwcs = "[Northwest Coast States] : ";
private String wcs = "[West Coast States] : ";
/* Area people number */
public static int necsPeople;
public static int scsPeople;
public static int msPeople;
public static int nwcsPeople;
public static int wcsPeople;
/* after quickSort, bucketSort, and radixSort, all sorted data will into the
required ArrayList */
private ArrayList<String> strlist = new ArrayList<>();
private ArrayList<String> quickList = new ArrayList<>();
private ArrayList<String> bucketList = new ArrayList<>();
private ArrayList<String> radixList = new ArrayList<>();
/* Access Method: get different area people's number */
public int getNecsPeople(){
    return necsPeople;
}
public int getScsPeople(){
    return scsPeople;
}
public int getMsPeople(){
    return msPeople;
}
public int getNwcsPeople(){
    return nwcsPeople;
}
public int getWcsPeople(){
    return wcsPeople;
}

```

```

/*
public class extractSSN {
    /* Area name */
    private String necs = "[Northeast Coast States] : ";
    private String scs = "[South Coast States] : ";
    private String ms = "[Middle States] : ";
    private String nwcs = "[Northwest Coast States] : ";
    private String wcs = "[West Coast States] : ";
    /* Area people number */
    public static int necsPeople;
    public static int scsPeople;
    public static int msPeople;
    public static int nwcsPeople;
    public static int wcsPeople;
    /* after quickSort, bucketSort, and radixSort, all sorted data will into the required ArrayList */
    private ArrayList<String> strlist = new ArrayList<>();
    private ArrayList<String> quickList = new ArrayList<>();
    private ArrayList<String> bucketList = new ArrayList<>();
    private ArrayList<String> radixList = new ArrayList<>();

    /* Access Method: get different area people's number */
    public int getNecsPeople() { return necsPeople; }
    public int getScsPeople() { return scsPeople; }
    public int getMsPeople() { return msPeople; }
    public int getNwcsPeople() { return nwcsPeople; }
    public int getWcsPeople() { return wcsPeople; }
}

```

The above 4 parts:

1. displaying the area name on the **User Interface**
2. will count the total numbers of different area's people
3. provides a data structure **ArrayList**, which can store 300 sorted SSN numbers into my **arrayList**
4. return the changed people number of different area

- Inside **extractSSN** class, I will create seven method to help implement different functioning:
 1. **public ArrayList<String> generateToRandom_SSN()**
 2. **public ArrayList<Integer> quickSortToFile()**
 3. **public ArrayList<String> quickSortToUI()**
 4. **public ArrayList<Integer> bucketSortToFile()**
 5. **public ArrayList<String> bucketSortToUI()**
 6. **public ArrayList<Integer> radixSortToFile()**
 7. **public ArrayList<String> radixSortToUI()**
- Originally, this seven methods' codes will write into the **Main** class; they help **Main** class implement different functioning.
- Now, from **method()** 1 to 7, I only write either one sentence or two sentences inside **try()** clause of different button event in the **Main** class to implement the user's desired functioning : this way offers the **highest possible degreee of encapsulation**; see the **codes** of 7 methods encapsulating in **extractSSN** class. I will further explain in Part 4, “**4. Classes/Subroutines/function call**”.

- Create a class **quick** which provide a method called **quickSort()** to help sort 300 SSN numbers

```
public class quick {

    //Create a new arrayList in order to add ssn numbers
    private ArrayList<Integer> arrlist;

    /**
     * The following procedure implements quickSort
     * @param arr the given arrayList
     * @param p the left position
     * @param r the right position
     */
    public void quickSort(ArrayList<Integer>arr, int p, int r){
        if(p < r){
            int q = Partition(arr, p, r);
            quickSort(arr, p, r-1);
            quickSort(arr, p+1, r);
        }
    }
}
```

- Create a class **bucket** which provide a method called **bucketSort()** to help sort 300 SSN numbers

```
/*
public class bucket{
    /**
     * Convert a Integer arrayList to a double array[]
     * @param intarr the given arrayList with Integer elements
     * @return a double array[]
     */
    public double[] convertArr(ArrayList<Integer> intarr){
        double [] darr = new double[intarr.size()];
        for(int i=0; i<intarr.size(); i++){
            darr[i] = (intarr.get(i))/ 1000000000.0;
        }
        return darr;
    }

    /**
     * BuckSort method to sort a double array
     * @param arr the required arrayList will order by calling bucketSort()
     * @return the new ArrayList with calling buckSort()
     */
    public void buckSort(double[] arr){
```

- Create a class **radix** which provide a method called **raidxSort()** to help sort 300 SSN numbers

```

public class radix {

    /**
     * @param arr the required arrayList to convert to a integer array
     * @return a new integer array
     */
    public int[] convertToArr(ArrayList<Integer> arr){
        int [] iarr = new int[arr.size()];
        for(int i=0; i<arr.size(); i++){
            iarr[i] = arr.get(i);
        }
        return iarr;
    }

    /**
     * Get the maximum from this array
     * @param arr the required array will be sorted
     * @param n the length of this array
     * @return a maximum
     */
    public int getMax(int arr[], int n){
        int max = arr[0];
        for(int i=1; i<n; i++){
            if(arr[i] > max){
                max = arr[i];
            }
        }
        return max;
    }
}

```

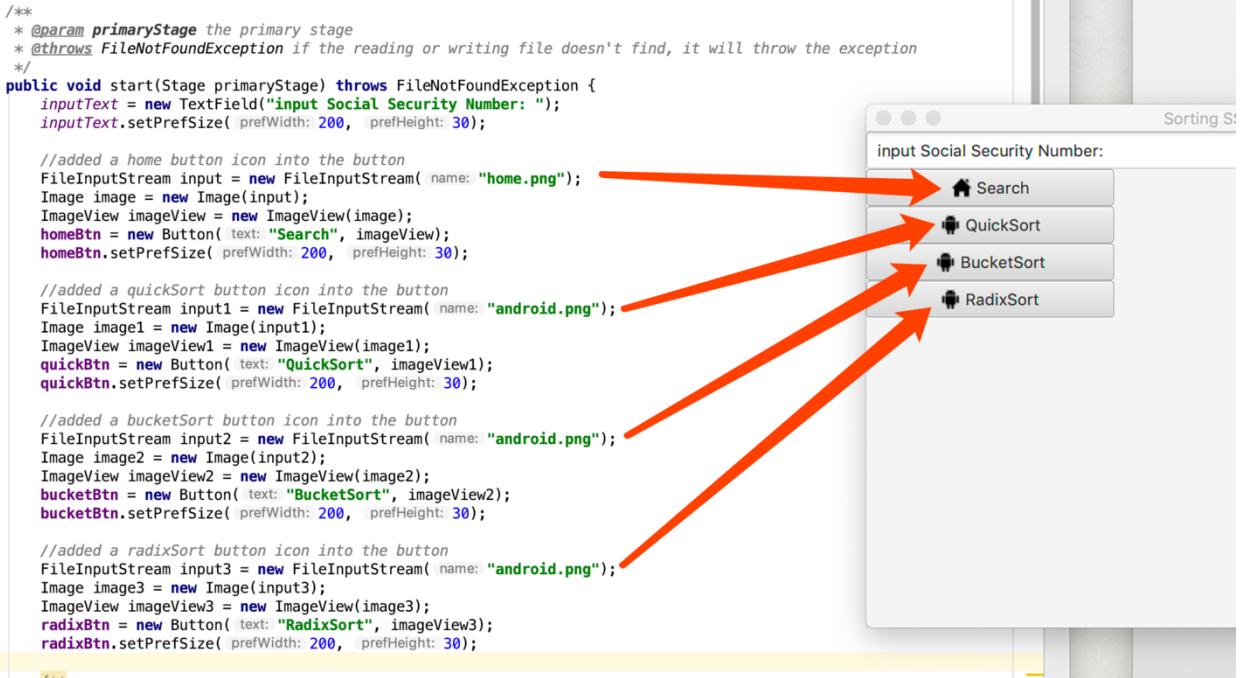
- Create a **Main** class with 4 arrayLists and 4 Buttons :

```

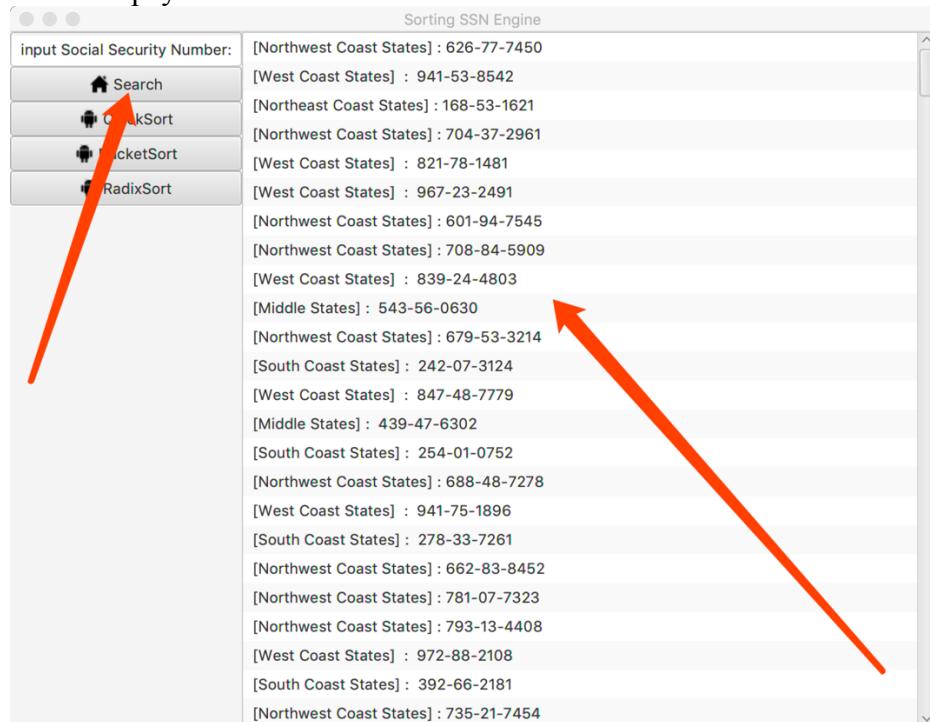
private ArrayList<String> homeList = new ArrayList<>();
private ArrayList<String> quicklist = new ArrayList<>();
private ArrayList<String> bucketlist = new ArrayList<>();
private ArrayList<String> radixlist = new ArrayList<>();
private Button homeBtn; //generate 300 random SSN numbers
private Button quickBtn;
private Button bucketBtn;
private Button radixBtn;

```

- Every button will add a .png file on the button, then the **User Interface** will show a button with the image (icon) :



- Inside **Main** class, there are 4 **arrayList<String>** to hold the sorted SSN numbers, this step's purpose is to one by one call the **ListView<String>** to add every record, finally all record will display on the **User Interface**:



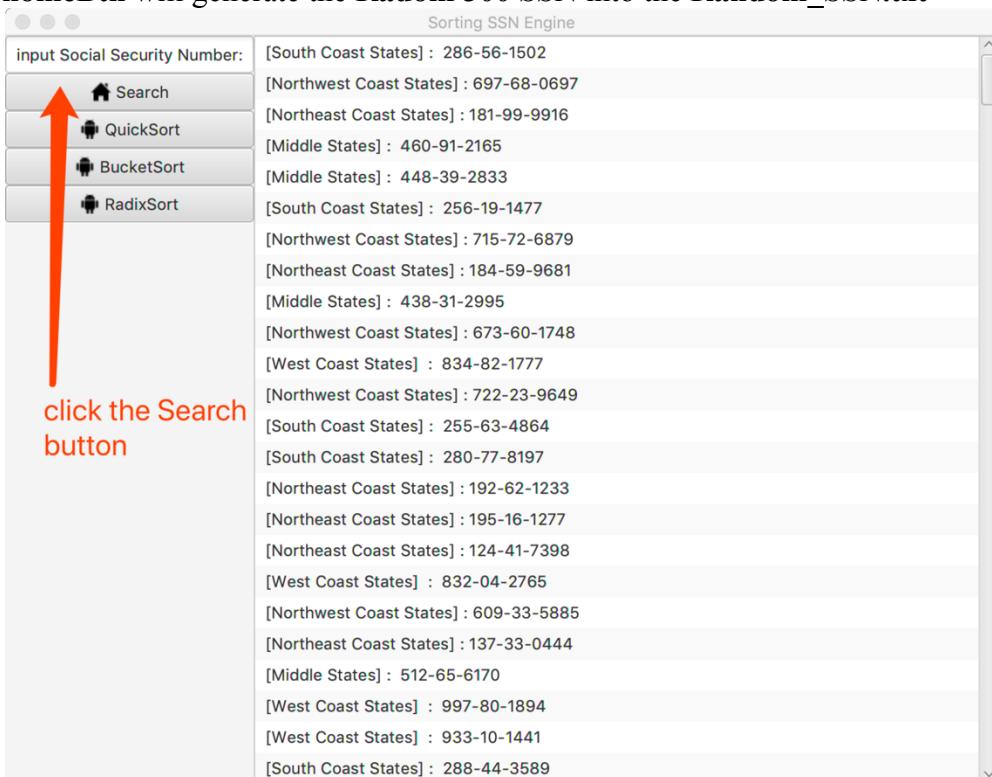
- In the Main class, there are 4 **button event** that will implement different functioning;
- A primary method **start(Stage ..)** to execute the main task.

```
public void start(Stage primaryStage) throws FileNotFoundException { ... }
```

- four different **button** to implement different desired functioning:

`homeBtn.setOnAction((ActionEvent event)-> { ... })`

homeBtn will generate the **Radom 300 SSN** into the **Random_SSN.txt**



```
quickBtn.setOnAction((ActionEvent event)->{ ... })
```

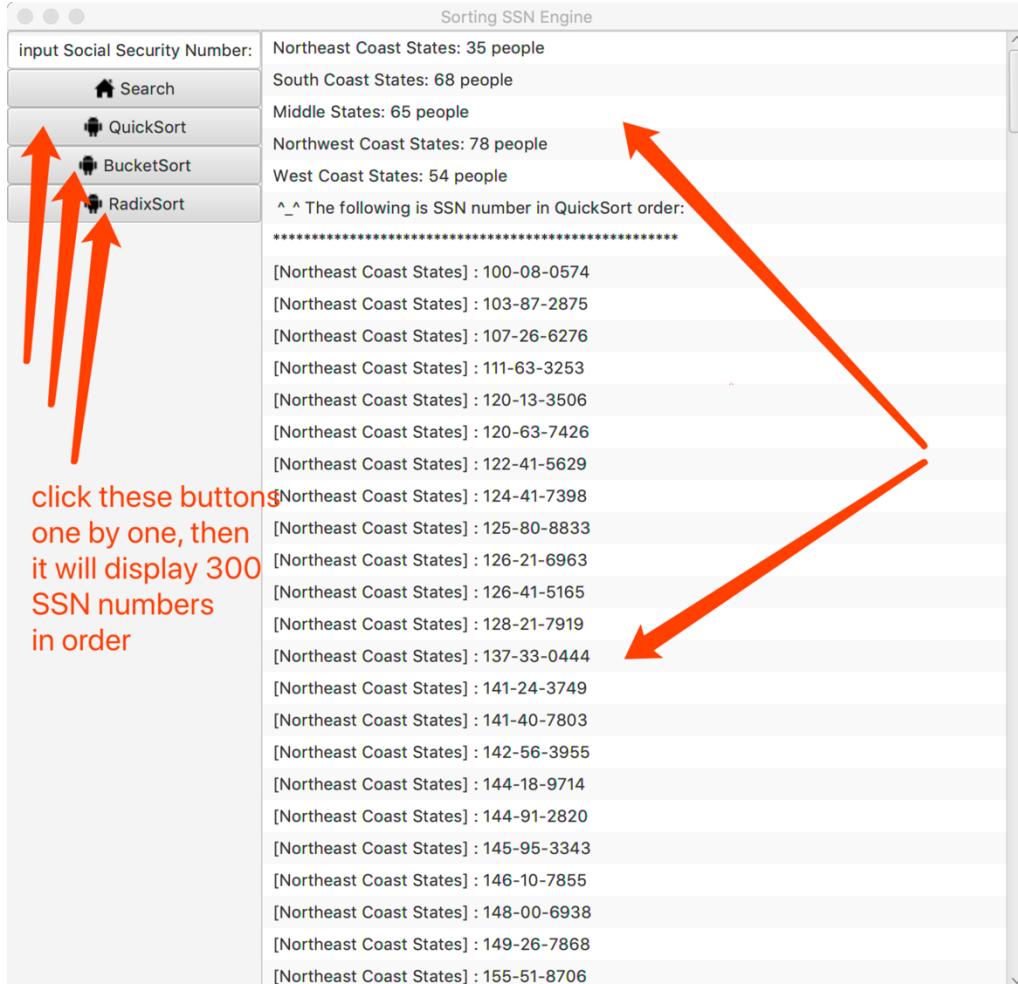
quickBtn will display the **300 SSN** on the **User Interface**, and it will write **300 SSN** into the **Quick_SSН.txt**

