Testing Document – Andrei Gulapa PROGRAM: Word Sort

Planning Phase

I will make a program to create a binary tree that allows you to analyze the frequency of given words in various ways.

Functions needed: One function for query, one function for node insertion, one function to make an array of words and their frequencies, one function to copy contents of binary tree to the array, one function for merge, one function for merge sort, one function to free the tree, and one function to free the array – a total of **8** functions.

The concept of the problem was easy to grasp but hard to implement. I feel like I was able to do this project more methodically as I've split up the task between several days and had plan out specific goals to reach for each of those days. Two of the main concepts that I struggled with this one was the array of pointers for my array and memory management.

Assistance Received

None

Debugging Phase

Making the insertion and query logic was not complicated for me as it was pretty straight-forward. Initially, I only had one struct to hold all of my node's items; however, I realized that it will be easier for me to keep track of stuff with two structs instead of one, so I made a struct item to hold the word and its frequency and the other struct is the actual node itself.

Tackling the second part of the program was trickier than I thought as I'm not really that confident yet when it comes to playing around with array of pointers. Apparently, I don't need to do a for-loop and malloc space for each single pointer of my array as it's holding a pointer to the root->item (I really don't know how to put it into words) and mallocing for the array of pointers (the one with **) is sufficient. At first I malloced the **array then did a for-loop for each array[i] but that gave me memory issues when freeing them, at least that's what Valgrind told me.

Freeing the arrays and tree were tricky as well as I have to be wary on which order to free them since they're holding pointers. To avoid memory leaks, I free'd the binary tree first before freeing my array. After that, I had no more issues with my memory.

Testing Phase

- My first test case will test will check my program's ability to do an alphabetical sort if all the frequencies for each node are similar.
- My second test case will test if my program works if there is only one node.
- My third test case will test query calls with an empty tree.
- My fourth test case will be a normal test case.

First Test Case Input	**First Test Case Output**
15	abba 1
1 marmalade	bob 1
1 nee	con 1
1 abba	def 1
1 con	eueue 1
1 def	fola 1
1 zebra	heehee 1
1 meep	jol 1
1 jol	marmalade 1
1 unich	meep 1
1 eueue	nee 1
1 bob	que 1
1 fola	soy 1
1 heehee	unich 1
1 que	zebra 1
1 soy	
Second Test Case Input	**Second Test Case Output**
1	lmao 1
1 lmao	
Third Test Case Input	**Third Test Case Output**
5	-1 -1
2 test	-1 -1
2 hi	-1 -1
2 hello	-1 -1
2 bye	-1 -1
2 hee	
Fourth Test Case Input	**Fourth Test Case Output**
25	1 1
1 moana	-1 -1
1 helix	1 2
1 bob	2 1
2 helix	2 1
2 knees	1 2
1 knees	-1 -1
2 knees	bob 5
1 moana	helix 3
1 pop	pop 3
1 pop	knees 2
1 bob	moana 2

1 helix	belch 1
2 helix	knee 1
1 knees	shenandoah 1
2 pop	
1 pop	
1 belch	
1 shenandoah	
1 bob	
2 shenandoah	
1 helix	
1 knee	
1 bob	
1 bob	
2 christmas	