202502 - UECS2453 Problem Solving with Data Structures and Algorithms

PRACTICAL ASSIGNMENT - MARKING REPORT

1. PERSONAL DATA

Group number: 44				
No	Name	ID	Programme	Total Marks
1.	Benjamin Lau Yi En	2104665	SE	
2.	Seah Jun Wei	2206981	SE	
3.	Soo Kok Leang	2206123	SE	
4.	Yong Sze Ting	2205922	SE	

2. SUBMISSION STATUS

No soft copy/ Upload wrong file(s)	Late submission of softcopy	No hardcopy	Late submission of hardcopy	No issue

3. COMPILATION AND RUNNING

Does not compile/Bytecode & batch file do not work	Compile but no output/ wrong output/ run-time error	Compile and produce output

4. PRESENTATION OF SOURCE CODES(3%)

(a) Indent Style (1.5%) □ Poor □ Inconsistent □ Good
(b) Identifier names (1.5%) □ Poor choice □ Meaningful □ Meaningful and good naming convention

5. PROGRAM COMPONENT (57% + 3%)

Program Components	Missing/ Does not work	Major errors	Minor errors	Not robust	No issue/ Excellent design	Max marks	Marks obtained
Framework Design (Use of interfaces and abstract classes)						10	
Classes for storing objects (data structures/containers)						12	
Bin Packing Algorithms (at least 2)						16	
Test program (main program, set of bins and set of objects)						14	
Exception and error handling						5	
Presentation of source codes						3	
					Total	60	

6. REPORT AND OTHER COMPONENT (40%)

Components	Missing	Poor	Average	Good	Excellent	Max marks	Marks obtained
The proposed solution and design (data structures and algorithms)						8	
Discussion (efficiency and complexities)						12	
Flowchart						5	
UML Diagram						5	
Sample input and test cases						5	
Screenshots						5	
					Total	40	

Table of Contents

Introduction

The bin packing problem is a classic combinatorial optimization problem that includes packing items of varying sizes into a finite number of bins of fixed capacity. The main goal is to minimize the bins number used to pack all the items. In this report, the chosen real-life example that is associated with bin-packing problem is the optimization of delivery box packing for an e-commerce business specializing in skincare products. The objective is to efficiently pack different-sized skincare products into the smallest possible number of delivery boxes, hence reducing packaging costs and minimizing waste.

Proposed Solution

In this report, Array List and Tree Map are the data structures used to manage the storage of bins and items, whereas First Fit (FF) and Next Fit (NF) are the packing algorithms implemented and evaluated to solve the bin-packing problem. Specifically, the Array List is paired with the First Fit algorithm, while the Stack is paired with the Next Fit algorithm, to analyze the performance of each combination in terms of packing efficiency.

Data Structures

The ArrayList is a dynamic, resizable array that stores the elements in a sequential and indexed order. Meanwhile, s Stack is a linear data structure that operates based on a specific sequence for performing actions. This sequence can follow either the LIFO (Last In, First Out) or FILO (First In, Last Out) principle. In LIFO, the most recently added element is the first to be removed, while in FILO, the earliest added element is the last to be removed. The differences between both data structures are discussed in Table 1.

Feature	ArrayList	Stack
Implementation	Dynamic array	Linear data structure
Ordering	Insertion order	LIFO (Last In First Out) or FILO (First In Last Out)
Access: Time Complexity	O(1) for indexed access	O(1) for accessing top element
Traversal: Time Complexity	O(n)	O(n) if full traversal needed
Search: Time Complexity	O(n)	O(n)
Insertion: Time Complexity	O(1) for insertion at end / O(n) for insertion elsewhere	O(1) (push operation)
Duplicates allowed	Yes	Yes

Table 1: Differences between ArrayList and Stack

The ArrayList provides a simple, linear structure for sequential access, making it ideal for scenarios requiring dynamic resizing and quick indexed access. On the other hand, operates on a LIFO principle, making it well-suited for tasks that require Last In, First Out behaviour. Choosing the right structure depends on the specific needs of the application, such as access patterns, performance, and the problem being solved.

Bin Packing Strategies

The main differences between both algorithms are discussed in Table 2.

Feature	First Fit (FF)	Next Fit (NF)
Approach	Place each item into the first bin that has enough space	Place each item into the current bin if it fits. Otherwise, closes the current bin and opens a new one
Bin Selection	Scan bins in the order they were opened and selects the first suitable bin	Considers only the current open bin for each item
Efficiency	Slower due to more search, especially when dealing with large number of bins	Faster due to minimal search operations and requires only constant additional space as it maintains only one open bin at any time
Space Utilization	Better space utilization due to tendency of filling bins more completely before opening a new one	May leave more unused space in bins
Time Complexity (Worst Case)	O(n^2)	O(n)
Time Complexity (Best Case)	O(n)	O(n)
Time Complexity (Optimized Implementation)	O(n log n)	O(n)

Table 2: Differences between First Fit (FF) and Next Fit (NF) algorithms

The implementation and explanation of each algorithm will be discussed in detailed individually in the following sections.

First Fit

The First Fit algorithm is a simple and straightforward heuristic approach to solve the bin-packing problem. It processes items sequentially, placing item into the first bin that has sufficient remaining capacity. If no existing bin can accommodate the item, a new bin is used. In the context of delivery box packing for an e-commerce business specializing in skincare products, the First Fit algorithm works as well:

1. Initialize a list of empty boxes.
2. Search the list of existing boxes in order.
3. Place the skincare product into the first box that can fit in.
4. If no suitable box is found, a new box will be created and the skincare product will be placed into it.

To analyze the time complexity of First Fit algorithm, code structure of First Fit is shown:

```java
public ArrayList<Box> FirstFitPacking(ArrayList<Item>
items, double boxCapacity) {
        ArrayList<Box> boxes = new ArrayList<>();
        //Iterate through each item to place it into box
        for (Item item : items) {
            boolean placed = false;
            //Try to place item in an existing box
            for (Box box : boxes) {
                if (box.addItem(item)) {
                    placed = true;
                    break; //Stop checking other box
                }
            }
            //If item does not fit in any existing box,
create a new box
            if (!placed) {
                Box newBox = new Box(boxCapacity);
                newBox.addItem(item);
                boxes.add(newBox);
            }
        }

        return boxes;
    }
```

In the First Fit Algorithm,

The outer loop `for (Item item : items)` iterates over all the items in the item list, so it will run n times.

The inner loop `for (Box box : boxes)` check if the current item fits in any of the existing boxes. In the worst case, the number of boxes (m) will be equal to the number of items. The other operations in the algorithm are all O(1) operation.

```
    //Method to add an item into the box
    public boolean addItem(Item item) {
        if(item.getVolume() < remainingCapacity)
{ //Check if item fits
            items.add(item);
//Add item to the box
            remainingCapacity -= item.getVolume();
//reduce remaining capacity
            return true;
        }
        return false; //Item does not fit
    }
```

Inside of the addItem method,

`if(item.getVolume() < remainingCapacity)` simple comparison => O(1)

`items.add(item)` adding to an ArrayList => O(1)

`remainingCapacity -= item.getVolume()` subtraction => O(1)

hence, time complexity of the method adds up will be O(1).

Worst-case Scenario:

In the worst case when no items can fit in existing boxes and every item creates a new box:

1. For the first item, the inner loop runs 0 times as there are no boxes yet
2. For the second item, the inner loop runs 1 time since there is only 1 box
3. For the third item, the inner loop runs 2 times and so on

Through this we can conclude that for the nth item, the inner loop runs n-1 times. Thus, the number of operations across all iterations of the inner loop is the sum of the series

$$0 + 1 + 2 + \cdots + (n - 1)$$

The sum of the first n-1 integers is

$$\frac{(n-1)\ \times n}{2} = O(n^2)$$

Therefore, the time complexity of the First Fit algorithm is O(n^2) in the worst case.

Best-case Scenario:

On the other hand, in the best-case scenario occurs when item fits into the first box checked requiring minimal search. This means that the inner loop runs only 1 time for each item as the item will fit into the first available box. The outer loop will still run for n times which is once for every item in the list. Lastly, the addItem runs O(1) for each item. Thus, the total time complexity is:

$$O(n) \times O(1) = O(n)$$

Therefore, the time complexity of the First Fit algorithm is O(n) in the best case.

Optimization with Data Structure:

The time complexity can be optimized by using data structure like Self-Balancing Binary Search Trees to keep track of remaining capacities could reduce the search time for a suitable box from O(m) to O(log m). Hence, the time complexity will be:

$$\log m = \log n \ \Rightarrow O(n \ \times \log n) = O(n \log n)$$

Next Fit

The Next Fit algorithm is also a heuristic approach, which places each item into the current open bin if it fits. Otherwise, it closes the current bin and opens a new one. In the context of delivery box packing for an e-commerce business specializing in skincare products, the Next Fit algorithm works as follows:

1. Initialize an empty box.
2. Check if the skincare product fits into the current open box.
3. If it fits, the skincare product is placed into the current box.
4. If it does not fit, the current box is closed, a new box is opened and the skincare product is placed into the new box.

To analyze the time complexity of Next Fit algorithm, code structure of Next Fit is shown:

```
    @Override
    public Stack<Bin> allocateItem(ArrayList<Item>
items, double binCapacity){
        Stack<Bin> binStack = new Stack<>();
        Bin currentBin = new Bin(binCapacity);
        for (Item item: items) {
            if (!currentBin.addItem(item)) {
                binStack.push(currentBin);
                currentBin = createNewBin();
                currentBin.addItem(item);

            }
        }
        binStack.push(currentBin);
        return binStack;

    }
```

In the Next Fit Algorithm,

The outer loop `for (Item item: items)` iterates through each item in the items list, so it will run for n times.

The operations `if (!currentBin.addItem(item))`, pushing bin to stack, creating new bin, adding new item to the new bin and pushing the final bin onto stack all have Time Complexity of O(1). They are all constant-time operations.

Worst-Case Scenario:

The worst-case scenario occurs when no item fits in the current bin and a new bin is created for each item. In this case, the algorithm must create a new bin for each item, push the old bin onto the stack, and then add the item to the newly created bin. The outer loop run n times, and each iteration performs the constant-time operations within the loop, so the time complexity is:

$$O(n) \times O(1) = O(n)$$

Thus, the time complexity of Next Fit Algorithm is O(n) in the worst case.

Best-Case Scenario:

The best-case scenario occurs when every item fits into the first bin it tries and no new bin is created. The outer loop will still run for n times to iterate through the item in the list. The addItem and the stack operations are both O(1). Since the algorithm only checks the current bin for each item and does not need to create new bins, the time complexity is:

$$O(n)$$

Thus, the time complexity of Next Fit Algorithm is O(n) in the best case which is similar to the worst-case scenario.

Optimization with Data Structure:

The Next Fit algorithm operates by maintaining a single open bin and placing each item into this bin if it fits. If an item doesn't fit, a new bin is opened. Given that Next Fit only keeps track of one open bin at a time, introducing more complex or advanced data structures doesn't provide a performance benefit. Therefore, the time complexity will remain as:

$$O(n)$$

Thus, the simplicity of Next Fit leads to its O(n) time complexity, and there is no room for optimization through different data structures.

Flowchart

```
public boolean addItem(Item item)
```

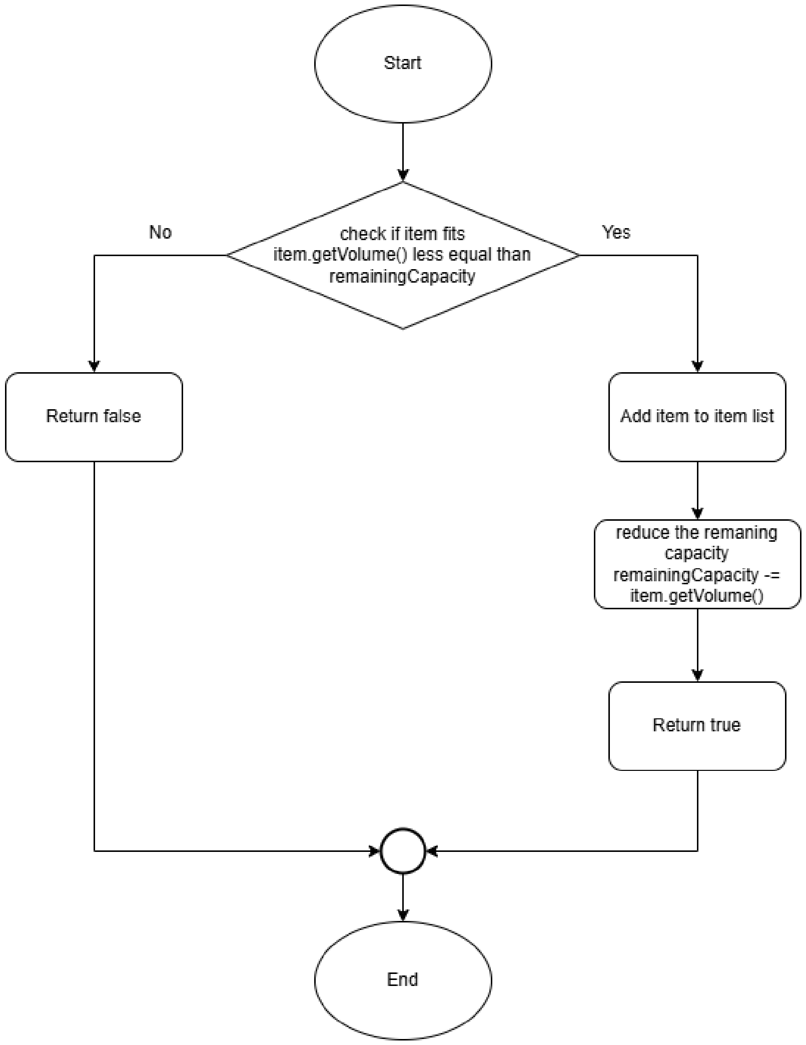
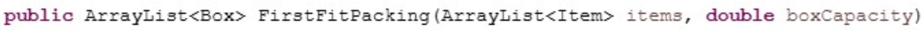


