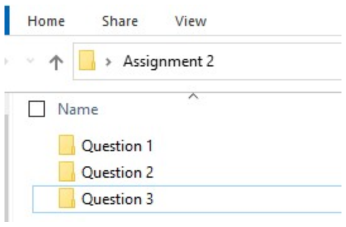
Data Structures and Algorithms: Assignment 2

Due: 7th of June - 100 marks possible (20% of final grade)

When submitting, **zip up your entire Netbeans projects** (**Try to create and include your ".jar" files of your Noise Removing and Maze projects**) into a folder with the following structured file name:

lastname_firstname_studentID.zip (using your own name and ID)



Assignment 2.zip contains (see the image on the side):

- Question 1 folder (for entire question 1 project)
- Question 2 folder (for entire question 2 project)
- Question 3 folder (for entire question 3 project)

and submit the Assignment 2 zipped folder on AUT Canvas BEFORE the due date and time. Late submissions will incur a penalty. Antiplagiarism software will be used on all assignment submissions, so make sure you submit **YOUR OWN** work and **DO NOT SHARE** your answer with others. Any breaches will be reported and will result in an automatic fail and possible removal from the course.

The projects source code templates are given. Please **DO NOT** change the package name.

Question 1) Binary Tree and Memo Sorting (50%)

The purpose of this question is to have an opportunity to understand and manipulate a binary tree and utilise the binary tree to create a program that sorts memos by date or title and can search the memo by a searching key. The binary tree is built by a key of node (Key can be Date type date, or String type title)

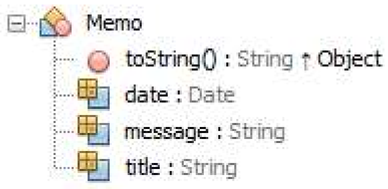
Memo Class

Create a Memo Class which stores memo's **date, title** and **message**.

The date type must be java.util.Date; (See example of converting java String to java Date.)

Memo Class has a "**toString**" method which returns date, title, message of the memo and each of the details should be on separated lines. Such as:

```
Sun Nov 19 00:00:00 NZDT 2023

Work 06

message 3
```



```
Memo
  toString() : String ↑ Object
  date : Date
  message : String
  title : String
```

Node Class

Create a Node Class which has element, key, and linkers parts.

The “**element**” of a Node class is a generic type. It references to any types of objects. In this project, it references to a Memo object.

The “**key**” is for comparing different nodes. It must be a Comparable Object.

If the key is a Date type, then the tree must be built by the dates of memos.

If the key is a String type, then the tree must be built by the titles of memos.

Therefore the key must be a generic type.

Node Class has a node linker named “**left**” which references to a node that has smaller key value.

Node Class has a node linker named “**Right**” which references to a node that has greater key value.

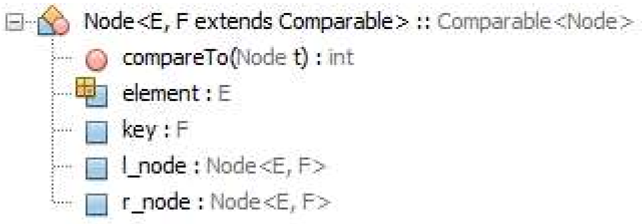
Node Class has a “**compareTo**” method. “**compareTo**” method takes a Node object in and compares to the current node by their keys.

It returns:

= 0: if current node’s element equals to the argument node’s element.

< 0: if current node’s element is less than the argument node’s element.

> 0: if current node’s element is greater than the argument node’s element.



```
Node<E, F extends Comparable> :: Comparable<Node>
    compareTo(Node t) : int
    element : E
    key : F
    l_node : Node<E, F>
    r_node : Node<E, F>
```

BinaryTree Class

Create a BinaryTree Class which builds and manages a binary tree.

BinaryTree Class has a Node object named “**root**” to reference to the root of a binary tree.

BinaryTree Class has an int variable named “**number_of_nodes**” to store the number of nodes that a binary tree has.

