

Alexander Molodyh

Class: CS361

Date: June 6, 2017

CS361 Lab3

While executing the Radixsort, Binsort, and Mergesort algorithms alongside each other. I have noticed that Binsort performs much better than Radixsort and Mergesort algorithms. When the list becomes much larger, Binsort isn't affected as much as Radixsort or Mergesort. On smaller list sizes the Mergesort beats both Radixsort and Binsort. But when the list starts becoming very large, then the Binsort outperforms both the Radixsort and Mergesort. According to the recorded data, Radix sort is also slower than Mergesort. I doesn't matter if the list size is small or large, Mergesort is consistently faster than Radixsort by a small amount. Because of Binsort's linear time of sorting the data, it is and will be faster than any sorting algorithm such as: Quicksort, Mergesort, and Radixsort.

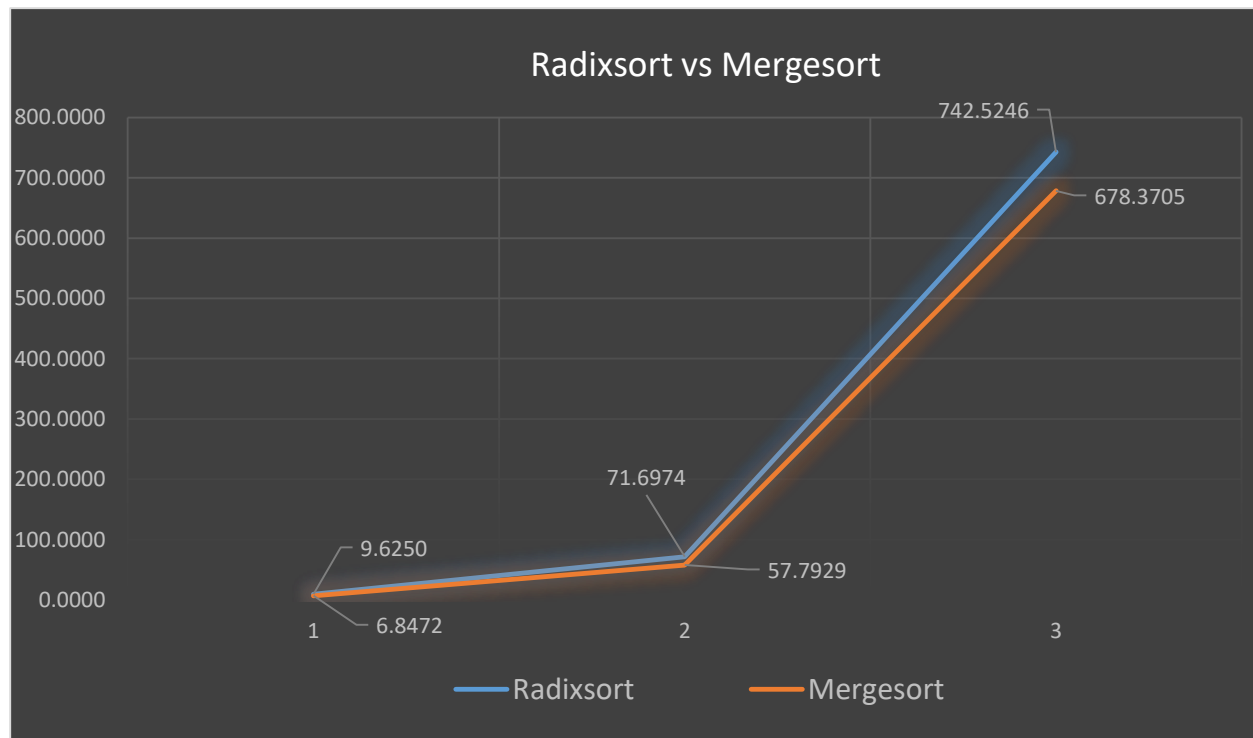
The following table contains the results data from running the Radixsort, Binsort, Mergesort data. The sorting algorithms where ran 3 times on sizes of:

