CSC2023: Algorithm Design and Analysis
Assignment 2: Glass Cutting Problem

Aim:

a. To assess student ability to develop new algorithms following the principles of a chosen Algorithm Design Technique.
b. To strengthen student understanding of the Greedy technique for designing algorithms.
c. To provide experience of evaluating and comparing different algorithms.

General:
Consider the following variant of the *Bin Packing problem*, called the *Glass Cutting Problem*.

We have an unlimited supply of two-dimensional glass **sheets**, each sheet having a width of `W` and a height of `H`. We also have a request of `N` rectangular **shapes**, with widths $w_1, w_2, \ldots, w_N$ (where $w_i <= W, i=1,\ldots,N$) and heights $h_1, h_2, \ldots, h_N$ (where $h_i <= H, i=1,\ldots,N$).

We use a glass cutting machine that conducts a specific type of cut, namely the guillotine cut. A guillotine cut is a cut from one edge of a sheet to the opposite edge, parallel to the remaining edges. In other words, when applied to a rectangle, the cut is of guillotine type if it produces two new rectangles. The cutting patterns for the sheets are produced in at most two stages (Figure 1). In the first stage, parallel (horizontal) guillotine cuts are produced on a sheet to produce a set of 'shelves'. In the second stage, vertical guillotine cuts are made on each shelf to produce the required shapes. If the shapes in a shelf are not all of the same height, additional trimming of the shapes is required (Figure 2).

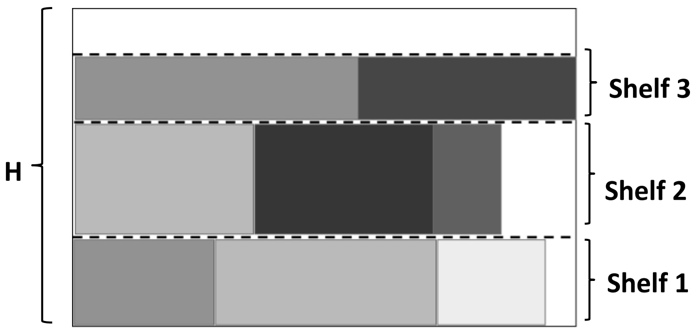


Figure 1

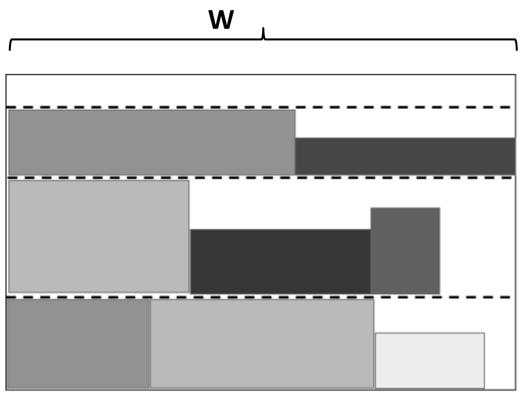


Figure 2

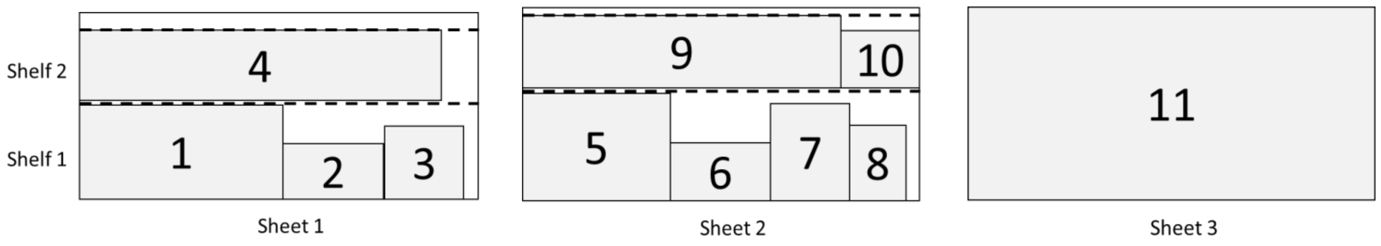
Shelves, parallel to the horizontal axis, have a height equal to the height of the leftmost shape in the shelf, and a width equals to `W` (the width of the sheet). All shapes in a shelf are placed such that the bottom side is on the shelf floor. There can be as many shelves in a sheet as needed as long as the total height of all shelves does not exceed `H` (the height of the sheet).

<u>The problem</u> is to place all the shapes on the smallest number of sheets. Therefore, you need to design an algorithm that lays out the shapes across as few sheets as possible under the following rules:

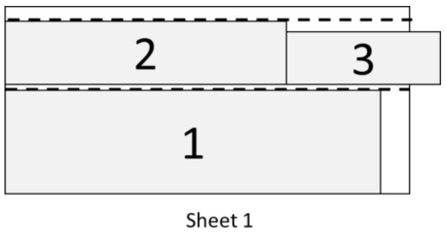
A. A shape is placed either at the bottom left corner of a sheet (starting the first shelf on the sheet) or to the right of another shape, if there is sufficient space in the shelf.
B. If a shape does not fit in a shelf, one rotates the shape and tries to fit it in the shelf.
C. If a shape still does not fit in a shelf, one can start a new shelf directly on top of the current shelf against the left side of the sheet, if there is enough space in the sheet. First, try to create a new shelf with the shape in its original orientation, and if it does not fit, one can rotate the shape and try to create the new shelf.
D. At most one shape is placed directly to the right of any other shape (see example **5** for clarification).
E. The total height of all shelves in a sheet does not exceed H
F. The number of shapes placed on a sheet cannot exceed `L` (where `L` is given).

