Lab Report 01 Justin Schlag

**1. Problem**:

This lab has the goal of sorting a list of words based on three different criteria. These criteria include the number of vowels, the number of consonants, and length of word. I needed to build a program that copies the original text file of words and sorts and copies them. The program should handle any words, and return sorted results, displaying them in the correct order.

**2. Solution Description**:

The program solves the problem by implementing three static methods in the WordHelper class: sortByVowels(), sortByConsonants(), and sortByLength(). Each method creates a copy of the input array to prevent modifying the original array, and uses a bubble sort to sort the copied array based on the given criteria:

sortByVowels() counts vowels and sorts words from smallest to largest based on this count.

sortByConsonants() calculates consonants by subtracting the vowel count from the word length and sorts words accordingly.

sortByLength() sorts the words by their total length in ascending order.

The code uses helper methods like countVowels() to simplify vowel and consonant counts. It sorts and returns the arrays according to the requirements.

**3. Problems Encountered**

Array referencing issues: I didn't copy the input array at first, which caused the sorting methods to modify from the original array. I fixed this problem by creating a different copy of the array inside each sorting method.

Off-by-one error in the bubble sort loop: The sorting loop did not correctly handle the final element because of an incorrect boundary condition (i < copy.length - 1). After adjusting the loop structure, the sorting worked as expected.

Also, I initially missed counting 'y' as a vowel. Once I added Y to the condition in countVowels() the output matched the expected results.

**4. What are the advantages of using an array?**

Fast access: Elements can be accessed directly using an index. Predictable structure: Arrays have a fixed size and contiguous memory allocation, making operations like iteration efficient. Simpler syntax: Arrays provide straightforward operations for indexing and looping.

**5. What are the disadvantages of using an array?**

Arrays have a fixed size. The size of an array is determined at creation and cannot be changed dynamically, which limits flexibility. They are hard to resize: To increase or decrease the array size, you must create a new array and copy the elements, which is inefficient. Limited built-in functionality: Unlike some other data structures like lists, arrays do not support advanced operations like dynamic insertion or deletion.

**6. Describe the range of valid indices for an array.**

The valid indices for an array in Java range from 0 to array.length - 1.

Accessing an index outside this range (e.g., -1 or array.length) will result in an ArrayIndexOutOfBoundsException.

**7.**

Syntax error: The expression "a" + i creates a **String**, but the array abcs is of type **char**.This will cause a **type mismatch error** because you cannot assign a String to a char array element.

Logic Error: The condition i <= abcs.length is incorrect. The valid index range for the array is 0 to abcs.length - 1. When i equals abcs.length, it will attempt to access abcs[26], causing an **ArrayIndexOutOfBoundsException** at runtime.

Logic Error: for (int i = abcs.length - 1; i >= 0; i -= 4) This loop is valid, but it skips four characters per iteration. This means it will only print every fourth letter starting from z.

Final Fixed Code:

```
char[] abcs = new char[26];

// Fill the array with letters 'a' to 'z'

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

   abcs[i] = (char) ('a' + i);

}
```

```
// Print every fourth letter in reverse order

for (int i = abcs.length - 1; i >= 0; i -= 4) {

  System.out.println(abcs[i]);

}
```

**8.**

Logic Error: The condition i = 0 in the loop is a logic error. This line reinitializes i to 0 on every iteration, causing an infinite loop.

The correct condition should increment i in each iteration

Currently, the program counts values divisible by 4 or 5, but not both.
For example, if array[i] = 80, it is divisible by both 4 and 5, but only gets counted under div4.

Check divisibility for both conditions inside the loop

Final fixed code:

```
int[] array = {36, 25, 80, 95, 54};

int div4 = 0;

int div5 = 0;


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

  if (array[i] % 4 == 0 && array[i] % 5 == 0) {

    div4++;

    div5++;

  } else if (array[i] % 4 == 0) {

    div4++;

  } else if (array[i] % 5 == 0) {

    div5++;

  }
}
```

```
System.out.println("Number of values divisible by 4: " + div4);

System.out.println("Number of values divisible by 5: " + div5);
```

**9.**

Logic error: int[] ret = array;

This line does not create a new copy of the array. Instead, it assigns the reference of the original array to ret, meaning both ret and array refer to the same array in memory. Any changes to ret will affect the original array.

Fix:

Create a new array and manually copy each element from the original array:

```
public static int[] copyIntArray(int[] array) {

  int[] ret = new int[array.length];  // Create a new array of the same length

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

    ret[i] = array[i];  // Copy each element

  }

  return ret;

}
```

**10.**

Logic Error: Incorrect Index Assignment

doubledArray[i + array.length] = array[i] * 2.0;

This line is intended to copy double the values of the original array to the second half of doubledArray.

However, the index i + array.length is incorrect for accessing doubledArray. While array.length is correct for offsetting, doubledArray will only be partially initialized before the second loop starts.

**Fix:**
Initialize and manipulate doubledArray in a single loop:

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

   doubledArray[i] = array[i];

   doubledArray[i + array.length] = array[i] * 2.0;

}
```