BinaryTree Class has a Node array named “**nodeList**” to store sorted nodes.

BinaryTree Class has an “**addElement**” method. It takes an element and a key then it creates a Node object. The Node object loads element and key and is passed to addNode() method to attach on the tree.

BinaryTree Class has an “**addNode**” method. It takes a root and a new node to create a tree when the tree is empty or adds a new node to the binary tree.

BinaryTree Class has a “**reverseOrder**” method. It manipulates the binary tree. By default, if we call in-order traversal method, it displays nodes’ details in the order of from smallest key value to the largest key value. If reverseOrder method has been called, then traversal method displays nodes’ details in the order of from largest key value to the smallest key value.

The time complexity (Big *O*) of “**reverseOrder**” method must be *n*. Please do not rebuild your tree. This would be $n\log_2 n$. Comments your solution, please.

BinaryTree Class has a “**searchElement**” method. It takes a key which is a generic type then creates a target node object and loads key to the target node. The target node is passed to the searchNode() method to do the searching job. “**searchElement**” method returns a generic type element if searchNode() returns a Node object. Otherwise, it returns null. The target node only contains a key value for searching.

For example:

```
Node targetNode = new Node();

targetNode.key = “title 01”;

Node resultNode = tree.searchNode(tree.root, targetNode);
```

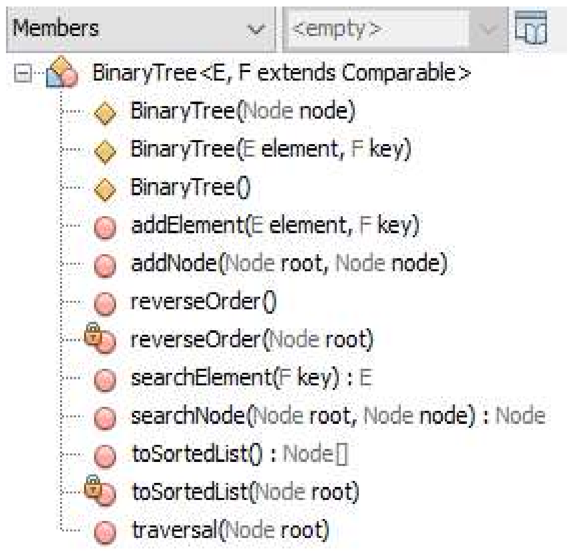
BinaryTree Class has a “**searchNode**” method. It takes a root and target node. It returns a node if it finds the node. Otherwise, it returns null.

In this assignment, The searchingResultNode.element references to a Memo object which contains all the details of memo.

For better understanding of adding and searching memo, please see diagrams after MemoManager class.

BinaryTree Class has a “**toSortedList**” method. It travels each node of the current tree and stores the nodes to the “**nodeList**”.

BinaryTree Class has a "**traversal**" method. It travels each node on the current tree and display node.elements' details in the order of from smallest key value to the largest key value, or from the largest key value to the smallest key value (it depends on whether reverseOrder() has been called).



```
Members                    <empty>
BinaryTree<E, F extends Comparable>
    BinaryTree(Node node)
    BinaryTree(E element, F key)
    BinaryTree()
    addElement(E element, F key)
    addNode(Node root, Node node)
    reverseOrder()
    reverseOrder(Node root)
    searchElement(F key) : E
    searchNode(Node root, Node node) : Node
    toSortedList() : Node[]
    toSortedList(Node root)
    traversal(Node root)
```

Helps on reverseOrder() method

If you do not know how this can be done, please draw two binary trees with the same data inputs.

When you draw the second tree, you put a node with a bigger key value on the left and smaller key value on the right.