- 100,000
- 1,000,000
- 10,000,000

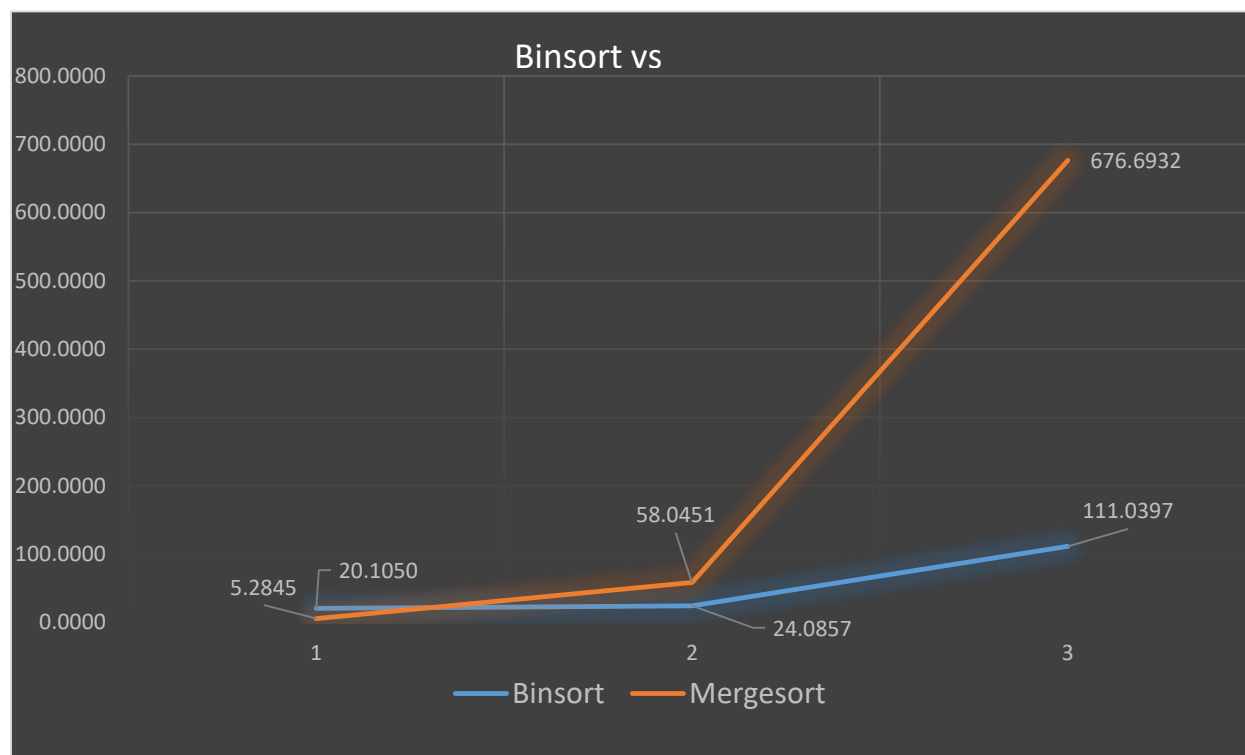
Then an average runtime is calculated from the three columns of each sorting algorithm.

Algorithm	Array Size	Run #1	Run #2	Run #3	Avg Runtime
Radixsort	100,000	14.4804	7.4447	6.9501	9.6250
Mergesort	100,000	8.9508	6.1846	5.4062	6.8472
Radixsort	1,000,000	67.7946	71.8768	75.4207	71.6974
Mergesort	1,000,000	56.9583	57.0155	59.4051	57.7929
Radixsort	10,000,000	700.4251	793.6850	733.4637	742.5246
Mergesort	10,000,000	678.3160	678.4101	678.3855	678.3705
Binsort	100,000	22.1938	25.9605	12.1608	20.1050
Mergesort	100,000	5.3574	5.0942	5.4018	5.2845
Binsort	1,000,000	23.2534	22.1499	26.8540	24.0857
Mergesort	1,000,000	57.0474	58.1578	58.9301	58.0451
Binsort	10,000,000	111.1672	110.7620	111.1899	111.0397
Mergesort	10,000,000	676.2537	677.0539	676.7719	676.6932

The following chart represents the runtime difference between the Radixsort and Mergesort algorithms.



The following chart represents the average runtime between the Binsort and Mergesort algorithms.



