# Comp151 Lab10

In Lab10 you will implement **five** applications: two (#2 and #5) that utilize a **sorted list**, and three (#1, #3, and #4) that utilize **efficient searching algorithms**. Please read the instructions carefully.

1. Implement *trinarySearch* method.

The **recursive** **trinarySearch** method returns true or false depending if the element was found or not.

The trinarySearch method works in a similar manner to a binary search except it uses two mid values that “divide” the array into three portions. So, it needs to consider three recursive scenarios:

* desired item is smaller than the element at index mid1
* desired item is greater than the element at index mid2
* desired item is smaller than the element at index mid2 but is greater than the element at index mid1

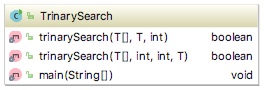
Utilize compareTo method, save the returned value(s) and use them in comparisons.

Use the following formulas to calculate mid indexes:

int mid1 = left + (right - left)/3;

int mid2 = right – (right - left)/3;

### UML diagram:



### See sample run:

Accounts are:

[0] 5658845

[1] 8080152

[2] 1005231

[3] 4520125

[4] 4562555

[5] 6545231

[6] 7895122

[7] 5552012

[8] 3852085

[9] 8777541

[10] 5050552

[11] 7576651

[12] 8451277

[13] 7825877

[14] 7881200

[15] 1302850

[16] 1250255

[17] 4581002

Sorted accounts are:

[0] 1005231

[1] 1250255

[2] 1302850

[3] 3852085

[4] 4520125

[5] 4562555

[6] 4581002

[7] 5050552

[8] 5552012

[9] 5658845

[10] 6545231

[11] 7576651

[12] 7825877

[13] 7881200

[14] 7895122

[15] 8080152

[16] 8451277

[17] 8777541

trinarySearch: element 7881200 is found true

PASS

trinarySearch: element 7881199 is found false

PASS

trinarySearch: element 7881201 is found false

PASS

trinarySearch: element 2222222 is found false

PASS

trinarySearch: element 9999999 is found false

PASS

trinarySearch: element 0000000 is found false

PASS

trinarySearch: element 1111111 is found false

PASS

trinarySearch: element 1005231 is found true

PASS

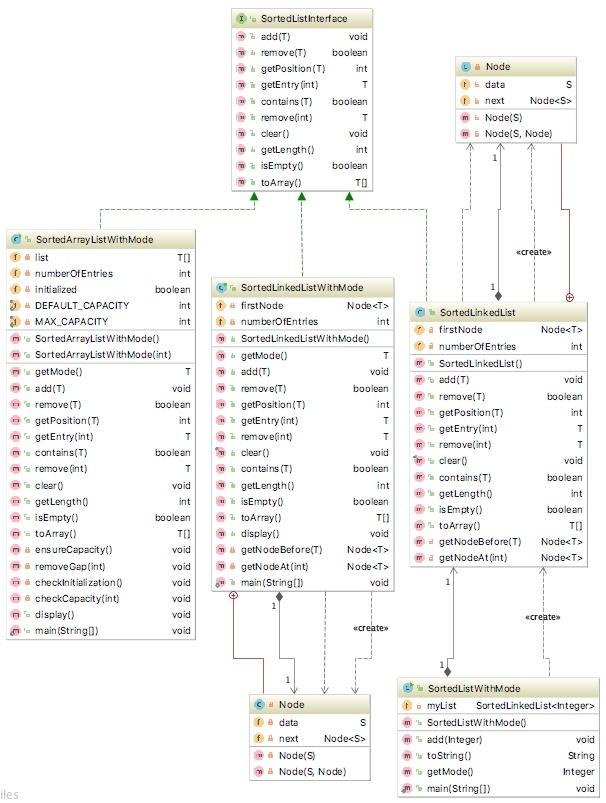
trinarySearch: element 8777541 is found true

PASS

\*\*\* Done \*\*\*

1. The *mode* of a list of values is the value having the greatest frequency. In your pre-lab you wrote an algorithm to find the mode of the sorted list. Your task is to implement this method in three ways:
   1. Assuming an array-based implementation manipulate **the array directly** - implement method getMode in SortedArrayListWithMode.java. Please note that the list is 1 based.
   2. Assuming a linked implementation manipulate **the linked list directly** - implement method getMode in SortedLinkedListWithMode.java.
   3. Use **only** **sorted list operations** like getEntry and getLength - implement method getMode in SortedListWithMode.java. Please note that the list is 1 based.

