Lab 2 – Report

2SI4

Dates:

- Started: 02/02/2015 – Monday 2, 2015
- Halfway: 04/02/2015 – Wednesday 4, 2015
- Finished: 05/02/2015 – Thursday 5, 2015

Sources:

- I referred to the lecture slides

Description of Algorithms:

- insert(String newword)
    - First this method checks to see if the list is empty. If it is then this method creates a new node at the beginning of the list and adds newword there. If the list if not empty, the method checks to see if newword already exists. If so then the method doesn't do anything. Otherwise; it adds one to the size of the list. After adding one to the size of the list I use a tracker node to go through each node in the list. I first check to see if newword should be added in the beginning of the list, then I check if it needs to be added in the end of the list and finally I check to see if it has to be added in the middle. After the method knows where to add newword it creates a new node where newword is supposed to be inserted and adds newword there.
    - The worst case memory usage is Θ(1) and the worst case run time is Θ(n). The worst case run time is because insert uses method find which has a run time of n, and insert itself has a run time of n.

List = {"A","K","T"} "H"

{"DN", "A", "K", "T"} "H"

{"DN", "A", <- "H", "K", "T"} "H"

- mergeTo(WordLinkedList that)
    - This method first checks to see if this list is empty. If so then it transfers all the words from that to this. It also checks to see if this is empty, if so then it just returns. Last it checks to see if both are not empty. If so then it determines which list comes first alphabetically and sets the head to that. It then runs through to determine which word comes next from this or that, then it adds to the merge list. The values are incremented and the tracker nodes are moved accordingly.
    - The worst case memory usage is Θ(1). The worst case run time is Θ(n). This is because this method goes through this and that list.

List = {“A”,”K”,”T”,”F”} List = {“L”,”M”}

{“DN”, “A”,”F”,”K”,”T”} {“L”,”M”}

{“DN”, “A”,”F”,”K” <- “L”,”T”} {”M”}

{“DN”, “A”,”F”,”K”. “L”, <- “M”,”T”} { }

- WordLinkedList(String[] arrayOfWords)
    - In this constructor the linked list is initialized to 0 and we make a dummy node. A while loop is then used to go through each element in the array and we call the insert function for each work in the array to insert the words into the linked list.
    - The worst case run time is the same as the length of the array. Therefore, the worst case run time is Θ(n).

List = {“A”,”K”,”T”,”F”}

{“DN”} {“A”,”K”,”T”,”F”}

{“DN” <- “A”} {“A”,”K”,”T”,”F”}

{“DN”, “A” <- “K”} {“A”,”K”,”F”,”T”}

{“DN”, “A” <-“T”, “K”} {“A”,”K”,”T”,”F”}