CS 3310 - Data & File Structures
Instructor: Ajay Gupta, Western Michigan University
Lab TA: Yu Guo

Brennan C. Muir

Assignment 1 - Linked Lists and Arrays

PHASE 1: SPECIFICATION

Write a JAVA application to solve the following problem:

1. Generate n random integers from 1 to m
2. Store the generated numbers to a linked list
3. Count how many numbers in the linked list are larger than 50
    a. If there are more than 5 integers that are larger than 50, sort the data in the linked list in a non-decreasing order, then delete the fifth element.
    b. Otherwise, sort the linked list in a non-increasing order and delete the second element
4. Insert the number 10 to linked list in its correct place
5. Then, output the data in your final linked list
6. Redo steps 2-5 with array instead of linked list
7. Perform a theoretical complexity analysis of your design and then express that using asymptotic notation as a function of the input size
8. Empirically measure time and space complexity of your code.

PHASE 2: DESIGN

My program is composed of 3 classes: Node, linkedList and LinkedListsMain. Node handles control of the node within the linked list. Class linkedList is in charge of any commands done to the list. Finally, LinkedListsMain is what drives the program and takes input from the user. LinkedListsMain also controls the linked list and array, making alterations as requested.

PHASE 3: RISK ANALYSIS

It’s possible that overloading the program with too many digits could affect performance.

PHASE 4: VERIFICATION

From the tests that I have run, I did not run into any issues regarding my algorithms.

PHASE 5: TESTING

Integers Generated	**Max Integer**	**Linked List Time (sec)**	**Array Time (sec)**
10	**100**	**0.00407941**	**9.69846E-4**
100	**1,000**	**0.004840782**	**4.9174E-4**
1,000	**10,000**	**0.030573402**	**0.019082931**
10,000	**100,000**	**0.078569023**	**0.005869988**
100,000	**1,000,000**	**1.490390556**	**0.016509491**

PHASE 6: REFINING THE PROGRAM

I would maybe have made a “prettier” output

PHASE 7: PRODUCTION

I prepared a copy of the entire program for Lab TA’s evaluation, as specified by the TA. Then, I sent electronically the copy to the Lab TA.

PHASE 8: MAINTENANCE

To fully benefit from the program evaluation feedback received from the Lab TA, I will perform program maintenance