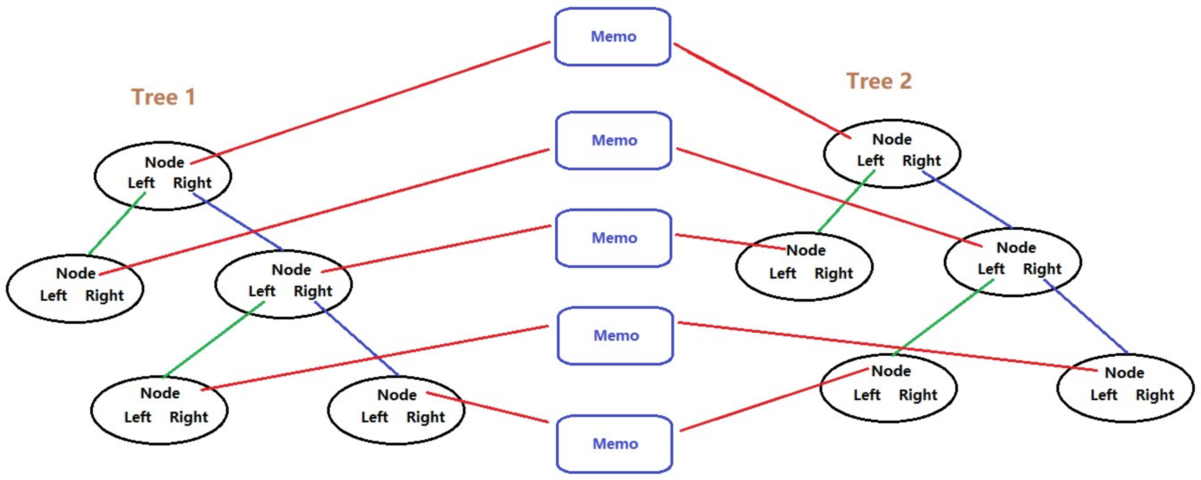
Check the difference between these two trees. I am happy to discuss it with you in the lab.

Wish it helps 😊

Fully comments on the methods of addNode(), searchNode(), reverseOrder() and traversal().

You may lose 20% of the marks if you don't give the comments to those methods.

MemoManager Class



MemoManager class has a BinaryTree object named "**bTreeDate**" which stores Nodes. All nodes are arranged by the memo's date (node.key is Date type).

MemoManager class has a BinaryTree object named "**bTreeTitle**" which stores Nodes. All nodes are arranged by the memo's title (node.key is String type).

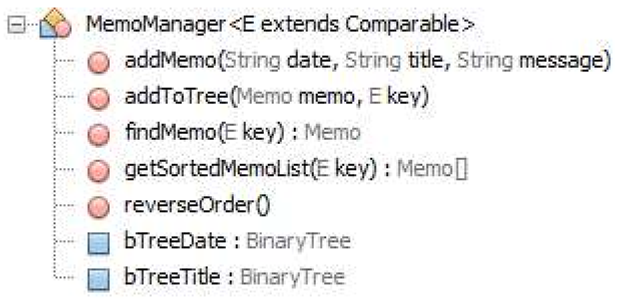
MemoManager class has a method named "**addMemo**" which takes a String date, a String title, and a String message to create a Memo object then calls addToTree() twice and passes the Memo object with different keys (date and title) to add on different trees.

MemoManager class has a method named "**addToTree**" which takes a Memo object and a key (the type of key can be Date or String) to add to the bTreeDate and bTreeTitle

MemoManager class has a method named "**findMemo**" which takes a searching key and returns the memo that matches the searching key by calling binaryTree. searchElement(key). If the memo does not exist, it returns null.

MemoManager class has a method named "**getSortedMemoList**" which takes a key (the type of key can be Date or String) and returns a memo array. The elements in the memo array must be sorted.

MemoManager class has a method named "**reverseOrder**" which calls the BinaryTree objects to reverse the order of the trees.