Figure 1.1: Flowchart of addItem method used by First Fit Algorithm and Next Fit Algorithm



```
public ArrayList<Box> FirstFitPacking(ArrayList<Item> items, double boxCapacity)
```

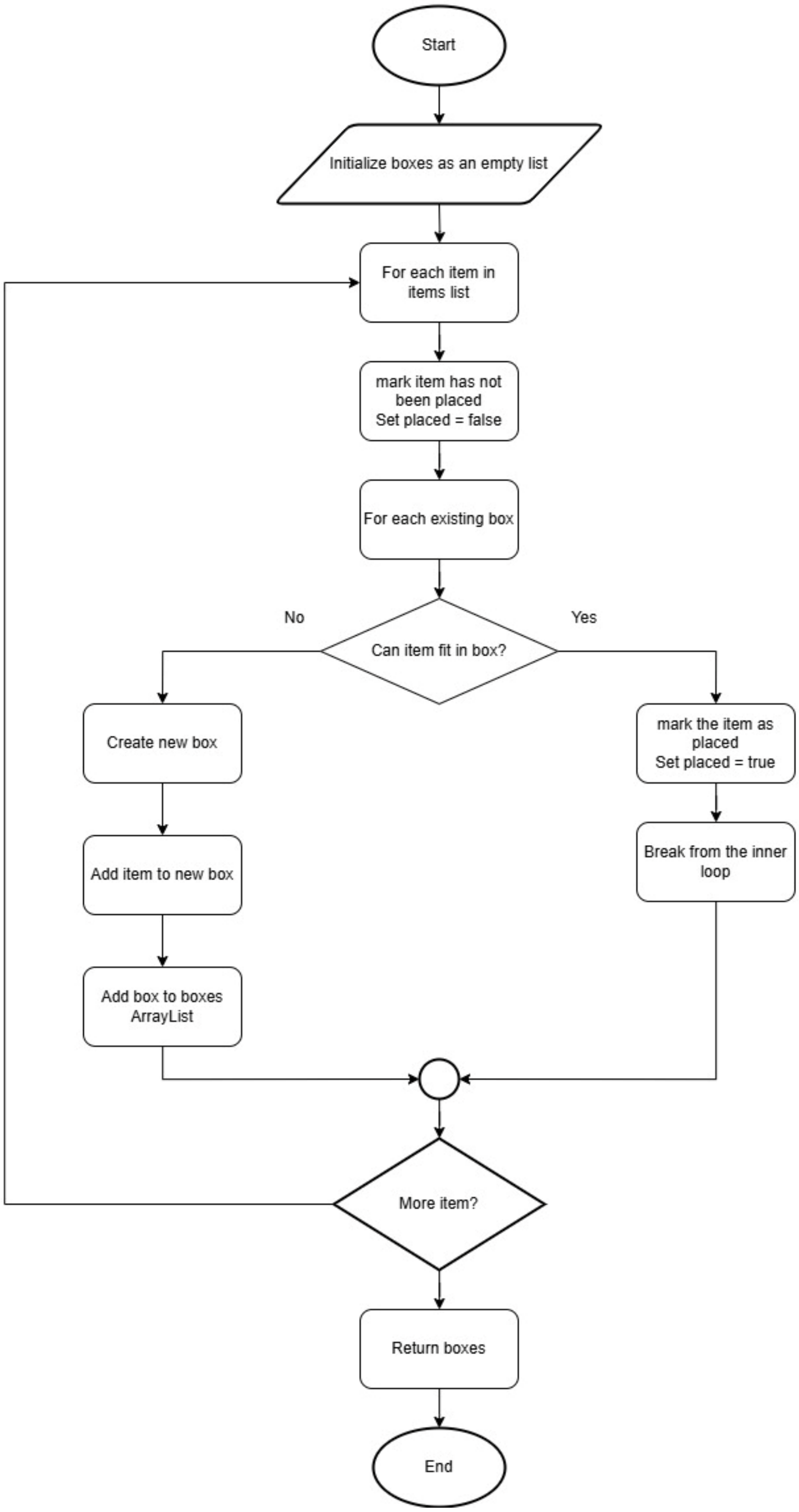
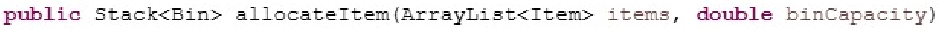


Figure 1.2: Flowchart of First Fit Algorithm



```
public Stack<Bin> allocateItem(ArrayList<Item> items, double binCapacity)
```

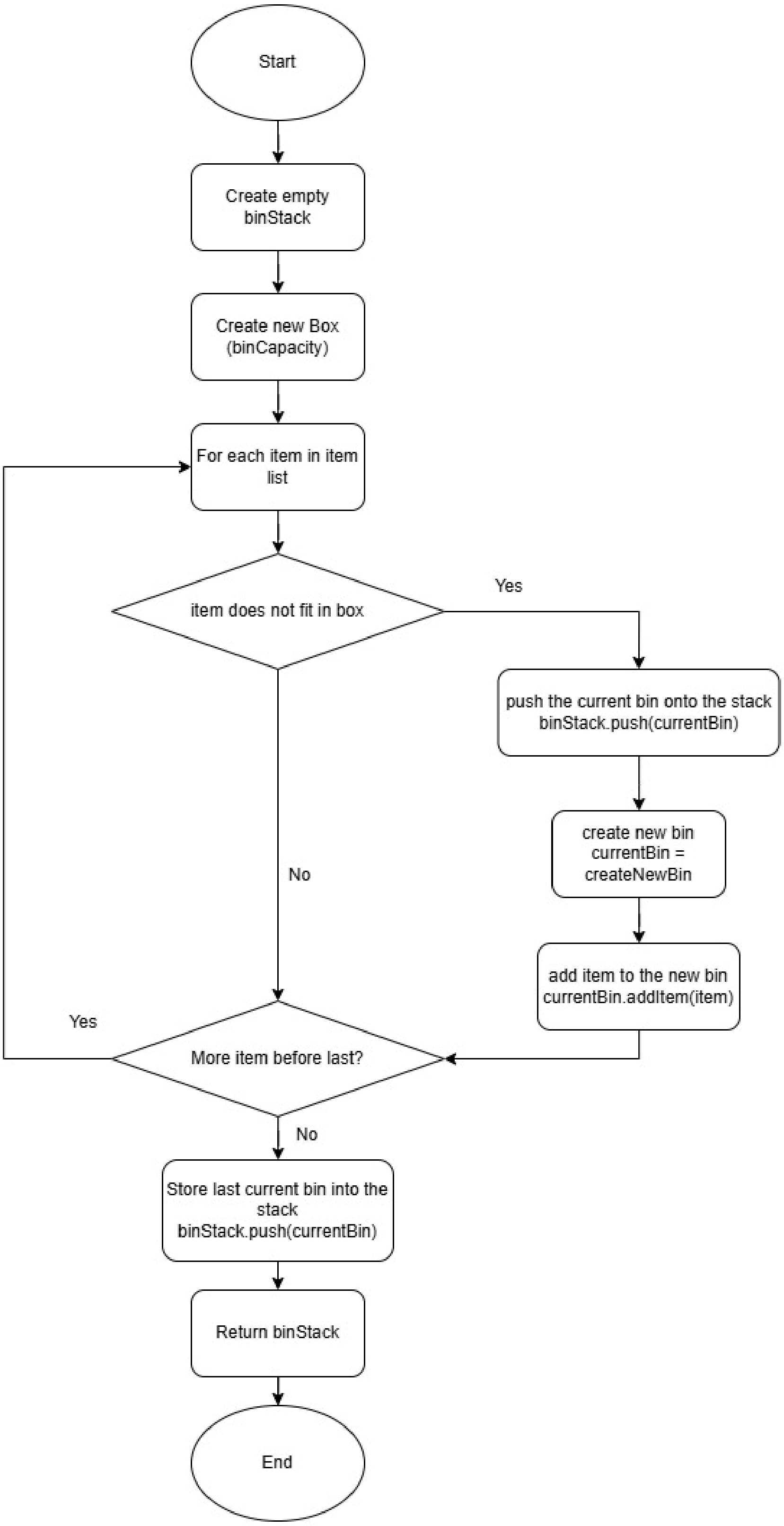

Start
Create empty binStack
Create new Box (binCapacity)
For each item in item list
item does not fit in box
Yes
push the current bin onto the stack binStack.push(currentBin)
create new bin currentBin = createNewBin
add item to the new bin currentBin.addItem(item)
No
Yes
More item before last?
No
Store last current bin into the stack binStack.push(currentBin)
Return binStack
End

Figure 2.1: Flowchart of Next Fit Algorithm

UML Diagram

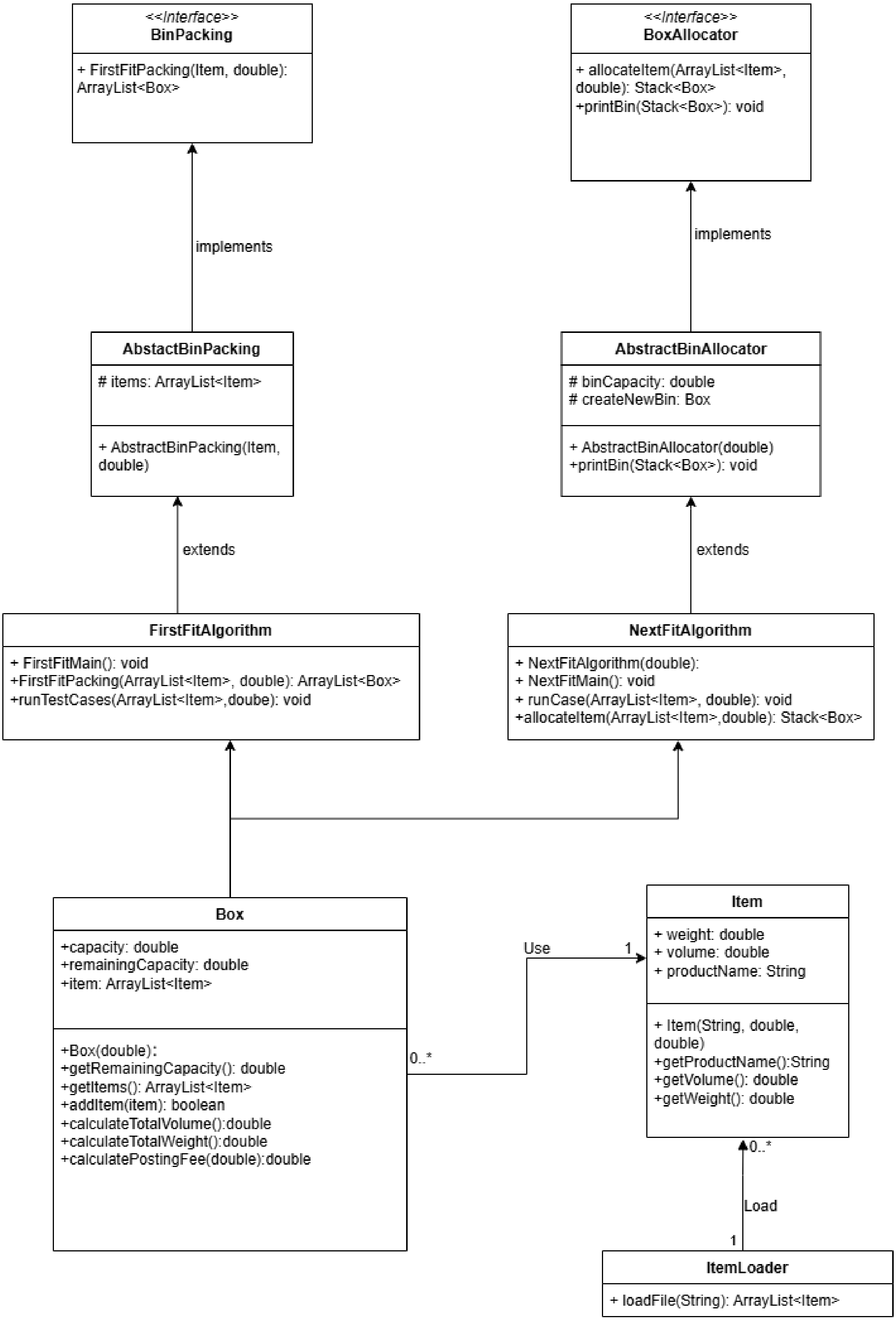
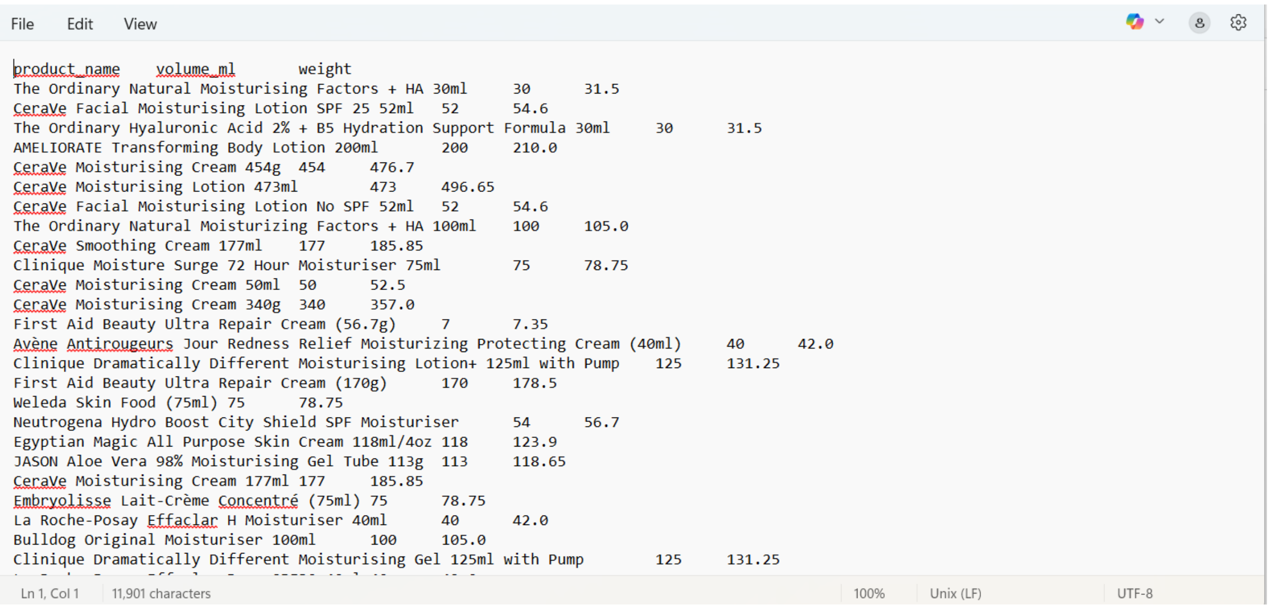