SOURCE CODE:

```
/*
 * Brennan Muir
 * 9/7/2016
 * CS3310
 * Instructor: Ajay Gupta
 * TA: Yu Guo
 * Assignment 1 - Linked Lists and Arrays
 *
 */

package wmich.edu.cs3310.brennanmuir;

import java.io.*;
import java.util.*;
import java.util.concurrent.TimeUnit;

//  Class Node
class Node {
	protected int data;
	protected Node next, prev;

	// Constructor
	public Node() {
		next = null;
		prev = null;
		data = 0;
	}

	// Constructor
	public Node(int d, Node n, Node p) {
		data = d;
		next = n;
		prev = p;
	}

	// Function to set link to next node
	public void setLinkNext(Node n) {
		next = n;
	}

	// Function to set link to previous node
	public void setLinkPrev(Node p) {
		prev = p;
	}

	// Funtion to get link to next node
	public Node getLinkNext() {
		return next;
	}

	// Function to get link to previous node
	public Node getLinkPrev() {
		return prev;
	}

	// Function to set data to node
	public void setData(int d) {
		data = d;
	}

	// Function to get data from node
	public int getData() {
		return data;
	}
}

// Class linkedList
class linkedList {
	protected Node start;
	protected Node end;
	public int size;

	// Constructor
	public linkedList() {
		start = null;
		end = null;
```

```
        size = 0;
}

// Function to check if list is empty
public boolean isEmpty() {
        return start == null;
}

public void clear() {
        start = null;
}

// Function to get size of list
public int getSize() {
        return size;
}

// Function to insert element at beginning
public void insertAtStart(int val) {
        Node nptr = new Node(val, null, null);
        if (start == null) {
                start = nptr;
                end = start;
        } else {
                start.setLinkPrev(nptr);
                nptr.setLinkNext(start);
                start = nptr;
        }
        size++;
}

// Function to insert element at end
public void insertAtEnd(int val) {
        Node nptr = new Node(val, null, null);
        if (start == null) {
                start = nptr;
                end = start;
        } else {
                nptr.setLinkPrev(end);
                end.setLinkNext(nptr);
                end = nptr;
            }
            size++;
        }

        // Function to insert element at position
        public void insertAtPos(int val, int pos) {
            Node nptr = new Node(val, null, null);
            if (pos == 1) {
                insertAtStart(val);
                return;
            }
            Node ptr = start;
            for (int i = 2; i <= size; i++) {
                if (i == pos) {
                    Node tmp = ptr.getLinkNext();
                    ptr.setLinkNext(nptr);
                    nptr.setLinkPrev(ptr);
                    nptr.setLinkNext(tmp);
                    tmp.setLinkPrev(nptr);
                }
                ptr = ptr.getLinkNext();
            }
            size++;
        }

        // Function to delete node at position
        public void deleteAtPos(int pos) {
            if (pos == 1) {
                if (size == 1) {
                    start = null;
                    end = null;
                    size = 0;
                    return;
                }
                start = start.getLinkNext();
                start.setLinkPrev(null);
                size--;
            return;
        }
        if (pos == size) {
            end = end.getLinkPrev();
            end.setLinkNext(null);
            size--;
        }
        Node ptr = start.getLinkNext();
        for (int i = 2; i <= size; i++) {
            if (i == pos) {
                Node p = ptr.getLinkPrev();
                Node n = ptr.getLinkNext();

                p.setLinkNext(n);
                n.setLinkPrev(p);
                size--;
                return;
            }
            ptr = ptr.getLinkNext();
        }
    }

    // Function to display status of list
    public void display() {
        // if 0 elements
        if (size == 0) {
            System.out.print("empty\n");
            return;
        }
        // if 1 element
        if (start.getLinkNext() == null) {
            System.out.println(start.getData());
            return;
        }
        Node ptr = start;
        System.out.print(start.getData() + "->");
        ptr = start.getLinkNext();
        while (ptr.getLinkNext() != null) {
            System.out.print(ptr.getData() + "->");
```

```
                ptr = ptr.getLinkNext();
            }
            System.out.print(ptr.getData() + "\n");
        }

}

// Class LinkedListsMain
public class LinkedListsMain {
    public static int[] randGenerator(int num, int maxNum) {
        Random random = new Random();
        int randNum;
        int numCount = 0;
        int[] firstArray = new int[num];

        for (int i = 0; i < num; i++) {
            int minNum = 1;

            randNum = random.nextInt(maxNum - minNum + 1) + minNum;
            firstArray[i] = randNum;

            // Check to see if numbers are >50, throw into another array and
            // count
            if (randNum > 50) {
                numCount++;
            }
        }
        return firstArray;
    }

    public static void main(String[] args) {
        int maxNum, minNum, randNum;
        long arrayStart, arrayEnd, listStart, listEnd;
        Scanner kb = new Scanner(System.in);
        int counter = 0;

        // Creating object of linkedList
        linkedList list = new linkedList();
```

```
// Get user input
System.out.println("How many integers would you like to generate?");
int num = kb.nextInt();
System.out.println("Enter a maximum integer: ");
maxNum = kb.nextInt();
// Fill the array with our random numbers
int[] integers = randGenerator(num, maxNum);
// Count how many digits we have >50
for (int i = 0; i < num; i++) {
        if (integers[i] > 50) {
                counter++;
        }
}
int[] firstArray = new int[num];
int ten = 10;
for (int i = 0; i < num; i++) {
        firstArray[i] = integers[i];
}
System.out.println();
// Start list timer
listStart = System.nanoTime();

// Loop that fills our pre-generated random numbers into a list
for (int i = 0; i < num; i++) {