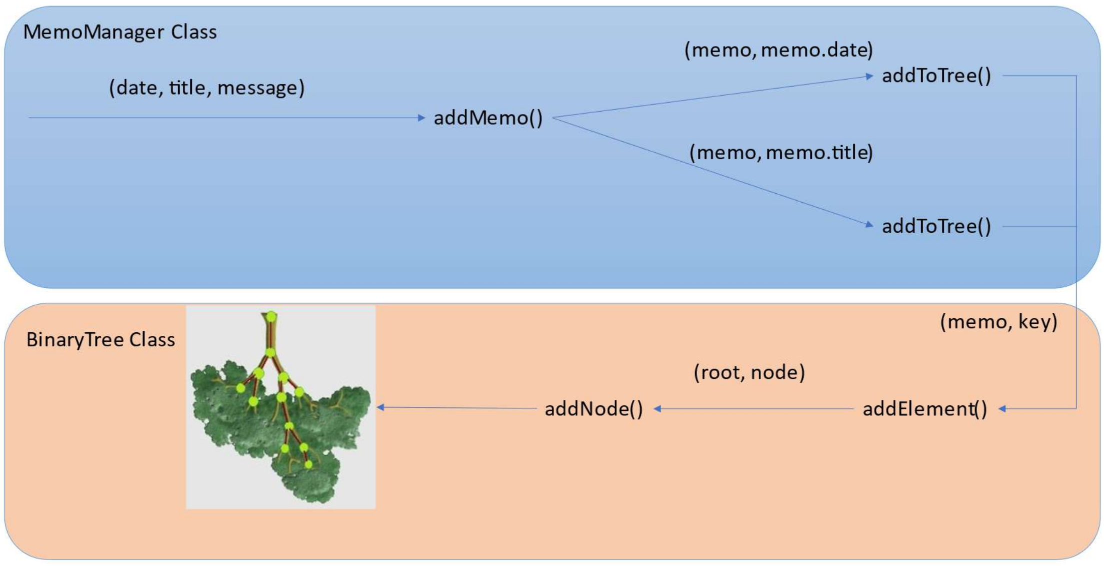
```
MemoManager <E extends Comparable>
    addMemo(String date, String title, String message)
    addToTree(Memo memo, E key)
    findMemo(E key) : Memo
    getSortedMemoList(E key) : Memo[]
    reverseOrder()
    bTreeDate : BinaryTree
    bTreeTitle : BinaryTree
```

You can add more methods or fields as you wish, but you must pass the testing code (MemoApp.java).

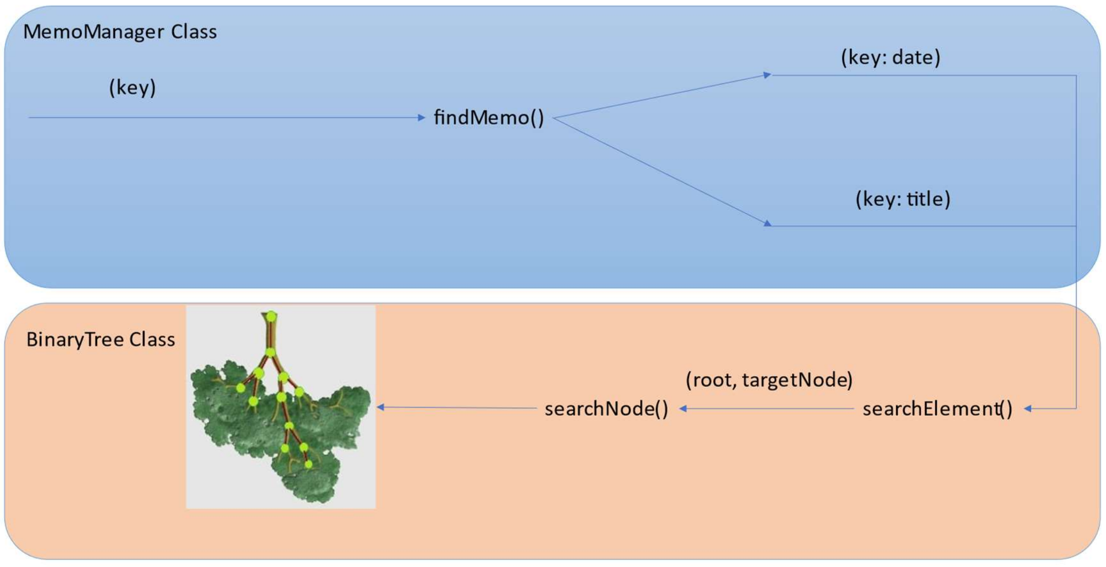
Java String to Date

```
Date date;

DateFormat dateFormat = new SimpleDateFormat("yyyy/MM/dd");

try

{

        date = dateFormat.parse(date);

}

catch (ParseException ex)

{

        Logger.getLogger(MemoManager.class.getName()).log(Level.SEVERE, null, ex);

}
```