<<Interface>>
BinPacking
+ FirstFitPacking(Item, double): ArrayList<Box>
<<Interface>>
BoxAllocator
+ allocateItem(ArrayList<Item>, double): Stack<Box>
+printBin(Stack<Box>): void
implements
implements
AbstactBinPacking
# items: ArrayList<Item>
+ AbstractBinPacking(Item, double)
AbstractBinAllocator
# binCapacity: double
# createNewBin: Box
+ AbstractBinAllocator(double)
+printBin(Stack<Box>): void
extends
extends
FirstFitAlgorithm
+ FirstFitMain(): void
+FirstFitPacking(ArrayList<Item>, double): ArrayList<Box>
+runTestCases(ArrayList<Item>,doube): void
NextFitAlgorithm
+ NextFitAlgorithm(double):
+ NextFitMain(): void
+ runCase(ArrayList<Item>, double): void
+allocateItem(ArrayList<Item>,double): Stack<Box>
Box
+capacity: double
+remainingCapacity: double
+item: ArrayList<Item>
+Box(double):
+getRemainingCapacity(): double
+getItems(): ArrayList<Item>
+addItem(item): boolean
+calculateTotalVolume():double
+calculateTotalWeight():double
+calculatePostingFee(double):double
Use
1
0..*
Item
+ weight: double
+ volume: double
+ productName: String
+ Item(String, double, double)
+getProductName():String
+getVolume(): double
+getWeight(): double
0..*
Load
1
ItemLoader
+ loadFile(String): ArrayList<Item>

Figure 3: UML Diagram of the application

Sample Input Data and Test Cases

The dataset used in the case study contains 200 skincare products data collected for an e-commerce delivery box optimization problem. Each product is described by the name of skincare products (product_name), volume of the product in millilitres (volume_ml) and weight of product in grams (weight).



```
product_name    volume_ml       weight
The Ordinary Natural Moisturising Factors + HA 30ml     30      31.5
CeraVe Facial Moisturising Lotion SPF 25 52ml   52      54.6
The Ordinary Hyaluronic Acid 2% + B5 Hydration Support Formula 30ml     30      31.5
AMELIORATE Transforming Body Lotion 200ml       200     210.0
CeraVe Moisturising Cream 454g  454     476.7
CeraVe Moisturising Lotion 473ml        473     496.65
CeraVe Facial Moisturising Lotion No SPF 52ml   52      54.6
The Ordinary Natural Moisturizing Factors + HA 100ml    100     105.0
CeraVe Smoothing Cream 177ml    177     185.85
Clinique Moisture Surge 72 Hour Moisturiser 75ml        75      78.75
CeraVe Moisturising Cream 50ml  50      52.5
CeraVe Moisturising Cream 340g  340     357.0
First Aid Beauty Ultra Repair Cream (56.7g)     7       7.35
Avène Antirougeurs Jour Redness Relief Moisturizing Protecting Cream (40ml)     40      42.0
Clinique Dramatically Different Moisturising Lotion+ 125ml with Pump    125     131.25
First Aid Beauty Ultra Repair Cream (170g)      170     178.5
Weleda Skin Food (75ml) 75      78.75
Neutrogena Hydro Boost City Shield SPF Moisturiser      54      56.7
Egyptian Magic All Purpose Skin Cream 118ml/4oz 118     123.9
JASON Aloe Vera 98% Moisturising Gel Tube 113g  113     118.65
CeraVe Moisturising Cream 177ml 177     185.85
Embryolisse Lait-Crème Concentré (75ml) 75      78.75
La Roche-Posay Effaclar H Moisturiser 40ml      40      42.0
Bulldog Original Moisturiser 100ml      100     105.0
Clinique Dramatically Different Moisturising Gel 125ml with Pump        125     131.25
```

Figure 4: Screenshot of sample input data (skincare_products_with_weight.txt)

To evaluate which packing algorithm performs better for packing skincare products of varying sizes, a series of test cases were designed and conducted. Different box sizes were used in these test cases to assess the efficiency of each algorithm as each box size increases in minimizing the number of boxes required, maximizing box space utilization, and reducing overall shipping costs.

To compare the overall shipping costs of both algorithms, the postage fee rate is set as Table 3.

Weight Range (kg)	Postage Rate Applied
Less than 0.5kg	RM 0.65 per kg
Between 0.501 kg and 1 kg	RM 0.70 per kg
Between 1 kg and 2 kg	RM 1.00 per kg
Between 2 kg and 5 kg	RM 1.50 per kg
Between 5kg and 10kg	RM 2.00 per kg
Between 10kg and 15kg	RM 3.00 per kg
Between 15kg and 20kg	RM 28.00 per kg

Table 3: Postage Fee Rate

Test case 1: Box capacity is fixed at 500 cm^3

Test case 2: Box capacity is fixed at 700 cm^3

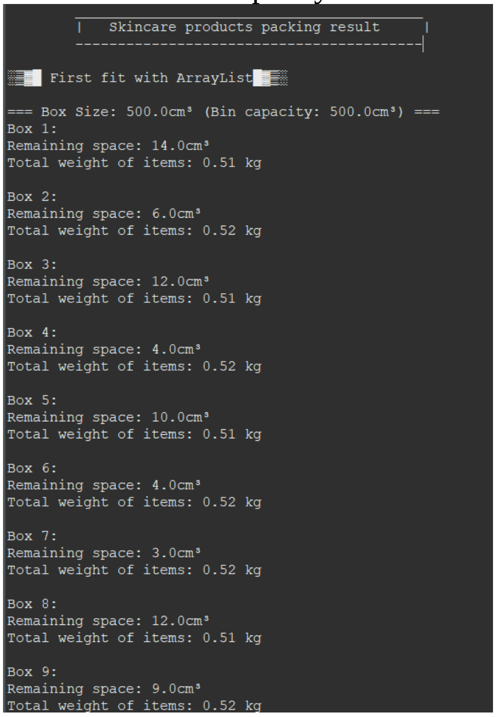
Test case 3: Box capacity is fixed at 900 cm^3

Test case 4: Box capacity is fixed at 1200 cm^3

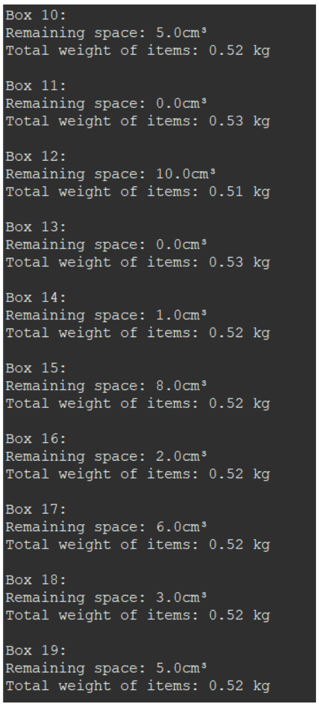
Test case 5: Box capacity is fixed at 1500 cm^3

Sample Outputs

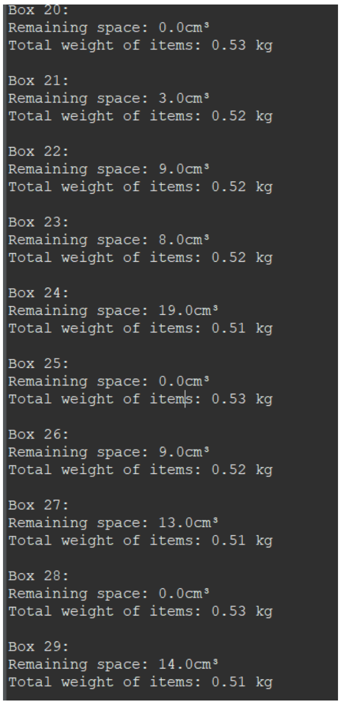
First Fit with ArrayList
Test case 1: Box capacity is fixed at 500 cm^3



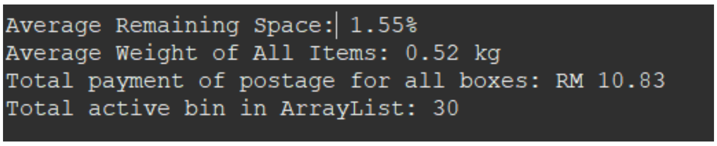
```
        ________________________________________
        |   Skincare products packing result    |
        ----------------------------------------|

░▒▓█ First fit with ArrayList█▓▒░

=== Box Size: 500.0cm³ (Bin capacity: 500.0cm³) ===
Box 1:
Remaining space: 14.0cm³
Total weight of items: 0.51 kg

Box 2:
Remaining space: 6.0cm³
Total weight of items: 0.52 kg

Box 3:
Remaining space: 12.0cm³
Total weight of items: 0.51 kg

Box 4:
Remaining space: 4.0cm³
Total weight of items: 0.52 kg

Box 5:
Remaining space: 10.0cm³
Total weight of items: 0.51 kg

Box 6:
Remaining space: 4.0cm³
Total weight of items: 0.52 kg

Box 7:
Remaining space: 3.0cm³
Total weight of items: 0.52 kg

Box 8:
Remaining space: 12.0cm³
Total weight of items: 0.51 kg

Box 9:
Remaining space: 9.0cm³
Total weight of items: 0.52 kg
```



```
Box 10:
Remaining space: 5.0cm³
Total weight of items: 0.52 kg

Box 11:
Remaining space: 0.0cm³
Total weight of items: 0.53 kg

Box 12:
Remaining space: 10.0cm³
Total weight of items: 0.51 kg

Box 13:
Remaining space: 0.0cm³
Total weight of items: 0.53 kg

Box 14:
Remaining space: 1.0cm³
Total weight of items: 0.52 kg

Box 15:
Remaining space: 8.0cm³
Total weight of items: 0.52 kg

Box 16:
Remaining space: 2.0cm³
Total weight of items: 0.52 kg

Box 17:
Remaining space: 6.0cm³
Total weight of items: 0.52 kg

Box 18:
Remaining space: 3.0cm³
Total weight of items: 0.52 kg

Box 19:
Remaining space: 5.0cm³
Total weight of items: 0.52 kg
```

