# Comp151 Lab10

In Lab10 you will implement **five** applications:

* #2 utilizes a **sorted list**,
* #1, 3, 4, and 5 utilize **efficient searching algorithms**.

Please read the instructions carefully.

1. Implement *ternarySearch* method.

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

The ternarySearch 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:

A screenshot of a cell phone

Description automatically generated

### 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

ternarySearch: element 7881200 is found true

PASS

ternarySearch: element 7881199 is found false

PASS

ternarySearch: element 7881201 is found false

PASS

ternarySearch: element 2222222 is found false

PASS

ternarySearch: element 9999999 is found false

PASS

ternarySearch: element 0000000 is found false

PASS

ternarySearch: element 1111111 is found false

PASS

ternarySearch: element 1005231 is found true

PASS

ternarySearch: element 8777541 is found true

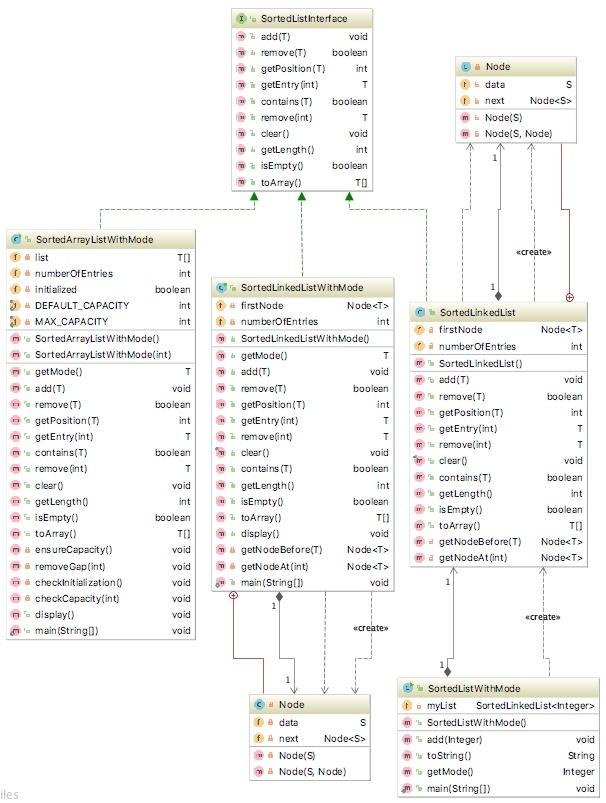
PASS

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

Process finished with exit code 0

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.

### 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.

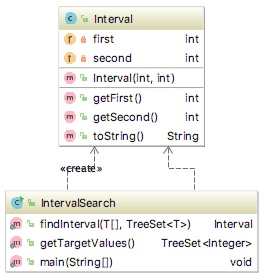
For example, given the array: 5 8 10 13 15 20 22 26 and the target values [8, 2, 9, 17], the range is (-1, 5).

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 #1 (no seed):

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 \*\*\*

### See sample run #2 (seed set to 7 and added tracing):

How many elements in the array?

17

The sorted data is:

[0]=6 [1]=10 [2]=15 [3]=19 [4]=19 [5]=23 [6]=31 [7]=40 [8]=40 [9]=44 [10]=44 [11]=46 [12]=54 [13]=55 [14]=60 [15]=62 [16]=63

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

44

Target list is [44]

Search on the smallest value

Left Boundary set to 9

Search on the largest value

Left: 9 Middle: 12 Right: 16

Left: 9 Middle: 10 Right: 11

Left: 9 Middle: 10 Right: 11

Right Boundary set to 10

The interval is: (9, 10)

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

19

Target list is [19]

Search on the smallest value

Left Boundary set to 3

Search on the largest value

Left: 3 Middle: 9 Right: 16

Left: 3 Middle: 5 Right: 8

Left: 3 Middle: 3 Right: 4

Left: 3 Middle: 3 Right: 4

Right Boundary set to 4

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.