Examples of correct and incorrect shape placement for guillotine glass cutting:

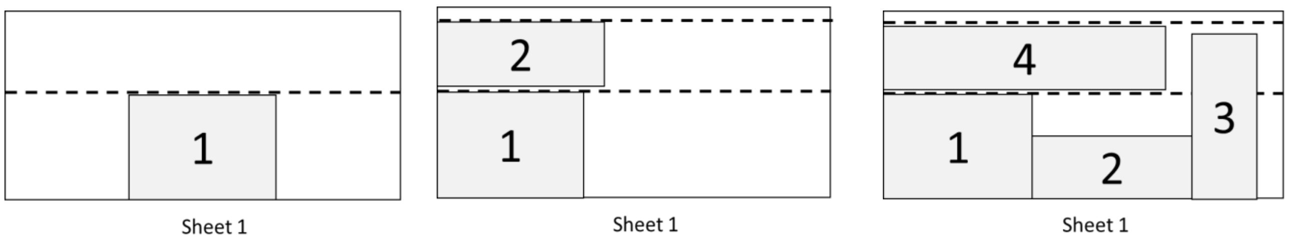
1. **Correct:**



2. **Incorrect (rules A and C violated): A new sheet should be used.**



3. **Incorrect (rule A or C violated)**

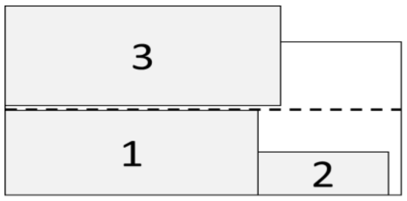


Shape 1 is not placed at the bottom left corner

There is a sufficient space for shape 2 to be placed next to shape 1

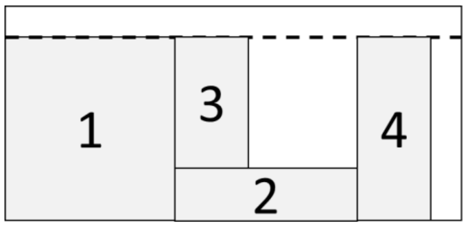
Insufficient space for shape 3 in shelf 1

4. **Incorrect (rules C and E violated): A new sheet should be used (insufficient space in sheet 1 to create a new shelf for shape 3).**



Sheet 1

5. **Incorrect (role D violated)**

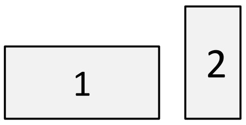


Sheet 1

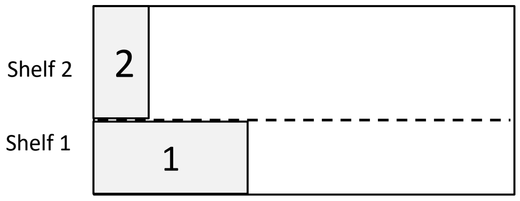
Shape 3 should not be placed on top of shape 2. At most one shape is placed directly to the right of shape 1.

6. **Incorrect (rule B violated): Shape 2 can be rotated to fit in shelf 1.**

Input shapes:



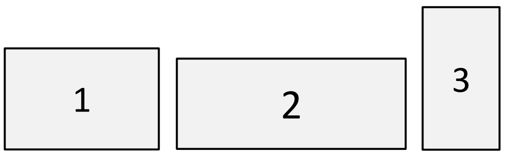
Output (**Incorrect**):



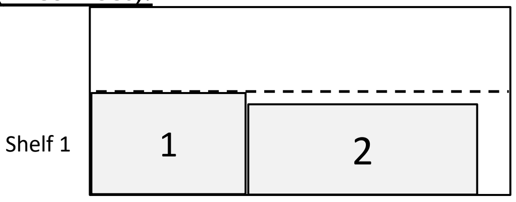
Sheet 1

7. **Incorrect (rule B violated): Shape 3 can be rotated to create a new shelf in sheet 1.**

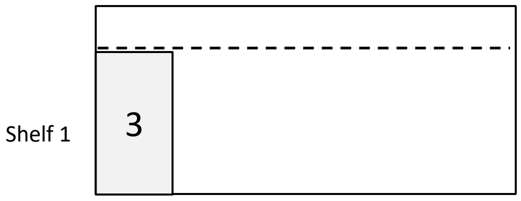
Input shapes:



Output (**Incorrect**):