The following screenshot is the output of an array being sorted with the Radixsort of size 10,000,000.

```

Run SortingHelper
"C:\Program Files\Java\jdk1.8.0_73\bin\java" ...

Please select an option from the following list
Run RadixSort(R)
Run BinSort(B)
Set array size(S)
To exit, enter(Q)
Enter your choice here: S
10000000
Array size is now: 10000000
New size is: 10000000
You must enter a valid decision!!

Please select an option from the following list
Run RadixSort(R)
Run BinSort(B)
Set array size(S)
To exit, enter(Q)
Enter your choice here: R
Is array sorted after RadixSort? Yes
Time it took to sort: 744.50283

Is array sorted after MergeSort? Yes
Time it took to sort: 647.501481

Please select an option from the following list
Run RadixSort(R)
Run BinSort(B)
Set array size(S)
To exit, enter(Q)
Enter your choice here:

```

The code for the previous screenshot is:

```

public static void main(String[] args)
{
    int arraySize = 10000000;
    InputRoutine inputRoutine = new InputRoutine(arraySize);
    QuickMergeSort qmSort = new QuickMergeSort();
    SortManager sm = new SortManager();
    FA dfa;

    State q0 = State.makeState(0, true);
    State q1 = State.makeState(1);

    ArrayList<State> Q = new ArrayList<>();
    Q.add(q0);
    Q.add(q1);

    ArrayList<State> F = new ArrayList<>();
    F.add(q0);

    String language = "00001001";

```

```

State[][] states = {{q0, q1}, {q0, q1}};

int[] largeArray = inputRoutine.getIntegerList();
boolean done = false;

while(!done)
{
    String sizeResponse = "";
    int[] integerList = inputRoutine.getIntegerList();

    System.out.println("\nPlease select an option from the following list");
    System.out.println("Run RadixSort(R)\nRun BinSort(B)");
    System.out.println("Set array size(S)");
    System.out.println("Run DFA(D)");
    System.out.println("To exit, enter(Q) ");