```
bucketBtn.setOnAction((ActionEvent event)->{ ... })
```

bucketBtn will display the **300 SSN** on the **User Interface**, and it will write **300 SSN** into the **Quick_SSН.txt**

```
radixBtn.setOnAction((ActionEvent event)->{ ... })
```

radixBtn will display the **300 SSN** on the **User Interface**, and it will write **300 SSN** into the **Radix_SSН.txt**



- The following shows that these **button event** can also write all SSN data into the required .txt files.

GoogleSortingSSN [-~/Desktop/GoogleSortingSSN] - .../Bucket_SSN.txt [GoogleSortingSSN]

Project / Random_SSN.txt / Quick_SSN.txt / Radix_SSN.txt / Bucket_SSN.txt

1: Project 2: Favorites

Bucket_SSN.txt

	Random_SSN.txt	Quick_SSN.txt	Radix_SSN.txt	Bucket_SSN.txt
1	286-56-1582	100-08-0574	101-90-9865	101-90-9865
2	697-68-0697	103-87-2875	126-73-7228	128-43-2989
3	181-99-9916	107-26-6276	127-86-5510	126-73-7228
4	460-91-2165	111-63-3253	128-43-2989	127-86-5509
5	448-39-2833	120-13-3506	128-55-3735	128-55-3735
6	256-19-1477	120-63-7425	132-36-6052	132-36-6052
7	715-72-6879	122-41-1049	132-36-6052	132-88-9974
8	184-59-9681	124-18-7398	138-64-4089	138-64-4089
9	438-31-2995	125-80-8833	139-30-4864	139-30-4864
10	673-60-1448	126-21-6963	140-88-9128	140-88-9128
11	834-82-1777	126-41-5165	147-80-1452	148-28-0648
12	722-53-9649	128-21-7919	148-28-0648	147-80-1452
13	205-63-4864	137-33-0444	149-37-9602	149-37-9602
14	280-77-8197	141-24-1749	152-13-8972	152-81-6111
15	102-62-1233	143-19-7893	152-81-6111	152-13-8972
16	195-16-1277	142-56-3935	160-43-3879	160-43-3879
17	124-41-7238	144-18-9714	162-99-7788	162-99-7788
18	832-0-2765	144-91-2820	168-48-2191	168-48-2191
19	60-33-5885	145-95-3343	179-75-5261	179-75-5261
20	137-33-0444	146-10-7855	180-25-8197	182-08-1548
21	512-65-6170	148-00-6938	182-08-1548	182-25-8197
22	997-80-1860	149-26-7868	182-21-0302	182-21-0302
23	933-19-1441	155-51-8706	182-28-7508	182-28-7508
24	110-44-3589	155-86-9646	187-69-4215	187-69-4215
25	883-20-4328	158-70-1301	192-25-3845	192-25-3845
26	605-86-9312	159-28-4829	194-08-7798	194-08-7798
27	432-16-1919	163-75-9172	195-84-0017	194-08-7798
28	662-99-8024	167-19-7012	195-90-7475	195-90-7475
29	884-16-9255	167-78-0887	196-01-4755	195-90-7475
30	882-48-4382	177-71-1911	199-29-2871	199-29-2871
31	312-87-4984	181-99-9916	199-81-6809	199-81-6809
32	981-32-2982	184-59-9681	205-45-6243	205-45-6243
33	530-46-3853	188-70-5434	208-27-8622	208-27-8622
34	405-40-1865	192-62-1233	211-86-7669	211-86-7669
35	832-70-6908	195-16-1277	212-23-1849	212-23-1849
36	752-54-2279	202-51-8688	220-20-6128	220-20-6128
37	719-70-8419	203-18-0607	227-95-2011	229-89-2455
38	384-64-2529	207-52-3989	228-30-7388	227-95-2011
39	961-45-2297	210-46-8678	229-89-2455	228-30-7388
40	144-91-2820	212-60-4405	229-90-7645	229-90-7645
41	257-22-3245	213-56-6285	230-35-4093	231-00-6729
42	103-87-2875	222-12-7369	231-00-6729	230-35-4093
43	645-27-3433	222-78-2597	232-36-3810	232-36-3810
44	826-32-9408	223-03-4125	233-66-8257	233-66-8257
45	438-76-0397	225-92-5133	237-44-1620	237-44-1620
46	212-60-4405	228-01-2945	238-01-9610	238-01-9610
47	282-22-1876	229-52-6559	240-68-1351	241-24-1458
48	682-95-4588	230-72-4842	241-24-1458	240-68-1351
49	348-00-8257	233-98-7327	242-63-0138	242-63-0138
50	736-56-7435	234-34-3524	245-90-1680	245-90-1680
51	502-79-5382	243-44-7375	251-90-2504	253-16-3662
52	283-34-3330	244-30-2213	253-16-3663	251-90-2504
53	609-65-2346	244-67-5205	254-74-1648	254-74-1648
54	429-24-1258	246-16-7988	256-31-2191	256-31-2191
55	390-75-6234	247-47-1324	256-44-1055	256-44-1055
56	159-28-4829	249-64-2474	259-76-2921	259-76-2921
57	145-95-3343	250-99-5037	266-75-4880	267-32-9435
58	891-53-5140	251-94-0384	267-32-9435	266-75-4880
59	755-25-6118	255-63-4864	269-73-5807	269-73-5807
60	690-68-6199	256-19-1477	270-05-6444	272-66-4217
61	791-76-8483	256-32-0032	272-66-4217	270-05-6444
62	867-06-0792	257-20-3245	273-28-7360	273-28-7360
63	937-06-4783	264-65-0751	273-93-5938	273-93-5938
64	149-26-7868	266-63-7113	275-53-9624	275-53-9624

Project / Random_SSN.txt / Quick_SSN.txt / Radix_SSN.txt / Bucket_SSN.txt

1: Project 2: Favorites

Bucket_SSN.txt

All files are up-to-date (19 minutes ago)

1:12 LF UTF-8 4 spaces

3. Implementation

ssnCrawler class has a method **void generate()** , which generate 300 SSN numbers.

```
int randomSSN = 100000000 + new Random().nextInt(900000000);
```

this statement will generate a 9-place number, from 0 ~ 999-99-9999, Note: it **will not** generate with “000” at the beginning.

The screenshot shows a Java development environment with two panes. On the left, the code editor displays `ssnCrawler.java` containing the `ssnCrawler` class. The `generate()` method is highlighted, showing its implementation. A red arrow points from the line `int randomSSN = 100000000 + new Random().nextInt(900000000);` in the code to the corresponding line in the generated file. On the right, the file `Random_SSN.txt` is shown, containing a list of 300 randomly generated SSN numbers. The red arrow starts near the bottom of the code editor and points directly to the first line of the text file.

```
SSN > ssnCrawler
nCrawler.java x
Random_SSN.txt x
package sortingSSN;

import java.io.File;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.Random;

/**
 * A class ssnCrawler can generate random 300 SSN number
 */
public class ssnCrawler {

    /**
     * This method to help user generate 300 SSN number
     * @throws FileNotFoundException if the System can not find the .txt file, it will
     */
    public static void generate() throws FileNotFoundException {
        String outFile = "transfer.txt";
        File outFile = new File(outFile);
        PrintWriter output = new PrintWriter(outFile);
        /**
         * generating 300 Random SSN numbers
         */
        for(int index=1; index<301; index++){
            int randomSSN = 100000000 + new Random().nextInt( bound: 900000000 );
            output.println(randomSSN);
        }
        output.close();
        System.out.println("Generate SSN numbers successfully");
    }

    /**
     * Testing the generate() method if it generating successfully
     * @param args
     */
    public static void main(String[] args) {
        try{
            generate();
        }catch (IOException e){
        }
    }
}
```

	Random_SSN.txt
1	286-56-1502
2	697-68-0697
3	181-99-9916
4	460-91-2165
5	448-39-2833
6	256-19-1477
7	715-72-6879
8	184-59-9681
9	438-31-2995
10	673-60-1748
11	834-82-1777
12	722-23-9649
13	255-63-4864
14	280-77-8197
15	192-62-1233
16	195-16-1277
17	124-41-7398
18	832-04-2765
19	609-33-5885
20	137-33-0444
21	512-65-6170
22	997-80-1894
23	933-10-1441
24	288-44-3589
25	883-20-4328
26	605-86-9312
27	432-16-1919
28	662-99-8024
29	804-16-9255
30	882-48-4382
31	312-87-4984
32	901-32-2982
33	530-46-3853
34	405-40-1865
35	832-70-6908
36	752-54-2279
37	719-70-8419
38	384-64-2529
39	961-45-2297
40	144-91-2820
41	257-20-3245
42	103-87-2875
43	645-27-3433
44	826-32-9408
45	438-76-0397
46	212-60-4405
47	282-22-1876
48	682-95-4588
49	348-00-8257
50	736-56-7435
51	502-79-5382
52	283-34-3330
53	609-65-2346

Implement extractSSN class's functioning:

1. `public ArrayList<String> generateToRandom_SSN()`

The screenshot shows a Java IDE interface with several tabs at the top: Main.java, ssnCrawler.java, SSN.java, extractSSN.java (which is the active tab), and quick.java. Below the tabs is the code for the extractSSN.java file. The code implements the `generateToRandom_SSN()` method. It reads from a file named `transfer.txt`, processes each line to generate a random SSN (with specific rules for different ranges of numbers), and writes the results to a file named `Random_SSN.txt`. The generated SSNs are listed on the right side of the screenshot.

```
private ArrayList<String> strlist = new ArrayList<>();
private ArrayList<String> quickList = new ArrayList<>();

public ArrayList<String> inputToArrayList() throws FileNotFoundException{
    ssnCrawler sc = new ssnCrawler();
    sc.generate();

    String inFile = "transfer.txt";
    File inputFile = new File(inFile);
    Scanner in = new Scanner(inputFile);
    String outFile = "Random_SSN.txt";
    PrintWriter out = new PrintWriter(outFile);

    while(in.hasNextInt()){
        String strValue;
        String str = in.nextLine();
        String first = str.substring(0, 3);
        int temp = Integer.parseInt(first);
        if(temp <= 199){
            strValue = nes + str.substring(0, 3) + "-" + str.substring(3, 5) + "-" + str.substring(5);
        }else if(temp >199 & temp <= 399){
            strValue = scs + str.substring(0, 3) + "-" + str.substring(3, 5) + "-" + str.substring(5);
        }else if(temp >399 & temp <= 599){
            strValue = ms + str.substring(0, 3) + "-" + str.substring(3, 5) + "-" + str.substring(5);
        }else if(temp >599 & temp <= 799){
            strValue = nws + str.substring(0, 3) + "-" + str.substring(3, 5) + "-" + str.substring(5);
        }else{
            strValue = wcs + str.substring(0, 3) + "-" + str.substring(3, 5) + "-" + str.substring(5);
        }

        out.println(str.substring(0, 3) + "-" + str.substring(3, 5) + "-" + str.substring(5));
        strlist.add(strValue);
    }
    System.out.println("Written successfully");
    in.close();
    out.close();
    return strlist;
}

public ArrayList<String> quickList() throws FileNotFoundException{
    ArrayList<Integer> temp = new ArrayList<>();
    String inFile = "Random_SSN.txt";
    File inputFile = new File(inFile);
    Scanner in = new Scanner(inputFile);
    String outFile = "Quick_SSN.txt";
    PrintWriter out = new PrintWriter(outFile);