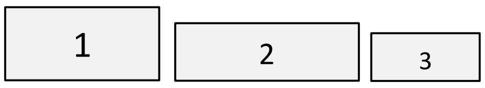
Sheet 1



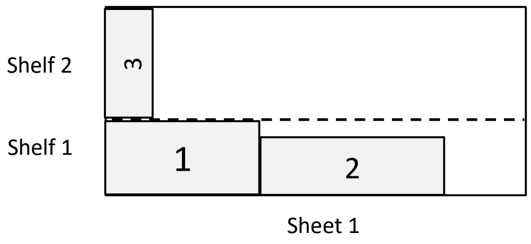
Sheet 2

8. **Incorrect (rule C violated): Shape 3 in the second shelf should be placed using its original orientation**

Input shapes:



Output (**Incorrect**):



For solving the Glass Cutting Problem (GCP), we will use two algorithms:

- NFGCP: based on Next Fit strategy for solving Bin Packing Problems.
- FFGCP: based on First Fit strategy for solving Bin Packing Problems.

You have been provided with a template to start with (see the zip file in the **Assessment and Feedback** section on the Blackboard page for CSC2023) which contains:

A. Basic classes: A class `Shape`, which represents a single shape requested by the customer. A class `Shelf`, which contains a list of shapes. A class `Sheet` which represents a single sheet supplied by the manufacturer. Those classes contain basic methods to be used in the assignment.

- A class `Generator` that has the methods signature for providing a list of shapes.
- A class `Algorithms` that has the method signatures for Next Fit and First Fit algorithms.
- Test classes `CorrectnessTest`, `PerformanceTest` and `SortedTest`: Contain instructions on how to set up your tests.

Tasks (to be implemented in the Java programming language and Eclipse framework):

A. **Design and implement algorithms for the glass cutting problem**. Consider the two aforementioned greedy algorithms NFGCP and FFGCP. To do that you need to:

Task 1 (2 marks): Provide an output method for displaying detailed information about the content of each sheet.

Task 2 (4 marks): In the `Generator` class: Complete the method for randomly generating suitable sequences of integers which can be used later for testing and measuring performance of the two algorithms. The test data should represent a list of shapes requested by the customers. All test data for performance analysis must be generated within your program.

Task 3 (12 marks): In the `Algorithms` class: Implement the two algorithms (NFGCP and FFGCP). The methods should be able to place all the shapes according to NextFit and FirstFit and obeying the rules mentioned above. **Note: do not sort the shapes**.

B. Testing and performance analysis:

Task 4 (8 marks): In the `CorrectnessTest` class, validate the correctness of the two algorithms (look for normal cases and border cases). Document this in your report.

Task 5 (10 marks): In the `PerformanceTest` class, implement two ways of measuring the performance of the two algorithms: one based on timing their runs, and the other on the number of sheets used to place all the shapes. Generate **tabulated** results for the overall comparisons. This should be based on running the tests multiple times for different requests (with different values of `N`). **Comment** on the results obtained (which one is faster and which one uses less sheets). **Note: do not sort the shapes**.

Task 6 (10 marks): The previous test assumes that the shapes are not sorted. The manufacturer would like to see if sorting shapes before placing them could help in minimizing the waste (that is, use less sheets of glass). Therefore in the `SortedTest` class, compare the number of sheets used by the NFGCP when a sorted list of shapes is passed to the algorithm, against a non-ordered list. You can sort based on height, width, or any function of **h** and **w** in both increasing and decreasing order. Generate **tabulated** results for the overall comparisons. **Comment** on your findings to inform the manufacturer about your result: Was sorting helpful? If so, what type of sorting is most beneficial and why (give at least one reason)? Do the same for FFGCP.

C. Code quality (4 marks)

You should observe good programming practice when writing Java code (such as appropriate choice of variable names, indentation, comments, etc.).

Notes:

- Additional comments are provided in the template to help you in implementing the tasks.
- The test class should have appropriate test code that runs the performance analysis multiple times for multiple requests' sizes. The markers should be able to re-run your test.
- You must not change the methods signature in the Algorithm, Sheet and Generator classes. **If we cannot test your implementation because you have changed the method signatures then that corresponding task will score 0.**
- Your code must be your own work and will be checked using plagiarism detection tools.

Marking Scheme:

Java code	**36%**
Generating data	8%
Algorithms & methods Implementation	28%
Testing	**56%**
Correctness testing	16%
Performance analysis	20%
Sorting Test	20%
Code quality	**8%**

The completed assignment must be submitted to NESS in a **single .zip file** by

3 pm on Friday 13th December 2019.

Your submission folder should include the following:

- Source code for the eight Java classes.
- A Word document or PDF file including:
- ➢ Correctness testing
- ➢ The tabulated results of the overall comparisons for Tasks 5.
- ➢ Comments on the results that you have obtained for Tasks 5.
- ➢ The tabulated results of the overall comparisons for Tasks 6 (For NFGCP and FFGCP).
- ➢ Comments on the results that you have obtained for Tasks 6 (For NFGCP and FFGCP).

This assignment is worth **50%** of your final coursework mark for this module.