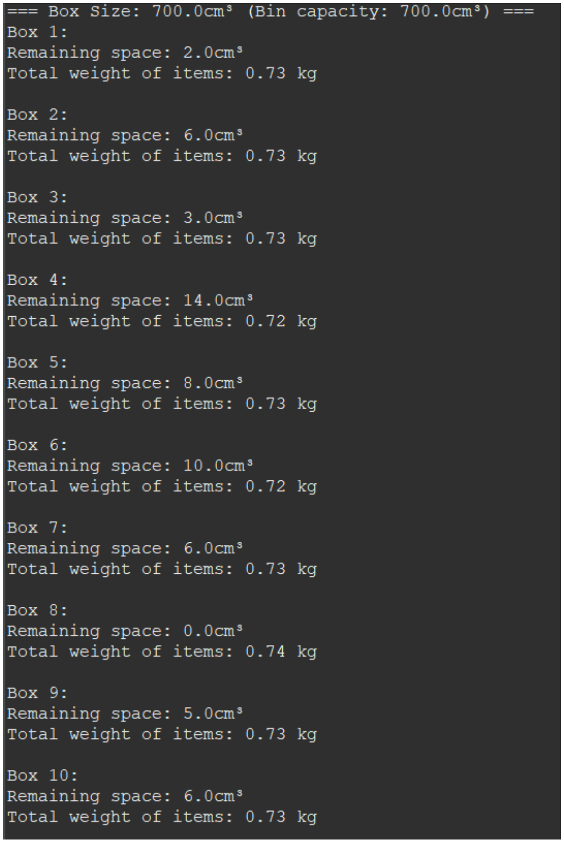


```
Box 20:
Remaining space: 0.0cm³
Total weight of items: 0.53 kg

Box 21:
Remaining space: 3.0cm³
Total weight of items: 0.52 kg

Box 22:
Remaining space: 9.0cm³
Total weight of items: 0.52 kg

Box 23:
Remaining space: 8.0cm³
Total weight of items: 0.52 kg

Box 24:
Remaining space: 19.0cm³
Total weight of items: 0.51 kg

Box 25:
Remaining space: 0.0cm³
Total weight of items: 0.53 kg

Box 26:
Remaining space: 9.0cm³
Total weight of items: 0.52 kg

Box 27:
Remaining space: 13.0cm³
Total weight of items: 0.51 kg

Box 28:
Remaining space: 0.0cm³
Total weight of items: 0.53 kg

Box 29:
Remaining space: 14.0cm³
Total weight of items: 0.51 kg
```

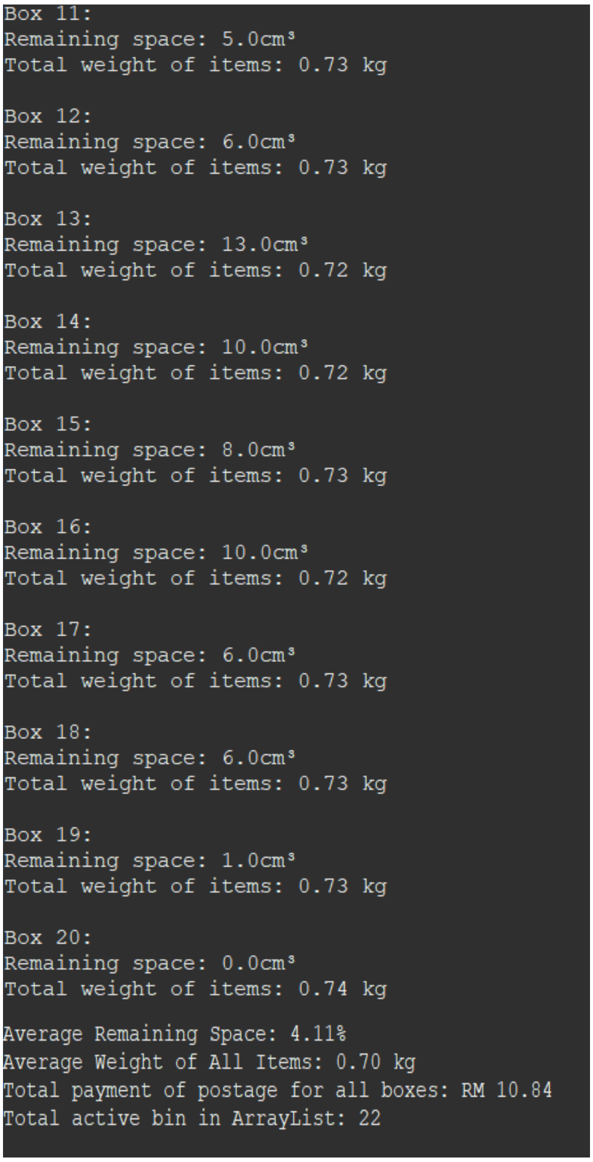


```
Average Remaining Space: 1.55%
Average Weight of All Items: 0.52 kg
Total payment of postage for all boxes: RM 10.83
Total active bin in ArrayList: 30
```

Test case 2: Box capacity is fixed at 700 cm^3

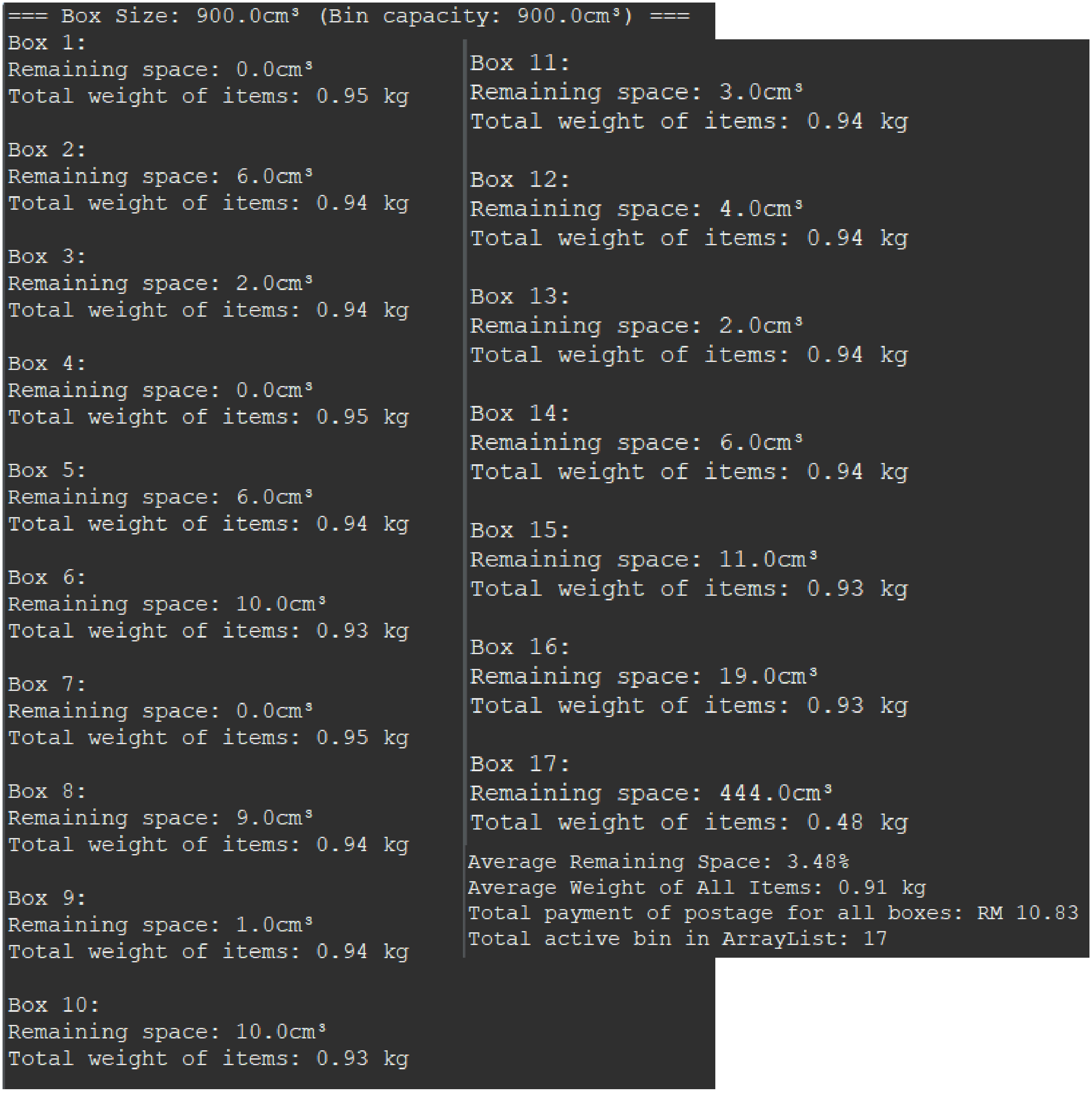


```
=== Box Size: 700.0cm³ (Bin capacity: 700.0cm³) ===
Box 1:
Remaining space: 2.0cm³
Total weight of items: 0.73 kg

Box 2:
Remaining space: 6.0cm³
Total weight of items: 0.73 kg

Box 3:
Remaining space: 3.0cm³
Total weight of items: 0.73 kg

Box 4:
Remaining space: 14.0cm³
Total weight of items: 0.72 kg

Box 5:
Remaining space: 8.0cm³
Total weight of items: 0.73 kg

Box 6:
Remaining space: 10.0cm³
Total weight of items: 0.72 kg

Box 7:
Remaining space: 6.0cm³
Total weight of items: 0.73 kg

Box 8:
Remaining space: 0.0cm³
Total weight of items: 0.74 kg

Box 9:
Remaining space: 5.0cm³
Total weight of items: 0.73 kg

Box 10:
Remaining space: 6.0cm³
Total weight of items: 0.73 kg
```



```
Box 11:
Remaining space: 5.0cm³
Total weight of items: 0.73 kg

Box 12:
Remaining space: 6.0cm³
Total weight of items: 0.73 kg

Box 13:
Remaining space: 13.0cm³
Total weight of items: 0.72 kg

Box 14:
Remaining space: 10.0cm³
Total weight of items: 0.72 kg

Box 15:
Remaining space: 8.0cm³
Total weight of items: 0.73 kg

Box 16:
Remaining space: 10.0cm³
Total weight of items: 0.72 kg

Box 17:
Remaining space: 6.0cm³
Total weight of items: 0.73 kg

Box 18:
Remaining space: 6.0cm³
Total weight of items: 0.73 kg

Box 19:
Remaining space: 1.0cm³
Total weight of items: 0.73 kg

Box 20:
Remaining space: 0.0cm³
Total weight of items: 0.74 kg

Average Remaining Space: 4.11%
Average Weight of All Items: 0.70 kg
Total payment of postage for all boxes: RM 10.84
Total active bin in ArrayList: 22
```

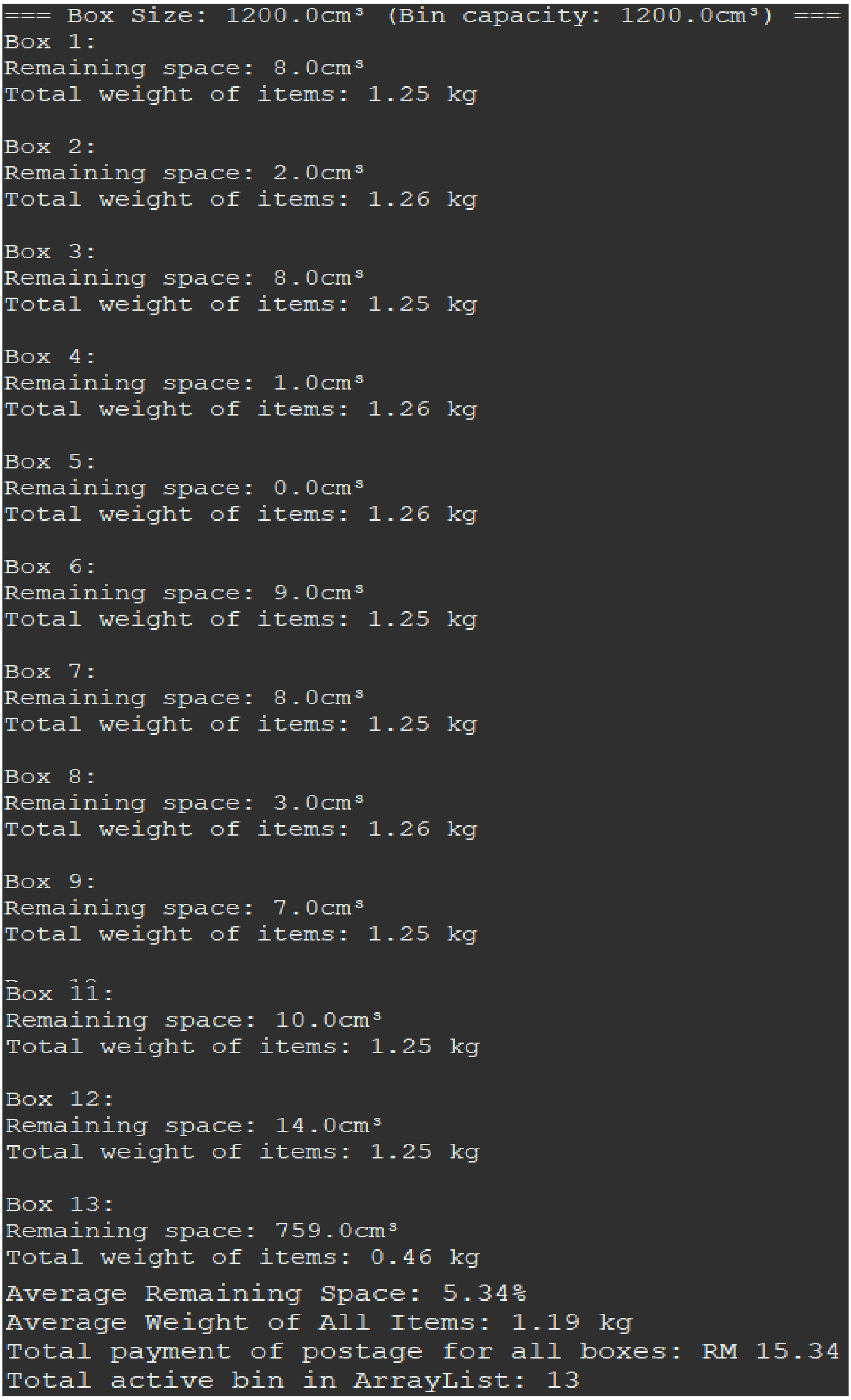
Test case 3: Box capacity is fixed at 900 cm^3



```
=== Box Size: 900.0cm³ (Bin capacity: 900.0cm³) ===
Box 1:
Remaining space: 0.0cm³
Total weight of items: 0.95 kg

Box 2:
Remaining space: 6.0cm³
Total weight of items: 0.94 kg

Box 3:
Remaining space: 2.0cm³
Total weight of items: 0.94 kg

Box 4:
Remaining space: 0.0cm³
Total weight of items: 0.95 kg

Box 5:
Remaining space: 6.0cm³
Total weight of items: 0.94 kg

Box 6:
Remaining space: 10.0cm³
Total weight of items: 0.93 kg

Box 7:
Remaining space: 0.0cm³
Total weight of items: 0.95 kg

Box 8:
Remaining space: 9.0cm³
Total weight of items: 0.94 kg

Box 9:
Remaining space: 1.0cm³
Total weight of items: 0.94 kg

Box 10:
Remaining space: 10.0cm³
Total weight of items: 0.93 kg
```