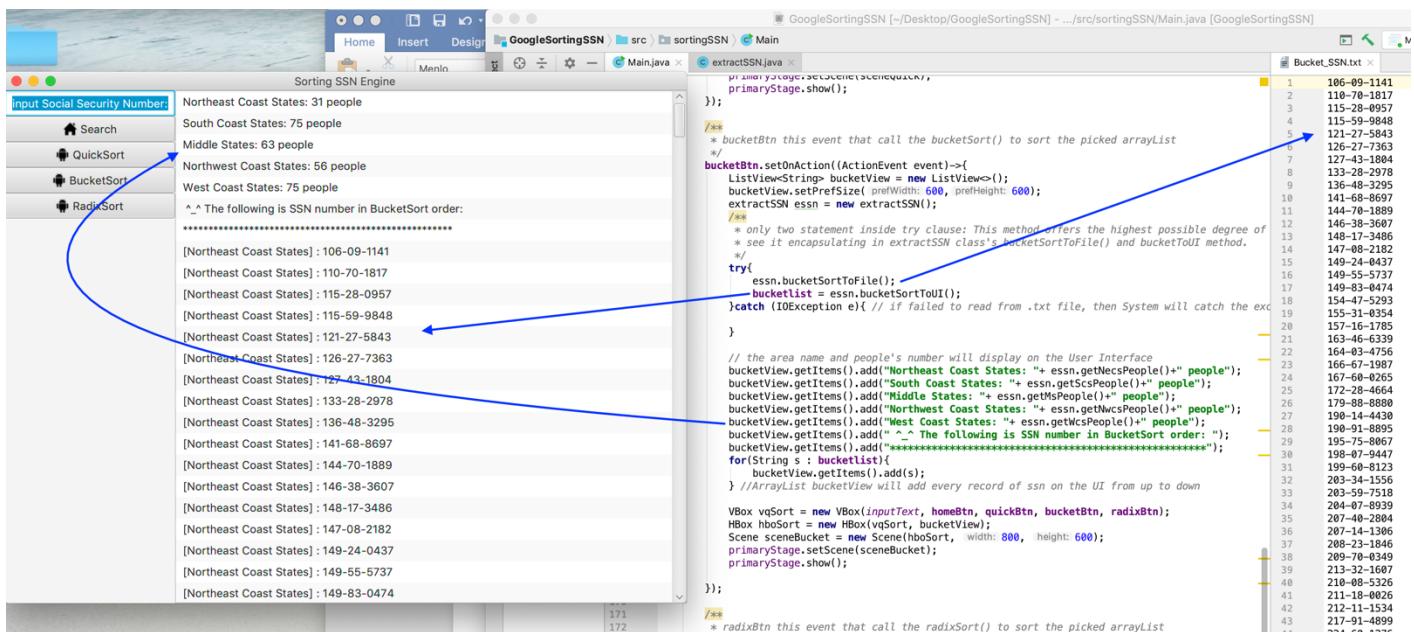
    while(in.hasNextInt()){
    }
}
```

Random_SSN.txt

Line	SSN
2	317-21-3982
3	577-17-4504
4	372-30-0755
5	735-01-2446
6	994-53-5578
7	201-11-8628
8	920-66-7752
9	335-41-7616
10	540-22-2461
11	203-37-7691
12	892-56-0074
13	745-54-0277
14	551-47-3058
15	341-38-0282
16	675-14-7272
17	400-35-5426
18	735-98-9168
19	789-95-3541
20	224-44-1750
21	207-46-1593
22	993-34-9019
23	744-01-1923
24	248-16-1130
25	603-01-0680
26	413-13-0763
27	167-44-5213
28	752-13-4157
29	930-46-5816
30	120-09-4275
31	188-15-7844
32	253-93-3010
33	558-11-9721
34	734-58-1067
35	963-01-1614
36	980-99-7246
37	606-34-2322
38	358-49-1540
39	387-24-4930
40	366-97-2816
41	254-64-9831
42	469-82-8185
43	835-04-6427
44	577-60-0736
45	919-97-5090
46	617-81-4371
47	470-54-6250
48	821-15-9218
49	159-98-6482
50	422-34-4696
51	215-48-9857
52	299-57-9217
53	018-06-2020

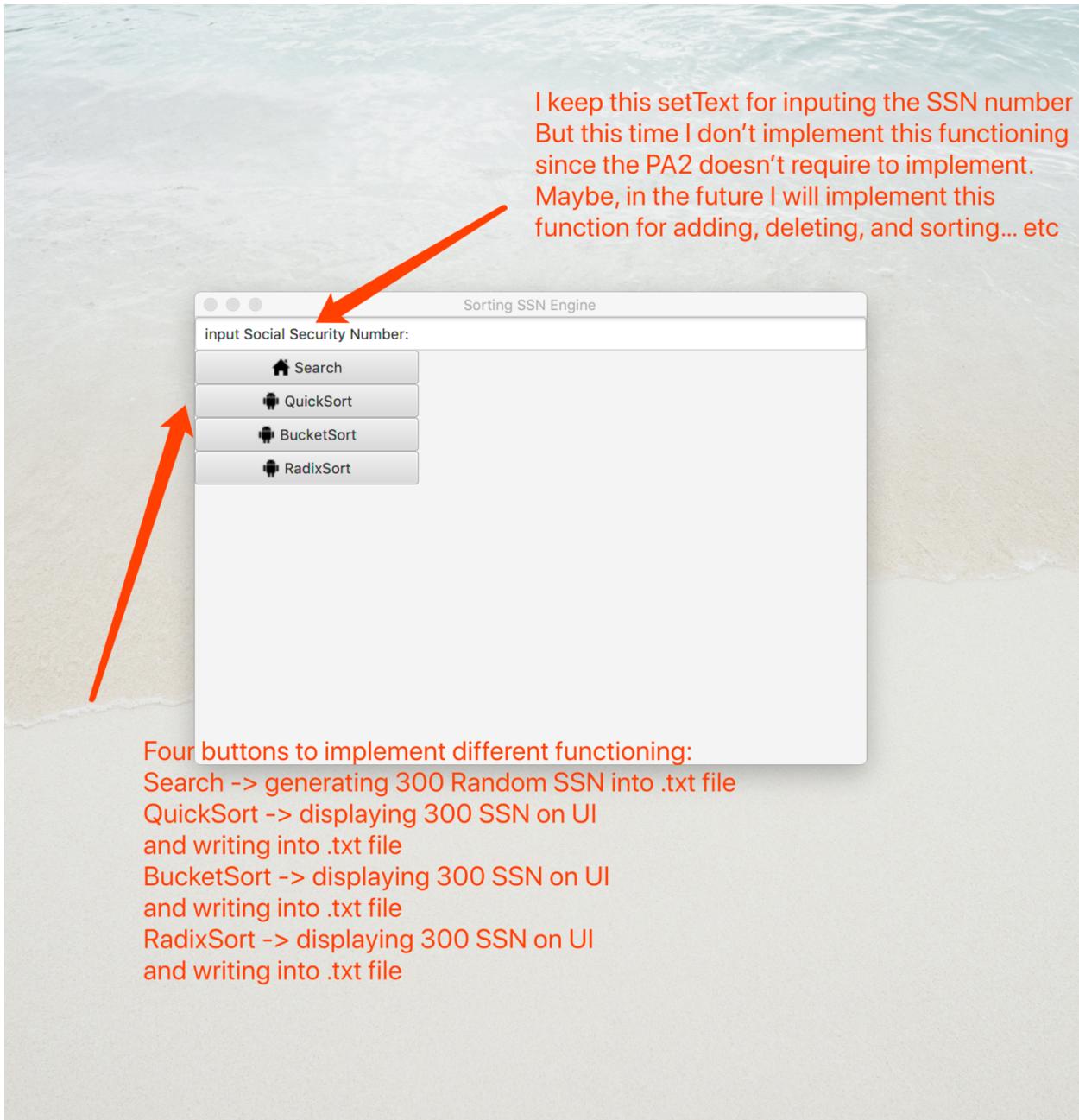
2. `public ArrayList<Integer> quickSortToFile()`
`public ArrayList<String> quickSortToUI()`

- This two methods either write **300 SSN** to the .txt file or display 300 SSN on the **User Interface**, see the result in the following:

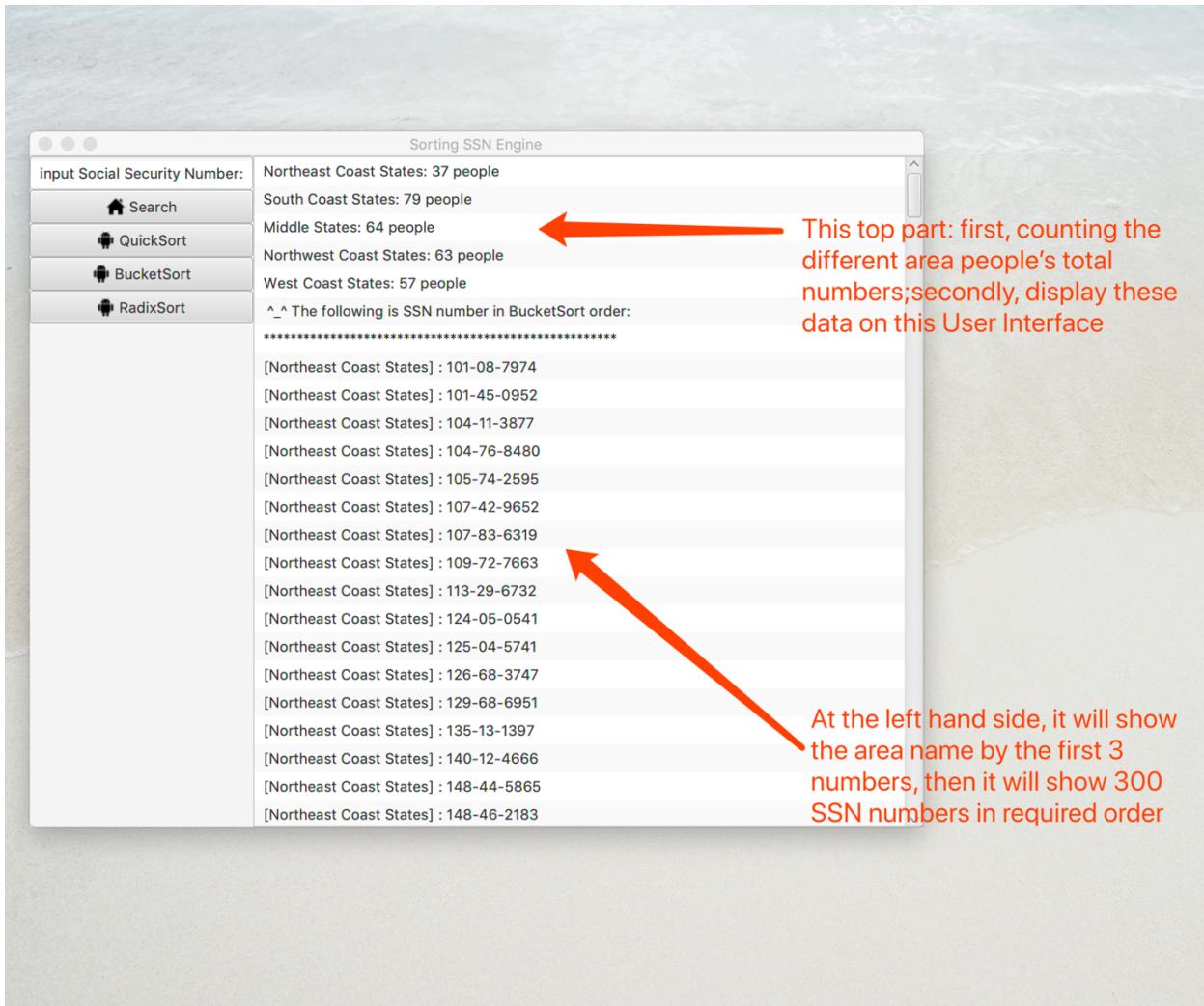


- Then I implement **quick** class for the **quickSort()** method
The main functioning, I will show in the **Part 4: A list of classes/subroutines**
- I also implement **bucket** class for the **bucketSort()** method
The main functioning, I will show in the **Part 4: A list of classes/subroutines**
- I implement **radix** class for the **radixSort()** method
The main functioning, I will show in the **Part 4: A list of classes/subroutines**

- After finished the above working, the **Main** class can implement four different required functioning:



- The main User Interface will show two parts:
 1. Button component with image icon
 2. Console will display SSN numbers, Area name, and the number of people



To summary: so far so good, I have **implemented** all class's functioning.
 In chapter 4, I will further explain my every class's method, class's subroutines, and every function calls.

4. classes/subroutines/function calls:

- In the **extractSSN** class, there is a method `ArrayList<String> generateToRandom_SSN()`
 1. Creating a **object** of **ssnCrawler**; calling the **generate()** method to generate 300 **Random SSN** numbers.
 2. The **generate()** will write 300 SSN into the **transfer.txt** ; this step's purpose is to make **transfer.txt** be a buffered .txt file. Because without **transfer.txt**, we only read data from **Random_SSN.txt** so that we can't see the change after we call **quickSort()**, **bucketSort()**, and **radixSort()**. Therefore, we can see the different and new SSN numbers.

The screenshots show the interface of the 'Sorting SSN Engine' application. The left sidebar contains buttons for 'Search', 'QuickSort', 'BucketSort', and 'RadixSort'. The right pane displays population data for coast states and lists SSN numbers in sorted order.

Top Screenshot (QuickSort output):

Coast State	Population
Northeast Coast States	30 people
South Coast States	75 people
Middle States	53 people
Northwest Coast States	75 people
West Coast States	67 people

^_ The following is SSN number in QuickSort order:

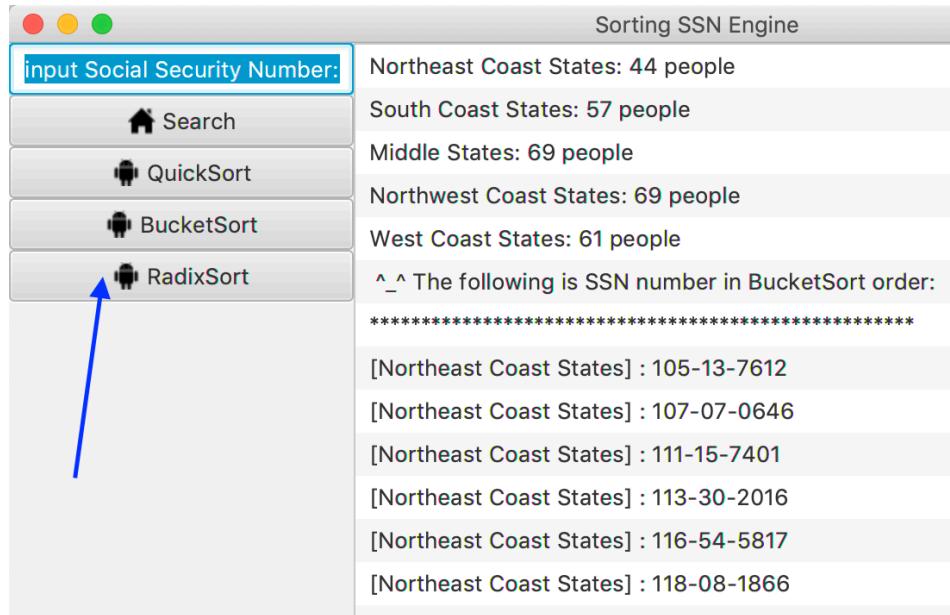
```
*****  
[Northeast Coast States] : 101-35-6786  
[Northeast Coast States] : 103-42-3589  
[Northeast Coast States] : 104-78-6544  
[Northeast Coast States] : 106-20-5505  
[Northeast Coast States] : 106-61-3712
```

Bottom Screenshot (BucketSort output):

Coast State	Population
Northeast Coast States	40 people
South Coast States	58 people
Middle States	72 people
Northwest Coast States	52 people
West Coast States	78 people

^_ The following is SSN number in BucketSort order:

```
*****  
[Northeast Coast States] : 103-08-5368  
[Northeast Coast States] : 104-90-5977  
[Northeast Coast States] : 105-43-3694  
[Northeast Coast States] : 114-29-5951  
[Northeast Coast States] : 122-36-0980  
[Northeast Coast States] : 122-03-7026
```



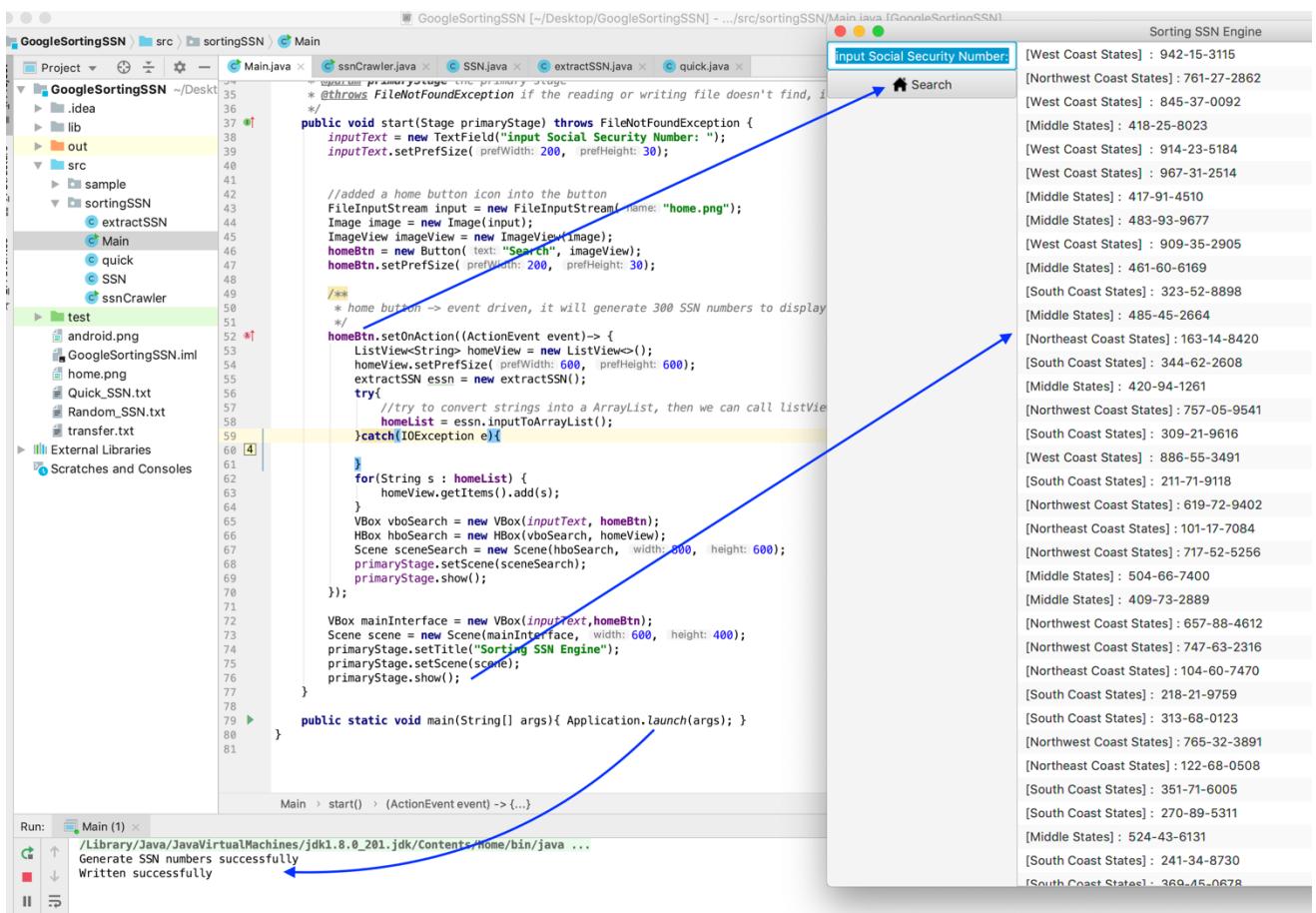
- Inside every method, there is **while()** clause, its purpose is to compare first 3 numbers:

