**Student Name: Weight: 15%**

**Student ID:** **Marks:** **/100**

# Assignment: Complexity and Sorting

## Description

In this assignment, you’ll work in groups to create a sorting algorithm for geometric shapes. You’ll also implement and perform experimental analysis on six sorting algorithms.

## Equipment and Materials

For this assignment, you will need:

* Java IDE

## Instructions

This assignment consists of three parts, to be completed outside of class time. See the course outline and Brightspace for due dates. Complete this assignment with your assigned group.

#### Part A: Create a Sorting Application (95 marks)

1. Using the specifications below, create an application to sort objects. You’ll then implement and perform experimental analysis on six sorting algorithms:

* Bubble
* Insertion
* Selection
* Merge
* Quick
* A sorting algorithm of your choice

You must research the sorting algorithm of your choice and ensure that it’s significantly different from the other five. Include a detailed description of this sort’s algorithm and a complexity analysis in your submission.

#### Part B: Complete a Peer Assessment (5 marks)

Each student must also complete a peer assessment of their group members. Your instructor will provide further submission details.

#### Part C: Complete an Application Evaluation as a Group

After completing your sorting application, check your work against the provided marking criteria. Your instructor will refer to your group’s self-evaluation when grading the assignment and will provide further feedback and grade adjustments as needed. Your instructor is responsible for awarding the group’s final grade.

1. Open the Marking Criteria document (MarkingCriteria\_Assignment1.docx) and save a copy with your group’s name.
2. As a group, discuss how well you met each criterion and assign yourselves a mark for each row in the table. You may include a short, point form, explanation for your mark in the Notes column.
3. Save this file for submission to Brightspace along with your completed code.

## Part A: Application Specifications

**Important:** Read the specifications very carefully. If you are uncertain about any of the requirements, discuss it with your instructor.

1. Create an abstract class in Java that represents a three-dimensional geometric shape. Using the Strategy pattern discussed in class:

* Implement the **compareTo()** method of the Comparable interface to compare two shapes by their height, and the **compare()** method of the Comparator interface to compare two shapes by their base area and volume.
* The compare type will be provided as input from the command line to your program: **h** for height, **v** for volume, and **a** for base area via the -t option.

1. Write a testing program that will read a text file (entered at the command line via the -f option) of random shapes that adds them to an array (not ArrayList).

**Notes:**

* All shapes must be manipulated as elements of the corresponding collection.
* The first value in the data file contains the number of shapes in that file.
* A shape in the file is represented as follows (all values are separated by spaces):
* One of: Cylinder, Cone, Prism or Pyramid
* Cylinders and Cones are followed by a double value representing the height and another double value representing radius.

e.g., Cylinder 9431.453 4450.123 or Cone 674.2435 652.1534

* Pyramids are followed by a double value representing the height and another double value representing edge length.

e.g., Pyramid 6247.53 2923.456

* Prisms are specified by the type of base polygon (SquarePrism, TrianglarPrism, PentagonalPrism, OctagonalPrism), a double value representing the height and another double value representing edge length.

e.g., SquarePrism 8945.234 3745.334

* See the formulas at the end of this document for more information.

1. Your testing application should then invoke the utility methods to re-arrange the figures according to the compare type from the largest to smallest (descending order).
2. Your testing program should print the time that it took to sort the collection of objects for each of the six sorting algorithms, including a measurement unit (e.g., milliseconds).

**Note:** Have this benchmarking done outside of the sorting algorithms such that the sorting code can be executed without this explicit output.

1. The program should also print the first sorted value and last sorted value, and every thousandth value in between.
2. Implement the sorting algorithms as part of a utility class. The utility class must sort a collection of Comparables.

**Note:** Ensure the utility class isn’t dependent on the testing application and that it can be re-used in the future. Invoke your sorting methods like the Arrays.sort() method.

1. To run the program, the user should enter a command via command line in the following format:

java -jar sort.jar **-ffile\_name -tv -sb**

* -f or -F followed by file\_name (the file name and path)
* -t or -T followed by v (volume), h (height) or a (base area)
* -s or -S followed by b (bubble), s (selection), i (insertion), m (merge), q (quick) or z (your choice of sorting algorithm)

The program must be order and case insensitive. The following are examples of valid inputs:

* java -jar sort.jar -fpolyfor1.txt -Tv -Sb
* java -jar sort.jar -ta -sQ -f"res\polyfor3".txt
* java -jar sort.jar -tH -F"C:\temp\polyfor5.txt" –sB

1. If the user enters an incorrect command line argument according to the rules above, the program should display a helpful message to help the user to correct the error.

## Deliverables

**Note:** This is a group assessment, so only one submission is required per group. See Brightspace for the exact due date and time.

Your group’s submission should include a zipped folder named as the assignment number and your group number (e.g., A1Group3.zip). The zipped folder should contain the following items:

* An executable Java Archive file (.jar) for your sorting application called **Sort.jar**.
* A **readMe.txt** file with instructions on how to install and use your sorting program.
* The project should have completed javadoc using the “-private” option when generated. Place the output in the doc directory of the project.
* A folder containing the complete Eclipse project directory.
* At the root of the project directory, include a **readMe.txt** file describing the completeness of the assignment (as a percentage) and a list of known deficiencies and/or missing functionalities.

**Note:** Do NOT include the test data files in your upload (e.g., polyNameBIG.txt).

* A **mySort.txt** file that provides a detailed description of the sorting algorithm of your choice along with its complexity analysis.

Your analysis should include the steps of the algorithm in pseudocode, with the number of operations performed in each step. Make sure to write this in your own words and DO NOT copy or rephrase from another source.

* The completed Marking Criteria document containing your group’s self evaluation.

No late assignments will be accepted.

# Formulas

**Cylinder**

* 𝑏𝑎𝑠𝑒 𝑎𝑟𝑒𝑎 = 𝜋𝑟 2
* 𝑣𝑜𝑙𝑢𝑚𝑒 = 𝜋𝑟 2ℎ

**Cone**

* 𝑏𝑎𝑠𝑒 𝑎𝑟𝑒𝑎 = 𝜋𝑟 2
* 𝑣𝑜𝑙𝑢𝑚𝑒 = 1 3 𝜋𝑟 2ℎ

**Pyramid**

* 𝑏𝑎𝑠𝑒 𝑎𝑟𝑒𝑎 = 𝑠 2
* 𝑣𝑜𝑙𝑢𝑚𝑒 = 1 3 𝑠 2ℎ

**Prisms**

**Square Base**

* 𝑏𝑎𝑠𝑒 𝑎𝑟𝑒𝑎 = 𝑠 2
* 𝑣𝑜𝑙𝑢𝑚𝑒 = 𝑠 2ℎ

**Equilateral Triangle Base**

* 𝑏𝑎𝑠𝑒 𝑎𝑟𝑒𝑎 𝐴 = 𝑠 2√3 4
* 𝑣𝑜𝑙𝑢𝑚𝑒 = Aℎ

**Pentagon Base**

* 𝑏𝑎𝑠𝑒 𝑎𝑟𝑒𝑎 𝐴 = 5𝑠 2 tan (54°) 4
* 𝑣𝑜𝑙𝑢𝑚𝑒 = Aℎ

**Octagon Base**

* 𝑏𝑎𝑠𝑒 𝑎𝑟𝑒𝑎 𝐴 = 2(1 + √2)s 2
* 𝑣𝑜𝑙𝑢𝑚𝑒 = Aℎ 4.