Unit 9. Sorting, Searching, and Recursion Programs.

Program 1. Write a utility class, MyUtil, that has these two methods:

```
public class MyUtil {
        public static int linearSearch( Object [] a, Object target ){ ... }
        public static int binarySearch( Comparable [] a, Comparable target ){ ... }
}
```

Notice that the linearSearch method with any Objects but the binarySearch method only works with Comparable objects. If you are not sure why that is, be sure to ask.

Test your methods using this code.

```
public class MyUtilRunner {
        public static void main(String[] args) {
                String [] arr1 = { "walrus", "zoo", "elephant", "bear", "panda" };
                int index = MyUtil.linearSearch( arr1, "bear" );
                System.out.println( "Linear search: bear was found at index " + index );
                index = MyUtil.linearSearch( arr1, "ox" );
                System.out.println( "Linear search: ox was found at index " + index );

                String [] arr2 = { "A", "B", "F", "H", "Q", "T" };
                index = MyUtil.binarySearch( arr2, "B" );
                System.out.println( "Binary search: B was found at index " + index );
                index = MyUtil.binarySearch( arr2, "S" );
                System.out.println( "Binary search: S was found at index " + index );

                int [] nums = { -5, 0, 33, 34, 46, 51, 84, 199 };
                index = MyUtil.binarySearch( nums, 84 );
                System.out.println( "Binary search: 84 was found at index " + index );
        }
}
```

The above should print:

```
Linear search: bear was found at index 3
Linear search: ox was found at index -1
Binary search: B was found at index 1
Binary search: S was found at index -1
Binary search: 84 was found at index 6
```

Warning. You will get warnings when you right the above code. We are using an older version of the Comparable interface where the parameter is an Object. The newer version is a generic and looks like this:

```
public interface Comparable <T>{
        public int compareTo( <T> obj );
}
```

but writing this kind of method is beyond the scope of this course so we use the old version.

Program 2. Add two more static methods to MyUtil:

```
public static void selectionSort( Comparable [] a ) { ... }
public static void insertionSort( Comparable [] a ) { ... }
```

You should write private helper methods in implementing these methods.
Use the following code to test your solutions.

```
public class MyUtilRunner {
    public static void main(String[] args) {
        String [] arr3 = { "G", "R", "A", "E", "Z", "T" };
        System.out.println( "\n*** Testing selectionSort of Comparables" );
        MyUtil.selectionSort( arr3 );
        for ( String s : arr3 )
        System.out.print( s + " " );

        String [] arr4 = { "K", "Y", "W", "N", "E", "X" };
        System.out.println( "\n*** Testing insertionSort of Comparables" );
        MyUtil.insertionSort( arr4 );
        for ( String s : arr4 )
        System.out.print( s + " " );
    }
}
```

The above should print:

```
*** Testing selectionSort of Comparables
A E G R T Z
*** Testing insertionSort of Comparables
E K N W X Y
```

Program 3. When this program is completed, the user can enter any series of characters and the program will display what words, if any, can be generated using all of those letters. For instance, enter

```
deswe
```

and the program displays:

```
3 matches
sewed swede weeds
```

First you need to copy the file words.txt into the folder where your code will be. If you cannot find the file then go to mrsawyer.com and look at the Unit 5 section for AP Computer Science. There's link to the file.

Second, we are going to use the WordList class that we used back in Unit 5. The code for that is on the next page.

```
import java.io.*;
import java.util.Scanner;

public class WordList {
    public static String [] getWords( String filename ) {
        String [] list = null;
        try {
            Scanner read = new Scanner( new File( filename ) );
            int count = 0;
            while ( read.hasNext() ){
                String temp = read.nextLine();
                count++;
            }

            list = new String[ count ];
            read = new Scanner( new File( filename ) );
            count = 0;
            while ( read.hasNext() ){
                list[count] = read.nextLine();
                count++;
            }
        } catch ( IOException e1 ) {
            System.out.println( "problem reading file" );
        }
        return list;
    }
}
```

try catch is required when reading from a file

There is one word per line in the file. So we loop through once to count the lines/words in the file

Create an array of the correct length and read through the file again, this time copying the words into the array.