Add a memo



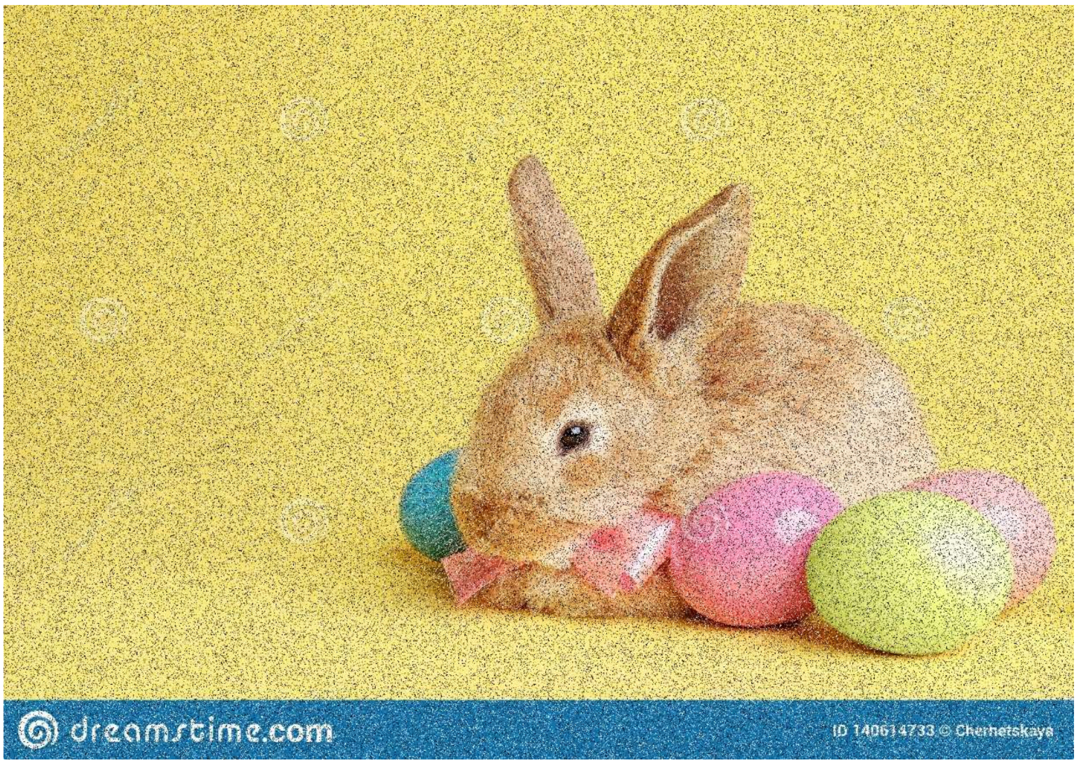
Search a memo



Question 2) Sorting: Noise Removing Application (20%)

Problem: **Salt-and-pepper noise**, also known as impulse noise, is a form of noise sometimes seen on digital images. This noise can be caused by sharp and sudden disturbances in the image signal. It presents itself as sparsely occurring white and black pixels. (See more information: https://en.wikipedia.org/wiki/Salt-and-pepper_noise#:~:text=Salt%2Dand%2Dpepper%20noise%2C,occurring%20white%20and%20black%20pixels.)