```

•
if(temp <= 199){ //store every ssn number with the area name
    strValue = necs + str.substring(0, 3) + "-" + str.substring(3,5)+ "-" +
    str.substring(5);
}else if(temp >199 && temp <= 399){
    strValue = scs + str.substring(0, 3) + "-" + str.substring(3,5)+ "-" +
    str.substring(5);
}else if(temp >399 && temp <= 599){
    strValue = ms + str.substring(0, 3) + "-" + str.substring(3,5)+ "-" +
    str.substring(5);
}else if(temp >599 && temp <= 799){
    strValue = nwcs + str.substring(0, 3) + "-" + str.substring(3,5)+ "-" +
    str.substring(5);
}else{
    strValue = wcs + str.substring(0, 3) + "-" + str.substring(3,5)+ "-" +
    str.substring(5);
}
//write every ssn number into the Random_SSN.txt file
out.println(str.substring(0, 3) + "-" + str.substring(3,5)+ "-" +
    str.substring(5));

```

- We can distinguish the different area name:
[Northeast Coast States]
[South Coast States]
[Middle States]
[Northwest Coast States]
[West Coast States]
- `out.println()` will write all data into the .txt file



The screenshot shows a Java development environment with the following details:

- Project Structure:** GoogleSortingSSN /src/sortingSSN/Main.java
- Main.java Code:**

```

public void start(Stage primaryStage) throws FileNotFoundException {
    inputText = new TextField("input Social Security Number: ");
    inputText.setPrefSize(200, 30);

    //added a home button icon into the button
    FileInputStream input = new FileInputStream("home.png");
    Image image = new Image(input);
    ImageView imageView = new ImageView(image);
    homeBtn = new Button( text: "Search", imageView);
    homeBtn.setPrefSize(200, 30);

    /**
     * home button -> event driven, it will generate 300 SSN numbers to display
     */
    homeBtn.setOnAction(ActionEvent event-> {
        ListView<String> homeView = new ListView<>();
        homeView.setPrefSize(600, 600);
        extractSSN essn = new extractSSN();
        try{
            //try to convert strings into a ArrayList, then we can call listView
            homeList = essn.inputToArrayList();
        }catch(IOException e){

        }

        for(String s : homeList){
            homeView.getItems().add(s);
        }
        VBox vboSearch = new VBox(inputText, homeBtn);
        HBox hboSearch = new HBox(vboSearch, homeView);
        Scene sceneSearch = new Scene(hboSearch, width: 600, height: 600);
        primaryStage.setScene(sceneSearch);
        primaryStage.show();
    });

    VBox mainInterface = new VBox(inputText, homeBtn);
    Scene scene = new Scene(mainInterface, width: 600, height: 400);
    primaryStage.setTitle("Sorting SSN Engine");
    primaryStage.setScene(scene);
    primaryStage.show();
}

public static void main(String[] args){ Application.launch(args); }

```
- Terminal Output:**

```

Main > start() > (ActionEvent event) ->{...}
Generate SSN numbers successfully
Written successfully

```
- Output Window:** Shows a list of generated SSN numbers categorized by region.

Region	SSN Number
[West Coast States]	: 942-15-3115
[Northwest Coast States]	: 761-27-2862
[West Coast States]	: 845-37-0092
[Middle States]	: 418-25-8023
[West Coast States]	: 914-23-5184
[West Coast States]	: 967-31-2514
[Middle States]	: 417-91-4510
[Middle States]	: 483-93-9677
[West Coast States]	: 909-35-2905
[Middle States]	: 461-60-6169
[South Coast States]	: 323-52-8898
[Middle States]	: 485-45-2664
[Northeast Coast States]	: 163-14-8420
[South Coast States]	: 344-62-2608
[Middle States]	: 420-94-1261
[Northwest Coast States]	: 757-05-9541
[South Coast States]	: 309-21-9616
[West Coast States]	: 886-55-3491
[South Coast States]	: 211-71-9118
[Northwest Coast States]	: 619-72-9402
[Northeast Coast States]	: 101-17-7084
[Northwest Coast States]	: 717-52-5256
[Middle States]	: 504-66-7400
[Middle States]	: 409-73-2889
[Northwest Coast States]	: 657-88-4612
[Northwest Coast States]	: 747-63-2316
[Northeast Coast States]	: 104-60-7470
[South Coast States]	: 218-21-9759
[South Coast States]	: 313-68-0123
[Northwest Coast States]	: 765-32-3891
[Northwest Coast States]	: 122-68-0508
[South Coast States]	: 351-71-6005
[South Coast States]	: 270-89-5311
[Middle States]	: 524-43-6131
[South Coast States]	: 241-34-8730
[South Coast States]	: 369-15-0678

```

public ArrayList<Integer> quickSortToFile()
public ArrayList<String> quickSortToUI()

```

this two method's purpose are to write all data into .txt file and on the User Interface:

The screenshot shows two Java code files in an IDE:

- extractSSN.java** (left):


```

public ArrayList<Integer> quickToFile() throws FileNotFoundException{
    /* initialize every area people's numbers */
    ArrayList<Integer> tempList = new ArrayList<>();
    String inFilepath = "transfer.txt";
    File inputFile = new File(infilepath);
    Scanner in = new Scanner(inputFile);
    String outfileName = "Quick_SSN.txt";
    PrintWriter out = new PrintWriter(outfileName);

    while(in.hasNextInt()){
        int tempTotalNum = in.nextInt();
        tempList.add(tempTotalNum);
    }
    quick ql = new quick();
    ql.quickSort(tempList, 0, tempList.size()-1);
    for(int i=0; i<tempList.size(); i++){
        String str = "";
        str = str + tempList.get(i);
        out.println(str.substring(0, 3) + "-" + str.substring(3,5) + "-" + str.substring(5));
    }
    System.out.println("Written successfully");
    in.close();
    out.close();
    return tempList;
}

public ArrayList<String> quickToUI() throws FileNotFoundException{
    ArrayList<Integer> tempList = quickToFile();
    ncsPeople = 0;
    mcsPeople = 0;
    wcsPeople = 0;

    String strValue;
    for(int i=0; i<tempList.size(); i++){
        String str = "";
        str = str + tempList.get(i);
        String first = str.substring(0, 3);
        int temp = Integer.parseInt(first);
        if(temp < 199){
            ncsPeople++;
            strValue = ncs + str.substring(0, 3) + "-" + str.substring(3,5) + "-000";
        }else if((temp >199 && temp <= 399){
            scsPeople++;
            strValue = scs + str.substring(0, 3) + "-" + str.substring(3,5) + "-000";
        }else if((temp >399 && temp <= 599){
            mcsPeople++;
            strValue = mcs + str.substring(0, 3) + "-" + str.substring(3,5) + "-000";
        }else if((temp >599 && temp <= 799){
            wcsPeople++;
            strValue = wcs + str.substring(0, 3) + "-" + str.substring(3,5) + "-000";
        }
        quickList.add(strValue);
    }
}
            
```
- Main.java** (right):


```

/*
 * homeBtn.setAction(ActionEvent event)-> {
    ListView<String> homeView = new ListView<>();
    homeView.setPrefSize( preWidth: 600, prefHeight: 600 );
    extractSSN essn = new extractSSN();
    try{
        //try to convert strings into a ArrayList, then we can call listVi
        homeList = essn.inputToArrayList();
    }catch(IOException e){}

    for(String s : homeList) {
        homeView.getItems().add(s);
    }

    VBox vBoxSearch = new VBox(inputText, homeBtn, quickBtn);
    HBox hbSort = new HBox(vBoxSearch, homeView);
    Scene sceneSearch = new Scene(hbSort, width: 800, height: 600);
    primaryStage.setScene(sceneSearch);
    primaryStage.show();
});

quickBtn.setOnAction(ActionEvent event)->{
    ListView<String> quickView = new ListView<>();
    quickView.setPrefSize( preWidth: 600, prefHeight: 600 );
    extractSSN essn = new extractSSN();

    try{
        essn.quickToFile();
        quickList = essn.quickToUI();
    }catch(IOException e){}

    quickView.getItems().addAll("Northeast Coast States: " + essn.getNcsPeople());
    quickView.getItems().addAll("Middle States: " + essn.getMcsPeople());
    quickView.getItems().addAll("Northwest Coast States: " + essn.getWcsPeople());
    quickView.getItems().addAll("West Coast States: " + essn.getWcsPeople());
    quickView.getItems().add("^.^ The following is SSN number in QuickSort order:");
    quickView.getItems().addAll("*****");
    for(String s : quickList){
        quickView.getItems().add(s);
    }

    VBox vqSort = new VBox(inputText, homeBtn, quickView);
    HBox hbSort = new HBox(vqSort, quickView);
    Scene sceneQuick = new Scene(hbSort, width: 800, height: 600);
    primaryStage.setScene(sceneQuick);
    primaryStage.show();
});

VBox mainInterface = new VBox(inputText,homeBtn, quickBtn);
Scene scene = new Scene(mainInterface, width: 600, height: 400);
primaryStage.setTitle("Sorting SSN Engine");
primaryStage.setScene(scene);
primaryStage.show();
}

public static void main(String[] args){ Application.launch(args); }
            
```

Annotations in the screenshot highlight specific parts of the code:

- A blue arrow points from the line `quick ql = new quick();` in `extractSSN.java` to the declaration of `quick ql` in `Main.java`.
- A blue arrow points from the line `quick ql.quickSort(tempList, 0, tempList.size()-1);` in `extractSSN.java` to the `quickSort` method in `Main.java`.
- A blue arrow points from the line `System.out.println("Written successfully");` in `extractSSN.java` to the output window.
- A blue arrow points from the line `quickList.add(strValue);` in `extractSSN.java` to the `quickList` declaration in `Main.java`.
- A blue arrow points from the line `quickList = essn.quickToUI();` in `Main.java` to the `quickToUI` method in `extractSSN.java`.
- A blue arrow points from the line `quickView.getItems().addAll("Northeast Coast States: " + essn.getNcsPeople());` in `Main.java` to the `getNcsPeople` method in `extractSSN.java`.
- A blue arrow points from the line `quickView.getItems().addAll("Middle States: " + essn.getMcsPeople());` in `Main.java` to the `getMcsPeople` method in `extractSSN.java`.
- A blue arrow points from the line `quickView.getItems().addAll("Northwest Coast States: " + essn.getWcsPeople());` in `Main.java` to the `getWcsPeople` method in `extractSSN.java`.
- A blue arrow points from the line `quickView.getItems().addAll("West Coast States: " + essn.getWcsPeople());` in `Main.java` to the `getWcsPeople` method in `extractSSN.java`.
- A blue arrow points from the line `quickView.getItems().add("^.^ The following is SSN number in QuickSort order:");` in `Main.java` to the output window.
- A blue arrow points from the line `quickView.getItems().addAll("*****");` in `Main.java` to the output window.
- A blue arrow points from the line `for(String s : quickList){` in `Main.java` to the `quickList` declaration in `extractSSN.java`.
- A blue arrow points from the line `quickView.getItems().add(s);` in `Main.java` to the output window.
- A blue arrow points from the line `primaryStage.show();` in `Main.java` to the output window.

The output window displays the results of the sorting process:

```

Input Social Security Number
-----
Northeast Coast States: 39 people
South Coast States: 51 people
Middle States: 67 people
Northwest Coast States: 67 people
West Coast States: 76 people
^.^ The following is SSN number in QuickSort order:
*****
[Northeast Coast States]: 101-36-1326
[Northeast Coast States]: 102-10-7757
[Northeast Coast States]: 105-37-5820
[Northeast Coast States]: 109-01-8543
[Northeast Coast States]: 110-41-1019
[Northeast Coast States]: 116-49-7428
[Northeast Coast States]: 116-58-5511
[Northeast Coast States]: 118-70-8291
[Northeast Coast States]: 119-60-0491
[Northeast Coast States]: 122-08-3093
[Northeast Coast States]: 125-95-7913
[Northeast Coast States]: 128-12-8241
[Northeast Coast States]: 130-93-8784
[Northeast Coast States]: 132-62-2682
[Northeast Coast States]: 139-23-1948
[Northeast Coast States]: 140-19-7951
[Northeast Coast States]: 142-47-0921

```

```

public ArrayList<Integer> bucketSortToFile()
public ArrayList<String> bucketSortToUI()

```

the two method's purpose are to write all data into .txt file and on the User Interface:

The diagram illustrates the flow of data from Java code to a user interface and finally to a sorting table.

Java Code (Left):

```

quickView.getItems().add("South Coast States: " + quickView.getItems().size());
quickView.getItems().add("Middle States: " + essn.getItems().size());
quickView.getItems().add("Northwest Coast States: " + quickView.getItems().size());
quickView.getItems().add("West Coast States: " + essn.getItems().size());
quickView.getItems().add("The following is SS");
for(String s : quickList){
    quickView.getItems().add(s);
}
//ArrayList<String> quickView will add every record of s
VBox vgSort = new VBox(inputText, homeBtn, quickBtn);
HBox hboSort = new HBox(vgSort, quickView);
Scene sceneQuick = new Scene(hboSort, width: 800, height: 600);
primaryStage.setScene(sceneQuick);
primaryStage.show();
});

/**
 * bucketBtn this event that call the bucketSort() to sort the array
 */
bucketBtn.setOnAction(ActionEvent event) ->
{
    ListView<String> bucketView = new ListView<String>();
    bucketView.setPrefSize(prefWidth: 600, prefHeight: 60);
    extractSSN essn = new extractSSN();
    try{
        essn.bucketSortToFile();
        bucketlist = essn.bucketSortToUI();
    } catch (IOException e){ // if failed to read from file
    }
    // the area name and people's number will display
    bucketView.getItems().add("Northeast Coast States: " + bucketView.getItems().size());
    bucketView.getItems().add("South Coast States: " + quickView.getItems().size());
    bucketView.getItems().add("Middle States: " + essn.getItems().size());
    bucketView.getItems().add("West Coast States: " + essn.getItems().size());
    bucketView.getItems().add("The following is SS");
    for(String s : bucketlist){
        bucketView.getItems().add(s);
    }
    //ArrayList<String> bucketView will add every record of s
    VBox vgSort = new VBox(inputText, homeBtn, quickBtn);
    HBox hboSort = new HBox(vgSort, bucketView);
    Scene sceneBucket = new Scene(hboSort, width: 800, height: 600);
    primaryStage.setScene(sceneBucket);
    primaryStage.show();
};

/**
 * radixBtn this event that call the radixSort() to sort the array
 */

```

User Interface (Center):

A JavaFX application window titled "Sorting SSN Engine". It contains a search bar labeled "input Social Security Number:" and a list of buttons:

- Search
- QuickSort
- BucketSort (highlighted)
- RadixSort

Sorting Table (Right):

The table displays the following data:

Region	People
Northeast Coast States	27 people
South Coast States	76 people
Middle States	52 people
Northwest Coast States	67 people
West Coast States	78 people

Below the table, a note states: "The following is SSN number in BucketSort order: [list of SSN numbers]".

