Glass Cutting Algorithms report

Correctness testing

My correctness testing goes through the if statements on my algorithms and checks that each statement causes the correct behaviour, these if statements are formed using the rules set out for the algorithms so, through doing this, I have ensured that my algorithms follow all the rules set out. Doing this first before any other testing allowed me to find any errors or unexpected behaviour in my algorithms such as my next fit making a new shelf after already placing the shape on a different shelf.

```
*****************************************
********** Correctness testing ************
*****************************************

----------------- SINGLE SHAPE TEST -----------------
Test data: [HEIGHT: 1, WIDTH: 1, AREA: 1]

resulting shelves (nextFit): [HEIGHT: 1, USED WIDTH: 1, SHAPES STORED: 1]
resulting sheets (nextFit): [HEIGHT: 250, WIDTH: 300, SHELF AMOUNT: 1, SHAPE
AMOUNT: 1, REMAINING HEIGHT: 249]
Passed

resulting shelves (firstFit): [HEIGHT: 1, USED WIDTH: 1, SHAPES STORED: 1]
resulting sheets (firstFit): [HEIGHT: 250, WIDTH: 300, SHELF AMOUNT: 1, SHAPE
AMOUNT: 1, REMAINING HEIGHT: 249]
Passed

----------------- MULTIPLE SHAPES TEST -----------------
Test data: [HEIGHT: 1, WIDTH: 1, AREA: 1, HEIGHT: 1, WIDTH: 1, AREA: 1]

resulting shelves (nextFit): [HEIGHT: 1, USED WIDTH: 2, SHAPES STORED: 2]
resulting sheets (nextFit): [HEIGHT: 250, WIDTH: 300, SHELF AMOUNT: 1, SHAPE
AMOUNT: 2, REMAINING HEIGHT: 249]
Passed

resulting shelves (firstFit): [HEIGHT: 1, USED WIDTH: 2, SHAPES STORED: 2]
resulting sheets (firstFit): [HEIGHT: 250, WIDTH: 300, SHELF AMOUNT: 1, SHAPE
AMOUNT: 2, REMAINING HEIGHT: 249]
Passed

----------------- MULTIPLE SHAPES ON MULTIPLE SHELVES TEST (HEIGHT) ------------
-----
Test data: [HEIGHT: 1, WIDTH: 1, AREA: 1, HEIGHT: 2, WIDTH: 1, AREA: 2]

resulting shelves (nextFit): [HEIGHT: 1, USED WIDTH: 1, SHAPES STORED: 1, HEIGHT:
2, USED WIDTH: 1, SHAPES STORED: 1]
resulting sheets (nextFit): [HEIGHT: 250, WIDTH: 300, SHELF AMOUNT: 2, SHAPE
AMOUNT: 2, REMAINING HEIGHT: 247]
Passed

resulting shelves (firstFit): [HEIGHT: 1, USED WIDTH: 1, SHAPES STORED: 1, HEIGHT:
2, USED WIDTH: 1, SHAPES STORED: 1]
resulting sheets (firstFit): [HEIGHT: 250, WIDTH: 300, SHELF AMOUNT: 2, SHAPE
AMOUNT: 2, REMAINING HEIGHT: 247]
```

```
Passed

------------------ MULTIPLE SHAPES ON MULTIPLE SHELVES TEST (WIDTH) --------------
----
Test data: [HEIGHT: 1, WIDTH: 1, AREA: 1, HEIGHT: 1, WIDTH: 300, AREA: 300]

resulting shelves (nextFit): [HEIGHT: 1, USED WIDTH: 1, SHAPES STORED: 1, HEIGHT:
1, USED WIDTH: 300, SHAPES STORED: 1]
resulting sheets (nextFit): [HEIGHT: 250, WIDTH: 300, SHELF AMOUNT: 2, SHAPE
AMOUNT: 2, REMAINING HEIGHT: 248]
Passed

resulting shelves (firstFit): [HEIGHT: 1, USED WIDTH: 1, SHAPES STORED: 1, HEIGHT:
1, USED WIDTH: 300, SHAPES STORED: 1]
resulting sheets (firstFit): [HEIGHT: 250, WIDTH: 300, SHELF AMOUNT: 2, SHAPE
AMOUNT: 2, REMAINING HEIGHT: 248]
Passed

------------------ NEW SHEET ON LIMIT REACHED TEST ------------------
Test data: [HEIGHT: 1, WIDTH: 1, AREA: 1, HEIGHT: 1, WIDTH: 1, AREA: 1, HEIGHT: 1,
WIDTH: 1, AREA: 1, HEIGHT: 1, WIDTH: 1, AREA: 1, HEIGHT: 1, WIDTH: 1, AREA: 1,
HEIGHT: 1, WIDTH: 1, AREA: 1, HEIGHT: 1, WIDTH: 1, AREA: 1, HEIGHT: 1, WIDTH: 1,
AREA: 1, HEIGHT: 1, WIDTH: 1, AREA: 1, HEIGHT: 1, WIDTH: 1, AREA: 1, HEIGHT: 1,
WIDTH: 1, AREA: 1, HEIGHT: 1, WIDTH: 1, AREA: 1, HEIGHT: 1, WIDTH: 1, AREA: 1,
HEIGHT: 1, WIDTH: 1, AREA: 1, HEIGHT: 1, WIDTH: 1, AREA: 1, HEIGHT: 1, WIDTH: 1,
AREA: 1, HEIGHT: 1, WIDTH: 1, AREA: 1, HEIGHT: 1, WIDTH: 1, AREA: 1, HEIGHT: 1,
WIDTH: 1, AREA: 1, HEIGHT: 1, WIDTH: 1, AREA: 1, HEIGHT: 1, WIDTH: 1, AREA: 1]

resulting shelves (nextFit): [HEIGHT: 1, USED WIDTH: 20, SHAPES STORED: 20]
[HEIGHT: 1, USED WIDTH: 1, SHAPES STORED: 1]
resulting sheets (nextFit): [HEIGHT: 250, WIDTH: 300, SHELF AMOUNT: 1, SHAPE
AMOUNT: 20, REMAINING HEIGHT: 249, HEIGHT: 250, WIDTH: 300, SHELF AMOUNT: 1, SHAPE
AMOUNT: 1, REMAINING HEIGHT: 249]
Passed

resulting shelves (firstFit): [HEIGHT: 1, USED WIDTH: 20, SHAPES STORED: 20]
[HEIGHT: 1, USED WIDTH: 1, SHAPES STORED: 1]
resulting sheets (firstFit): [HEIGHT: 250, WIDTH: 300, SHELF AMOUNT: 1, SHAPE
AMOUNT: 20, REMAINING HEIGHT: 249, HEIGHT: 250, WIDTH: 300, SHELF AMOUNT: 1, SHAPE
AMOUNT: 1, REMAINING HEIGHT: 249]
Passed

------------------ FIRST FIT WORKS AS EXPECTED (SHELVES) ------------------
Test data: [HEIGHT: 100, WIDTH: 100, AREA: 10000, HEIGHT: 50, WIDTH: 50, AREA:
2500, HEIGHT: 150, WIDTH: 50, AREA: 7500, HEIGHT: 50, WIDTH: 50, AREA: 2500,
HEIGHT: 50, WIDTH: 250, AREA: 12500]

resulting shelves (firstFit): [HEIGHT: 100, USED WIDTH: 200, SHAPES STORED: 3,
HEIGHT: 150, USED WIDTH: 300, SHAPES STORED: 2]
resulting sheets (firstFit): [HEIGHT: 250, WIDTH: 300, SHELF AMOUNT: 2, SHAPE
AMOUNT: 5, REMAINING HEIGHT: 0]
Passed

------------------ FIRST FIT WORKS AS EXPECTED (SHEETS) ------------------
Test data: [HEIGHT: 100, WIDTH: 100, AREA: 10000, HEIGHT: 50, WIDTH: 50, AREA:
2500, HEIGHT: 130, WIDTH: 50, AREA: 6500, HEIGHT: 50, WIDTH: 50, AREA: 2500,
HEIGHT: 50, WIDTH: 250, AREA: 12500, HEIGHT: 150, WIDTH: 150, AREA: 22500, HEIGHT:
20, WIDTH: 50, AREA: 1000]
```

```
resulting shelves (firstFit): [HEIGHT: 100, USED WIDTH: 250, SHAPES STORED: 4,
HEIGHT: 130, USED WIDTH: 300, SHAPES STORED: 2]
[HEIGHT: 150, USED WIDTH: 150, SHAPES STORED: 1]
resulting sheets (firstFit): [HEIGHT: 250, WIDTH: 300, SHELF AMOUNT: 2, SHAPE
AMOUNT: 6, REMAINING HEIGHT: 20, HEIGHT: 250, WIDTH: 300, SHELF AMOUNT: 1, SHAPE
AMOUNT: 1, REMAINING HEIGHT: 100]
Passed
```