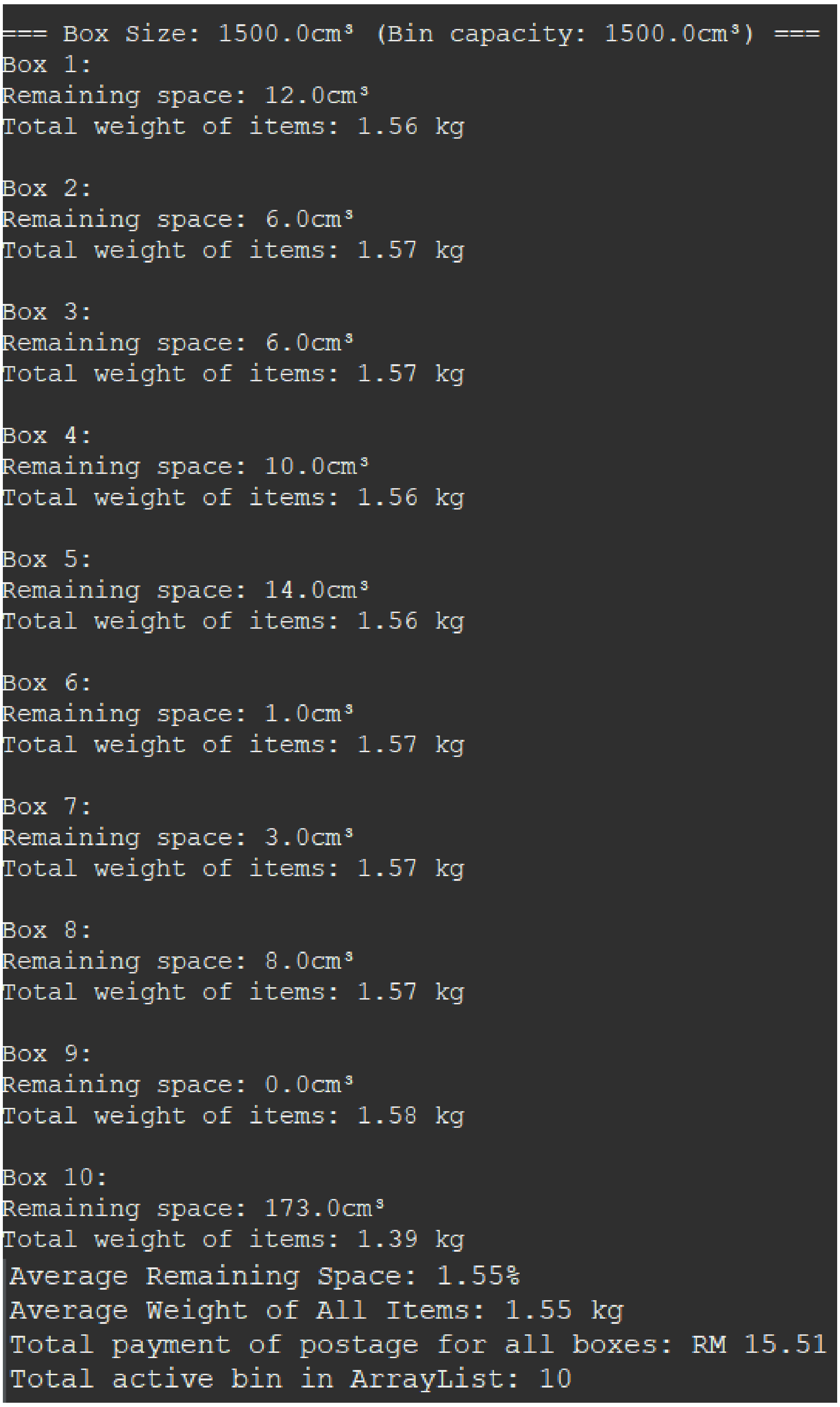
```
Box 11:
Remaining space: 3.0cm³
Total weight of items: 0.94 kg

Box 12:
Remaining space: 4.0cm³
Total weight of items: 0.94 kg

Box 13:
Remaining space: 2.0cm³
Total weight of items: 0.94 kg

Box 14:
Remaining space: 6.0cm³
Total weight of items: 0.94 kg

Box 15:
Remaining space: 11.0cm³
Total weight of items: 0.93 kg

Box 16:
Remaining space: 19.0cm³
Total weight of items: 0.93 kg

Box 17:
Remaining space: 444.0cm³
Total weight of items: 0.48 kg
Average Remaining Space: 3.48%
Average Weight of All Items: 0.91 kg
Total payment of postage for all boxes: RM 10.83
Total active bin in ArrayList: 17
```

Test case 4: Box capacity is fixed at 1200 cm^3



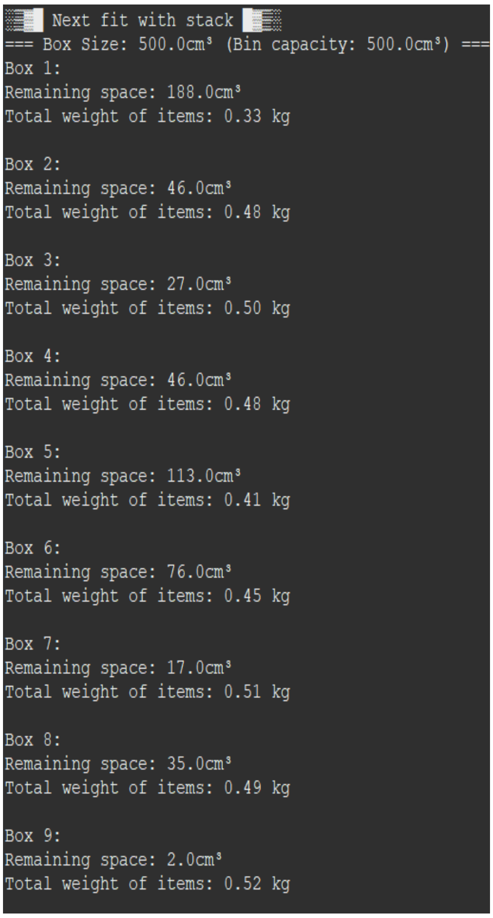
```
=== Box Size: 1200.0cm³ (Bin capacity: 1200.0cm³) ===
Box 1:
Remaining space: 8.0cm³
Total weight of items: 1.25 kg

Box 2:
Remaining space: 2.0cm³
Total weight of items: 1.26 kg

Box 3:
Remaining space: 8.0cm³
Total weight of items: 1.25 kg

Box 4:
Remaining space: 1.0cm³
Total weight of items: 1.26 kg

Box 5:
Remaining space: 0.0cm³
Total weight of items: 1.26 kg

Box 6:
Remaining space: 9.0cm³
Total weight of items: 1.25 kg

Box 7:
Remaining space: 8.0cm³
Total weight of items: 1.25 kg

Box 8:
Remaining space: 3.0cm³
Total weight of items: 1.26 kg

Box 9:
Remaining space: 7.0cm³
Total weight of items: 1.25 kg

Box 11:
Remaining space: 10.0cm³
Total weight of items: 1.25 kg

Box 12:
Remaining space: 14.0cm³
Total weight of items: 1.25 kg

Box 13:
Remaining space: 759.0cm³
Total weight of items: 0.46 kg
Average Remaining Space: 5.34%
Average Weight of All Items: 1.19 kg
Total payment of postage for all boxes: RM 15.34
Total active bin in ArrayList: 13
```

Test case 5: Box capacity is fixed at 1500 cm^3

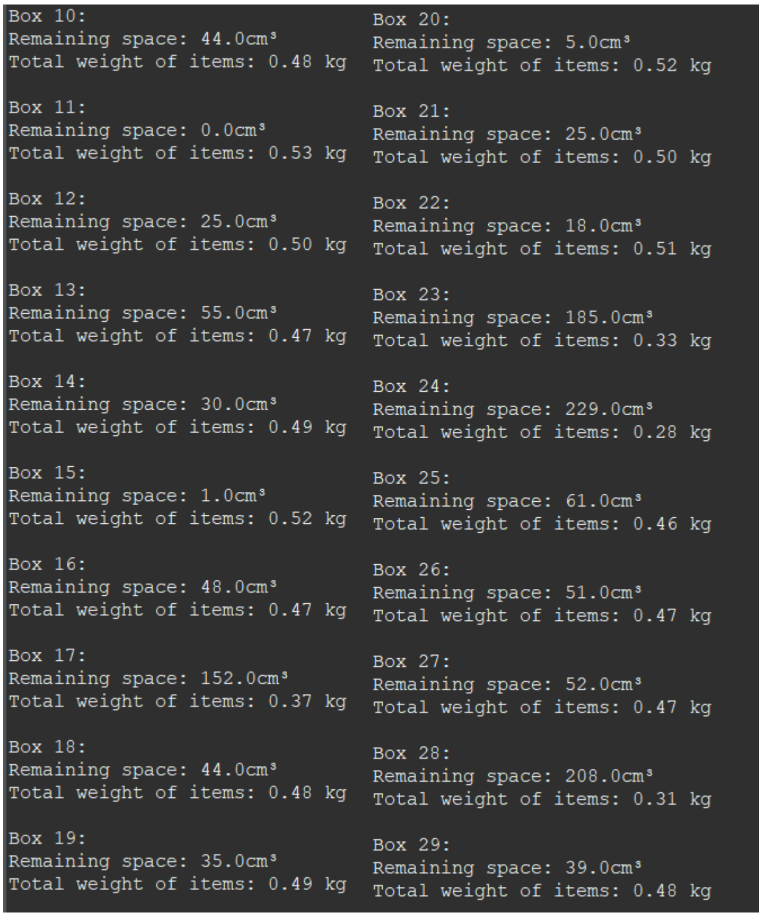


```
=== Box Size: 1500.0cm³ (Bin capacity: 1500.0cm³) ===
Box 1:
Remaining space: 12.0cm³
Total weight of items: 1.56 kg

Box 2:
Remaining space: 6.0cm³
Total weight of items: 1.57 kg

Box 3:
Remaining space: 6.0cm³
Total weight of items: 1.57 kg

Box 4:
Remaining space: 10.0cm³
Total weight of items: 1.56 kg

Box 5:
Remaining space: 14.0cm³
Total weight of items: 1.56 kg

Box 6:
Remaining space: 1.0cm³
Total weight of items: 1.57 kg

Box 7:
Remaining space: 3.0cm³
Total weight of items: 1.57 kg

Box 8:
Remaining space: 8.0cm³
Total weight of items: 1.57 kg

Box 9:
Remaining space: 0.0cm³
Total weight of items: 1.58 kg

Box 10:
Remaining space: 173.0cm³
Total weight of items: 1.39 kg
Average Remaining Space: 1.55%
Average Weight of All Items: 1.55 kg
Total payment of postage for all boxes: RM 15.51
Total active bin in ArrayList: 10
```