40

Target list is [40]

Search on the smallest value

Left Boundary set to 7

Search on the largest value

Left: 7 Middle: 11 Right: 16

Left: 7 Middle: 8 Right: 10

Left: 7 Middle: 8 Right: 10

Right Boundary set to 8

The interval is: (7, 8)

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

19 44

Target list is [19, 44]

Search on the smallest value

Left Boundary set to 3

Search on the largest value

Left: 3 Middle: 9 Right: 16

Left: 3 Middle: 9 Right: 16

Right Boundary set to 10

The interval is: (3, 10)

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

20 41

Target list is [20, 41]

Search on the smallest value

Left Boundary set to 4

Search on the largest value

Left: 4 Middle: 10 Right: 16

Left: 4 Middle: 6 Right: 9

Left: 7 Middle: 8 Right: 9

Left: 9 Middle: 9 Right: 9

Left: 9 Middle: 9 Right: 8

Right Boundary set to 9

The interval is: (4, 9)

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

1 2

Target list is [1, 2]

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.

1 44

Target list is [1, 44]

Search on the smallest value

Left Boundary set to -1

Search on the largest value

Left: -1 Middle: 7 Right: 16

Left: 8 Middle: 12 Right: 16

Left: 8 Middle: 9 Right: 11

Left: 8 Middle: 9 Right: 11

Right Boundary set to 10

The interval is: (-1, 10)

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

90 95

Target list is [90, 95]

The interval is: (17, 17)

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

1 90

Target list is [1, 90]

Search on the smallest value

Left Boundary set to -1

Search on the largest value

Right Boundary set to 17

The interval is: (-1, 17)

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

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

Process finished with exit code 0

1. Open Search2D class and implement **efficient,** **non-recursive** search method that searches for a desired item in a rectangular two-dimensional array of integers where elements are sorted within each row and within each column (see the following two-dimensional array).

**int** matrix[][] = {  
 {10, 20, 21, 40},  
 {15, 25, 26, 45},  
 {27, 29, 30, 48},  
 {32, 33, 34, 50}};

If the element is found, the method prints its position, otherwise it prints "not found".

The algorithm **must** take into consideration the **nature of the sorted data**.

### See sample run :

\*\*\* These should be successful searches \*\*\*

Searching for 10

checking 40

checking 21

checking 20

checking 10

10 found at [0, 0]

Searching for 20

checking 40

checking 21

checking 20

20 found at [0, 1]

Searching for 21

checking 40

checking 21

21 found at [0, 2]

Searching for 40

checking 40

40 found at [0, 3]

Searching for 15

checking 40

checking 21

checking 20

checking 10

checking 15

15 found at [1, 0]

Searching for 25

checking 40

checking 21

checking 26

checking 25

25 found at [1, 1]

Searching for 26

checking 40

checking 21

checking 26

26 found at [1, 2]

Searching for 45

checking 40

checking 45

45 found at [1, 3]

Searching for 27

checking 40

checking 21

checking 26

checking 30

checking 29

checking 27

27 found at [2, 0]

Searching for 29

checking 40

checking 21

checking 26

checking 30

checking 29

29 found at [2, 1]

Searching for 30

checking 40

checking 21

checking 26

checking 30

30 found at [2, 2]

Searching for 48

checking 40

checking 45

checking 48

48 found at [2, 3]

Searching for 32

checking 40

checking 21

checking 26

checking 30

checking 34

checking 33

checking 32

32 found at [3, 0]

Searching for 33

checking 40

checking 21

checking 26

checking 30

checking 34

checking 33

33 found at [3, 1]

Searching for 34

checking 40

checking 21

checking 26

checking 30

checking 34

34 found at [3, 2]

Searching for 50

checking 40

checking 45

checking 48

checking 50

50 found at [3, 3]

\*\*\* These should be unsuccessful searches \*\*\*

Searching for 28

checking 40

checking 21

checking 26

checking 30

checking 29

checking 27

checking 32

28 not found

Searching for 5

checking 40

checking 21

checking 20

checking 10

5 not found

Searching for 100

checking 40

checking 45

checking 48

checking 50

100 not found found