Performance testing

My performance testing was generally what I expected with the number of sheets used for nextFit, as in, the amount of sheets output was just over half the number of shapes input, but for firstFit I did notice that the amount of sheets output was actually just over quarter of the shapes input, this is around double the efficiency of nextFit. One thing I did not expect was the time taken by each algorithm, nextFit was incredibly fast no matter the size of the dataset, whereas firstFit really slowed down as the number of shapes increased. There is a stark difference between even the first test for both, with firstFit taking ~45 times longer than nextFit.

```
******************************************
********** Performance analysis ************
******************************************

--------------------------- NEXT FIT TESTS ----------------------------
Shapes: 10000|      Time (ms): 18, Sheets: 5523
Shapes: 20000|      Time (ms): 20, Sheets: 10973
Shapes: 30000|      Time (ms): 11, Sheets: 16496
Shapes: 40000|      Time (ms): 29, Sheets: 21938
Shapes: 50000|      Time (ms): 11, Sheets: 27361
--------------------------- FIRST FIT TESTS ---------------------------
Shapes: 10000|      Time (ms): 813, Sheets: 2925
Shapes: 20000|      Time (ms): 3885, Sheets: 5737
Shapes: 30000|      Time (ms): 7405, Sheets: 8517
Shapes: 40000|      Time (ms): 17392, Sheets: 11277
Shapes: 50000|      Time (ms): 35831, Sheets: 13964
```