UML diagram:

****

### See sample run:

---> mode is null; mode count is 0

The mode of the empty list should be null, got: null

The data has 1 element(s): 9

---> mode is 9; mode count is 1

The mode should be 9, got: 9

The data has 2 elements: 9 13

---> mode is 9; mode count is 1

The mode should be 9, got: 9

The data has 3 elements: 9 13 13

---> mode is 13; mode count is 2

The mode should be 13, got: 13

The data has 3 elements: 9 9 13

---> mode is 9; mode count is 2

The mode should be 9, got: 9

The data has 10 elements: 0 1 2 3 4 5 6 7 8 9

---> mode is 0; mode count is 1

The mode should be 0, got: 0

The data has 55 elements: 0 1 1 2 2 2 3 3 3 3 4 4 4 4 4 5 5 5 5 5 5 6 6 6 6 6 6 6 7 7 7 7 7 7 7 7 8 8 8 8 8 8 8 8 8 9 9 9 9 9 9 9 9 9 9

---> mode is 9; mode count is 10

The mode should be 9, got: 9

The data has 133 elements: 0 1 1 2 2 2 3 3 3 3 4 4 4 4 4 5 5 5 5 5 5 6 6 6 6 6 6 6 7 7 7 7 7 7 7 7 8 8 8 8 8 8 8 8 8 9 9 9 9 9 9 9 9 9 9 9 10 10 11 11 11 12 12 12 12 13 13 13 13 13 14 14 14 14 14 14 15 15 15 15 15 15 15 16 16 16 16 16 16 16 16 17 17 17 17 17 17 17 17 17 18 18 18 18 18 18 18 18 18 18 19 19 19 19 19 19 19 19 19 19 19 20 20 20 20 20 20 20 20 20 20 20 20

---> mode is 20; mode count is 12

The mode should be 20, got: 20

The data has 147 elements: 0 1 1 2 2 2 3 3 3 3 4 4 4 4 4 5 5 5 5 5 5 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 7 7 7 7 7 7 7 7 8 8 8 8 8 8 8 8 8 9 9 9 9 9 9 9 9 9 9 9 10 10 11 11 11 12 12 12 12 13 13 13 13 13 14 14 14 14 14 14 15 15 15 15 15 15 15 16 16 16 16 16 16 16 16 17 17 17 17 17 17 17 17 17 18 18 18 18 18 18 18 18 18 18 19 19 19 19 19 19 19 19 19 19 19 20 20 20 20 20 20 20 20 20 20 20 20

---> mode is 6; mode count is 21

The mode should be 6, got: 6

\*\*\* Done \*\*\*

1. Implement *interpolationSearch* method

An **interpolation search** assumes that the data in an array is sorted and uniformly distributed.

Whereas a *binary search* always looks at the *middle* item in an array, an *interpolation search* looks where the sought-for item is more likely to occur. For example, if you searched your telephone book for Victoria Appleseed, you probably would look near its beginning rather than its middle. And if you discovered many Appleseeds, you would look near the last Appleseed.

Hence the difference between the *binary search* and the *interpolation search* is that the *binary search* always splits the array in half and inspects the middle element, where the *interpolation search* calculates a probable position *p*, where the value should be found in accordance to the distribution of values, and splits the array at *p*. If the array contains numbers 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 and we are looking for 9 the binary search needs three steps – split at 5, split at 8, split at 9 (found). The interpolation search calculates the probable position *p* (index 9) and immediately finds the value. The expected [complexity](http://www.programming-algorithms.net/article/44682/Asymptotic-complexity) of the interpolation search in O(log(log n)).