Example (image with Salt-and-pepper noise)



An effective noise reduction method for this type of noise is a median filter.

Median Filter:

A program reads a pixel and its 8 surrounding pixels (This 3x3 block of pixels is called sliding window in digital image processing) then find the median of these 9 pixels. Finally, the median replaces the central pixel. Sliding window moves onto the next pixel and repeats this process until all pixels has been changed (To make the assignment simple, it excludes the bounding pixels).

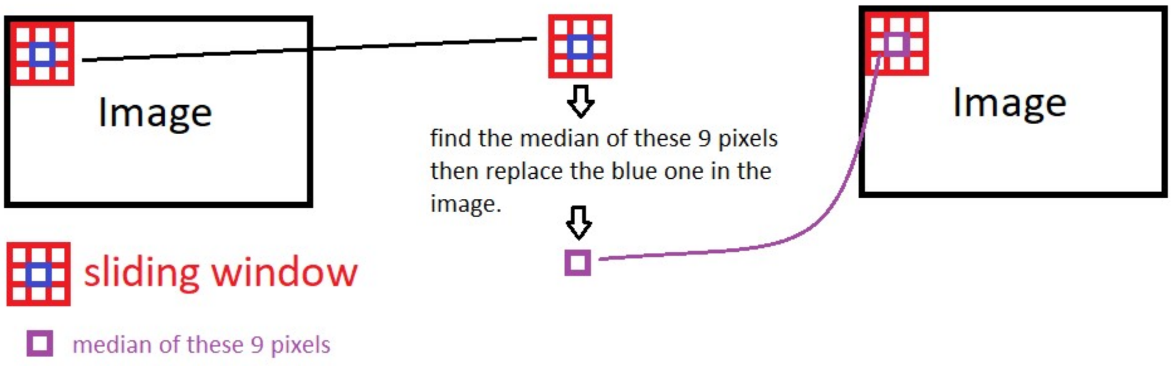


Image
find the median of these 9 pixels then replace the blue one in the image.
Image
sliding window
median of these 9 pixels

Median: the middle score when all the scores are arranged **in order** of size from the smallest to the highest. If there are an even number of scores, then it is the average of the two middle scores (It will not be the case in this assignment).

The code of reading image, getting pixels, saving pixels to an image has been done for you. Your task is to find median for given pixels. In order to find the median, you need to sort an array of 9 pixels.

Example (output image after removing noise)

Your task:

The assignment project question 2 contains two .java files. “**ImageProcess.java**” and “**NoiseRemoving.java**”. If you run the project, it loads an image and generates a .jpg file, named “noise_removed.jpg”, but the generated image still contains noise for now. You need to complete the method of “cleanNoise” to clean the noise in the ImageProcess class.

“**ImageProcess.java**” deals with an image. It has a method named “**cleanNoise**”. There is a gap in the method. You need to add your SortArray Class to the project and fill the gap to complete the ImageProcess Class (3%).

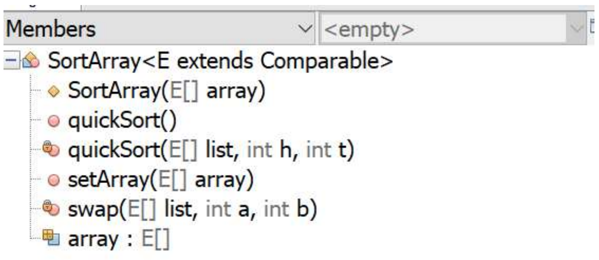
You need to complete the “**SortArray**” Class in the package. It takes a generic Comparable type of array and sorts array in order.

SortArray Class has an “**array**” field. It stores a reference of an array. (2%)

SortArray Class has “**setArray**” method. It takes a reference of an array and passes the reference to the “array” field. (2%)

SortArray Class has “**quickSort**” method. It runs quick sort algorithm and sorts array in order (4%). Comments this method (2%) Check quicksort() method with your lab exercise.

