10 LABS x 2 hours 15 minutes

- Lab 1: OOP Reviews & Arrays
- Lab 2: Simple sorting
- Lab 3: Stacks & Queues
- Lab 4: Linked List
- Lab 5: Recursion
- Lab 6: Trees
- Lab 7: Hash Tables
- Lab 8: Graph
- Lab 9: Exam
- Lab 10: Project Presentation

There are 8 practical labs (30%):

- Select 3 random submissions to mark
- If you miss a lab or a submission: that lab will be selected to mark

Lab 9 will be a practical exam (35%)

- You can use your laptop to code
- You are only allow to use the following IDE:
  - NetBeans
  - VS Code
  - BlueJ
  - IntelliJ
- You must DISCONNECT your laptop from the Internet

Lab 10 is the project presentation (35%)

Deadline to submit your work on Blackboard: 3 days from the lab day

- i.e., Lab day is Monday => deadline is Wednesday (mid-night)

Assignments submission guide

- Create the folder with a name like: ***StudentID_Name_Lab#***, (e.g. ***01245_VCThanh_Lab1***) to contain your assignment with subfolders:
  - Problem_01 (sometimes Problem_i or Problem_Array)
  - Problem_02 (sometimes Problem_ii or Problem_Queue)
  - etc.
- Compress (.zip) and Submit the whole folder with the same name (i.e., ***01245_ VCThanh_Lab1.zip***) to Blackboard
- Students **not** following this rule **will get their marks deducted**

6. (optional): Advanced Sorting

6.1. Objectives

- Merge sort
- Shell sort
- Quick sort

6.2. Problem 1: PartitionApp.java

- Add counters for the number of comparisons and swaps and display them after partitioning.
- Investigate the relationship between the index of partitioning, the number of comparison, and the number of swaps.
- Do the previous exercise with different pivots:
    - beginning, end, or middle of the interval;
    - selected at random from the interval or from a larger interval;
    - last item in the array.
- Compute the average number of comparisons and swaps over 100 runs.

6.3. Merge sort, Shell sort, Quick sort

Code the following program as in the text book:

- mergeSort.java (listing 6.6, page 288)
- shellSort.java (listing 7.1, page 321)
- quickSort3.java (listing 7.5, page 351)

In each sorting algorithm, add counters for the number of comparisons, copies, and swaps and display them after sorting.

Create an array of integer numbers, fill the array with random data and print the number of **comparisons, copies, and swaps** made for sorting 10000, 15000, 20000, 25000, 30000, 35000, 40000, 45000 and 50000 items and fill in the table below. Analyze the trend for the three different algorithms.

Compare with the same table in Lab 2 with simple sorting algorithms.

COPIES/ COMPARISONS/ SWAPS			
	Merge Sort	**Shell Sort**	**Quick Sort**
10000			
15000			
20000			
25000			
30000			
35000			
40000			
45000			
50000			

7. Lab 6: Trees

7.1. Objectives

- Know how binary Tree works.
- Implementing additional methods to deal with binary trees in Java programming.

You are provided with two files

- Tree.java
- TreeApp.java

7.2. Problems 1

(10 points) Add a method that counts the elements in a binary tree into the *Tree Class*. Specifically, the method takes no parameters and returns an integer equal to the number of elements in the tree.

7.3. Problems 2

(10 points) Add a method that computes the height of a binary tree into the *Tree Class*. Specifically, this method has no parameters and returns an integer equal to the height of the tree.

7.4. Problems 2

(10 points) Add a method that counts a binary tree's leaves tree into the *Tree Class*. Specifically, this method has no parameters and returns an integer equal to the number of leaves in the tree.

7.5. Problems 3

(10 points) Add a method that determines whether a binary tree is fully balanced. This method takes no parameters and returns a Boolean value: true if the tree is fully balanced and false if not.

7.6. Problems 4

(10 points) Define two binary trees to be identical if both are empty or their roots are equal, their left subtrees are identical, and their right subtrees are identical. Design a method that determines whether two binary trees are identical *(this method takes a second binary tree as its only parameter and returns a Boolean value: true if the tree receiving the message is identical to the parameter, and false otherwise)*.

7.7. Huffman coding

(15 points) Draw a Huffman coding tree for the following text with **YOUR_NAME** below is your full name.

"I am a student at International University. My name is YOUR_NAME. I am working on a DSA lab"

7.8. TreeApp.java

Count the number of comparisons in find(), insert(), and delete() and evaluate their efficiency.

Create an option to clear the tree.

Create an option to insert random items and examine the resulting trees.

Create methods for finding the minimal and maximal item.

Save the items in an array using traversal and then reinsert them in the tree. How does the tree change with different traversals?