        list.insertAtEnd(integers[i]);
}

// Display our initial Doubly-Linked List
System.out.println("\nDOUBLY-LINKED LIST SECTION\n");
System.out.println("Linked List:");
list.display();

ArrayList<Integer> sortedList = new ArrayList<Integer>(num);
Arrays.sort(firstArray);

// If counter>5, organize in ascending order
if (counter > 5) {
        System.out.println();
        System.out.println("\nGreater than 5");
```

```
		System.out.println("Linked List in ascending order:");

		list.clear();
		for (int i = 0; i < num; i++) {
			list.insertAtEnd(firstArray[i]);
		}
		// Show us our sorted list
		list.display();
		System.out.println();
		System.out.println("\nDELETE THE 5TH ELEMENT");
		System.out.println("Here are the values, in ascending order, with the 5th
element deleted:");
		if (5 < 1 || 5 > list.getSize())
			System.out.println("Invalid position\n");
		else
			list.deleteAtPos(5);

		list.display();

		// Display and clear list

		System.out.println();
		// Add 10
		list.clear();
		System.out.println("Output of the final linked list:");
		sortedList.add(ten);
		Collections.sort(sortedList);

		for (int sorted : firstArray) {
			sortedList.add(sorted);
		}

		Collections.sort(sortedList);
		for (Integer nums : sortedList) {
			list.insertAtEnd(nums);
		}
		list.deleteAtPos(5);
		list.display();
		}
		// If counter <= 5, organize in descending order
		else {
			System.out.println();
			System.out.println("\nLess than or equal to 5");
			System.out.println("Linked List in descending order:");
			list.clear();
			Arrays.sort(firstArray);
			for (int sorted : firstArray) {
				sortedList.add(sorted);
			}

			Collections.sort(sortedList, Collections.reverseOrder());
			for (Integer nums : sortedList) {
				list.insertAtEnd(nums);
			}
			list.display();

			System.out.println();
			System.out.println("\nDELETE THE 2ND ELEMENT");
			System.out.println("Here are the values, in descending order, with the 2nd
element deleted:");
			if (2 < 1 || 2 > list.getSize())
				System.out.println("Invalid position\n");
			else
				list.deleteAtPos(2);

			// Display and clear list
			list.display();
			list.clear();
			System.out.println();
			// Add 10
			System.out.println("Output of the final linked list:");
			sortedList.add(ten);
			Collections.sort(sortedList);
			Collections.sort(sortedList, Collections.reverseOrder());
			for (Integer nums : sortedList) {
				list.insertAtEnd(nums);
			}
```

```
            list.deleteAtPos(2);

            list.display();

        }

        // End list timer
        listEnd = System.nanoTime();
        long listTime = (listEnd - listStart);
        long durationInMs = TimeUnit.NANOSECONDS.toMillis(listTime);
        System.out.println("\nList time in seconds " + (listTime /(Math.pow(10, 9))));

        // ~~~~~~~~~~~~~~~~~~~~~~~~~~ARRAY
SECTION~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
        //
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

        System.out.println("\n\nARRAY SECTION\n");
        // Start array timer
        arrayStart = System.nanoTime();

        // Display our array
        System.out.println("Array:");

        // Loop that fills our pre-generated random numbers into an array
        System.out.print("[");
        for (int i = 0; i < num; i++) {
            firstArray[i] = integers[i];
            System.out.print(firstArray[i] + ",");
        }
        System.out.print("]");
        ArrayList<Integer> sortedArray = new ArrayList<Integer>(num);

        // If counter>5, organize in ascending order
        if (counter > 5) {
            System.out.println();
            System.out.println("\nThere are " + counter + " integers > 50.");
            System.out.println("Here are the values in ascending order:");
            Arrays.sort(firstArray);
```

```
			for (int sorted : firstArray) {
				System.out.print(sorted + " ");
				sortedArray.add(sorted);
			}
			System.out.println();
			System.out.println("\nDELETE THE 5TH ELEMENT");
			System.out.println("Here are the values, in ascending order, with the 5th
element deleted:");
			sortedArray.remove(4);
			for (Integer number : sortedArray) {
				System.out.print(number + " ");
			}
			// Add 10
			System.out.println("\n\nOutput of the final linked list:");
			sortedArray.add(10);
			Collections.sort(sortedArray);

			for (Integer number : sortedArray) {
				System.out.print(number + " ");
			}

		}
		// If counter <= 5, organize in descending order
		else {
			System.out.println();
			System.out.println("\nThere are " + counter + " integers <= 50.");
			System.out.println("Here are the values in descending order:");
			Arrays.sort(firstArray);
			for (int sorted : firstArray) {
				sortedArray.add(sorted);
			}

			Collections.sort(sortedArray, Collections.reverseOrder());
			for (Integer nums : sortedArray) {
				System.out.print(nums + " ");
			}
			System.out.println();
			System.out.println("\nDELETE THE 2ND ELEMENT");
```

```
                              System.out.println("Here are the values, in descending order, with the 2nd
element deleted:");
                              sortedArray.remove(1);
                              for (Integer number : sortedArray) {
                                        System.out.print(number + " ");
                              }
                              // Add 10
                              System.out.println("\n\nOutput of the final array:");
                              sortedArray.add(10);
                              Arrays.sort(firstArray);
                              Collections.sort(sortedArray, Collections.reverseOrder());

                              for (Integer number : sortedArray) {
                                        System.out.print(number + " ");
                              }

                    }
                    // Calculate time and print
                    arrayEnd = System.nanoTime();
                    long arrayTime = (arrayEnd - arrayStart);
                    durationInMs = TimeUnit.NANOSECONDS.toMillis(arrayTime);
                    System.out.println("\n\nList time in seconds " + (arrayTime/(Math.pow(10, 9))));

          }
}
```