DO NOT COPY THE CODE FROM THE CHAPTER 3 EXAMPLES



```
SortArray<E extends Comparable>
    SortArray(E[] array)
    quickSort()
    quickSort(E[] list, int h, int t)
    setArray(E[] array)
    swap(E[] list, int a, int b)
    array : E[]
```

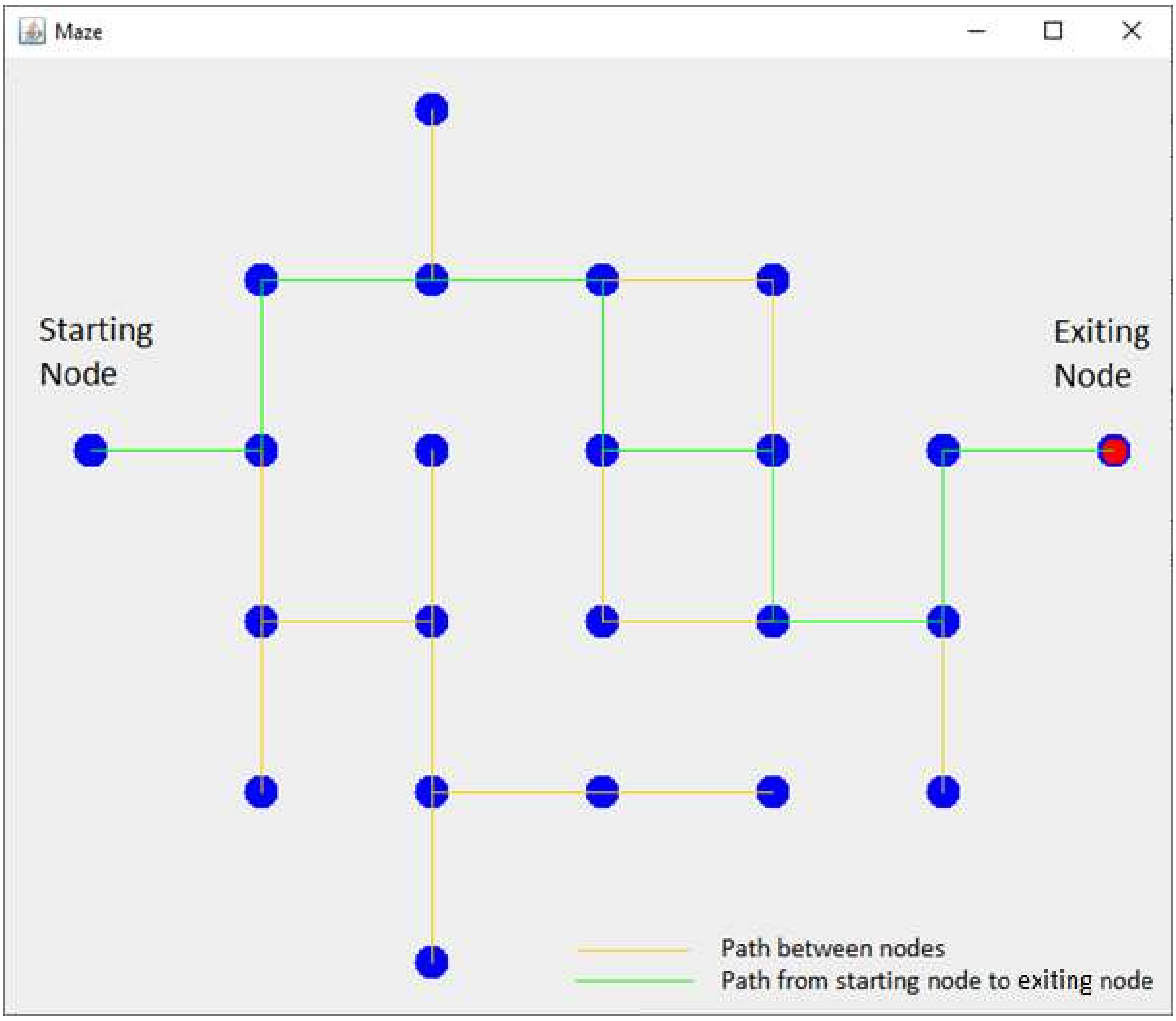
Answer following questions in your Comments at the beginning of your SortArray Class code.

