CS 3310 – Data and File Structures, Instructor: *<Gupta>*, Western Michigan University Lab TA: *<YG>*

Parker Loomis

**SOFTWARE LIFE CYCLE REPORT – FOR LAB ASSIGNMENT** *4*

**PHASE 1: SPECIFICATION (“What do we build?”)**

* 1)  Read the data in *NameList.txt* file and store the names in a linked-list.
* 2)  Sort the data in the linked-list with the 4 types of sorting methods (bubble sort, selection sort, merge sort, binary-insertion sort (i.e., insertion sort that uses binary search to find the insert position).
* 3)  Besides string type, please let your application handle integer, float, double, char, in case we may need to use those in the future.
* 4)  Please make your application could handle even billions of data (find out what is the maximum size your application can handle in case you cannot get billion size to run at all on your machine).
* 5)  Make sure your application is robust, readable, and has a friendly UI.
* 6)  Analyze these 4 sorting methods (by time and space complexity) empirically as well as theoretically, and specify which is the best one to help YG when linked list is used to store data.
* 7) Repeat steps 1-6 using array-lists instead of linked-lists (find out what is the maximum size your application can handle in case you can not get billion size). For array-lists since you may not know the time for each insertion / deletion for array-lists, assume they are O(1) without loss in generality.

**PHASE 2: DESIGN**

**Program should have 7 classes**

**1. Main method**

**2. LinkedList**

**3. Node**

**4. A class for each of the sorts. Bubble, Insertion, Selection, and Merge.**

**PHASE 3: RISK ANALYSIS (“What can go wrong, and how bad can it be?”)**

No risks (to timetable, cost, human health, etc.) are identified by me.

**PHASE 4: VERIFICATION (“Are the algorithms correct?”)**

The algorithm has only one execution path (a sequential execution). Correctness of the path has been verified by me by analyzing its steps, and their completeness w.r.t. the Specification.

**PHASE 5: CODING**

**package** application;

**import** java.io.File;

**import** java.util.LinkedList;

**import** java.util.Scanner;

**import** queues.Queues;

**import** sorting.Bubble;

**import** sorting.Insertion;

**import** sorting.Merge;

**import** sorting.Selection;

**import** stacks.Stack;

**public** **class** Controller {

**public** Controller() {

// **TODO** Auto-generated constructor stub

}

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

*readInputFile*();

}

@SuppressWarnings({ "unchecked", "static-access" })

**public** **static** **void** readInputFile() {

File file = **new** File("NameList.txt");

/\*\*Read input file

\*

\*/

@SuppressWarnings("rawtypes")

// LinkedList l = new LinkedList();

LinkedList l = **new** LinkedList();

**try** {

Scanner kbd = **new** Scanner(file);

**while** (kbd.hasNextLine()) {

Object i = kbd.next();

l.add(i);

/\*add to linked list

\*

\*/

}

kbd.close();

} **catch** (Exception e) {// catch if file is not found

System.***out***.println("Error: File Not Found");

}

Object[] array = **new** Object[l.size()];

**try** {

Scanner kbd = **new** Scanner(file);

**int** j = 0;

**while** (kbd.hasNextLine()) {

Object i = kbd.next();

array[j] = i;

/\*\*add to an array

\*

\*/

j++;

}

kbd.close();

} **catch** (Exception e) {// catch if file is not found

System.***out***.println("Error: File Not Found");

}

@SuppressWarnings("rawtypes")

Bubble bubble = **new** Bubble();

Selection sel = **new** Selection();

Merge merge = **new** Merge();

Insertion insert = **new** Insertion();

// print(array);

// print(l);

// insert.sort(array);

// insert.sort(l);

// sel.selectionSort(array);

// sel.selection(l);

// merge.mergeSort(array);

// merge.mergelist(l);

// bubble.bubbleSort(array);

// bubble.bubble(l);

// print(array);

// print(l);