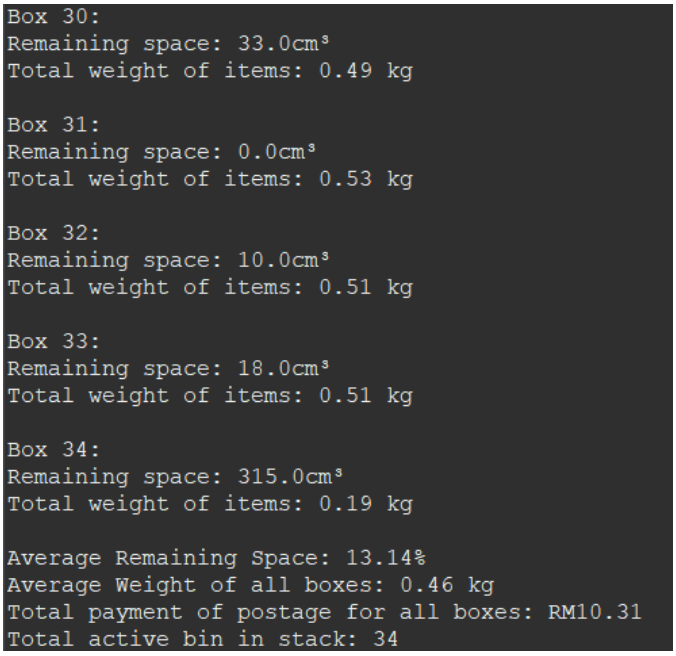
Next Fit with stack
Test case 1: Box capacity is fixed at 500 cm^3



```
░▒▓█ Next fit with stack █▓▒░
=== Box Size: 500.0cm³ (Bin capacity: 500.0cm³) ===
Box 1:
Remaining space: 188.0cm³
Total weight of items: 0.33 kg

Box 2:
Remaining space: 46.0cm³
Total weight of items: 0.48 kg

Box 3:
Remaining space: 27.0cm³
Total weight of items: 0.50 kg

Box 4:
Remaining space: 46.0cm³
Total weight of items: 0.48 kg

Box 5:
Remaining space: 113.0cm³
Total weight of items: 0.41 kg

Box 6:
Remaining space: 76.0cm³
Total weight of items: 0.45 kg

Box 7:
Remaining space: 17.0cm³
Total weight of items: 0.51 kg

Box 8:
Remaining space: 35.0cm³
Total weight of items: 0.49 kg

Box 9:
Remaining space: 2.0cm³
Total weight of items: 0.52 kg
```

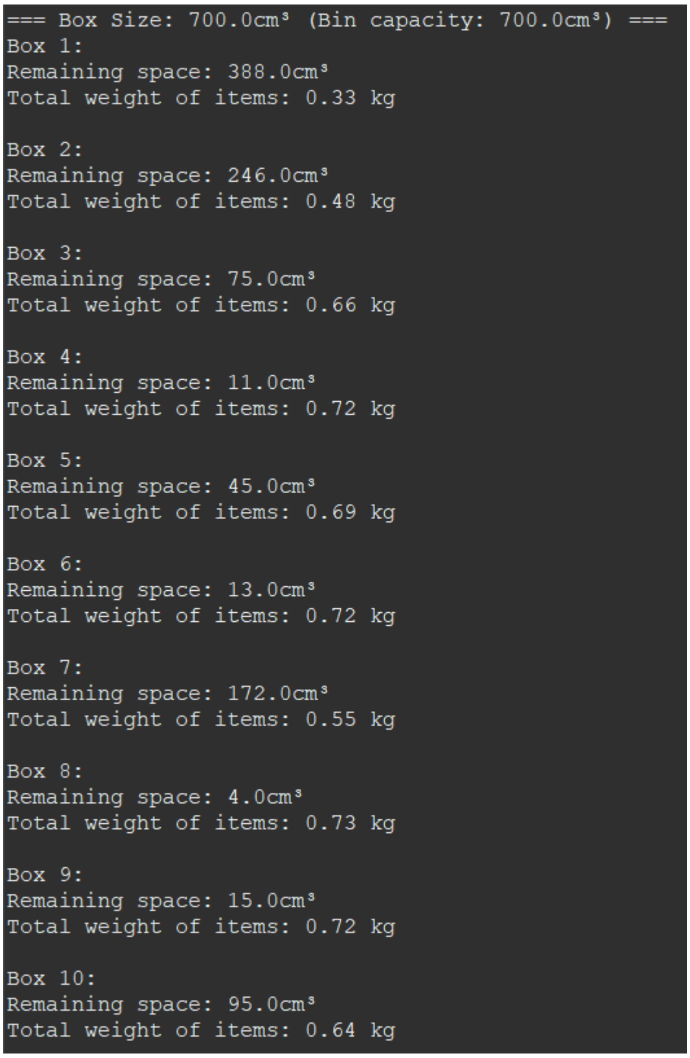


```
Box 10:                           Box 20:
Remaining space: 44.0cm³          Remaining space: 5.0cm³
Total weight of items: 0.48 kg    Total weight of items: 0.52 kg

Box 11:                           Box 21:
Remaining space: 0.0cm³           Remaining space: 25.0cm³
Total weight of items: 0.53 kg    Total weight of items: 0.50 kg

Box 12:                           Box 22:
Remaining space: 25.0cm³          Remaining space: 18.0cm³
Total weight of items: 0.50 kg    Total weight of items: 0.51 kg

Box 13:                           Box 23:
Remaining space: 55.0cm³          Remaining space: 185.0cm³
Total weight of items: 0.47 kg    Total weight of items: 0.33 kg

Box 14:                           Box 24:
Remaining space: 30.0cm³          Remaining space: 229.0cm³
Total weight of items: 0.49 kg    Total weight of items: 0.28 kg

Box 15:                           Box 25:
Remaining space: 1.0cm³           Remaining space: 61.0cm³
Total weight of items: 0.52 kg    Total weight of items: 0.46 kg

Box 16:                           Box 26:
Remaining space: 48.0cm³          Remaining space: 51.0cm³
Total weight of items: 0.47 kg    Total weight of items: 0.47 kg

Box 17:                           Box 27:
Remaining space: 152.0cm³         Remaining space: 52.0cm³
Total weight of items: 0.37 kg    Total weight of items: 0.47 kg

Box 18:                           Box 28:
Remaining space: 44.0cm³          Remaining space: 208.0cm³
Total weight of items: 0.48 kg    Total weight of items: 0.31 kg

Box 19:                           Box 29:
Remaining space: 35.0cm³          Remaining space: 39.0cm³
Total weight of items: 0.49 kg    Total weight of items: 0.48 kg
```

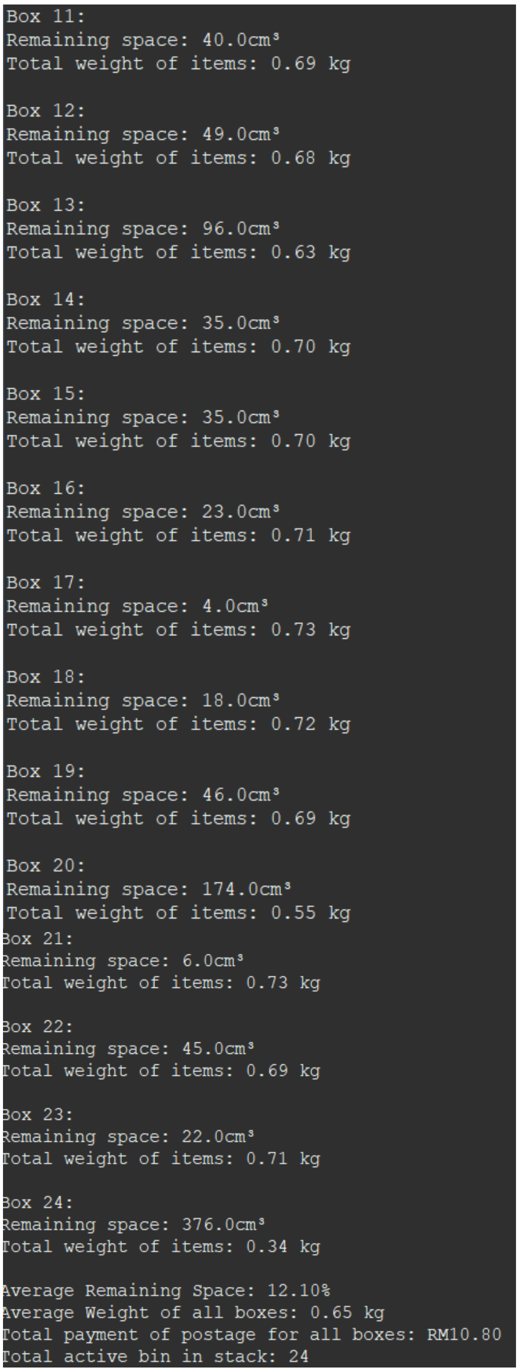


```
Box 30:
Remaining space: 33.0cm³
Total weight of items: 0.49 kg

Box 31:
Remaining space: 0.0cm³
Total weight of items: 0.53 kg

Box 32:
Remaining space: 10.0cm³
Total weight of items: 0.51 kg

Box 33:
Remaining space: 18.0cm³
Total weight of items: 0.51 kg

Box 34:
Remaining space: 315.0cm³
Total weight of items: 0.19 kg

Average Remaining Space: 13.14%
Average Weight of all boxes: 0.46 kg
Total payment of postage for all boxes: RM10.31
Total active bin in stack: 34
```

Test case 2: Box capacity is fixed at 700 cm^3

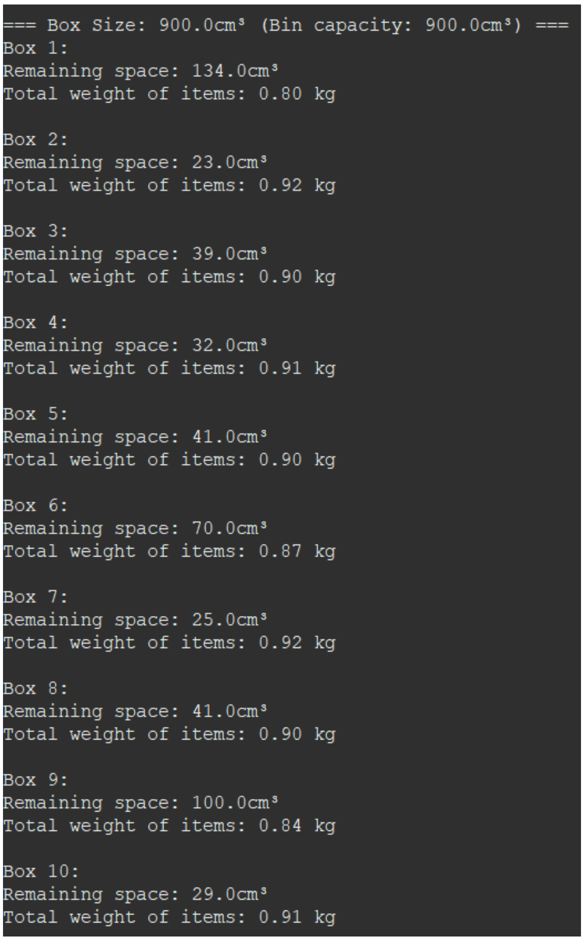


```
=== Box Size: 700.0cm³ (Bin capacity: 700.0cm³) ===
Box 1:
Remaining space: 388.0cm³
Total weight of items: 0.33 kg

Box 2:
Remaining space: 246.0cm³
Total weight of items: 0.48 kg

Box 3:
Remaining space: 75.0cm³
Total weight of items: 0.66 kg

Box 4:
Remaining space: 11.0cm³
Total weight of items: 0.72 kg

Box 5:
Remaining space: 45.0cm³
Total weight of items: 0.69 kg

Box 6:
Remaining space: 13.0cm³
Total weight of items: 0.72 kg

Box 7:
Remaining space: 172.0cm³
Total weight of items: 0.55 kg

Box 8:
Remaining space: 4.0cm³
Total weight of items: 0.73 kg

Box 9:
Remaining space: 15.0cm³
Total weight of items: 0.72 kg

Box 10:
Remaining space: 95.0cm³
Total weight of items: 0.64 kg
```

