1. **What is the big-O for the single‐linked list get operation?**

It is the speed of the order of the Nodes. Depending on the implementation, the big-O for the single-linked list get operation has a time complexity, which is the amount of time it takes in the computer for it to run an algorithm. Since it always starts on the head node, it will continue along certain nodes until it reaches the wanted index. However if the implementation is difficult, it will go through every node just to get the index.

1. **Draw a single‐linked list of Integer objects containing the integers 5, 10, 7, and 30 and referenced by head. Complete the following fragment, which adds all Integer objects in a list. Your fragment should walk down the list, adding all integer values to sum.**

head -> [5] -> [10] -> [7] -> [30] -> null

int sum = 0

Node current = head;

while (current != null) {

sum += current.data;

current = current.next;

}

System.out.println(“the sum” + sum);

1. **Using the single‐linked list shown in Figure 2.16, and assuming that head references the first Node and tail references the last Node, write statements to do each of the following:**
   1. **Insert "Bill" before "Tom".**

Node newNode = new Node (“bill”);

Node current = head;

while (current != null && !current.data.equals(“tom”)) {

current = current.next }

if (current != null) {

Node temp = current.next;

Current.Next = newNode;

newNode.next = temp;

}

* 1. **Insert "Sue" before "Sam".**

Node newNode = new Node (“sue”);

Node current = head;

while (current != null && !current.data.equals(“sam”)) {

current = current.next }

if (current != null) {

Node temp = current.next;

Current.Next = newNode;

newNode.next = temp;

}

* 1. **Remove "Bill".**

Node current = head;

Node prev = null;

while (current != null && !current.data.equals(“bill”)) {

prev = current;

current = current.next;

}

If (current != null) {

If (prev == null) {

head = current.next;

} else {

prev.next = current.next;

}

}

* 1. **Remove "Sam".**

Node current = head;

Node prev = null;

while (current != null && !current.data.equals(“sam”)) {

prev = current;

current = current.next;

}

If (current != null) {

If (prev == null) {

head = current.next;

} else {

prev.next = current.next;

}

}

1. **Implement the same Linked List using Java API.**

import java.util.LinkedList;

public class Main {

public static void main(String[] args) {

LinkedList<String> linkedList = new LinkedList<>();

linkedList.add("Tom");

linkedList.add("Sam");

int indexTom = linkedList.indexOf("Tom");

linkedList.add(indexTom, "Bill");

int indexSam = linkedList.indexOf("Sam");

linkedList.add(indexSam, "Sue");

linkedList.remove("Bill");

linkedList.remove("Sam");

System.out.println("Modified LinkedList: " + linkedList);

}

}

1. **Find the differences between ArrayList vs LinkedList. Create a program to compare the performance of both using different size of input (think of how the Big O notation measure the complexity).**

ArrayList - > using Array

Array -> fixated sequence of elements (collection of numbers in order or in a rule)

LinkedList -> Instead of using array, use nodes to store data

Nodes -> Data structure that contains data

import java.util.ArrayList;

import java.util.LinkedList;

import java.util.List;

public class ListPerformanceComparison {

public static void main(String[] args) {

int[] sizes = {1000, 10000, 100000};

for (int size : sizes) {

List<Integer> arrayList = new ArrayList<>();

List<Integer> linkedList = new LinkedList<>();

long startTime = System.nanoTime();

for (int i = 0; i < size; i++) {

arrayList.add(i);

}

long endTime = System.nanoTime();

long arrayAddTime = endTime - startTime;

startTime = System.nanoTime();

for (int i = 0; i < size; i++) {

linkedList.add(i);

}

endTime = System.nanoTime();

long linkedAddTime = endTime - startTime;

startTime = System.nanoTime();

for (int i = 0; i < size; i++) {

arrayList.get(i);

}

endTime = System.nanoTime();

long arrayGetTime = endTime - startTime;

startTime = System.nanoTime();

for (int i = 0; i < size; i++) {

linkedList.get(i);

}

endTime = System.nanoTime();

long linkedGetTime = endTime - startTime;

System.out.println("Size: " + size);

System.out.println("ArrayList add time: " + arrayAddTime + " ns");

System.out.println("LinkedList add time: " + linkedAddTime + " ns");

System.out.println("ArrayList get time: " + arrayGetTime + " ns");

System.out.println("LinkedList get time: " + linkedGetTime + " ns");

System.out.println();

}

}

}