```

public ArrayList<Integer> radixSortToFile()
public ArrayList<String> radixSortToUI()

```

this two method's purpose are to write all data into .txt file and on the User Interface:

The screenshot shows the Java code for the `extractSSN.java` file and its execution output. The code implements a radix sort algorithm to sort SSNs and their corresponding area names. It includes methods for reading from a file, performing radix sort, and writing results to both a file and the user interface.

Code Snippet:

```

public ArrayList<Integer> radixSortToFile() throws FileNotFoundException{
    /* initialize every area people's number */
    ArrayList<Integer> tempList = new ArrayList<Integer>();
    ArrayList<Integer> afterRadixSort = new ArrayList<Integer>();

    String inFile = "transfer.txt";
    File inputFile = new File(inFile);
    Scanner in = new Scanner(inputFile);
    String outFileName = "Radix_SSN.txt";
    PrintWriter out = new PrintWriter(outFileName);

    while(in.hasNextInt()){
        int tempTotalNum = in.nextInt();
        tempList.add(tempTotalNum);
    }

    /* Now create a new radix object, then calling the quickSort to sort */
    radix rx = new radix();
    int [] iarr = rx.convertToArr(tempList); //convert the arrayList to a
    rx.radixSort(iarr, tempList.size()); // call radixSort to sort the temporary array
    for(int i : iarr){
        afterRadixSort.add(i);
    } //call the an arrayList afterRadixSort to add temporary array's in

    //write every ssn number into the Radix_SSN.txt file
    for(int i=0; < afterRadixSort.size(); i++){
        String str = "";
        str = str + afterRadixSort.get(i);
        out.println(str.substring(0, 3) + "-" + str.substring(3, 5) + "-" +
    }

    System.out.println("Written successfully");
    in.close();
    out.close();
    return afterRadixSort;
}

/*
 * A method using radixSort, then display all data on the User Interface
 * @return A arrayList with this format "xxx-xx-xxxx", when it will write
 * @throws FileNotFoundException if the System cannot find the .txt file,
 */
public ArrayList<String> radixSortToUI() throws FileNotFoundException{
    ArrayList<Integer> tempUI = radixSortToFile();
    int ncsPeople = 0;
    int scsPeople = 0;
    int nwcsPeople = 0;
    int wcsPeople = 0;

    String strValue;
    for(int i=0; < tempUI.size(); i++){
        String str = "";
        str = str + tempUI.get(i);
        String first = str.substring(0, 3);
        int temp = Integer.parseInt(first);
        if(temp <= 199){ //store every ssn number with the area name
            ncsPeople++;
            strValue = ncsPeople + str.substring(0, 3) + "-" + str.substring(3, 5) + "-" +
        }
    }
}

```

Execution Output:

	Output
input Social Security Number:	Northeast Coast States: 42 people South Coast States: 68 people Middle States: 61 people Northwest Coast States: 65 people West Coast States: 64 people
Search	^ The following is SSN number in Bucke
QuickSort	[Northeast Coast States]: 100-90-2598
BucketSort	[Northeast Coast States]: 103-04-8667
RadixSort	[Northeast Coast States]: 103-31-9047
	[Northeast Coast States]: 104-32-0608
	[Northeast Coast States]: 109-26-8978
	[Northeast Coast States]: 110-73-1221
	[Northeast Coast States]: 114-35-6008
	[Northeast Coast States]: 118-59-0740
	[Northeast Coast States]: 130-34-6245
	[Northeast Coast States]: 131-54-9536
	[Northeast Coast States]: 132-87-5374
	[Northeast Coast States]: 136-41-4428
	[Northeast Coast States]: 139-92-1960
	[Northeast Coast States]: 140-22-9492
	[Northeast Coast States]: 140-52-4185
	[Northeast Coast States]: 145-41-0355
	[Northeast Coast States]: 146-44-5404

GoogleSortingSSN [~/Desktop/GoogleSortingSSN] - .../Random_SSN.txt [G]

Project Z: Structure 1: Favorites

GoogleSortingSSN ~/Desktop

- lib
- out
- src
 - sample
 - sortingSSN
 - extractSSN
 - Main
 - quick
 - SSN
 - ssnCrawler
 - test
 - transfer.txt
- External Libraries
- Scratches and Consoles

Random_SSN.txt x

Input Social Security Number: 106-55-3707

Search QuickSort

Sorting SSN Engine

1	106-55-3707	[Northeast Coast States] : 106-55-3707
2	589-23-3848	[Middle States] : 589-23-3848
3	951-71-8652	[West Coast States] : 951-71-8652
4	165-54-9928	[Northeast Coast States] : 165-54-9928
5	738-19-1024	[Northwest Coast States] : 738-19-1024
6	263-40-5989	[South Coast States] : 263-40-5989
7	999-25-1895	[West Coast States] : 999-25-1895
8	979-60-3792	[Northeast Coast States] : 979-60-3792
9	172-63-8210	[Northwest Coast States] : 172-63-8210
10	701-07-9470	[South Coast States] : 701-07-9470
11	372-31-1968	[Northeast Coast States] : 372-31-1968
12	139-02-9661	[South Coast States] : 139-02-9661
13	387-92-0477	[West Coast States] : 387-92-0477
14	796-24-4900	[Northeast Coast States] : 796-24-4900
15	511-43-2340	[Northwest Coast States] : 511-43-2340
16	140-38-5955	[South Coast States] : 140-38-5955
17	908-81-0976	[West Coast States] : 908-81-0976
18	516-03-3873	[Middle States] : 516-03-3873
19	552-22-4192	[Northeast Coast States] : 552-22-4192
20	733-79-6189	[South Coast States] : 733-79-6189
21	159-71-2690	[Northwest Coast States] : 159-71-2690
22	487-07-6082	[Middle States] : 487-07-6082
23	570-07-0546	[Northeast Coast States] : 570-07-0546
24	930-64-9755	[South Coast States] : 930-64-9755
25	390-83-2901	[West Coast States] : 390-83-2901
26	172-38-3291	[Northeast Coast States] : 172-38-3291
27	434-24-7719	[Middle States] : 434-24-7719
28	373-53-6321	[Northeast Coast States] : 373-53-6321
29	666-59-1716	[South Coast States] : 666-59-1716
30	843-12-1358	[Middle States] : 843-12-1358
31	143-04-1857	[Northeast Coast States] : 143-04-1857
32	890-78-4768	[South Coast States] : 890-78-4768
33	533-05-4987	[Northwest Coast States] : 533-05-4987
34	981-71-1038	[Northeast Coast States] : 981-71-1038
35	886-42-3887	[South Coast States] : 886-42-3887
36	145-67-7650	[Middle States] : 145-67-7650
37	458-69-2280	[Northeast Coast States] : 458-69-2280
38	931-37-0811	[South Coast States] : 931-37-0811
39	426-78-4227	[Middle States] : 426-78-4227
40	710-66-3760	[Northeast Coast States] : 710-66-3760
41	490-70-9648	[South Coast States] : 490-70-9648
42	505-18-4492	[Middle States] : 505-18-4492
43	590-47-1841	[Northeast Coast States] : 590-47-1841
44	880-69-6482	[South Coast States] : 880-69-6482
45	341-78-2540	[Middle States] : 341-78-2540
46	402-14-8718	[Northeast Coast States] : 402-14-8718
47	356-86-7041	[South Coast States] : 356-86-7041
48	664-55-8377	[Middle States] : 664-55-8377
49	936-34-9887	[Northeast Coast States] : 936-34-9887

GoogleSortingSSN [~/Desktop/GoogleSortingSSN] - .../Quick_SSN.txt [GoogleSortingSSN]

txt

Quick_SSN.txt x Random_SSN.txt x

Input Social Security Number: 104-07-1255

Search QuickSort

Sorting SSN Engine

1	104-07-1255	Northeast Coast States: 37 people
2	105-68-7534	South Coast States: 55 people
3	106-02-9874	Middle States: 70 people
4	106-23-1070	Northwest Coast States: 65 people
5	106-55-3707	West Coast States: 73 people
6	110-36-4766	
7	112-61-3042	
8	121-37-9722	
9	132-43-2175	
10	133-51-6130	
11	136-42-7333	
12	138-38-7225	
13	139-02-9661	
14	140-38-5955	
15	143-04-1857	
16	143-29-2747	
17	144-00-1917	
18	145-67-7650	
19	150-22-2495	
20	152-20-6916	
21	159-71-2690	
22	163-97-4394	
23	165-54-9928	
24	171-49-8730	
25	172-38-3291	
26	172-63-8210	
27	175-71-3015	
28	176-59-0963	
29	176-69-4934	
30	176-78-9668	
31	182-44-7612	
32	185-09-7796	
33	191-56-6630	
34	198-06-9540	
35	199-07-6171	
36	199-87-1929	
37	199-88-1864	
38	203-48-4754	
39	203-77-3678	
40	204-42-3321	

- Now I will explain how to implement **bucketSort()** method step by step:

```
/*
 * Convert a Integer arrayList to a double array[]
 * @param intarr the given arrayList with Integer elements
 * @return a double array[]
 */
public double[] convertArr(ArrayList<Integer> intarr){
    double [] darr = new double[intarr.size()];
    for(int i=0; i<intarr.size(); i++){
        darr[i] = (intarr.get(i))/ 1000000000.0;
    }
    return darr;
}
```

- Firstly, trying to **convert** the **ArrayList<Integer>** to the **double [] array**, this step's purpose is to satisfies $0 \leq A[i] < 1$
- Create a **double[] array** to receive the converted data
- Using original **arrayList**'s **size()**
 - For here, the original **arrayList**'s **size()** is **300**
 - So we set **n = size() = 300**
- Every int element / 1000000000.0**; this step's purpose is to make **every new element ≤ 1** and new element is a **double**
- Inside **buckSort(double [] arr)** method, we call a


```
ArrayList<Double> [] bucketList = new ArrayList[n];
```

bucketList is an **array**, which every **array[i]** contains a **arrayList**
 - Array[i].arrayList<Double>**

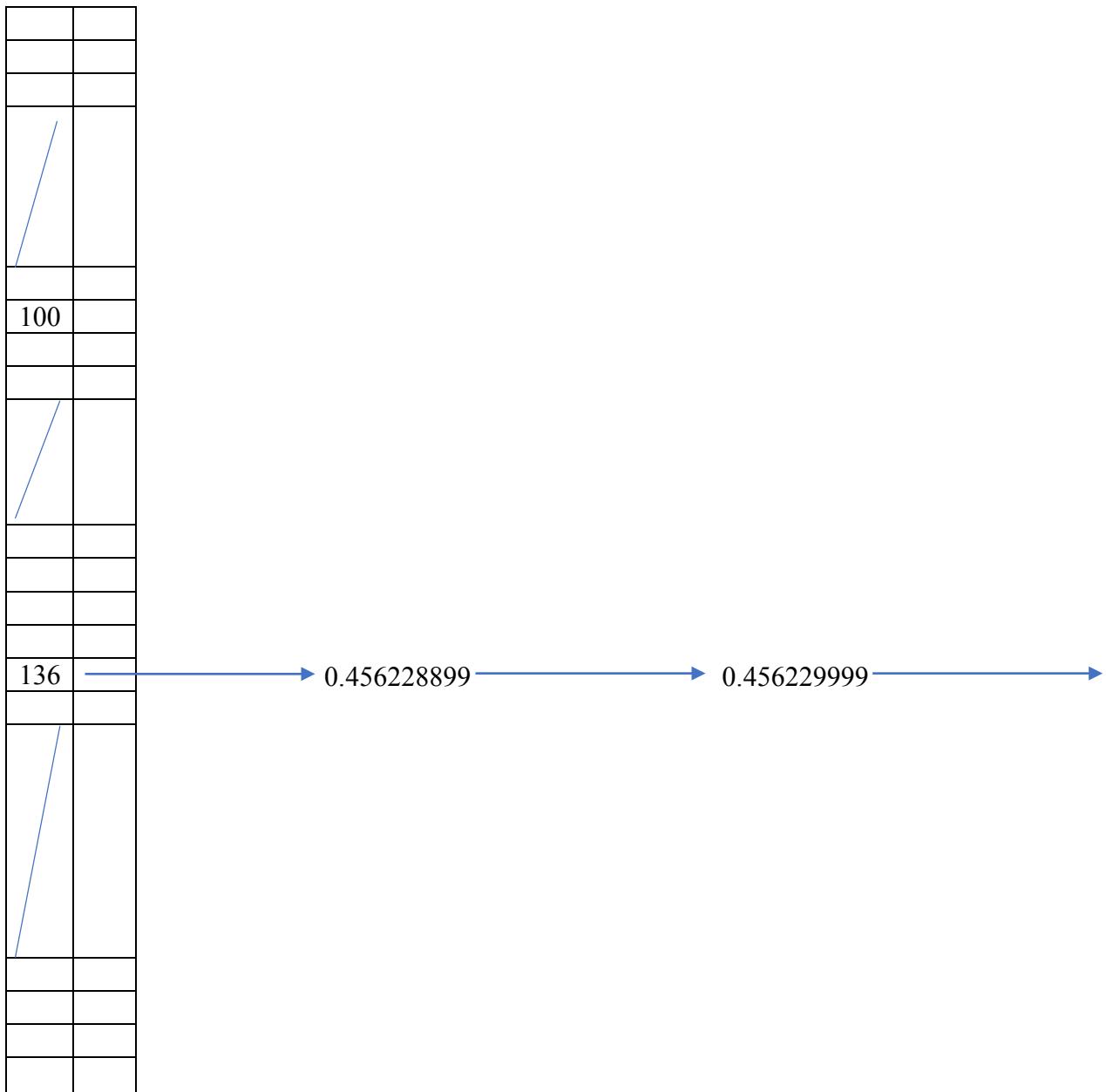
```
/*
 * insert every double element into the appropriate index of bucketList
 */
for(int i=0; i<n; i++){
    int bucketIndex = (int)(arr[i] * n);
    bucketList[bucketIndex].add(arr[i]);
}
```

- Set every **bucketIndex = every double element * 300**
- For example: **0.456228899 * 300 = 136.8686697**
- Convert: **(int) 136.8686697 => 136**
- Finally, **bucketIndex = 136**
- Calling the **array[] bucketList[136].add(double element);**

```
/*
 * calling the insertSort to sort every arrayList
 */
for(int i=0; i<n; i++){
    insertSort(bucketList[i]);
}
```

- loop a size = 300, everytime calling the **insertSort()** to help sort every bucket's double element:
 - We have known this step is stable.
 - The **insertSort()** will compare the every double element in this **bucketList[bucketIndex]**

7. the following is the graphically explanation:



```
/* concatenate all buckets */
int index = 0;
for(int i=0; i<n; i++){
    for(int j=0; j<bucketList[i].size(); j++){
        arr[index++] = bucketList[i].get(j);
    }
}
```

Finally, use **two for** loop to **concatenate** all buckets, this step is like 2D array operation.

- Done!

- Now I will explain how to implement **radixSort()** method step by step:

```
/*
 * @param arr the required arrayList to convert to a integer array
 * @return a new integer array
 */
public int[] convertToArr(ArrayList<Integer> arr){
    int [] iarr = new int[arr.size()];
    for(int i=0; i<arr.size(); i++){
        iarr[i] = arr.get(i);
    }
    return iarr;
}
```

⇒ Convert **ArrayList<Integer>** to **int[]**, the reason why I convert the arrayList because arrayList is difficult to **manipulate**.

```
/*
 * Get the maximum from this array
 * @param arr the required array will be sorted
 * @param n the length of this array
 * @return a maximum
 */
public int getMax(int arr[], int n){
    int max = arr[0];
    for(int i=1; i<n; i++){
        if(arr[i] > max){
            max = arr[i];
        }
    }
    return max;
}
```

⇒ Find the max length, this step is to find the max places
 ⇒ For example 10's => 100's => 1000's => 10000's =>

```
/*
 * The helper method to help radixSort to sort the SSN number
 * @param arr the required array
 * @param n the array's lenght
 * @param places the every digit place
 */
public void countSort(int arr[], int n, int places){
    int B[] = new int[n];
    int count[] = new int[10];
    /* initialize every element to be 0 */
    Arrays.fill(count, 0);

    /* count every digit from 0-9 that the number of times of appearance */
    for(int i=0; i<n; i++){
        count[(arr[i]/places)%10]++;
    }

    //updated the array C[]
    for(int j=1; j<10; j++){
        count[j] = count[j] + count[j-1];
    }
}
```

```

//input the sorted number into the appropriate index
for(int j=n-1; j>=0; j--){
    B[count[(arr[j]/places)%10]-1] = arr[j];
    count[ (arr[j]/places)%10 ]--;
}

//copy the new array to the original array
for(int i=0; i<n; i++){
    arr[i] = B[i];
}
}

```

In the **countSort()** method, count every digit from 0-9 that the times of this number's appearance:

- $\text{Arr}[i] / \text{places} \Rightarrow$ find the **nth** place
- $\%10 \Rightarrow$ find one of the 0-9 digit
- finally, update the times of this number appearing
- Put the sorted number into the appropriate index, then update the **count array**
- **Note:** copy the new array to the original array.

- The following table will show the theory to implement **radixSort()** :

Index	Original	One's place	1000's place	100xxxxxx's place
1	751345555	592679900
2	543456789	239671111		
3	689234545	961567711		
4	264245566	799990011		
5	837349988	393778811		
6	160783434	937882222		
7	239671111	177789732		
8	937882222	236567823		
9	592679900	160783434		
10	961567711	885993344		
11	799990011	412356784		
12	885993344	751345555		
13	393778811	689234545		
14	177789732	264245566		
15	236567823	747234567		
16	364117799	837349988		
17	412356784	543456789		
18	747234567	364117799		

5. Self-testing

- Testing **ssnCrawler.generate()** :
Generating 300 SSN numbers without sorting

The screenshot shows a Java development environment with two windows:

- ssnCrawler.java** (Left Window):


```
package sortingSSN;

import java.io.File;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.Random;

/**
 * A class ssnCrawler can generate random 300 SSN number
 */
public class ssnCrawler {