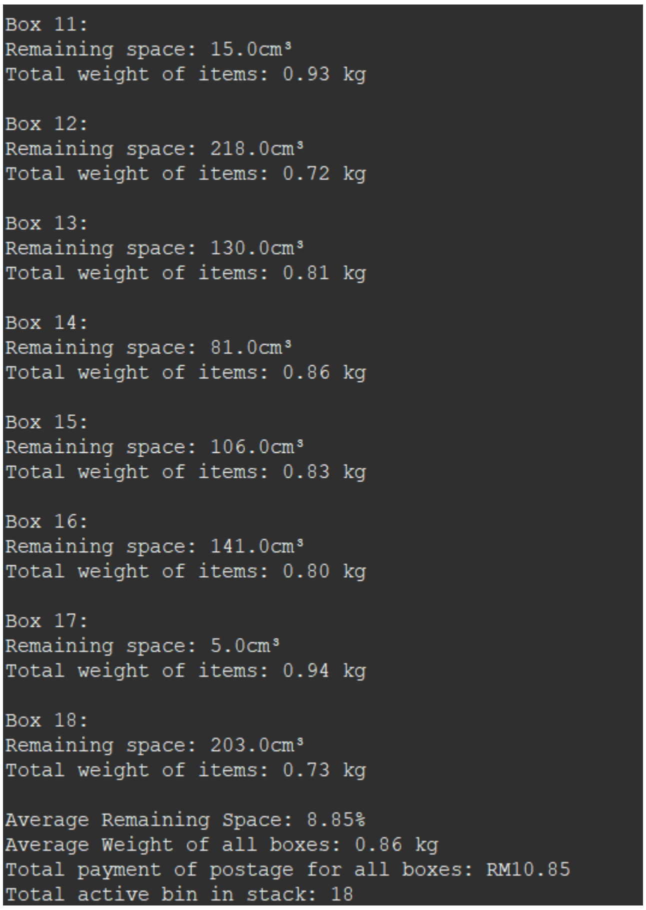


```
Box 11:
Remaining space: 40.0cm³
Total weight of items: 0.69 kg

Box 12:
Remaining space: 49.0cm³
Total weight of items: 0.68 kg

Box 13:
Remaining space: 96.0cm³
Total weight of items: 0.63 kg

Box 14:
Remaining space: 35.0cm³
Total weight of items: 0.70 kg

Box 15:
Remaining space: 35.0cm³
Total weight of items: 0.70 kg

Box 16:
Remaining space: 23.0cm³
Total weight of items: 0.71 kg

Box 17:
Remaining space: 4.0cm³
Total weight of items: 0.73 kg

Box 18:
Remaining space: 18.0cm³
Total weight of items: 0.72 kg

Box 19:
Remaining space: 46.0cm³
Total weight of items: 0.69 kg

Box 20:
Remaining space: 174.0cm³
Total weight of items: 0.55 kg
Box 21:
Remaining space: 6.0cm³
Total weight of items: 0.73 kg

Box 22:
Remaining space: 45.0cm³
Total weight of items: 0.69 kg

Box 23:
Remaining space: 22.0cm³
Total weight of items: 0.71 kg

Box 24:
Remaining space: 376.0cm³
Total weight of items: 0.34 kg

Average Remaining Space: 12.10%
Average Weight of all boxes: 0.65 kg
Total payment of postage for all boxes: RM10.80
Total active bin in stack: 24
```

Test case 3: Box capacity is fixed at 900 cm^3

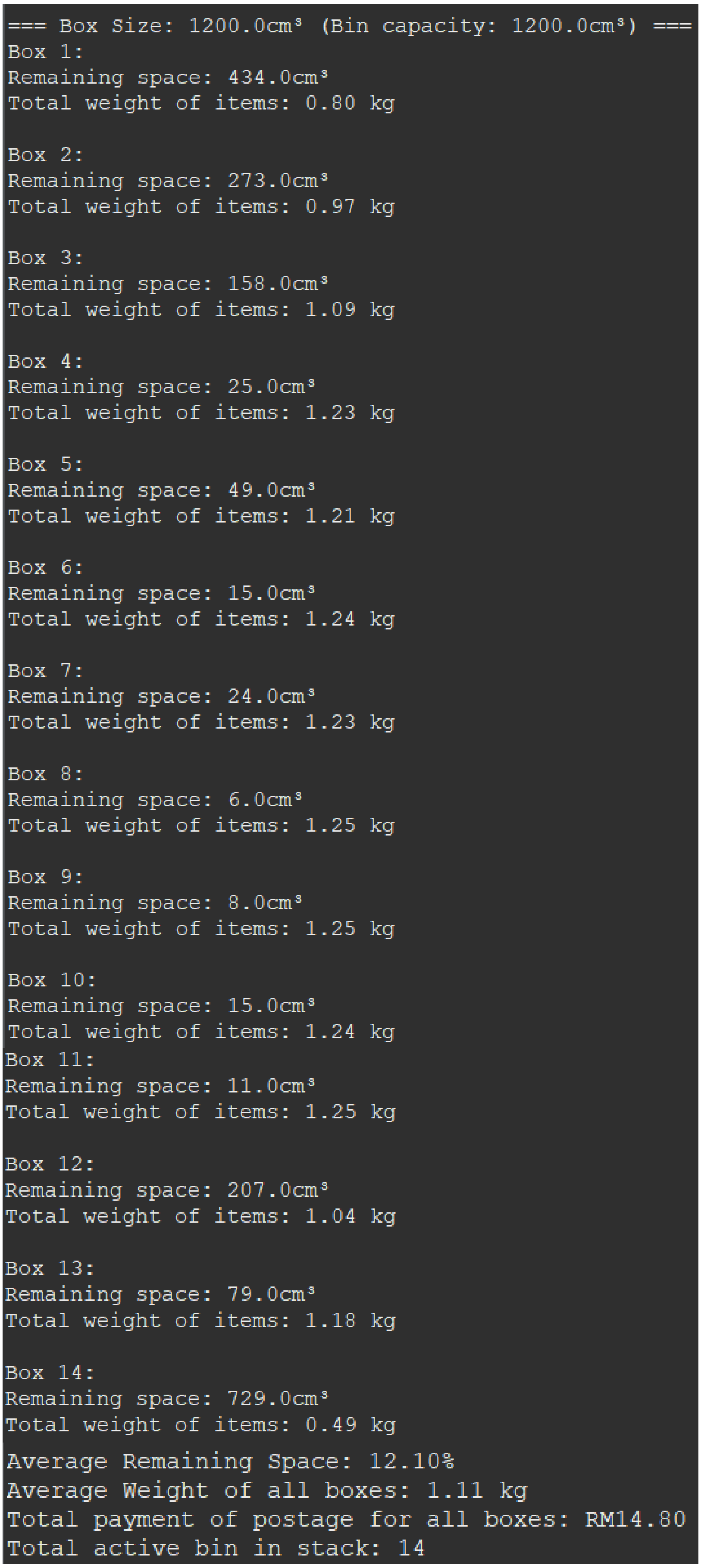


```
=== Box Size: 900.0cm³ (Bin capacity: 900.0cm³) ===
Box 1:
Remaining space: 134.0cm³
Total weight of items: 0.80 kg

Box 2:
Remaining space: 23.0cm³
Total weight of items: 0.92 kg

Box 3:
Remaining space: 39.0cm³
Total weight of items: 0.90 kg

Box 4:
Remaining space: 32.0cm³
Total weight of items: 0.91 kg

Box 5:
Remaining space: 41.0cm³
Total weight of items: 0.90 kg

Box 6:
Remaining space: 70.0cm³
Total weight of items: 0.87 kg

Box 7:
Remaining space: 25.0cm³
Total weight of items: 0.92 kg

Box 8:
Remaining space: 41.0cm³
Total weight of items: 0.90 kg

Box 9:
Remaining space: 100.0cm³
Total weight of items: 0.84 kg

Box 10:
Remaining space: 29.0cm³
Total weight of items: 0.91 kg
```



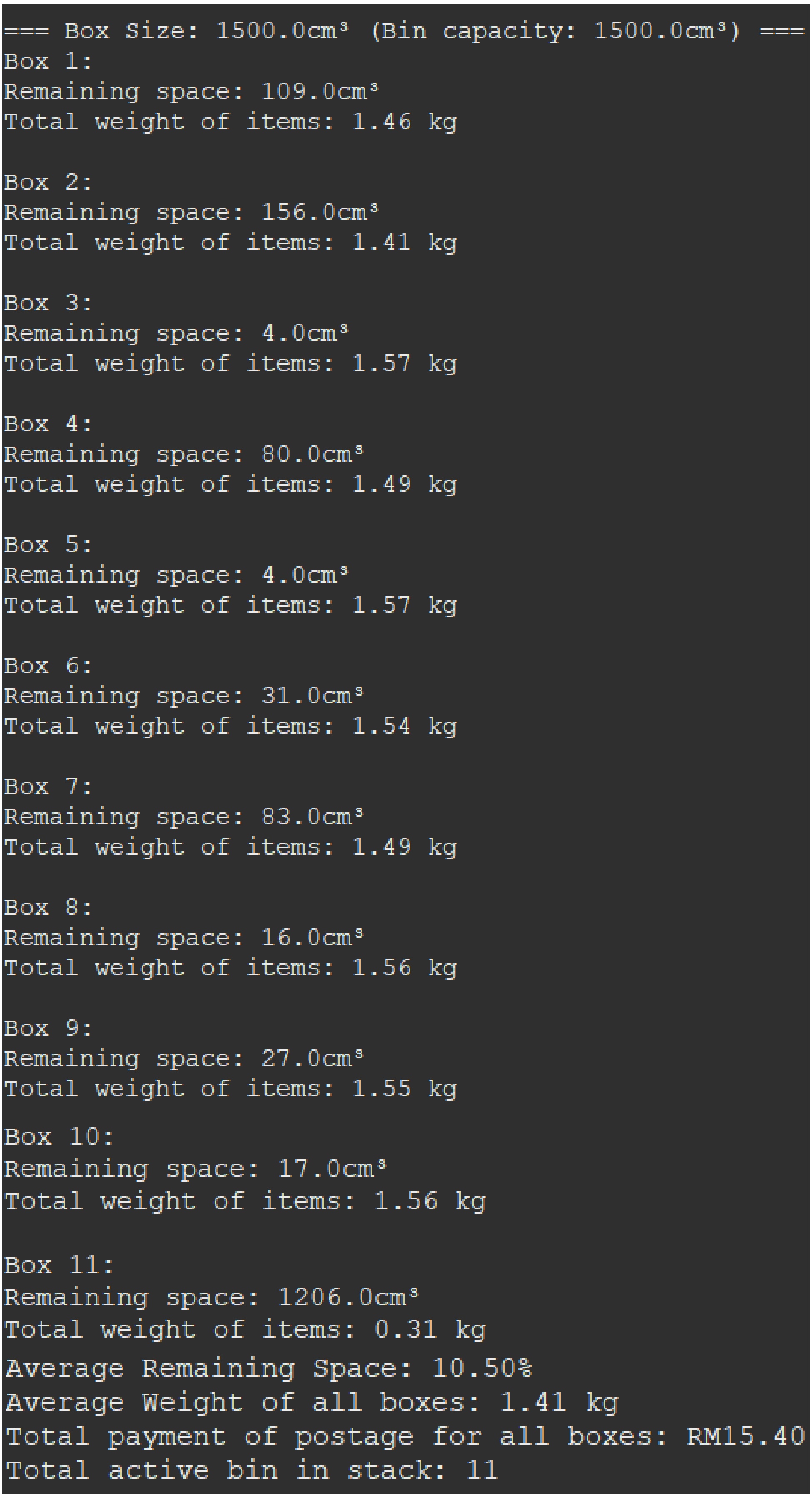
```
Box 11:
Remaining space: 15.0cm³
Total weight of items: 0.93 kg

Box 12:
Remaining space: 218.0cm³
Total weight of items: 0.72 kg

Box 13:
Remaining space: 130.0cm³
Total weight of items: 0.81 kg

Box 14:
Remaining space: 81.0cm³
Total weight of items: 0.86 kg

Box 15:
Remaining space: 106.0cm³
Total weight of items: 0.83 kg

Box 16:
Remaining space: 141.0cm³
Total weight of items: 0.80 kg

Box 17:
Remaining space: 5.0cm³
Total weight of items: 0.94 kg

Box 18:
Remaining space: 203.0cm³
Total weight of items: 0.73 kg

Average Remaining Space: 8.85%
Average Weight of all boxes: 0.86 kg
Total payment of postage for all boxes: RM10.85
Total active bin in stack: 18
```

Test case 4: box capacity is fixed at 1200 cm^3



```
=== Box Size: 1200.0cm³ (Bin capacity: 1200.0cm³) ===
Box 1:
Remaining space: 434.0cm³
Total weight of items: 0.80 kg

Box 2:
Remaining space: 273.0cm³
Total weight of items: 0.97 kg

Box 3:
Remaining space: 158.0cm³
Total weight of items: 1.09 kg

Box 4:
Remaining space: 25.0cm³
Total weight of items: 1.23 kg

Box 5:
Remaining space: 49.0cm³
Total weight of items: 1.21 kg

Box 6:
Remaining space: 15.0cm³
Total weight of items: 1.24 kg

Box 7:
Remaining space: 24.0cm³
Total weight of items: 1.23 kg

Box 8:
Remaining space: 6.0cm³
Total weight of items: 1.25 kg

Box 9:
Remaining space: 8.0cm³
Total weight of items: 1.25 kg

Box 10:
Remaining space: 15.0cm³
Total weight of items: 1.24 kg
Box 11:
Remaining space: 11.0cm³
Total weight of items: 1.25 kg

Box 12:
Remaining space: 207.0cm³
Total weight of items: 1.04 kg

Box 13:
Remaining space: 79.0cm³
Total weight of items: 1.18 kg

Box 14:
Remaining space: 729.0cm³
Total weight of items: 0.49 kg
Average Remaining Space: 12.10%
Average Weight of all boxes: 1.11 kg
Total payment of postage for all boxes: RM14.80
Total active bin in stack: 14
```

