Kyla Wilson

J00813814

Sep 17, 2020

Assignment 2

**Non optimized version**

N = 10,000

A screenshot of a cell phone

Description automatically generated

A screenshot of a cell phone

Description automatically generatedA screenshot of a cell phone

Description automatically generated

N = 100,000

**A screenshot of a cell phone

Description automatically generated**

**A screenshot of a cell phone

Description automatically generated**

**A screenshot of a cell phone

Description automatically generated**

**Optimized version**

N = 10,000

A screenshot of a cell phone

Description automatically generated

A screenshot of a cell phone

Description automatically generated

A screenshot of a cell phone

Description automatically generated

N = 100,000

A screenshot of a cell phone

Description automatically generated

A screenshot of a cell phone

Description automatically generated

A screenshot of a cell phone

Description automatically generated

Provide a description of your optimization strategy and explain how it could reduce

the run time at the cost of additional space, if any.

For my optimization stagey, I decided to add another hash table to store values of a sequence that was already found that way when if the element was found again it would not run the code that counted and compared the sequence count. The optimization code reduces the run time by very little if any where N equals a small value; On the other hand, for larger N values, the optimized version may be considered cause it would reduce runtime if the individual doesn’t mind the extra space complexity.

Present a table (the values of N as rows and the values of S as columns) for each of

the average largest sequence length and the average detection time (in milliseconds).

A screenshot of a cell phone

Description automatically generated

A screenshot of a cell phone

Description automatically generated

Discuss the impact of the size of the hash table on the average detection time.

From running the code, the size of the hash table really makes a speed difference. The bigger the table the faster the code completed. I think this is because there is more room in the hash table and the hash table indexing logic is defined as:  
 int hashIndex = data % tableSize;

Also, the insert in the linked list function is defined as:

void insert(int data){

Node\* currentNodePtr = headPtr->getNextNodePtr();

Node\* prevNodePtr = headPtr;

while (currentNodePtr != 0){

prevNodePtr = currentNodePtr;

currentNodePtr = currentNodePtr->getNextNodePtr();

}

Node\* newNodePtr = new Node();

newNodePtr->setData(data);

newNodePtr->setNextNodePtr(0);

prevNodePtr->setNextNodePtr(newNodePtr);

}

Upon examining the insert code, I can see that whatever that is inserted will be inserted as the last node. If table size is larger, when the insert code runs in the link list, it will be less work because it will not have to traverse through many data nodes to do the insertion due to the larger size of the hash table which means less nodes per linked list.