Sorted testing

For the sorted test I decided to create datasets of increasing size to get an idea of how the algorithms deal with both small and large datasets, both unsorted and sorted. I noticed that for nextFit when the data was sorted in ascending order the results were slightly better for the larger datasets and actually very slightly worse for the smaller datasets, although when the data was sorted in descending order next fit was slightly better for most datasets aside from 1000 shapes where it is ever so slightly worse. For firstFit, when the data was sorted in ascending order the optimality dropped, with about 6000 more sheets being used compared to the unsorted data for 50000 shapes, but the same amount being used for only 10 shapes, when the dataset is sorted in descending order the algorithm is a bit more optimal, with about 1000 less sheets being used for 50000 shapes and 1 less sheet being used for 10 shapes. For nextFit, sorting the shapes in either ascending or descending order beforehand has a small benefit but for firstFit, if any sorting is done on the dataset it should be in descending order for best performance. Sorting on small datasets also has negligible results, even in descending order.

```
******************************************
*************** Sorted Test ***************
******************************************
```

UNSORTED DATA RESULTS

Algorithm & data set	Average sheets used
nextFit (10 shapes)	8
nextFit (100 shapes)	53
nextFit (1000 shapes)	531
nextFit (10000 shapes)	5489
nextFit (50000 shapes)	27421
firstFit (10 shapes)	6
firstFit (100 shapes)	38
firstFit (1000 shapes)	311
firstFit (10000 shapes)	2902
firstFit (50000 shapes)	13997

SORTED DATA (ASCENDING) RESULTS

Algorithm & data set	Average sheets used
nextFit (10 shapes)	8
nextFit (100 shapes)	54
nextFit (1000 shapes)	541
nextFit (10000 shapes)	5394
nextFit (50000 shapes)	26478
firstFit (10 shapes)	8
firstFit (100 shapes)	44
firstFit (1000 shapes)	416
firstFit (10000 shapes)	4092
firstFit (50000 shapes)	20300

SORTED DATA (DESCENDING) RESULTS

Algorithm & data set	Average sheets used
nextFit (10 shapes)	7
nextFit (100 shapes)	57
nextFit (1000 shapes)	532
nextFit (10000 shapes)	5398
nextFit (50000 shapes)	26385
firstFit (10 shapes)	5
firstFit (100 shapes)	33
firstFit (1000 shapes)	272
firstFit (10000 shapes)	2620
firstFit (50000 shapes)	12906

The implementation

To begin with, I modified the generateShapeList method to create a random list of shapes, to do this I used the Random class and the nextInt method and generated a random number for the height with the upper bound set to the MAX_SIZE_HEIGHT along with adding 1 after to ensure that 0 wasn't created as a dimension and MAX_SIZE_HEIGHT was actually used. this is because Random.nextInt() takes in an upper bound as exclusive and will only generate up to the upper bound – 1. I did the same thing with the width and generated a shape using these new values and stored it in the list to be output. When implementing the algorithms, for ease of programming and readability I created

two methods in the Sheet class, shapeAmount and remainingHeight, shapeAmount returns the amount of shapes in a sheet by getting each shelf and the size of the shapes list in that shelf and adding it to a running total until all sheets are exhausted, remainingHeight just returns the height left on the sheet by getting the height and subtracting the result returned from allShelvesHeight. My nextFit algorithm simply loops through the list of shapes and checks if the shape will fit the shelf first, then checks if the shelf will fit the sheet if it hasn't already been added before proceeding to the next shape, the current sheet isn't changed until it cannot fit anymore shelves or the shape limit has been reached, then the algorithm will create a new sheet and place the shelf there. My firstFit algorithm is similar and I used nextFit as a base but now it also has a Boolean for if the shape has been added along with loops for traversing the lists of shelves and sheets, during the shape loop it will loop through the list of shelves to see if any shelf has space, if the shelf doesn't have space the shape Boolean remains false and the loop continues, if the shelf does have space the shape variable becomes true and the loop breaks, if after going through all shelves the shape Boolean is still false then a new shelf is created and the shape added to that, there is then a loop to check that the shelf will fit on the sheet, this is the same way as shapes are done where a shelf Boolean is set to false and a loop will run till the shelf either gets added or a new sheet must be created. I programmed these with the mindset that this is similar to bin packing on top of bin packing, the shelves are bins filled with shapes and the sheets are just bins filled with shelves. To sort the shapes I had to override compareTo, the method now compares the areas of the shapes, with larger input areas returning -1 and smaller ones, 1. I added toString methods to the Shape, Shelf, and Sheet classes displaying base attributes of the classes and what I thought would be the most relevant data such as the remaining height or the shapes stored etc.