    System.out.print("Enter your choice here: ");

    Scanner scan = new Scanner(System.in);
    String response = scan.next();

    //This will change the array size
    if(response.equalsIgnoreCase("S"))
    {
        try {arraySize = scan.nextInt();}
        catch(Exception e){};

        System.out.println("Array size is now: " + arraySize);
        inputRoutine.setArraySize(arraySize);
        inputRoutine.reReadFile();
        largeArray = inputRoutine.getIntegerList();

        System.out.println("New size is: " + largeArray.length);
    }

    //This will run the Radixsort
    if(response.equalsIgnoreCase("R"))
    {
        int[] copyList = Arrays.copyOf(integerList, integerList.length);
        integerList = sm.radixSort(integerList);

        System.out.println("Is array sorted after RadixSort? " +
            ((isSorted(integerList) ? " Yes" : " No")));
        System.out.println("Time it took to sort: " + sm.getRadixTime() + "\n");

        //Make a copy of the original array to be sorted in the Mergesort
        copyList = qmSort.mergeSort(copyList);

        System.out.println("Is array sorted after MergeSort? " +
            ((isSorted(copyList) ? " Yes" : " No")));
        System.out.println("Time it took to sort: " + qmSort.getMergeSortTime());
    }
    else if(response.equalsIgnoreCase("B"))//This will run the Binsort
    {
        int[] copyList = Arrays.copyOf(integerList, integerList.length);
        integerList = sm.binSort(integerList);

        System.out.println("Is array sorted after BinSort? " +
            ((isSorted(integerList) ? " Yes" : " No")));
        System.out.println("Time it took to sort: " + sm.getBinTime() + "\n");

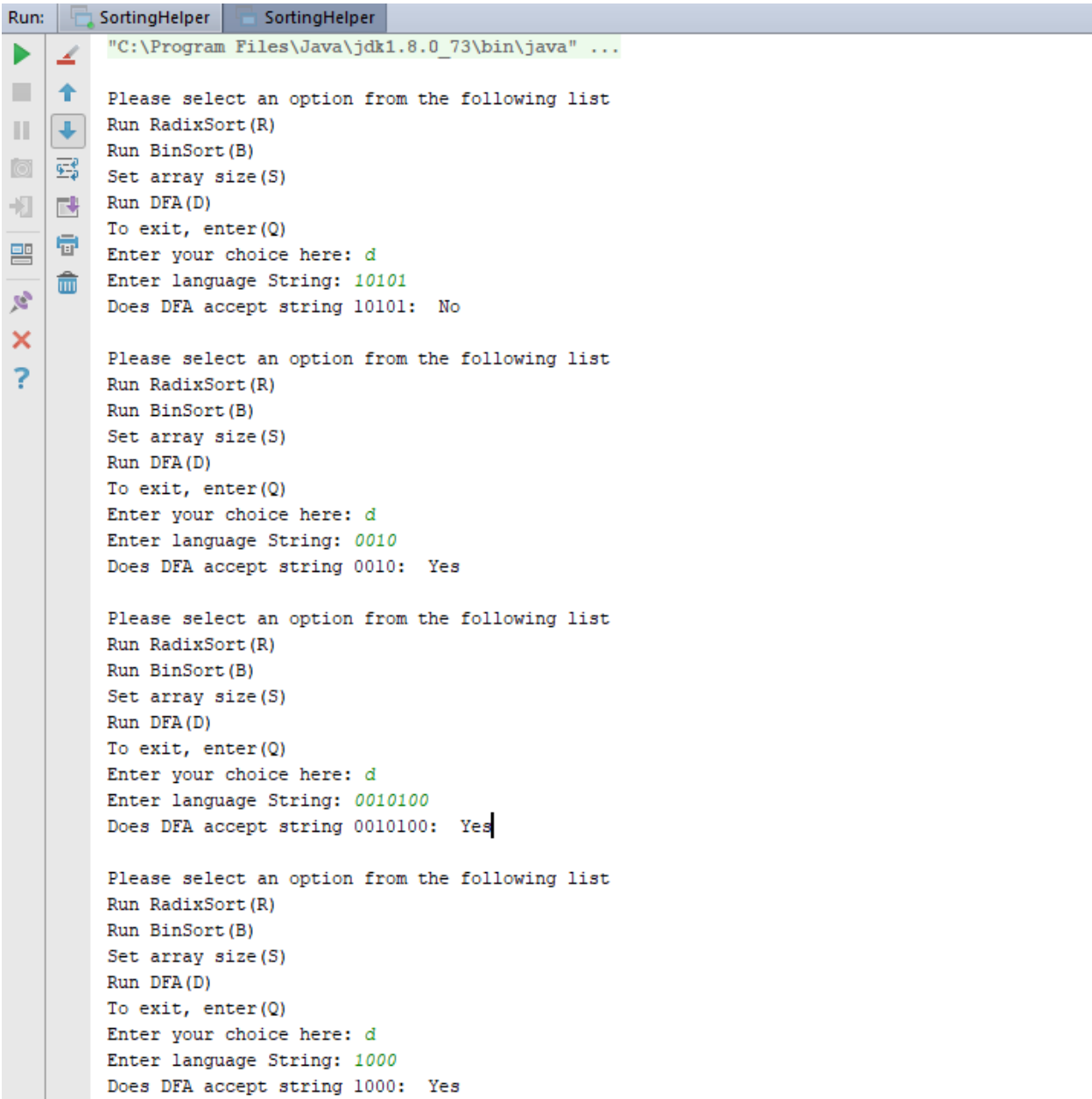
        //Make a copy of the original array to be sorted in the Mergesort
        copyList = qmSort.mergeSort(copyList);
    }
}