    /**
     * This method to help user generate 300 SSN number
     * @throws FileNotFoundException if the System can not find the .txt file, will
     */
    public static void generate() throws FileNotFoundException {
        String outFileName = "transfer.txt";
        File outFile = new File(outFileName);
        PrintWriter output = new PrintWriter(outFile);
        /**
         * generating 300 Random SSN numbers
         */
        for(int index=1; index<301; index++){
            int randomSSN = 10000000 + new Random().nextInt( bound: 900000000 );
            output.println(randomSSN);
        }
        output.close();
        System.out.println("Generate SSN numbers successfully");
    }

    /**
     * Testing the generate() method, if it generating successfully
     * @param args
     */
    public static void main(String[] args) {
        try{
            generate();
        }catch (IOException e){
        }
    }
}
```
- Random_SSN.txt** (Right Window):

237	938-57-7079
238	859-04-1754
239	599-60-5004
240	976-75-0438
241	302-61-7868
242	816-56-3204
243	849-65-4423
244	606-77-9905
245	971-72-9116
246	222-06-9707
247	338-12-9824
248	160-70-5682
249	289-91-5907
250	735-46-4344
251	942-45-9223
252	257-52-8292
253	284-90-7706
254	837-27-3258
255	709-77-0464
256	572-23-6515
257	195-47-9613
258	486-02-9942
259	783-55-9297
260	374-60-7362
261	452-10-1497
262	566-79-3879
263	759-87-5134
264	748-15-1129
265	983-85-0466
266	889-11-8511
267	523-25-8603
268	460-62-1027
269	540-20-0845
270	552-73-2718
271	614-70-3834
272	930-90-9109
273	912-55-7192
274	212-74-5070
275	136-41-4428
276	659-03-4190
277	140-22-9492
278	103-04-8667
279	295-59-6045
280	905-71-8905
281	566-94-8426
282	378-36-2492
283	243-23-8584
284	351-98-4392
285	786-54-4703
286	747-99-9402
287	363-02-7620
288	768-77-5189
289	358-89-4408
290	769-78-1983
291	566-16-4946
292	257-90-0315
293	562-76-1178
294	190-47-6199
295	384-33-2848
296	145-41-0355
297	753-23-4302
298	227-54-9669
299	752-75-1980
300	192-36-3286

Red arrows point from the `generate()` method in `ssnCrawler.java` to the first few lines of the `Random_SSN.txt` file, indicating that the generated SSNs are being written to this file.

- Testing Main class

1. testing four button icon:

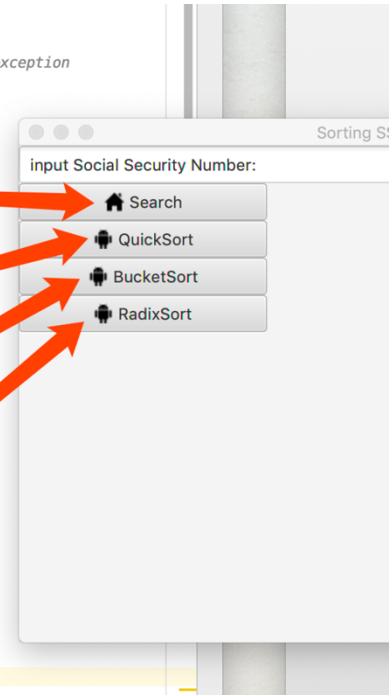
```
/*
 * @param primaryStage the primary stage
 * @throws FileNotFoundException if the reading or writing file doesn't find, it will throw the exception
 */
public void start(Stage primaryStage) throws FileNotFoundException {
    inputText = new TextField("input Social Security Number: ");
    inputText.setPrefSize( 200, prefHeight: 30);

    //added a home button icon into the button
    FileInputStream input = new FileInputStream( name: "home.png");
    Image image = new Image(input);
    ImageView imageView = new ImageView(image);
    homeBtn = new Button( text: "Search", imageView);
    homeBtn.setPrefSize( 200, prefHeight: 30);

    //added a quickSort button icon into the button
    FileInputStream input1 = new FileInputStream( name: "android.png");
    Image image1 = new Image(input1);
    ImageView imageView1 = new ImageView(image1);
    quickBtn = new Button( text: "QuickSort", imageView1);
    quickBtn.setPrefSize( 200, prefHeight: 30);

    //added a bucketSort button icon into the button
    FileInputStream input2 = new FileInputStream( name: "android.png");
    Image image2 = new Image(input2);
    ImageView imageView2 = new ImageView(image2);
    bucketBtn = new Button( text: "BucketSort", imageView2);
    bucketBtn.setPrefSize( 200, prefHeight: 30);

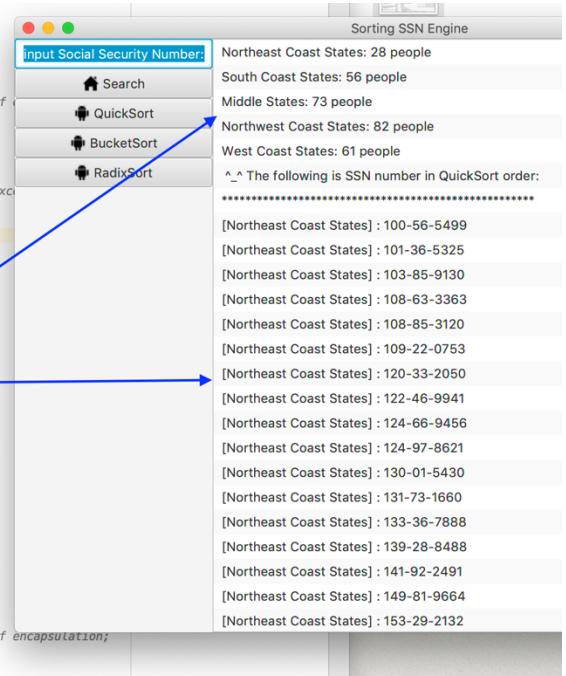
    //added a radixSort button icon into the button
    FileInputStream input3 = new FileInputStream( name: "android.png");
    Image image3 = new Image(input3);
    ImageView imageView3 = new ImageView(image3);
    radixBtn = new Button( text: "RadixSort", imageView3);
    radixBtn.setPrefSize( 200, prefHeight: 30);
}
```



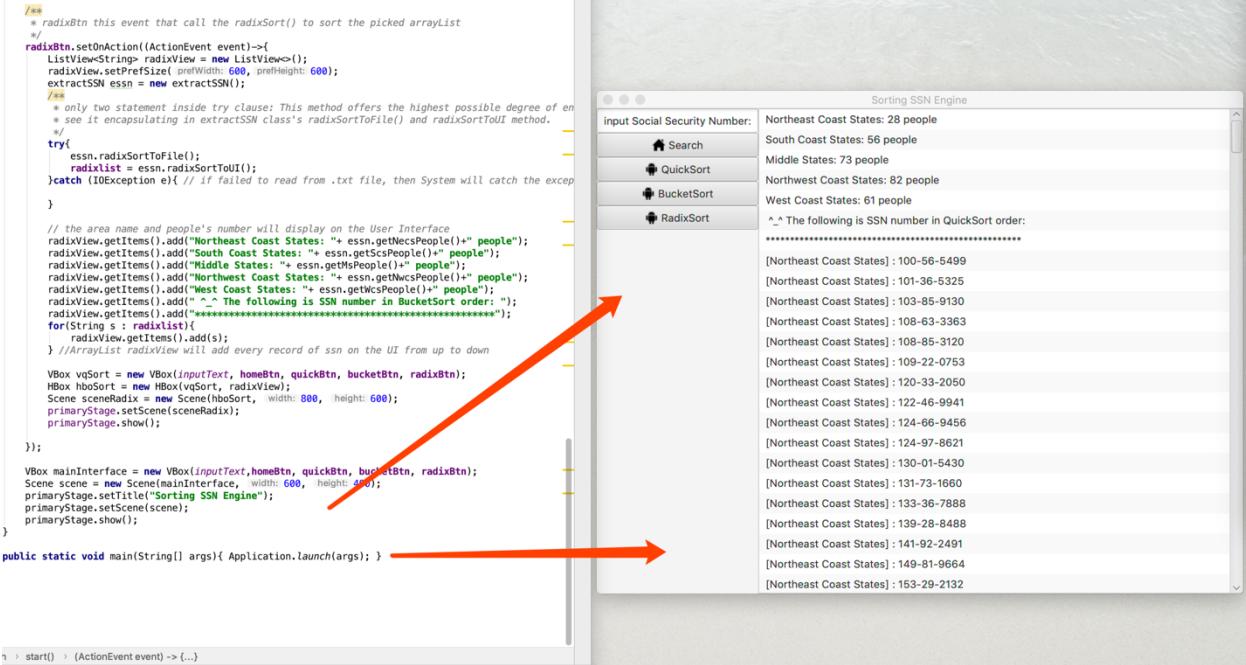
2. testing ListView() , which will drive the button event to display on the User Interface:

```
quickBtn.setOnAction((ActionEvent event)->{
    ListView<String> quickView = new ListView<>();
    quickView.setPrefSize( prefWidth: 600, prefHeight: 600);
    extractSSN essn = new extractSSN();
    /*
     * only two statement inside try clause: This method offers the highest possible degree of
     * encapsulating in extractSSN class's quickSortToFile() and quickSortToUI method.
     */
    try{
        essn.quickSortToFile();
        quickList = essn.quickSortToUI();
    }catch (IOException e){ // if failed to read from .txt file, then System will catch the exception
    }
    // the area name and people's number will display on the User Interface
    quickView.getItems().add("Northeast Coast States: "+ essn.getNecsPeople()+" people");
    quickView.getItems().add("South Coast States: "+ essn.getScsPeople()+" people");
    quickView.getItems().add("Middle States: "+ essn.getMsPeople()+" people");
    quickView.getItems().add("Northwest Coast States: "+ essn.getWcsPeople()+" people");
    quickView.getItems().add("West Coast States: "+ essn.getWcsPeople()+" people");
    quickView.getItems().add("~~~ The following is SSN number in QuickSort order: ");
    quickView.getItems().add("*****");
    for(String s : quickList){
        quickView.getItems().add(s);
    }
    //ArrayList quickView will add every record of ssn on the UI from up to down
    VBox vqSort = new VBox(inputText, homeBtn, quickBtn, bucketBtn, radixBtn);
    HBox hboSort = new HBox(vqSort, quickView);
    Scene sceneQuick = new Scene(hboSort, width: 800, height: 600);
    primaryStage.setScene(sceneQuick);
    primaryStage.show();
});

/*
 * bucketBtn this event that call the bucketSort() to sort the picked arrayList
 */
bucketBtn.setOnAction((ActionEvent event)->{
    ListView<String> bucketView = new ListView<>();
    bucketView.setPrefSize( prefWidth: 600, prefHeight: 600);
    extractSSN essn = new extractSSN();
    /*
     * only two statement inside try clause: This method offers the highest possible degree of encapsulation;
     * see it encapsulating in extractSSN class's bucketSortToFile() and bucketToUI method.
     */
    true;
})
```



- Testing `primaryStage.setTitle()` `primaryStage.setScene()``primaryStage.show()`
 Testing `Application.launch(args)`, the previous 4 method will construct a **primary stage** , you will see a **framework with title, button, icon, and data** :



The screenshot shows a Java code editor on the left and a running Java application window on the right. The code is for a radix sort application. It includes imports for `java.io`, `javafx.application`, `javafx.scene`, `javafx.scene.control`, `javafx.scene.layout`, and `javafx.stage`. The main logic is in the `main` method, which creates a `VBox` for input and buttons, and a `Scene` for the UI. The `Scene` contains a `Search` button, `QuickSort`, `BucketSort`, and `RadixSort` buttons. Below these are four text fields for region names and their people counts. A large list box displays sorted Social Security Numbers (SSNs) from the Northeast Coast States.

```

/*
 * radixBtn this event that call the radixSort() to sort the picked arrayList
 */
radixBtn.setOnAction(ActionEvent event) -> {
    ListView<String> radixView = new ListView<String>();
    radixView.setPrefSize( prefWidth: 600, prefHeight: 600);
    extractSSN essn = new extractSSN();
    radixView.getItems().addAll(extractSSN.getNecsPeople());
    radixView.getItems().addAll(extractSSN.getSoCoPeople());
    radixView.getItems().addAll(extractSSN.getMidPeople());
    radixView.getItems().addAll(extractSSN.getNWCoPeople());
    radixView.getItems().addAll(extractSSN.getWestCoastPeople());
    radixView.getItems().addAll(extractSSN.getBucketSortOrder());
    radixView.getItems().addAll(extractSSN.getRadixSortOrder());
    radixView.getItems().addAll(extractSSN.getRadixSortToFile());
    radixView.getItems().addAll(extractSSN.getRadixSortToUI());
}
try {
    essn.radixSortToFile();
    radixList = essn.radixSortToUI();
} catch (IOException e) { // if failed to read from .txt file, then System will catch the exception
}

// the area name and people's number will display on the User Interface
radixView.getItems().addAll("Northeast Coast States: " + essn.getNecsPeople());
radixView.getItems().addAll("South Coast States: " + essn.getSoCoPeople());
radixView.getItems().addAll("Middle States: " + essn.getMidPeople());
radixView.getItems().addAll("Northwest Coast States: " + essn.getNWCoPeople());
radixView.getItems().addAll("West Coast States: " + essn.getWestCoastPeople());
radixView.getItems().addAll("-----");
for(String s : radixList){
    radixView.getItems().add(s);
}
//ArrayList radixView will add every record of ssn on the UI from up to down

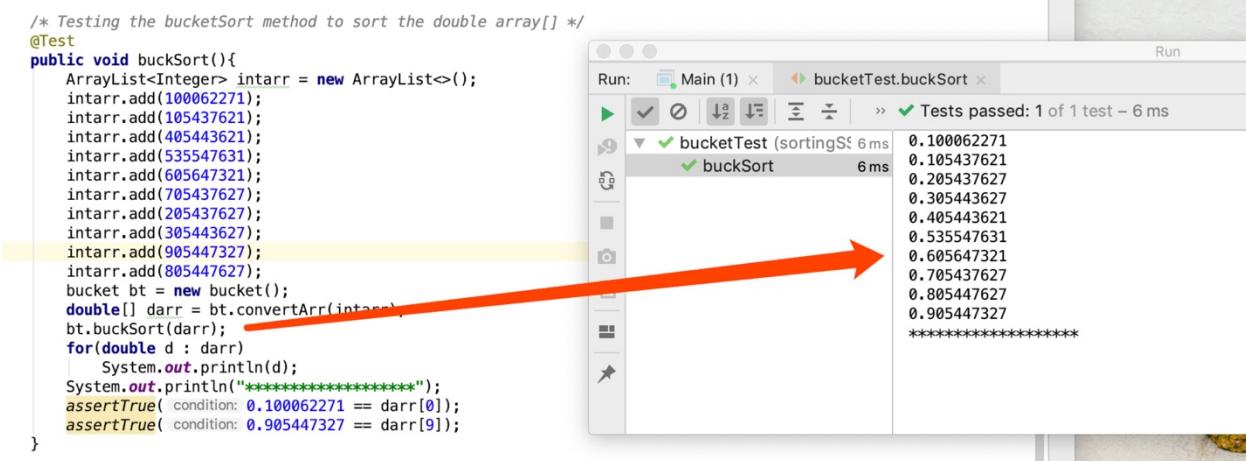
VBox vgsort = new VBox(inputText, homeBtn, quickBtn, bucketBtn, radixBtn);
Scene sceneRadix = new Scene(vgsort, width: 600, height: 600);
primaryStage.setScene(sceneRadix);
primaryStage.show();
}

VBox mainInterface = new VBox(inputText, homeBtn, quickBtn, bucketBtn, radixBtn);
Scene scene = new Scene(mainInterface, width: 600, height: 600);
primaryStage.setTitle("Sorting SSN Engine");
primaryStage.setScene(scene);
primaryStage.show();
}

public static void main(String[] args){ Application.launch(args); }

```

- Testing **bucketSort()** method:



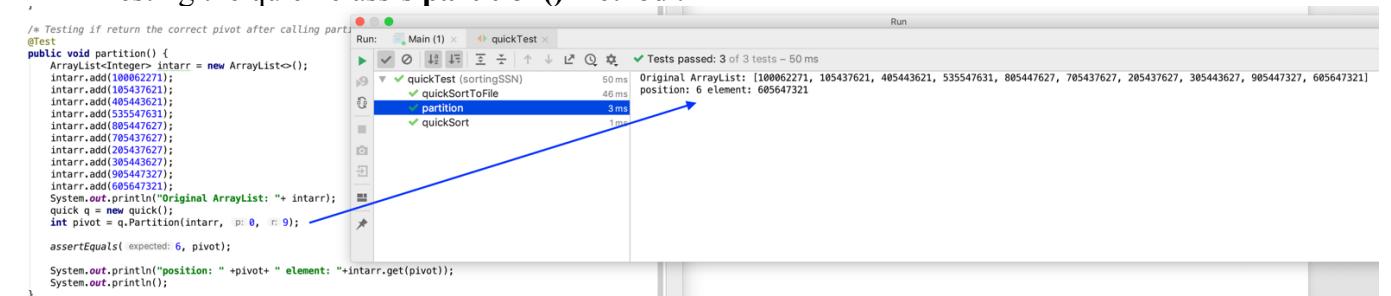
The screenshot shows a Java code editor on the left and a running Java application window on the right. The code is for a bucket sort test. It includes imports for `java.util`, `java.util.ArrayList`, `java.util.List`, and `java.util.stream`. The `bucketSort` method creates an array of integers, initializes a bucket, converts it to a double array, and then sorts it using the `buckSort` method. The sorted array is then printed to the console. The Java application window shows the sorted array results in the terminal.

