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

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)

Bonus Requirement for Final Lab Scores

Students have two options to secure a bonus that guarantees a minimum final lab score of 85 points:

1. **Online Problem-Solving Option:**
    - Solve at least 90 problems on leetcode.com, hackerrank.com, and hackerearth.com/practice from the first lab session through the end of the semester.
    - Only up to 45 "easy" problems will be counted toward this total.
    - Note: Any problems solved before the first lab session are not counted.
2. **Codeforces Rating Option:**
    - Secure at least 1400 points on Codeforces during the same period.
    - For those already above 1400, the student must achieve a minimum rating of 1550 by the end of the semester.

Meeting either of these criteria will ensure that the student receives a minimum final lab score of 85 points.

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.
- **Report:** Include a brief report (1–2 pages) summarizing your code, design choices, and any observations from your performance experiments.
- 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**

General Input Requirement
All programming problems must accept input from the keyboard, except for those problems that specifically require input via text files.

1. Lab 1: OOP Reviews & Arrays

1.1. Objectives

i. Review basic OOP and Java skills
ii. Arrays and operations with arrays: find, insert, delete
iii. Ordered arrays, binary search
iv. Implementing arrays is Java
v. Arrays of objects
vi. Efficiency of aray operations, big O notation

1.2. Problem 1: Simple application

i. Write a function to convert array to number. (10pts)

Suppose we have loaded an array with the digits of an integer, where the highest power is kept in position 0, next highest in position 1, and so on.

The ones position is always at position array.Length - 1:

Example input: 2, 0, 2, 5

Example output: 2025

Validation: Verify that every element is a valid digit (0–9). If not, handle the error appropriately (e.g., throw an exception or return an error code).

Negative Numbers: Extend the function so that an optional sign (e.g., a leading -) can be interpreted correctly.

Optional Extension: Support conversion of arrays representing hexadecimal numbers (digits 0–9 and letters A–F).

Documentation & Testing: Include unit tests that cover typical and edge-case scenarios in your report.

ii. Write a function to input a list of integer numbers and return the median of that list (10pts).

- **Data Types:** Allow the function to work with both integers and floating-point numbers.
- **Sorting & Even-Sized Lists:** Ensure the list is sorted; if the number of elements is even, return the average of the two middle numbers.
- **Error Handling:** Provide clear error messages or exceptions if the input list is empty or invalid.

iii. Find the min-gap (10pts)

Write a method named minGap that accepts an integer array and a number of elements as parameters and returns the minimum 'gap' between adjacent values in the array. The gap between two adjacent values in an array is defined as the second value minus the first value.

For example, suppose a variable called array is an array of integers that stores the following sequence of values:

int[] array = {1, 3, 6, 7, 12};

The first gap is 2 (3 - 1), the second gap is 3 (6 - 3), the third gap is 1 (7 - 6) and the fourth gap is 5 (12 - 7).

Thus, the call of minGap(array, n) should return 1 because that is the smallest gap in the array. If you are passed an array with fewer than 2 elements, you should return 0.

iv. GasMileage.java (input, the use of Scanner class to read numeric data)

What happens if you enter 20.5 for miles? Fix the problem in two ways.

What happens if enter miles and gallons on the same line?

Modify the program so that it reads two lines of inputs with car, miles, and gallons (separated by blanks) and prints mpg for each car.

Multiple Cars: Modify the program to read data for multiple cars, where each car's information is provided on a separate line.
Input Validation: Implement loops that prompt the user until valid input is received.
Extended Functionality: After calculating the miles-per-gallon (MPG) for each car, compute and display overall statistics (e.g., average MPG).
Documentation: explain how input errors are handled

v. Student.java, Students.java, students.txt (text files, loops, decision making, access modifiers)

Add public/private to the declaration of student names and print students with System.out.println(st.lname), explain.

Change the while-loop with a for-loop

Use grade to determine the type of the student: excellent (> 89), ok [60,89], and failure (< 60)

Do counting and averaging within each student type (excellent, ok, and failure)

Loop Transformation: Replace any while-loops with for-loops (and compare the pros and cons).

Sorting: Add functionality to sort student objects by last name or grade.

Error Handling: Include error handling for file I/O (e.g., if students.txt is missing or improperly

formatted).

1.3. Problem 2

i. Compile and Run the Example Programs

- Array, LowArray, HighArray
- OrderedArray
- ClassDataArray

ii. HighArray.java

Add a method called getMax() that returns the value of the highest key in the array, or –1 if the array is empty. Add some code in main() to exercise this method. You can assume all the keys are positive numbers.

Write a noDups() method for the HighArray class. This method should remove all duplicates from the array. That is, if three items with the key 17 appear in the array, noDups() should remove two of them. Don't worry about maintaining the order of the items. One approach is to first compare every item with all the other items and overwrite any duplicates with a null (or a distinctive value that isn't used for real keys). Then remove all the nulls. Of course, the array size will be reduced.

Insert 100 random items (generated with nextInt()) and find one of them chosen at random. Print the number of comparisons (add a counter of comparisons in class HighArray and set it in the find method).

Compute and print the average number of comparisons to find a random item over 100 trials.

Print the average number of comparisons to find a random item in arrays with 100, 200, 300,...,1000 items. Analyze the trend.

Modify the code that accepts input from keyboard.

iii. OrderedApp.java

- Insert 100 random items (generated with nextInt()) and find one of them chosen at random. Print the number of comparisons (add a counter of comparisons in class HighArray and set it in the find method).
- Compute and print the average number of comparisons to find a random item over 100 trials.
- Print the average number of comparisons to find a random item in arrays with 100, 200, 300,...,1000 items. Analyze the trend.
- Compare the complexity of linear (HighArrayApp.java) and binary (OrderedApp.java) search.

This problem does not need to accept input from keyboard.