```

```
        System.out.println("Is array sorted after MergeSort? " +
            ((isSorted(copyList) ? " Yes" : " No")));
        System.out.println("Time it took to sort: " + qmSort.getMergeSortTime());
    }
    else if(response.equalsIgnoreCase("D"))
    {
        System.out.print("Enter language String: ");
        language = scan.next();
        dfa = new FA(Q, language, states, q0, F);

        System.out.println("Does DFA accept string " + language + ": " +
            ((dfa.isValidString(language, true) ? " Yes" : " No")));
    }
    else if(response.equalsIgnoreCase("Q"))//Quit the program
        done = true;
    else
        System.out.println("You must enter a valid decision!!");
}
}
```

The following is a screenshot of the DFA output.



```
Run: SortingHelper SortingHelper
"C:\Program Files\Java\jdk1.8.0_73\bin\java" ...

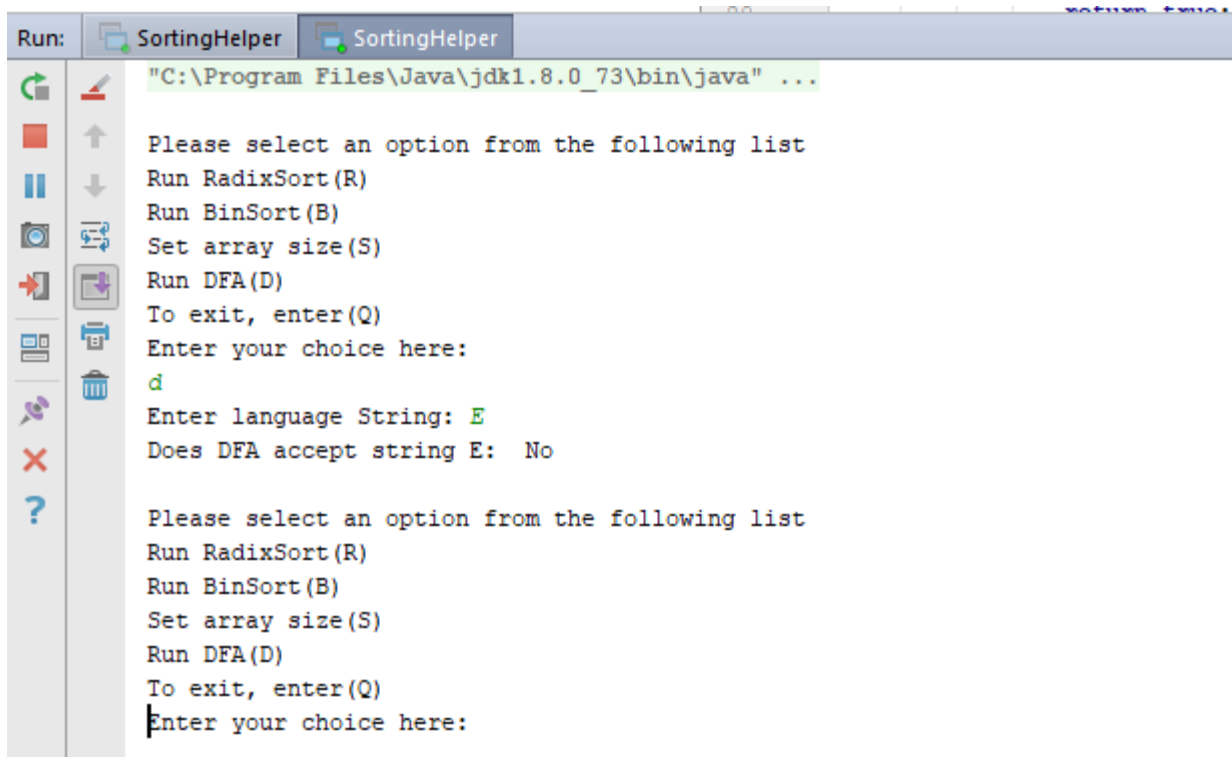
Please select an option from the following list
Run RadixSort(R)
Run BinSort(B)
Set array size(S)
Run DFA(D)
To exit, enter(Q)
Enter your choice here: d
Enter language String: 10101
Does DFA accept string 10101: No

Please select an option from the following list
Run RadixSort(R)
Run BinSort(B)
Set array size(S)
Run DFA(D)
To exit, enter(Q)
Enter your choice here: d
Enter language String: 0010
Does DFA accept string 0010: Yes

Please select an option from the following list
Run RadixSort(R)
Run BinSort(B)
Set array size(S)
Run DFA(D)
To exit, enter(Q)
Enter your choice here: d
Enter language String: 0010100
Does DFA accept string 0010100: Yes

Please select an option from the following list
Run RadixSort(R)
Run BinSort(B)
Set array size(S)
Run DFA(D)
To exit, enter(Q)
Enter your choice here: d
Enter language String: 1000
Does DFA accept string 1000: Yes
```

The following screenshot is from running the DFA on “E”.