```

/* Testing the bucketSort method to sort the double array[] */
@Test
public void buckSort(){
    ArrayList<Integer> intarr = new ArrayList<Integer>();
    intarr.add(100062271);
    intarr.add(105437621);
    intarr.add(405443621);
    intarr.add(535547631);
    intarr.add(605647321);
    intarr.add(705437627);
    intarr.add(205437627);
    intarr.add(305443627);
    intarr.add(905447327);
    intarr.add(805447627);
    bucket bt = new bucket();
    double[] darr = bt.convertArr(intarr);
    bt.buckSort(darr);
    for(double d : darr)
        System.out.println(d);
    System.out.println("*****");
    assertTrue( condition: 0.100062271 == darr[0] );
    assertTrue( condition: 0.905447327 == darr[9] );
}

```

- Testing the quick class's `partition()` method :



The screenshot shows a Java code editor on the left and a running Java application window on the right. The code is for a quick sort partition test. It includes imports for `java.util`, `java.util.ArrayList`, `java.util.List`, and `java.util.stream`. The `partition` method is tested with an array of integers. The Java application window shows the results of the partition test, including the original array list, pivot position, and the partitioned array.

```

/* Testing if return the correct pivot after calling partition */
@Test
public void partition() {
    ArrayList<Integer> intarr = new ArrayList<Integer>();
    intarr.add(100062271);
    intarr.add(105437621);
    intarr.add(405443621);
    intarr.add(535547631);
    intarr.add(605647321);
    intarr.add(705437627);
    intarr.add(205437627);
    intarr.add(305443627);
    intarr.add(905447327);
    intarr.add(805447627);
    System.out.println("Original ArrayList: " + intarr);
    quick q = new quick();
    int pivot = q.partition(intarr, p: 0, r: 9);
    assertEquals( expected: 6, pivot);
    System.out.println("position: " + pivot+ " element: "+intarr.get(pivot));
    System.out.println();
}

```

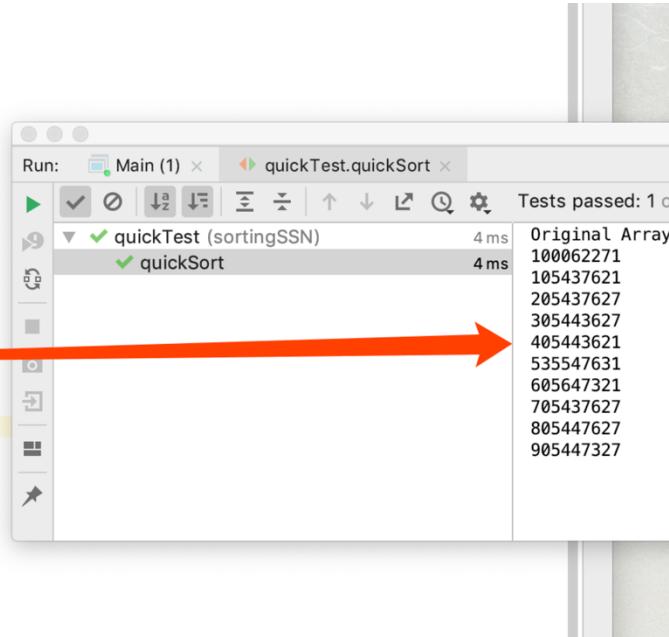
- Testing **quickSort()** method:

```

public void quickSort() {
    ArrayList<Integer> intarr = new ArrayList<>();
    intarr.add(100062271);
    intarr.add(105437621);
    intarr.add(405443621);
    intarr.add(535547631);
    intarr.add(605647321);
    intarr.add(705437627);
    intarr.add(205437627);
    intarr.add(305443627);
    intarr.add(905447327);
    intarr.add(805447627);
    System.out.println("Original ArrayList: " + intarr);
    ArrayList<Integer> temp = new ArrayList<>(intarr);
    Collections.sort(temp);
    quick q = new quick();
    q.quickSort(intarr, p: 0, r: 9);
    for(int i=0; i<intarr.size(); i++){
        System.out.println(intarr.get(i));
    }
    int min = intarr.get(0);
    int max = intarr.get(intarr.size()-1);

    assertEquals( expected: 10, intarr.size());
    assertEquals( expected: 100062271, min);
    assertEquals( expected: 905447327, max);
    assertEquals(temp, intarr);
    System.out.println();
}

```

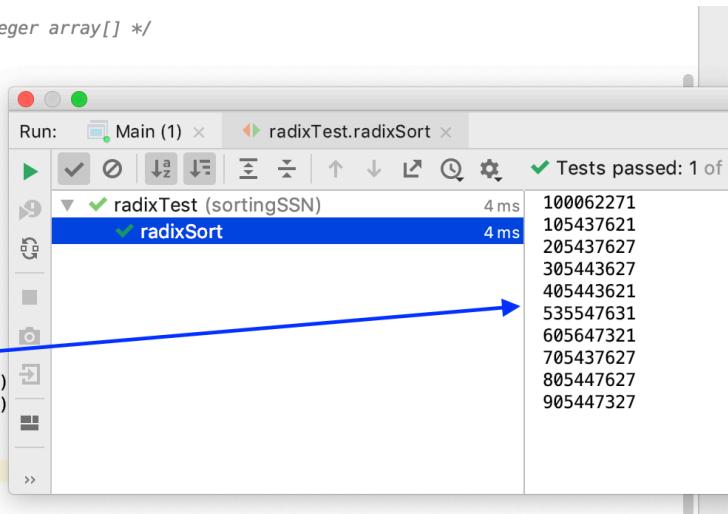


- Testing **radixSort()** method:

```

/* Testing radixSort method if sorting the integer array[] */
@Test
public void radixSort() {
    int [] intarr = new int [10];
    intarr[0] = 100062271;
    intarr[1] = 105437621;
    intarr[2] = 405443621;
    intarr[3] = 535547631;
    intarr[4] = 605647321;
    intarr[5] = 705437627;
    intarr[6] = 205437627;
    intarr[7] = 305443627;
    intarr[8] = 905447327;
    intarr[9] = 805447627;
    radix r = new radix();
    r.radixsort(intarr, n: 10);
    assertTrue( condition: 100062271 == intarr[0]);
    assertTrue( condition: 905447327 == intarr[9]);
    for(int i=0; i<intarr.length; i++){
        System.out.println(intarr[i]);
    }
}

```



To summary: the **remaining Running time** and some **detail**, I will illustrate in **Part 8: Running Time Big-O (Junit -Test)**

6. Problems encountered during implementation

- Since I design my program can do two things, I found that it is difficult to display all sorted data on the **UI** and **.txt file** at the **same time**.
- Originally, for example, I **combined AfterQuickSortToFile()** and **AfterQuickSortToUserInterface()** this two methods into one method:
 1. problem_1: if I did like this, it will increase its **running time** because I should use some **for loop**, and inside **for()** that I need to compare **300 SSN number**, whatever what functioning I want to implement, **it always occur at the same time**.
 2. problem_2: it is very hard to manipulate these data structures, since some **bucketSort radixSort** which I **convert** **arrayList** to **Array**; then after did this step, I convert the **Array** to **arrayList**.

This is a most troublesome case: you should care the data structure that you using and you should **focus on** which functioning that you want to implement: **on UI** and into **.txt file**.

- Look this following piece of codes:

```
/*
 * public int[] convertToArr(ArrayList<Integer> arr){
 *     int [] iarr = new int[arr.size()];
 *     for(int i=0; i<arr.size(); i++){
 *         iarr[i] = arr.get(i);
 *     }
 *     return iarr;
 * }
 */
/* Get the maximum from this array
 * @param arr the required array will be sorted
 * @param n the length of this array
 * @return a maximum
 */
public int getMax(int arr[], int n){
    int max = arr[0];
    for(int i=1; i<n; i++){
        if(arr[i] > max){
            max = arr[i];
        }
    }
    return max;
}

/*
 * The helper method to help radixSort to sort the SSN number
 * @param arr the required array
 * @param n the array's lenght
 * @param places the every digit place
 */
public void countSort(int arr[], int n, int places){
    int B[] = new int[n];
    int count[] = new int[10];
    /* initialize every element to be 0 */
    Arrays.fill(count, val: 0);

    /* count every digit from 0-9 that the number of times of appearance */
    for(int i=0; i<n; i++){
        count[ (arr[i]/places)%10 ]++;
    }

    //updated the array C[]
    for(int j=1; j<10; j++){
        count[j] = count[j] + count[j-1];
    }

    //input the sorted number into the appropriate index
    for(int j=n-1; j>=0; j--){
        B[count[(arr[j]/places)%10]-1] = arr[j];
        count[ (arr[j]/places)%10 ]--;
    }

    //copy the new array to the original array
    for(int i=0; i<n; i++){
        arr[i] = B[i];
    }
}
```

- And this following codes also illustrate the problem that I will encounter:

```

quickBtn.setOnAction((ActionEvent event)->{
    ListView<String> quickView = new ListView<>();
    quickView.setPrefSize(600,600);
    extractSSN essn = new extractSSN();

    try{
        essn.quickSortToFile();
        quicklist = essn.quickSortToUI();
    }catch (IOException e){ // if failed to read from .txt file, then System will
    catch the exception
    }

quickBtn.setOnAction((ActionEvent event)->{
    ListView<String> quickView = new ListView<>();
    quickView.setPrefSize(600,600);
    extractSSN essn = new extractSSN();

    try{
        essn.bucketSortToFile();
        bucketlist = essn.bucketSortToUI();
    }catch (IOException e){ // if failed to read from .txt file, then System will
    catch the exception
    }

bucketBtn.setOnAction((ActionEvent event)->{
    ListView<String> bucketView = new ListView<>();
    bucketView.setPrefSize(600,600);
    extractSSN essn = new extractSSN();

    try{
        essn.radixSortToFile();
        radixlist = essn.radixSortToUI();
    }catch (IOException e){ // if failed to read from .txt file, then System will
    catch the exception
    }
}

```

- Final strategy:
Encapsulation—separating one method into two methods, then encapsulating all these methods into the **extractSSN** class.

For example:

Only two statements(sentences) inside Main class's **try()** clause: this method offers the highest possible degree of encapsulation; see it encapsulating in **extractSSN** class's **bucketSortToFile()** and **bucketSortToUI** methods.

Look the following piece of codes:

```

/*
public ArrayList<Integer> bucketSortToFile() throws FileNotFoundException{
    /* initialize every area people's numbers */
    ArrayList<Integer> templist = new ArrayList<>();
    ArrayList<Integer> afterBucketSort = new ArrayList<>();

    String inFile = "transfer.txt";
    File inputFile = new File(inFile);
    Scanner in = new Scanner(inputFile);
    String outFile = "Bucket_SSN.txt";
    PrintWriter out = new PrintWriter(outFile);

    while(in.hasNextInt()){
        int tempTotalNum = in.nextInt();
        templist.add(tempTotalNum);
    }

    bucket bt = new bucket();
    double [] darr = bt.convertArr(templist); //covert an Integer arrayList to a double array[]
    bt.bucketSort(darr); //call bucketSort method to sort the double array[]
    for(double d : darr){
        int temp = (int) (d * 1000000000);
        afterBucketSort.add(temp);
    } //After called bucketSort method, convert double array[] to Integer arrayList

    //write every ssn number into the Bucket_SSN.txt file
    for(int i=0; i<afterBucketSort.size(); i++){
        String str = "";
        str = str + afterBucketSort.get(i);
        out.println(str.substring(0, 3) + "-" + str.substring(3,5) + "-" + str.substring(5));
    }
    System.out.println("Written successfully");
    in.close();
    out.close();
    return afterBucketSort;
}

public ArrayList<String> bucketSortToUI() throws FileNotFoundException{
    ArrayList<Integer> tempUI = bucketSortToFile();
    ncsPeople = 0;
    scsPeople = 0;
    msPeople = 0;
    nwcsPeople = 0;
    wcsPeople = 0;

    String strValue;
    for(int i=0; i<tempUI.size(); i++){
        String str = "";
        str = str + tempUI.get(i);
        String first = str.substring(0, 3);
        int temp = Integer.parseInt(first);
        if(temp <= 199){ //store every ssn number with the area name
            ncsPeople++;
            strValue = ncs + str.substring(0, 3) + "-" + str.substring(3,5) + "-" + str.substring(5);
        }else if(temp >199 && temp <= 399){
            scsPeople++;
            strValue = scs + str.substring(0, 3) + "-" + str.substring(3,5) + "-" + str.substring(5);
        }else if(temp >399 && temp <= 599){
            msPeople++;
            strValue = ms + str.substring(0, 3) + "-" + str.substring(3,5) + "-" + str.substring(5);
        }else if(temp >599 && temp <= 799){
            nwcsPeople++;
            strValue = nwcs + str.substring(0, 3) + "-" + str.substring(3,5) + "-" + str.substring(5);
        }else{
            wcsPeople++;
            strValue = wcs + str.substring(0, 3) + "-" + str.substring(3,5) + "-" + str.substring(5);
        }
        bucketList.add(strValue); //call the arrayList bucketList to add every ssn number with the area name
    }
    return bucketList;
}

```

- Change Data Type:

In **radix** class, I must convert the data type:

```
public class radix {

    /**
     * @param arr the required arrayList to convert to a integer array
     * @return a new integer array
     */
    public int[] convertToArr(ArrayList<Integer> arr){
        int [] iarr = new int[arr.size()];
        for(int i=0; i<arr.size(); i++){
            iarr[i] = arr.get(i);
        }
        return iarr;
    }
}
```

In **bucket** class, I must covert **ArrayList<Integer>** to **double array[]** :