Third, write the Word class. Each Word object has two instance variables, an actual word and a "sorted" word - a string where the letters have been arrange in order. For instance, the word "eat" in sorted form is "aet".

```
public class Word implements Comparable {
        private String my_word;
        private String my_sorted;

        public Word( String w ){
                my_word = w;
                my_sorted = sort( w );
        }

        private String sort( String s ){
                /* give a string, return a string with the same exact characters rearranged in
                        ascending order */
        }

        public int compareTo( Object obj ){
                // returns a value based only on comparing my_word variables
        }
```

Whenever you need to sort an array, use the MyUtil class.

continued on the next page.

```
    public boolean equals_sorted( Word w ){
        return my_sorted.equals( w.my_sorted );
    }

    public String toString(){
        return my_word;
    }
}
```

Fourth, follow this outline for the main method.

1) Call the getWords method of the WordList class. Use the returned String array to create an array of Word objects

2) Write a loop where the user is asked to enter a string of characters. The program lists the words that can be made by using all of those letters. If there are no words that are composed of those letters, display "No matches." Keep looping until the user hits the enter key without entering any letters.

3) Write the following method that will be called from the above loop.

   public static ArrayList<Word> getMatches(Word [] ray, Word target)

   Traverse the entire array looking for words that match the target (in their sorted forms). In other words, eat and tea are a match because their sorted forms are the same.

   This is a very inefficient approach but we don't have the time to get into better approaches (though think about how you could make this better).

4) This is not required but, if you have the time, try this. Find the word that has the most matches. According to my solution there is a collection of 6 letters that can be rearranged to form a total of 11 words from the text file.

Program 4. You will write a simplified version of the merge method. In this problem you are given an array that is divided into two sorted portions. You will write the method that merges the two sorted sections into one sorted array.

Complete the program below. Once you get it to work, you should change the contents of the array, and the call to the merge method, to fully test your solution.

```
public class Merging {
	public static void main( String [] args ) {
		int [] list = { 35, 78, 122, 502, 999, 56, 333 };
		merge( list, 4 );		// 4 because that's the last index of the first section
		for (int n = 0; n < list.length; n++ )
			System.out.print( list[n] + ", " );

		System.out.println( "\n" );

		int [] ray = { -4, 6, -11, 0, 3, 14, 45 };
		merge( ray, 1 );		// 1 because that's the last index of the first section
		for (int n = 0; n < ray.length; n++ )
			System.out.print( ray[n] + ", " );
	}



// Precondition: 0 <= middle < a.length
// Precondition: array a has two parts that have already been sorted: (1) from the beginning to
// middle and (2) from middle + 1 to the end of the array.
// Postcondition: array a is completely sorted.

	public static void merge (int [] a, int middle ) {
		// This method merges the two sorted sections into one sorted array
		// You must create a new temporary array, copy the items (in order) to this array,
		// and at the end copy the new sorted array into array a.
	}
}
```

Problem 5. Complete the program below. This is the complete implementation of the Merge Sort algorithm. To do that you must write a new merge method. It is similar to the original in that there are two sequential sections of sorted elements that must be merged together. The difference is that the starting index does not have to be zero and the ending index does not have to be the end of the array.

```
public class UsingMergeSort {
	public static void main( String [] args ) {
		int [] list = { 88, 34, 25, 25, 17, 2};
		System.out.print( "Original: " );
		for (int n = 0; n < list.length; n++ )
			System.out.print( list[n] + ", " );

		mergeSort( list, 0, list.length-1 );
		System.out.print( "\nSorted: " );
		for (int n = 0; n < list.length; n++ )
			System.out.print( list[n] + ", " );
	}
		// continued on the next page
```

```
	// Precondition: 0 <= start <= end < a.length
	// Postcondition: a[] is sorted in ascending order
	public static void mergeSort (int [] a, int start, int end ) {
		if ( start == end )
			return;

		int middle = (start + end)/2;
		mergeSort( a, start, middle );
		mergeSort( a, middle + 1, end  );
		merge( a, start, middle, end );

	}

// Precondition: 0 <= start <= middle < end <a.length
// Precondition: array a has two parts that have already been sorted: (1) from start to middle and
// (2) from middle + 1 to end
// Postcondition: array a is sorted from index start to end (inclusive)

	public static void merge( int [] a, int start, int middle, int end ) {
		// You write this code. It should be similar to your solution for problem 1.
	}
}
```