Instead of looking at the element a[mid] of an array a, as the binary search would, an interpolation search examines a[p], where *p* is calculated as follow:

**int p = left + ((desiredItem - a[left]) \* (right - left)/(a[right] - a[left]));**

Implement the interpolation search for an array using recursion in InterpolationSearch.java class.

### UML diagram:

### /Users/ania/Desktop/Untitled.jpeg

### See sample run :

Searching uniformly distributed sorted array: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22

--> p = 13

PASSES: 14 was found at index 13

PASSES: 1 was found at index 0

--> p = 21

PASSES: 22 was found at index 21

--> p = 19

PASSES: 20 was found at index 19

PASSES: 23456 was not found

PASSES: -6 was not found

--> p = 11

PASSES: 12 was found at index 11

Searching non-uniformly distributed sorted array: -10 -5 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 200 700 12345

--> p = 0

--> p = 1

--> p = 2

--> p = 3

--> p = 4

--> p = 5

--> p = 6

--> p = 7

--> p = 8

--> p = 9

--> p = 10

--> p = 11

--> p = 12

PASSES: 14 was found at index 13

PASSES: -10 was found at index 0

--> p = 21

PASSES: 12345 was found at index 21

--> p = 0

--> p = 1

--> p = 2

--> p = 3

--> p = 4

--> p = 5

--> p = 6

--> p = 7

--> p = 8

--> p = 9

--> p = 10

--> p = 11

--> p = 12

--> p = 13

--> p = 14

--> p = 15

--> p = 16

--> p = 17

--> p = 18

PASSES: 200 was found at index 19

PASSES: 23456 was not found

--> p = 0

PASSES: -6 was not found

--> p = 0

--> p = 1

--> p = 2

--> p = 3

--> p = 4

--> p = 5

--> p = 6

--> p = 7

--> p = 8

--> p = 9

--> p = 10

PASSES: 12 was found at index 11

\*\*\* Done \*\*\*

1. Implement *intervalSearch* method

Consider an array of *n* numerical values in sorted order and a list of numerical target values. Your goal is to compute **the smallest range** of array indices that contains all of the target values. Please note that:

* if a target value is smaller than data[0], the range should start with -1.
* if a target value is larger than data[n - 1], the range should end with n.

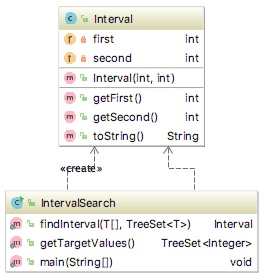
For example, given the array: 5 8 10 13 15 20 22 26 and the target values [8, 2, 9, 17], the range is (-1, 5).

Devise and implement an efficient algorithm that solves this problem.

HINT:

* 1. Find the max and min in the list of target values.
  2. Utilize the **iterative binary search algorithm** to find the position that those values would have if inserted into the list of sorted data.

### UML diagram:



### See sample run :

How many elements in the array?

13

The sorted data is:

[0]=8 [1]=8 [2]=10 [3]=11 [4]=16 [5]=22 [6]=27 [7]=35 [8]=36 [9]=36 [10]=45 [11]=48 [12]=50

Enter the list of integer values separated by spaces (all on one line), or just press enter if you are done.

2 3

Target list is [2, 3]

The interval is: (-1, -1)

Enter the list of integer values separated by spaces (all on one line), or just press enter if you are done.

60 70

Target list is [60, 70]

The interval is: (13, 13)

Enter the list of integer values separated by spaces (all on one line), or just press enter if you are done.

11 27

Target list is [11, 27]

The interval is: (3, 6)

Enter the list of integer values separated by spaces (all on one line), or just press enter if you are done.

15

Target list is [15]

The interval is: (3, 4)

Enter the list of integer values separated by spaces (all on one line), or just press enter if you are done.

9 29

Target list is [9, 29]

The interval is: (1, 7)

Enter the list of integer values separated by spaces (all on one line), or just press enter if you are done.

3 46

Target list is [3, 46]

The interval is: (-1, 11)

Enter the list of integer values separated by spaces (all on one line), or just press enter if you are done.