The following code is the Radixsort and Binsort.

```
public class SortManager
{
    private final int INCREMENTER_MOD = 10;
    private long radixTime = 0;
    private long binTime = 0;

    /**
     *radixSort will perform a Radixsort algorithm on the given array to sort it.
     * @param arr An integer array that will be sorted.
     */
    public int[] radixSort(int[] arr)
    {
        radixTime = getMillis();
        int n = arr.length;

        //Get the largest element in the list to know how long the number is
        int max = getLargestElement(arr, n);

        //Loop through the largest number length
        for(int i = 1; (max / i) > 0; i *= 10)
            arr = radixHelper(arr, n, i);

        radixTime = getMillis() - radixTime;

        return arr;
    }

    //gets the largest integer in the list and returns it
}
```



```

private int getLargestElement(int[] arr, int size)
{
    int largest = arr[0];

    for(int i = 0; i < size; i++)
        if(arr[i] > largest)
            largest = arr[i];

    return largest;
}

//This is a helper method for the Radixsort method
private int[] radixHelper(int[] arr, int n, int div)
{
    //Create helper array to store temporary output elements
    int[] helperArr = new int[n];
    int[] incrementerArr = new int[INCREMENTER_MOD];
    Arrays.fill(incrementerArr, 0);

    //Store the count of occurrences in incrementerArr
    for(int i = 0; i < n; i++)
        incrementerArr[ (arr[i] / div) % INCREMENTER_MOD ]++;

    //Change each index so that it contains the actual position of the current digit
    for(int i = 1; i < incrementerArr.length; i++)
        incrementerArr[i] += incrementerArr[i - 1];

    //Build the output array
    for(int i = n - 1; i >= 0; i--)
    {
        int helperIndex = incrementerArr[ (arr[i] / div) % INCREMENTER_MOD ] - 1;
        helperArr[ helperIndex ] = arr[i];
        incrementerArr[ (arr[i] / div) % INCREMENTER_MOD ]--;
    }

    //Copy the helperArr in to the arr to be returned
    for(int i = 0; i < n; i++)
        arr[i] = helperArr[i];

    return arr;
}

public static void print(int[] arr)
{
    for(int i = 0; i < arr.length; i++)
        System.out.print(arr[i] + ", ");
}

//////////Bin Sort//////////

/**
 * binSort performs a Binsort algorithm on the given array to be sorted
 * @param arr The array to be sorted.
 * @return An onteger array of sorted elements.
 */
public int[] binSort(int[] arr)
{
    binTime = getMillis();
    int n = arr.length;
    int arraySize = getLargestElement(arr, n);
    int[] helperArr = new int[arraySize + 1];
    Arrays.fill(helperArr, 0);

    /*

```

```

This increments the amount of same elements we have in to
the index of helperArr[with value of arr[i]].
*/
for(int i = 0; i < n; i++)
    helperArr[arr[i]]++;

int k = 0;
//Loop through the array size
for(int i = 0; i <= arraySize; i++)
{
    //loop through every index of helperArr
    for(int j = 0; j < helperArr[i]; j++)
    {
        //Set current index of arr to i index
        arr[k] = i;
        k++;
    }
}

binTime = getMillis() - binTime;

return arr;
}

```

The following code is for the Finite Automata:

```

public class FA
{
    private ArrayList<State> Q;
    private ArrayList<State> F;
    private State curState;

    private String language = "";
    private State[][] transFunction;
    private State[][][] nfaTransFunc;

    private final char E = 'E';
    private char curInput;

    private boolean isDFA = true;

    public FA(ArrayList<State> Q, String language, State[][] transFunction,
              State startState, ArrayList<State> F)
    {
        this.Q = Q;
        this.language = language;
        this.transFunction = transFunction;
        this.curState = startState;
        this.F = F;
    }

    public FA(ArrayList<State> Q, String language, State[][][] nfaTransFunc,
              State startState, ArrayList<State> F)
    {
        this.Q = Q;
        this.language = language;
        this.nfaTransFunc = nfaTransFunc;
        this.curState = startState;
        this.F = F;
    }

    /**
     * isValidString checks to see if the given string is a valid language of the current DFA.
     * @param language A string representing 0's and 1's.

```

```

* @param isDFA A boolean that determines if the current machine is a DFA or NFA.
* @return true if the string is valid and false otherwise.
*/
public boolean isValidString(String language, boolean isDFA)
{
    this.language = language;
    this.isDFA = isDFA;

    if(this.language != null)
    {
        if(!isDFA) //If current FA is not a DFA
        {
            //Check to see if language is not empty "E"
            if(language.length() < 2 && language.charAt(0) == 'E')
                return true;
        }
        else
        {
            //If language is empty and current machine is a DFA, return false
            if(language.length() < 2 && language.charAt(0) == 'E')
                return false;
            else
                return run(isDFA);
        }
    }
    return false;
}

private boolean run(boolean isDFA)
{
    if(isDFA)
    {
        return runDFA(language);
    }
    else
    {
        if(runNFA(language, this.curState) != null)
            return true;
    }
    return false;
}

private boolean runDFA(String language)
{
    //Loop through language string
    for(int i = 0; i < language.length(); i++)
    {
        this.curInput = language.charAt(i);
        int col = Character.getNumericValue(this.curInput);
        this.curState = getNextState(curState.getStateNum(), col);

        if(this.curState == null)
            return false;
    }

    //If we are not in the accept state when language sting is finished,
    //then we don't return true.
    if(F.contains(this.curState))
        return true;

    return false;
}

private State getNextState(int row, int col)