1. Is quick sort the best way of finding median? Why? (3%)
2. What is another good way of finding median? Please provide your explanation. (3%)

Finally, you need to design a GUI of noise removing application (1%). So, the application can load an image, clean the noise, and save to a new image. And produce a .jar file of this project.

*If you are interested in image process, you may try some open-source library. OpenCV for C/C++ or JavaCV for Java.

Question 3) Maze (30%)



You are going to design a maze program. It loads a maze from a maze text file then program walks through the maze and finds the path from starting vertex to the exiting vertex.

Maze text file format:

Number of edges, Number of columns, number of rows (Header of maze)

Vertex's name, x position, y position, next linked vertex's name, next linked vertex's name

…

Vertex's name, x position, y position, next linked vertex's name, next linked vertex's name

Example:

22,7,6

START,0,2,B,A

B,1,2,C,K

C,1,3,D,E

…

V,4,1,N,A

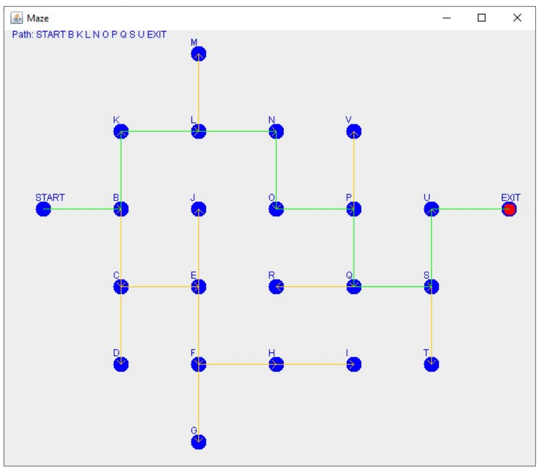
EXIT,6,2,A,A

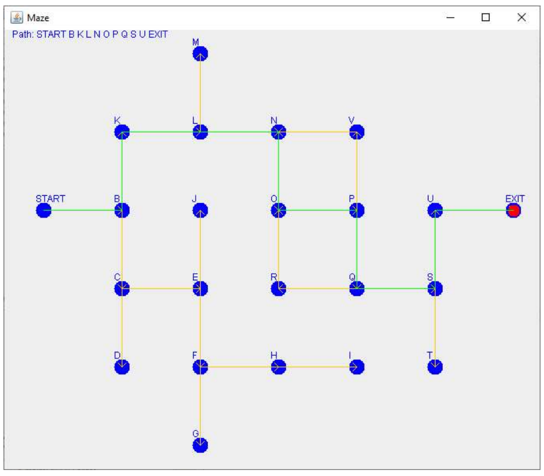
"A" is the same as null. It means not next linked vertex on this path (this path has no exit).

"W" links to exit.

Your task is to write a program. The program does:

- Loads a maze txt files (there are two txt files) (3%)
- Draws a maze on the panel (You are going to decide how to label the nodes). (3%)
- Walk through the maze and find path from start to exit. (5%) You need to show an animation of how your program finds a path from start to exit. (3%)
- Highlight the trail from "Start" to "Exit" on the panel (see image below). (3%)
- Display the path from "Start" to "Exit". (5%)
- make sure your program works for both txt files. (7%)
- GUI is provided. (1%)
- A jar file is created.


Path: START B K L N O P Q S U EXIT


Path: START B K L N O P Q S U EXIT

A demonstration is available during the lecture. If you missed the lecture, please ask the lecturer to run a demonstration in the lab.