```
/**************************************************/
/*** Purpose: Test different searching        ***/
/***          algorithms                      ***/
/***                                          ***/
/*** Author: Borja De La Viuda                ***/
/*** Date:                                    ***/
/***                                          ***/
/*** Note: Based on skeleton code provided by ***/
/*** Jason Steggles 23/11/2012                ***/
/**************************************************/

import java.io.*;
import java.text.*;
import java.util.*;

public class Search {

/** Global var for counting comparisons **/
public static int compSeq=0;
public static int compBin=0;
public static int compHash=0;

public static int totalHash=0;
public static int totalBin=0;
public static int totalSeq=0;

/** Array of values to be searched and size **/
private int[] A;
private int[] H;
 int size;
private int hSize;

/** Constructor **/
Search(int n, int hn)
{
    /** set size of array **/
    size = n;
    hSize = hn;

    /** Create arrays **/
    A = new int[size];

    H = new int[hSize];

    /** Initialize hash array **/
    /** Assume -1 indicates a location is empty **/
    for (int i=0; i<hSize; i++)
    {
        H[i] = -1;
    }
}


/*********************************************/
/*** Read a file of numbers into an array ***/
```

```
    /********************************************/
    public void readFileIn(String file)
    {
       try
       {
           /** Set up file for reading **/
           FileReader reader = new FileReader(file);
           Scanner in = new Scanner(reader);

           /** Loop round reading in data **/
           for (int i=0;i<size;i++)
           {
              /** Get net value **/
              A[i] = in.nextInt();
           }
        }
        catch (IOException e)
        {
           System.out.println("Error processing file " + file);
        }
    }


    /*********************/
    /*** Hash Function ***/
    /*********************/
    public int hash(int key)
    {
        return key%hSize;
    }


    /*****************************/
    /*** Display array of data ***/
    /*****************************/
    public void displayData(int line, String header)
    {
        /* ** Integer Formatter ** */
        NumberFormat FI = NumberFormat.getInstance();
        FI.setMinimumIntegerDigits(3);

        /** Print header string **/
        System.out.print("\n\n"+header);

        /** Display array data **/
        for (int i=0;i<size;i++)
        {
            /** New line? **/
            if (i%line == 0)
            {
                 System.out.println();
            }

            /** Display value **/
            System.out.print(FI.format(A[i])+" ");
        }
    }
```

```java
    /**************************/
    /*** Display hash array ***/
    /**************************/
    public void displayHash(int line)
    {
        /** Integer Formatter **/
        NumberFormat FI = NumberFormat.getInstance();
        FI.setMinimumIntegerDigits(3);

        /** Print header string **/
        System.out.print("\n\nHash Array of size " + hSize);

        /** Display array data **/
        for (int i=0;i<hSize;i++)
        {
            /** New line? **/
            if (i%line == 0)
            {
                 System.out.println();
            }

            /** Display value **/
            System.out.print(FI.format(H[i])+" ");
        }

    }

    /****************************
     * Sequential Search method**
     ****************************/
    public int seqSearch(int key)
    {
        compSeq = 0;
        //iterate through the array
        for(int i = 0; i < A.length;i++)
        {
            compSeq += 1;
            // key has been found
            if(A[i] == key)
            {
                totalSeq += compSeq;
                return i;

            }
            /* The value in the array is greater than key
             * since the array is sorted, key isn't in the array */
            else if(A[i]>key)
            {
                totalSeq += compSeq;
                return -1;
            }
        }

        return -1;
    }
    //get the total sequential comparisons
    public int getTotalSeq(){
        return totalSeq;
    }

    //get number of sequential comparisons
    public int getCompSeq(){
        return compSeq;
    }

    /****************************
     * Binary Search method**
     ****************************/
    //public search method
    public int binSearch(int key)
    {
        int r = A.length - 1;
        int l = 0;
        compBin = 0;
        return binaryRecursive(key,r,l);
    }

    //private recursive method to search the array
    private int binaryRecursive(int key, int r, int l)
    {

        int m = 0;
        //if pointers have crossed we haven't found the key
        if (r<l)
        {
            totalBin += compBin;
            return -1;
        }
        //calculate the median
        m = (r+l)/2;

        //check if median is the key
        if (A[m] == key)
        {
            compBin += 1;
            totalBin += compBin;
            return m;


        }
        compBin += 1;
        // go right of the array (key is greater than median)
        if (key >A[m]){
            compBin += 1;
            return binaryRecursive(key,r,m+1);

        }
        compBin += 1;