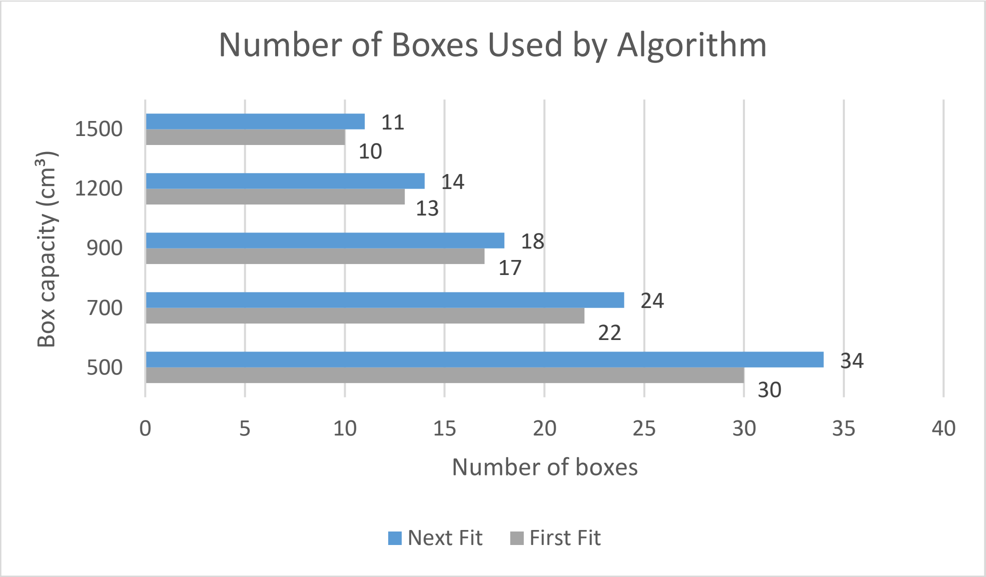
Test case 5: box capacity is fixed at 1500 cm^3



```
=== Box Size: 1500.0cm³ (Bin capacity: 1500.0cm³) ===
Box 1:
Remaining space: 109.0cm³
Total weight of items: 1.46 kg

Box 2:
Remaining space: 156.0cm³
Total weight of items: 1.41 kg

Box 3:
Remaining space: 4.0cm³
Total weight of items: 1.57 kg

Box 4:
Remaining space: 80.0cm³
Total weight of items: 1.49 kg

Box 5:
Remaining space: 4.0cm³
Total weight of items: 1.57 kg

Box 6:
Remaining space: 31.0cm³
Total weight of items: 1.54 kg

Box 7:
Remaining space: 83.0cm³
Total weight of items: 1.49 kg

Box 8:
Remaining space: 16.0cm³
Total weight of items: 1.56 kg

Box 9:
Remaining space: 27.0cm³
Total weight of items: 1.55 kg
Box 10:
Remaining space: 17.0cm³
Total weight of items: 1.56 kg

Box 11:
Remaining space: 1206.0cm³
Total weight of items: 0.31 kg
Average Remaining Space: 10.50%
Average Weight of all boxes: 1.41 kg
Total payment of postage for all boxes: RM15.40
Total active bin in stack: 11
```

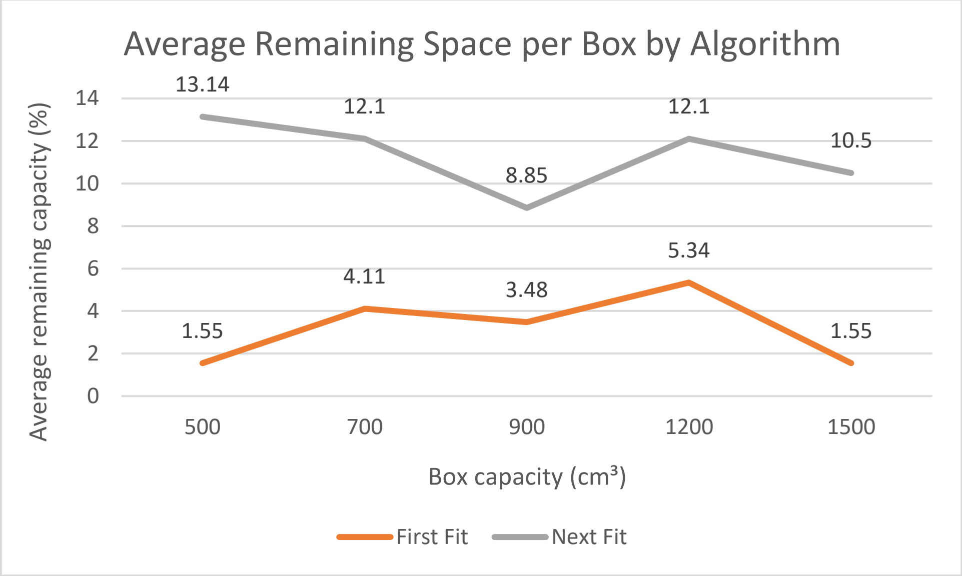
Discussion

1. Number of boxes used by First Fit and Next Fit for each box capacity



As we can observe via the chart, Next Fit consistently uses more boxes than First Fit across all box sizes. This difference can be due to the fundamental operational differences between the two algorithms. When dealing with smaller boxes (500cm^3), Next Fit requires significantly more boxes compared to First Fit. This is because when larger skincare products are packed, Next Fit tends to close the current box and move on to a new one even if there is still room in the current box. The algorithm's stricter packing approach leads to inefficient packing and a high number of boxes. When dealing with then largest box sizes (1500cm^3), although the gap of boxes number between two algorithms narrows, Next Fit still requires more boxes than First Fit. The gap is smaller since both algorithms benefit from larger box capacities, but Next Fit still tends to open new boxes too early when packing larger product.

2. Space Utilization and Waste Comparison

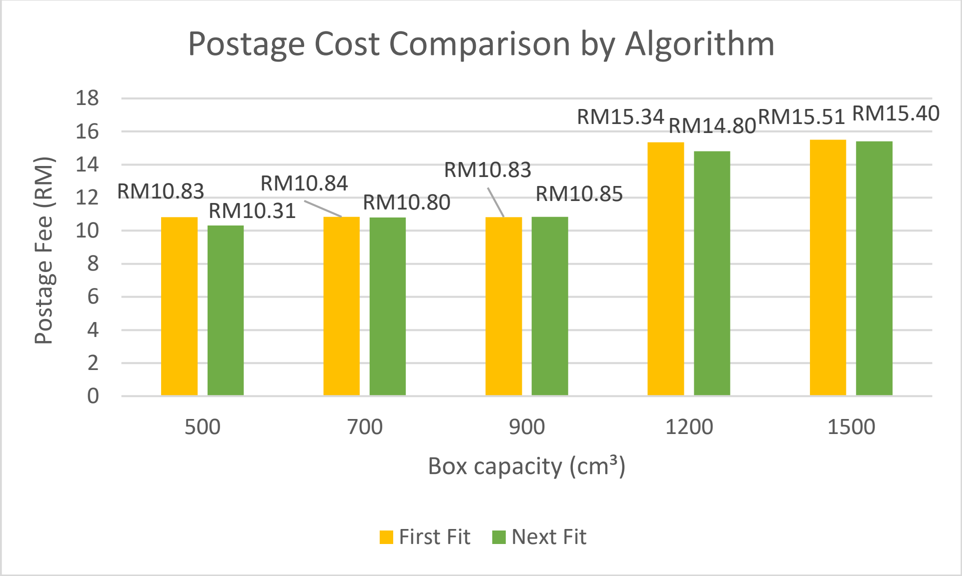


The line graph comparing average remaining space per box highlights the big difference in space utilization between the First Fit and Next Fit algorithms. First Fit consistently achieves lower remaining capacity percentages across all box sizes, indicating more efficient use of available space. Notably, for both the smallest (500 cm³) and largest (1500 cm³) box sizes, First Fit leaves only 1.55% of space unused, demonstrating outstanding packing efficiency. Even at its peak (1200 cm³), the remaining space reaches only 5.34%, showing that First Fit maintains a relatively low and stable level of unused capacity regardless of box size.

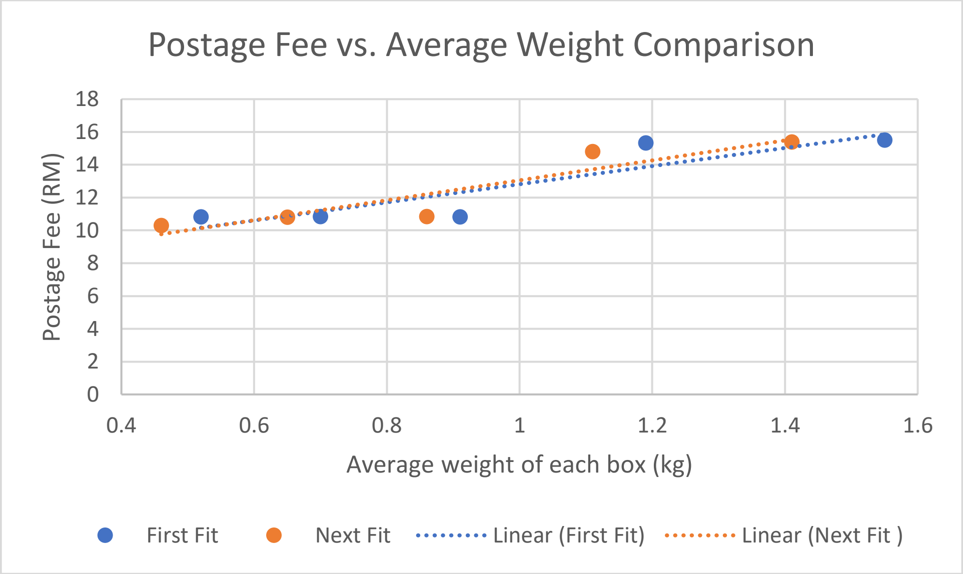
In contrast, Next Fit displays much higher remaining capacity across all box sizes. For the smallest box (500.0 cm³), the remaining space is 13.14%, which is over 8 times greater than the First Fit algorithm’s remaining space for the same size box. This trend continues for larger boxes, with the remaining space averaging 12.10% for both the 700.0 cm³ and 1200.0 cm³ boxes. Next Fit appears to leave significant gaps in the boxes, especially in smaller box sizes. This could be due to the packing of larger skincare products where the algorithm opens a new box when the current box cannot accommodate the item. As a result, Next Fit does not always make the best use of available space, leading to higher waste.

Overall, the graph clearly shows that First Fit not only minimizes box usage but also maximizes space efficiency, which is crucial for reducing packaging waste. Next Fit, while simpler in approach, is considerably less effective in utilizing box volume, leading to more waste and inefficiency.

3. Postage costs



The bar chart illustrates that First Fit generally results in higher postage fees compared to Next Fit across all box sizes, with the difference being more noticeable in the box sizes of 1200 cm³ and 500 cm³. Although Next Fit consistently has lower postage fees, the differences between the two algorithms are relatively small. This can be attributed to the fundamental operation of Next Fit, where it opens a new box whenever it detects that an item cannot fit into the current box, leading to boxes with lesser weight. As a result, Next Fit incurs a lesser postage fee regardless of the number of boxes used. To further investigate the relationship between postage fees and average weight of all boxes, a scatter plot was created, highlighting the trend that both postage fee and average weight increase together, with Next Fit showing a more gradual increase compared to First Fit.



The scatter plot illustrates the relationship between postage fees and average weight for both First Fit and Next Fit algorithms. First Fit is represented by a dotted line that shows a slightly steeper increase in postage fees, indicating that postage costs rise more rapidly as the weight of the box increases. This is because First Fit packs all the items into a box until it is full or no longer can fit any more, resulting in higher box weight and, consequently, higher postage fees. In contrast, Next Fit has a gentler, dotted line with a more gradual increase in postage fees as the weight of the boxes increases. This is due to Next Fit's approach of opening a new box whenever the current one is full, which generally leads to lighter boxes. Both algorithms show a positive correlation between average weight and postage fees, meaning that as the average weight of the box increases, the postage fee also increases. Therefore, Next Fit is more efficient in terms of cost relative to weight.

4. Flexibility in Handling Items of Different Sizes

First Fit can handle items of a wide of sizes better then Next Fit. This is because it attempts to fit skincare products into the first available box that has enough capacity. Since it fills up a box before moving to the next, it is capable of handling both small and large items without leaving much wasted space, especially when the items are appropriately sized to the box. This makes it more versatile in handling diverse sizes of items, ensuring that each box is filled as much as possible. In comparison, Next Fit is less flexible when dealing with varying product sizes. It opens a new box whenever the current box is full, even if there is still available capacity in the box. This can lead to inefficiency, especially when packing items that do not fill the box entirely, leading to wasteful use of space. This lack of adaptability makes it more difficult for Next Fit to efficiently handle a wide range of item sizes compared to the First Fit algorithm. Overall, First Fit's ability to adjust more easily to different item sizes enhances its flexibility, making it a better choice when packing a mix of small and large items.

Conclusion

The results indicate that the First Fit algorithm consistently outperforms the Next Fit algorithm across several key criteria. First Fit uses fewer boxes, achieves better space utilization and handles a wider range of product sizes more flexibly. Its ability to fill boxes more efficiently reduces packaging waste. Although Next Fit proves to be more cost-effective than First Fit, but the difference is not substantial. From computational perspective, First Fit may result in slightly higher computational overhead compared to Next Fit’s linear time complexity. However, the practical benefits in packing efficiency outweigh the additional computation in scenarios where minimizing the number of boxes and reducing waste space is prioritized. Overall, First Fit proves to be the more efficient, and adaptable packing algorithm, making it the preferred choice for packing skincare products of varying sizes.