```

```

{
    if(transFunction != null && transFunction[row][col] != null)
        return transFunction[row][col];
    else
        return null;
}
}

```

The following code is what I used to represent a State:

```

public class State
{
    private boolean acceptState = false;
    private String q = "q";
    private int stateNum;

    private State() {}

    private State(int stateNum) {this.stateNum = stateNum;}

    private State(int stateNum, boolean acceptState)
    {
        this.stateNum = stateNum;
        this.acceptState = acceptState;
    }

    public boolean isAcceptState() {return acceptState;}

    public String getState() {return q + stateNum;}

    public char getStateChar()
    {
        char c = q.charAt(0);
        return c;
    }

    public int getStateNum() {return stateNum;}

    @Override
    public boolean equals(Object o)
    {
        State tempState = (State) o;

        if(tempState.getState().equalsIgnoreCase(getState()) &&
            tempState.isAcceptState() == acceptState)
            return true;
        else
            return false;
    }

    @Override
    public int hashCode()
    {
        int result = 17;
        result = 31 * result + getState().hashCode();
        result = 31 * result + (acceptState ? 1 : 0);
        return result;
    }

    @Override
    public String toString()
    {
        String temp = "State: " + getState() + "\nAcceptState: " + acceptState;
        return temp;
    }
}

```

```

    }

    public static State makeState(int stateNum, boolean acceptState) {return new State(stateNum,
acceptState);}

    public static State makeState(int stateNum) {return new State(stateNum);}
}

```

The following code is for the InputRoutine class:

```

public class InputRoutine
{
    private String fileAddress = "../lab3_data.txt";
    private Long compareSum = new Long("49999995000000");
    private int[] integerList;
    private int[] copyList;
    private int arraySize = 0;
    Scanner fileScanner;
    File file;

    //////////////Public Constructors////////////////////////////////////
    public InputRoutine()
    {
        //If no getArrayLength is given for the array getArrayLength then we
        // CS set it to 10100000 as the default value(which is
        //ten million one hundred thousand).
        integerList = new int[10000000];
        copyList = Arrays.copyOf(integerList, integerList.length);
        this.arraySize = integerList.length;
        readInFile();
    }

    public InputRoutine(int arraySize)
    {
        this.arraySize = arraySize;
        this.fileAddress = fileAddress;
        integerList = new int[arraySize];
        copyList = new int[integerList.length];
        readInFile();
    }

    /**
     * getCompareSum returns the amount we want to compare our sum to.
     *
     * @return A Long object that represents the amount our list should sum up to.
     */
    public Long getCompareSum()
    {
        return compareSum;
    }

    /**
     * getSum sums up the total from the integerList.
     *
     * @return Returns an integer representing the sum of all integers from the integerList.
     */
    public long getSum()
    {
        long sum = 0;

        for(int i = 0; i < integerList.length; i++)
        {

```

```

        sum = sum + integerList[i];
    }

    return sum;
}

public int[] getIntegerList()
{
    return Arrays.copyOf(integerList, integerList.length);
}

public int[] getIntegerList(int start, int end)
{
    return Arrays.copyOfRange(integerList, start, end);
}

public void setArraySize(int arraySize)
{
    this.arraySize = arraySize;
}

public void reReadFile()
{
    integerList = new int[arraySize];
    readInFile();
}

/*
 * readFile opens a file and reads in all of the integers from the given file.
 */
private void readInFile()
{
    try
    {
        file = new File(fileAddress); //Create new file from the given file name
        fileScanner = new Scanner(file); //Create a file scanner to scan for all the integers
    }
    catch(FileNotFoundException e) //Throws a FileNotFoundException if the given file name
doesn't exist
    {
        System.out.println("The file " + fileAddress + " could not be found");
    }

    for(int i = 0; i < arraySize; i++)
    {
        try
        {
            integerList[i] = fileScanner.nextInt(); //Store the next integer from the file in to
the integerList
        }
        catch(NoSuchElementException e) {}

        i++;
    }

    fileScanner.close(); //close the scanner
}
}

```