        // go left of the array (key is less than median)
        if (key <A[m])
        {
            compBin += 1;
            return binaryRecursive(key,m-1,l);
        }

        return m;
    }

    public int getCompBin(){
        return compBin;
    }

    public int getTotalBin(){
        return totalBin;
    }

    /*****************************************
     * Hashing(Linear Probing) Search method**
     *****************************************/

    public void addToHash(int value)
    {
        H[hash(value)] = value;
    }

    public void readIntoHash (String file)
    {
           try
           {
               /** Set up file for reading **/
               FileReader reader = new FileReader(file);
               Scanner in = new Scanner(reader);

               //While there is still numbers to read into the array
               while(in.hasNext())
               {
                   int input = in.nextInt();

                   // If the hashed index is empty, store integer there
                      if(H[hash(input)] == -1)
                       {
                           addToHash(input);
                       }
                      // if not, go through the array till we find empty
                          space, unless array is full
                      else if (H[hash(input)] != -1)
                      {
                          for(int i = (hash(input) + 1); i<hSize;i++)
                          {
                              if (H[i] == (hSize-1))
                              {
                                  i = 0;
                              }
                              else if(H[i] == -1)
                              {
                                  H[i] = input;
                                  break;
                              }
                              else if (i ==hash(input))
                              {
```

```
                              System.out.print("Array is full!");
                          }
                      }
                  }



           }
        }


        catch (IOException e)
        {
           System.out.println("Error processing file " + file);
        }

}


public int hashSearch(int key)
{
    //Set the number of comparisons to 0;
    compHash =0;

        //hash the key to get the initial index
        int i = hash(key);

        // Add 1 to the comparison, as we are about to compare if the key
            is in the initial index
        compHash +=1;
        if ( H[i] == key)
        {
            //Found the key, total the comparisons and return index
            totalHash += compHash;
            return i;
        }

        //Not found yet, move on to next index and check if there, if not
            loop through the array
        i+=1;
        while(H[i] != key)
        {
            compHash +=1;
            //We have reached the end of the array, go to the beginning
            if (i == (hSize-1))
            {
                i = 0;
            }
            // We've reached the nearest empty space and haven't found
                the key, key is not in the array
            else if (H[i] == -1)
            {
                compHash +=1;
                totalHash += compHash;
                return -1;
            }
            //We've reached starting point and haven't found the key, it
```

```
                's not in the array
            else if( i == hash(key) )
            {
                totalHash += compHash;
                return -1;
            }

            i++;
        }
        //Found the key, exit loop and return index
        totalHash += compHash;
        return i;

    }
    //getter method to get total number of Hash comparisons
    public int getTotalHash(){
        return totalHash;
    }
    //getter method to get number of Hash comparisons
    public int getCompHash()
    {
        return compHash;
    }

    /***********************************************
     * Method to test the data + calculate totals**
     ***********************************************/

    //Method to test the different search algorithms
    public void testSearches(int[]test)
    {
        int [] toTest = test;
        for (int i= 0; i<toTest.length;i++)
        {
           System.out.println("\nSearching for number "+ toTest[i]+ ":");
           System.out.println("Sequential Search: "+ seqSearch(toTest[i])+ "
               Number of comparisons: " + getCompSeq());
           System.out.println("Binary Search: "+ binSearch(toTest[i])+ "
               Number of comparisons: " + getCompBin());
           System.out.println("Hash Search: "+ hashSearch(toTest[i])+ "
               Number of comparisons: " + getCompHash());
        }
    }


    // method to print out and calculate the averages
    public void getTotals()
    {
        System.out.println("Total Sequential comps: " + getTotalSeq());
        System.out.println("Avg Sequential comps after 10 tests: " +
            (getTotalSeq())/10);

        System.out.println("Total Binary comps: " + getTotalBin());
        System.out.println("Avg Binary comps after 10 tests: " + (getTotalBin
            ())/10);

        System.out.println("Total Hash comps: " + getTotalHash());
        System.out.println("Avg Hash comps after 10 tests: " + (getTotalHash
            ())/10);
    }

    } /*** End of class Search ***/
```