```
public class bucket{
    /**
     * Convert a Integer arrayList to a double array[]
     * @param intarr the given arrayList with Integer elements
     * @return a double array[]
     */
    public double[] convertArr(ArrayList<Integer> intarr){
        double [] darr = new double[intarr.size()];
        for(int i=0; i<intarr.size(); i++){
            darr[i] = (intarr.get(i))/ 1000000000.0;
        }
        return darr;
    }
}
```

To summary: there is still a way that I learn from textbook and the previous CS course, can address there problems.

7. Leasson Learned

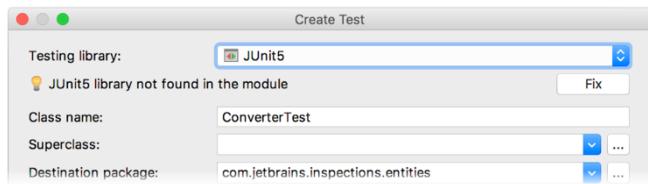
- Import **Javafx** library
- Improt **Java jdk 8**
- Import **Junit Test**
- Learned to debug some problems that I encountered
- Learned to encapsulate all methods into a class **extractSSN**
- Learned to convert data structure : convert **ArrayList to Array** and convert **Array to ArrayList**
- For **buketSort()**
 1. convert **ArrayList<Integer>** to **double[]**
 2. calling the **insertSort()** to help implement **bucketSort()**
 3. learned to explain and illustrate to show my design a bucket for **bucketSort()**
- For **radixSort()**
 1. convert **ArrayList<Integer>** to **int[]**
 2. calling the **countSort()** to help implement **raidxSort()**
 3. learned to explain and illustrate to define a digit for my **radixSort()**

8. Running Time Big-O (Junit -Test)

- **Install :**

Add a test library to the classpath when creating a test for a class

1. In the editor, place the cursor within the line containing the class declaration.
2. Press  to view the available [intention actions](#).
3. Select **Create Test**.
4. In the **Create Test** dialog, to the right of the text informing you that the corresponding library is not found, click **Fix**.



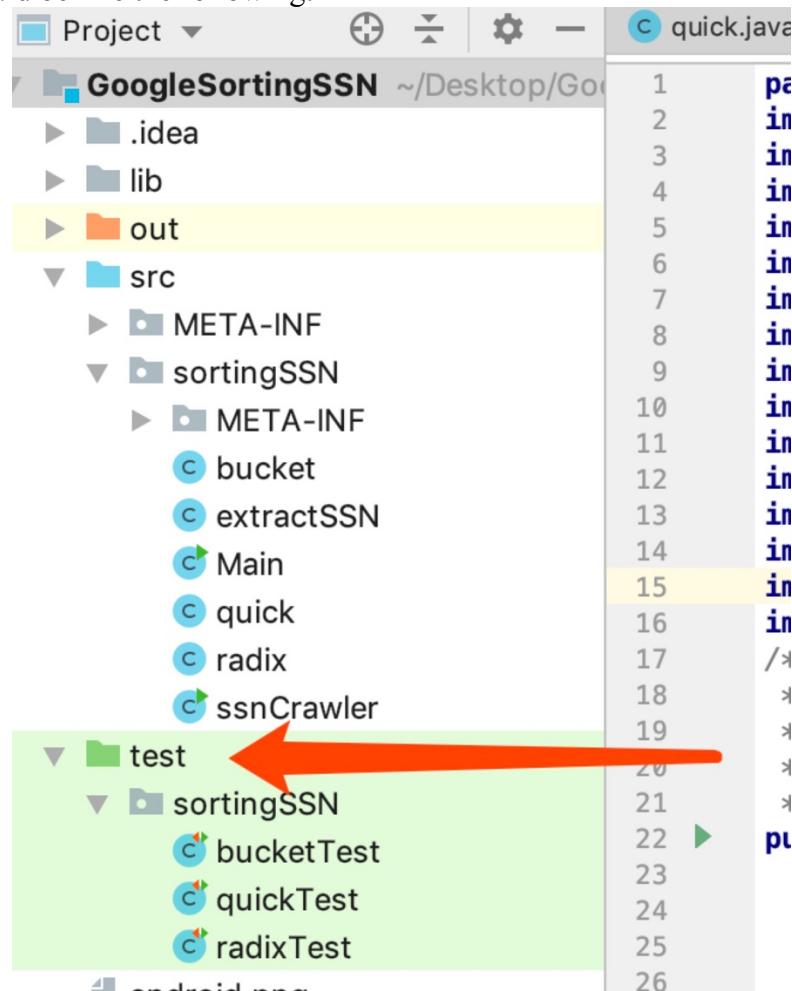
Add a test library to the classpath when writing the code for a test

1. In the source code of a test class, place the cursor within an unresolved reference to **TestCase** or annotation.
2. Press  to view the available [intention actions](#).
3. Select **Add <library> to classpath**.



- All **Junit Test** .java files should be inside the **test** folder
Note: the **folder name** depends on your choice

The path should be like the following:



```

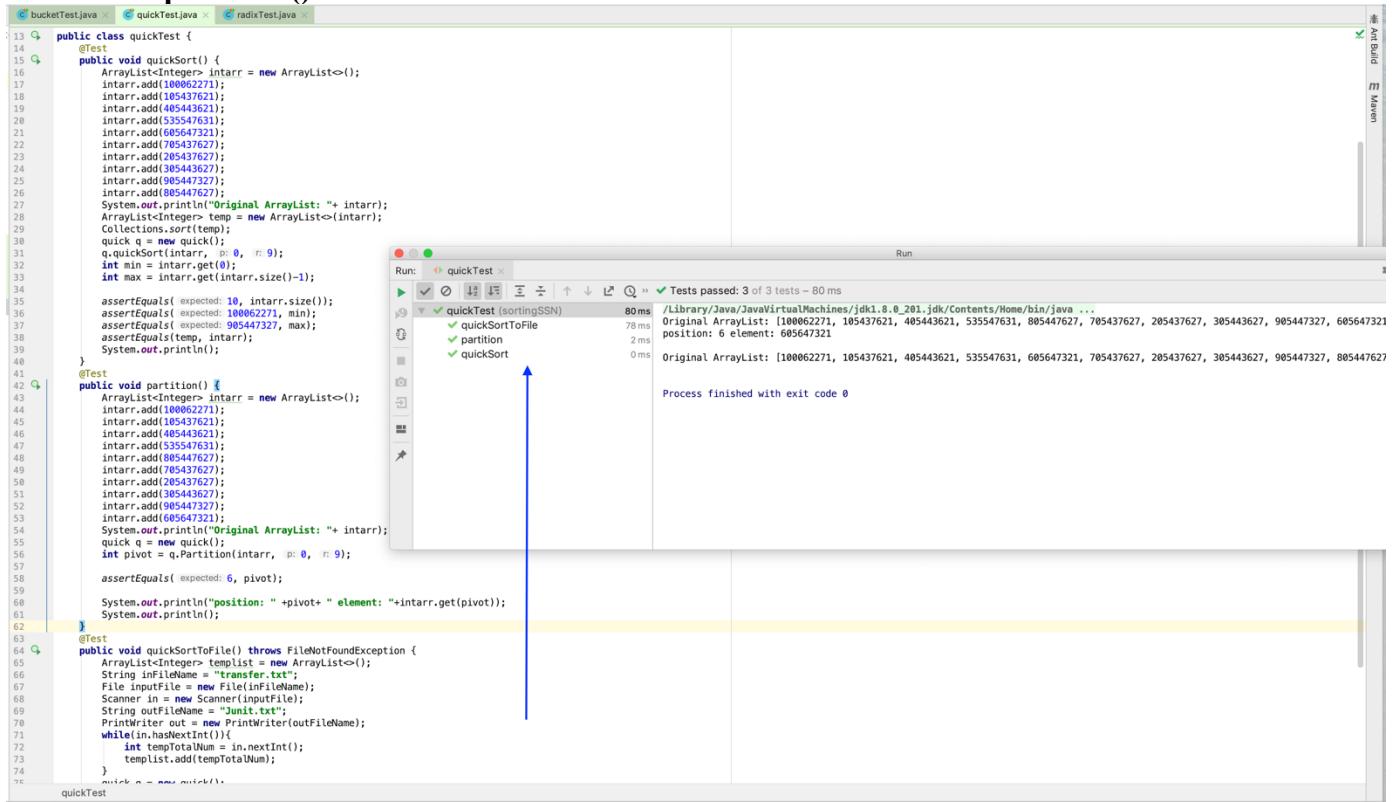
Project  +  -  ⚙  ⌂  quick.java
GoogleSortingSSN ~/Desktop/Go
  .idea
  lib
  out
  src
    META-INF
    sortingSSN
      META-INF
      bucket
      extractSSN
      Main
      quick
      radix
      ssnCrawler
    test
      sortingSSN
        bucketTest
        quickTest
        radixTest
  1  pa
  2  in
  3  in
  4  in
  5  in
  6  in
  7  in
  8  in
  9  in
 10  in
 11  in
 12  in
 13  in
 14  in
 15  in
 16  in
 17  /»
 18  »
 19  »
 20  »
 21  »
 22  ➤
 23  »
 24  »
 25  »
 26  »

```

- If you are installing like the above steps, that means you have installed successfully.

- The following I take it from 10 times's running, I extract the middle data from 10 times
- I picked the same 10 **SSN numbers**, then I will guarantee fairness.
- I picked the **300** SSN numbers that the **ssnCrawler** class generated, to write into the .txt file
- Including **quickSort** **bucketSort** **radixSort**

- For **quickSort()** :



The screenshot shows an IDE interface with two tabs: `bucketTest.java` and `quickTest.java`. The `quickTest.java` tab is active, displaying Java code for testing the `quickSort` method. The code includes assertions for the `partition` and `quickSort` methods. A blue arrow points from the text "Hence for here, QuickSort's running time:" to the test run output window.

```

13  public class quickTest {
14    @Test
15    public void quickSort() {
16      ArrayList<Integer> intarr = new ArrayList<>();
17      intarr.add(100062271);
18      intarr.add(105437621);
19      intarr.add(405443621);
20      intarr.add(535547631);
21      intarr.add(605647321);
22      intarr.add(705437627);
23      intarr.add(205437627);
24      intarr.add(905443627);
25      intarr.add(985447327);
26      intarr.add(1085447627);
27      System.out.println("Original ArrayList: " + intarr);
28      ArrayList<Integer> temp = new ArrayList<>(intarr);
29      Collections.sort(temp);
30      quick q = new quick();
31      q.quickSort(intarr, 0, 9);
32      int min = intarr.get(0);
33      int max = intarr.get(intarr.size() - 1);
34
35      assertEquals(expected: 10, intarr.size());
36      assertEquals(expected: 100062271, min);
37      assertEquals(expected: 905443627, max);
38      assertEquals(temp, intarr);
39      System.out.println();
40    }
41
42    @Test
43    public void partition() {
44      ArrayList<Integer> intarr = new ArrayList<>();
45      intarr.add(105437621);
46      intarr.add(405443621);
47      intarr.add(535547631);
48      intarr.add(605647321);
49      intarr.add(705437627);
50      intarr.add(205437627);
51      intarr.add(905443627);
52      intarr.add(985447327);
53      intarr.add(1085447627);
54      System.out.println("Original ArrayList: " + intarr);
55      quick q = new quick();
56      int pivot = q.partition(intarr, 0, 9);
57
58      assertEquals(expected: 6, pivot);
59
60      System.out.println("position: " + pivot + " element: " + intarr.get(pivot));
61      System.out.println();
62    }
63
64    @Test
65    public void quickSortToFile() throws FileNotFoundException {
66      ArrayList<Integer> tempList = new ArrayList<>();
67      String inFile = "transfer.txt";
68      File inputFile = new File(inFileName);
69      Scanner in = new Scanner(inputFile);
70      String outFileName = "Junit.txt";
71      PrintWriter out = new PrintWriter(outFileName);
72      while (in.hasNextInt()) {
73        int tempToIntNum = in.nextInt();
74        tempList.add(tempToIntNum);
75      }
76      quick q = new quick();
77    }
78  }

```

The test run output window shows the results of the `quickTest` run. It lists three tests: `quickTest (sortingSSN)`, `quickSortToFile`, and `partition`. The `quickTest` test has a duration of 80 ms. The output also shows the original array list and its sorted state.

Hence for here, QuickSort's running time:

- All quickTest : **80 ms**
- Partition: **2 ms**
- Since within Partition() , it will compare the data
 - ⇒ QuickSort's average running time is $O(n \lg n)$
 - ⇒ The worst case running time is $O(n^2)$

- For bucketSort()

The screenshot shows an IDE interface with two main panes. The left pane displays the Java code for the `bucketTest` class. The right pane shows the execution results in a 'Run' window.

```

13  public class bucketTest {
14      @Test
15      public void convert.ToDoubleArray(){
16          ArrayList<Integer> intarr = new ArrayList<>();
17          intarr.add(100062271);
18          intarr.add(105437621);
19          intarr.add(405443621);
20          intarr.add(535547631);
21          intarr.add(605647321);
22          intarr.add(705437627);
23          intarr.add(205437627);
24          intarr.add(305443627);
25          intarr.add(905447327);
26          intarr.add(805447627);
27          bucket bt = new bucket();
28          double[] darr = bt.convertArr(intarr);
29          for(double d : darr)
30              System.out.println(d);
31          System.out.println("*****");
32      }
33
34      @Test
35      public void buckSort(){
36          ArrayList<Integer> intarr = new ArrayList<>();
37          intarr.add(100062271);
38          intarr.add(105437621);
39          intarr.add(405443621);
40          intarr.add(535547631);
41          intarr.add(605647321);
42          intarr.add(705437627);
43          intarr.add(205437627);
44          intarr.add(305443627);
45          intarr.add(905447327);
46          intarr.add(805447627);
47          bucket bt = new bucket();
48          double[] darr = bt.convertArr(intarr);
49          bt.buckSort(darr);
50          for(double d : darr)
51              System.out.println(d);
52          System.out.println("*****");
53          assertTrue( condition: 0.100062271 == darr[0]);
54          assertTrue( condition: 0.905447327 == darr[9]);
55      }
56
57      @Test
58      public void buckSortToFile() throws FileNotFoundException {
59          ArrayList<Integer> templist = new ArrayList<>();
60          String inFileNamr = "transfer.txt";
61          File inputFile = new File(infileName);
62          Scanner in = new Scanner(inputFile);
63          String outfileName = "Junit.txt";
64          PrintWriter out = new PrintWriter(outfileName);
65          while(in.hasNextInt()){
66              int tempTotalNum = in.nextInt();
67              templist.add(tempTotalNum);
68          }
69          bucket bt = new bucket();
70          double[] darr = bt.convertArr(templist);
71          bt.buckSort(darr);
72          for(double d : darr){
73              int temp =(int) (d * 100000000);
74              out.println(temp);
75          }
76      }
    
```

The right pane shows the 'Run' window with the title 'bucketTest (sortingSSN)'. It displays the test results:

- Tests passed: 3 of 3 tests -
- /Library/Java/JavaVirtualMachine
- 0.100062271
- 0.105437621
- 0.205437627
- 0.305443627
- 0.405443621
- 0.535547631
- 0.605647321
- 0.705437627
- 0.805447627
- 0.905447327
- *****
- 0.100062271
- 0.105437621
- 0.205437627
- 0.305443627
- 0.405443621
- 0.535547631
- 0.605647321
- 0.705437627
- 0.805447627
- 0.905447327
- 0.805447627
- *****

Process finished with exit

Hence for here, BucketSort's running time:

- All bucketTest : **42 ms**
- buckSort: **5 ms**

⇒ bucketSort's worst case running time is $O(n)$

- For radixSort() :

The screenshot shows an IDE interface with several tabs at the top: Main.java, quick.java, quickTest.java, bucketTest.java, radix.java, radixTest.java, and Junit.txt. The radixTest.java tab is active, displaying Java code for testing radix sort. The code includes methods for getting maximum values, performing radix sort on an array, and writing sorted data to a file. A blue arrow points from the Junit.txt tab to the Run panel.

Junit.txt Content:

```

1 100062271
2 105447621
3 109817109
4 110171412
5 111201943
6 117213185
7 121556474
8 125193233
9 125633987
10 144193443
11 158727065
12 160065796
13 163002580
14 168149160
15 170028324
16 175212930
17 175968694
18 176113865
19 177174612
20 182754026
21 185790310
22 19122844

```

Run Panel Output:

Test	Time
radixTest (sortingSSN)	36 ms
radixSortToFile	34 ms
radixSort	2 ms
getMaximum	0 ms

Hence , RadixSort's running time:

- All quickTest : **36 ms**
- RadixSort: **2 ms**
 - ⇒ RadixSort's total running time is $\Theta(d(n+k))$
 - ⇒ When pass over n d-digit numbers that take time $\Theta(n+k)$
 - ⇒ Total time = $O(dn + dk)$
 - ⇒ Since d is constant and k= $O(n)$, takes $O(n)$ time
 - ⇒ Summary: Fast ! Stable ! Simple ! (Doesn't sort in place)

- Running time summary:
 - ⇒ Even though, we only pick random 300 SSN numbers, which are not representative.
 - ⇒ However, to be academic research, I still analyze their comparison.
 - ⇒ This is the course's goal——I learn from CS 146 Data Structure Algorithms
- Now I can summary the comparison between **QuickSort BucketSort RadixSort**:
 - ⇒ Since the **quickSort**'s worst case running time is $O(n^2)$
 - ⇒ **bucketSort**'s worst case running time is $O(n)$
 - ⇒ **radixSort**'s worst case running time is $O(n)$
 - ⇒ Thus, for my program, quickSort test is **80 ms**
 - ⇒ **bucketSort** and **radixSort** is faster than **quickSort**
- **Programming Assignment 2: Social Security Number**
is Done!

Thank you for reading !