}

**public** **static** **void** print(Object[] a) {

**for** (**int** i = 0; i < a.length; i++) {

System.***out***.print(i + "." + a[i] + " ");

}

}

**public** **static** **void** print(LinkedList a) {

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

System.***out***.print(i + "." + a.get(i) + " ");

}

}

}

**package** sorting;

**import** java.util.Collections;

**import** java.util.LinkedList;

**public** **class** Selection {

**public** Selection() {

// **TODO** Auto-generated constructor stub

}

// public static <E> void bubbleSort(E[] unsorted){

**public** **static** <E> **void** selectionSort(E[] list) {

**for** (**int** i = 0; i < list.length - 1; i++) {

**int** iSmallest = i;

**for** (**int** j = i + 1; j < list.length; j++) {

**if** (((Comparable) list[iSmallest]).compareTo((list[j])) > 0) {

iSmallest = j;

/\*\*Compare in order to sort

\*

\*/

}

}

E iSwap = list[iSmallest];

list[iSmallest] = list[i];

list[i] = iSwap;

}

}

**public** **void** selection(LinkedList l) {

Collections.*sort*(l);

}

}

**package** sorting;

**import** java.util.Collections;

**import** java.util.LinkedList;

**public** **class** Merge {

**public** Merge() {

// **TODO** Auto-generated constructor stub

}

**public** **static** **void** mergeSort(Object[] a) {

**if** (a.length >= 2) {

Object[] left = **new** String[a.length / 2];

Object[] right = **new** String[a.length - a.length / 2];

**for** (**int** i = 0; i < left.length; i++) {

left[i] = a[i];

}

**for** (**int** i = 0; i < right.length; i++) {

right[i] = a[i + a.length / 2];

}

*mergeSort*(left);

*mergeSort*(right);

*merge*(a, left, right);

}

}

**public** **static** **void** merge(Object[] result, Object[] left, Object[] right) {

**int** i1 = 0;

**int** i2 = 0;

**for** (**int** i = 0; i < result.length; i++) {

**if** (i2 >= right.length || (i1 < left.length && ((Comparable) left[i1]).compareTo(right[i1]) < 0)) {

/\*\*Compare in order to sort

\*

\*/

result[i] = left[i1];

i1++;

} **else** {

result[i] = right[i2];

i2++;

}

}

}

**public** **void** mergelist(LinkedList l) {

Collections.*sort*(l);

}

}

**package** sorting;

**import** java.util.Collections;

**import** java.util.LinkedList;

**import** java.util.List;

**import** java.util.ArrayList;

**public** **class** Bubble<T> {

**public** **static** <T> **void** bubbleSort(T[] unsorted) {

**for** (**int** iter = 1; iter < unsorted.length; iter++) {

**for** (**int** inner = 0; inner < (unsorted.length - iter); inner++) {

**if** ((((Comparable) (unsorted[inner])).compareTo(unsorted[inner + 1])) > 0) {

/\*\*Compare in order to sort

\*

\*/

T tmp = unsorted[inner];

unsorted[inner] = unsorted[inner + 1];

unsorted[inner + 1] = tmp;

}

}

}

}

**public** **void** bubble(LinkedList l) {

Collections.*sort*(l);

}

}

**package** sorting;

**import** java.util.Collections;

**import** java.util.LinkedList;

**public** **class** Insertion {

**public** Insertion() {

// **TODO** Auto-generated constructor stub

}

**public** **static** **void** sort(Object[] inputArray) {

Object temp;

**int** i;

**for** (**int** j = 1; j < inputArray.length; j++) {

temp = inputArray[j];

i = j - 1;

**while** (i >= 0) {

**if** (((Comparable) temp).compareTo(inputArray[i]) > 0) {

**break**;

/\*\*Compare in order to sort

\*

\*/

}

inputArray[i + 1] = inputArray[i];

i--;

}

inputArray[i + 1] = temp;

}

}

**public** **void** sort(LinkedList l) {

Collections.*sort*(l);