15 90

Target list is [15, 90]

The interval is: (3, 13)

Enter the list of integer values separated by spaces (all on one line), or just press enter if you are done.

\*\*\* Done \*\*\*

1. In certain computer networks, a message is not sent as a continuous stream of data. Instead it is divided into pieces, called *packets*, and sent a packet at a time. The packets might not arrive at their destination in the same order as the one in which they were sent. To enable the receiver to assemble the packets in their correct order, each packet contains a sequence number. For example, to send the message “I love Data Structures” three characters at a time, the packets would be created as follows:

0 I l

1 ove

2 Da

3 ta

4 Str

5 uct

6 ure

7 s

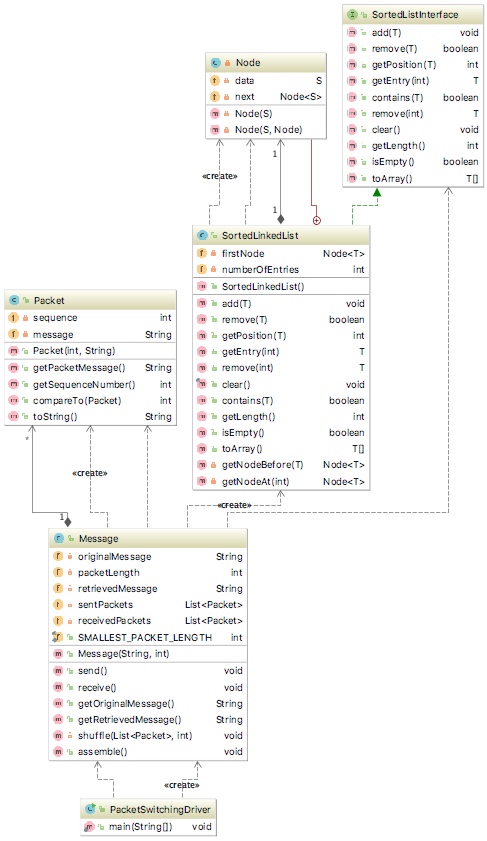
Regardless of when the packets arrive, the receiver can order the packets by their sequence numbers to determine the message.

Write an application that prompts the user for the message and the packet length. Next the application “sends” the message, “receives” the packets, and “assembles” the message back. The driver for the application is provided and should not be changed. You need to finish the Packet class and the Message class based on the provided UML diagram.

Note on the Message class:

* send method – breaks the message into packets of the given size
* shuffle method – this is a **recursive** method that rearranges the packet objects so they are in no particular order
* receive method – shuffles the packets (by calling the shuffle method) so they appear in random order
* assemble method – constructs the message back by using a **sorted list**

UML diagram:

****

### See sample run:

Run #1:

Enter the message to be sent

I love Data Structures

Enter the packet size (no smaller than 3)

3

The original message is "I love Data Structures"

Sent packets:[0 I l, 1 ove, 2 Da, 3 ta , 4 Str, 5 uct, 6 ure, 7 s]

Received packets:[5 uct, 4 Str, 6 ure, 1 ove, 0 I l, 7 s, 3 ta , 2 Da]

The message assembled back is "I love Data Structures"

\*\*\* Done \*\*\*

Run #2:

Enter the message to be sent

abc

Enter the packet size (no smaller than 3)

3

The original message is "abc"

Sent packets:[0 abc]

Received packets:[0 abc]

The message assembled back is "abc"

\*\*\* Done \*\*\*

Run#3:

Enter the message to be sent

ab

Enter the packet size (no smaller than 3)

3

The original message is "ab"

Sent packets:[0 ab]

Received packets:[0 ab]

The message assembled back is "ab"

\*\*\* Done \*\*\*

Run#4:

Enter the message to be sent

Happy Halloween !!!

Enter the packet size (no smaller than 3)

4

The original message is "Happy Halloween !!!"

Sent packets:[0 Happ, 1 y Ha, 2 llow, 3 een , 4 !!!]

Received packets:[0 Happ, 1 y Ha, 3 een , 2 llow, 4 !!!]

The message assembled back is "Happy Halloween !!!"

\*\*\* Done \*\*\*