}

}

**PHASE 6: TESTING (“Did we build it correctly?”)**

Before Sorting phase

0.Rawan

1.Brent

2.Larry

3.Andrew

4.Benjamin

5.Angelo

6.Andrew

7.Gerald

8.Kai

9.John

10.Connor

11.Adam

12.Daniel

13.Gideon

14.Parker

15.Alex

16.Joshua

17.Gregory

18.Justen

19.Nicholas

20.Matthew

21.Gabriel

22.Kenneth

23.Antonio

24.Brian

25.Brandon

26.James

27.Luis

28.Axel

29.Tyler

30.Vittini

31.James

32.Nicholas

33.Bishop

34.Justin

35.Eric

36.Nathan

37.Mariia

38.Kevin

39.Joshua

40.Brennan

41.Robert

42.Nicholas

43.Jaden

44.Joshua

45.Grace

46.Jacob

47.Mohammed

48.Cody

49.Katherine

After Sorting is complete

0.Adam

1.Alex

2.Andrew

3.Andrew

4.Angelo

5.Antonio

6.Axel

7.Benjamin

8.Bishop

9.Brandon

10.Brennan

11.Brent

12.Brian

13.Cody

14.Connor

15.Daniel

16.Eric

17.Gabriel

18.Gerald

19.Gideon

20.Grace

21.Gregory

22.Jacob

23.Jaden

24.James

25.James

26.John

27.Joshua

28.Joshua

29.Joshua

30.Justen

31.Justin

32.Kai

33.Katherine

34.Kenneth

35.Kevin

36.Larry

37.Luis

38.Mariia

39.Matthew

40.Mohammed

41.Nathan

42.Nicholas

43.Nicholas

44.Nicholas

45.Parker

46.Rawan

47.Robert

48.Tyler

49.Vittini

When adding number to test as well.

0.Rawan

1.Brent

2.Larry

3.Andrew

4.Benjamin

5.Angelo

6.Andrew

7.Gerald

8.Kai

9.John

10.Connor

11.Adam

12.Daniel

13.Gideon

14.Parker

15.Alex

16.Joshua

17.Gregory

18.Justen

19.Nicholas

20.Matthew

21.Gabriel

22.Kenneth

23.Antonio

24.99

25.5

26.6

27.1

28.0

29.4

30.3

31.8

32.Brian

33.Brandon

34.James

35.Luis

36.Axel

37.Tyler

38.Vittini

39.James

40.Nicholas

41.Bishop

42.Justin

43.Eric

44.Nathan

45.Mariia

46.Kevin

47.Joshua

48.Brennan

49.Robert

50.Nicholas

51.Jaden

52.Joshua

53.Grace

54.Jacob

55.Mohammed

56.Cody

57.Katherine

0.0

1.1

2.3

3.4

4.5

5.6

6.8

7.99

8.Adam

9.Alex

10.Andrew

11.Andrew

12.Angelo

13.Antonio

14.Axel

15.Benjamin

16.Bishop

17.Brandon

18.Brennan

19.Brent

20.Brian

21.Cody

22.Connor

23.Daniel

24.Eric

25.Gabriel

26.Gerald

27.Gideon

28.Grace

29.Gregory

30.Jacob

31.Jaden

32.James

33.James

34.John

35.Joshua

36.Joshua

37.Joshua

38.Justen

39.Justin

40.Kai

41.Katherine

42.Kenneth

43.Kevin

44.Larry

45.Luis

46.Mariia

47.Matthew

48.Mohammed

49.Nathan

50.Nicholas

51.Nicholas

52.Nicholas

53.Parker

54.Rawan

55.Robert

56.Tyler

57.Vittini

**PHASE 7: REFINING THE PROGRAM (“Add bells and whistles to the program”)**

No refinements are needed. In this program, I have already included all required features.

**PHASE 8: 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 9: MAINTENANCE**

I should create a better interface in order to make it easier to use and test. I also had trouble with sorting the linked lists. It was hard to figure out how to swap them since they are linked to the previous and next node in the list.