PRACTICAL ASSIGNMENT - MARKING REPORT

1. PERSONAL DATA

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Group number :** | | | | |
| No | Name | ID | Programme | Total Marks |
| 1. | EMILY TAN KIT LUM | 2305645 | SE |  |
| 2. | LIM SHI YAO | 2303328 | SE |  |
| 3. | NG KAI XIN | 2206605 | SE |  |
| 4. | NG LEE CHUAN | 2305659 | SE |  |

1. SUBMISSION STATUS

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

1. 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 |
|  |  |  |

1. PRESENTATION OF SOURCE CODES(3%)
2. Indent Style (1.5%) Poor Inconsistent Good
3. Identifier names (1.5%) Poor choice Meaningful Meaningful and good naming convention
4. 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 |  |

1. 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 |  |

**The Proposed Solution (Data Structure and Strategies)**

The bin packing problem is a combinatorial optimization problem that involves efficiently allocating objects of different sizes and volumes into bins with fixed capacity and minimizing the number of bins at the same time. The issue is discussed and summarized below:

Assume we are managing a warehouse which have various of items. Each of the bins *m* in the warehouse has a fixed size/capacity *c,* which is (120 \* 115 \* 120) // (135 \*110 \*100)cm. However, each of the items *n* has a different size *s*. Due to the fixed size of the bin but large numbers of items, the warehouse would like to pack and fit all of these items with minimum usage of bins. Thus, we must decide how to carefully allocate all the items into the bins based on their size without exceeding the capacity and ensure the minimum usage of bins at the same time. The Excel file below summarizes all the items we need to allocate together with their ID, height, weight, length and quantity.

Excel File: [Warehouse's Dataset](https://docs.google.com/spreadsheets/u/0/d/1lMeoHOT6utQwQyk5L6K1boew6XOslwz1/edit)

**Algorithms**

We proposed 2 solutions to solve the bin packing problem, which are the First Fit(FF) Algorithm and the First Fit Decreasing(FFD) Algorithm. FF and FFD Algorithms are 2 optimal algorithms in allocating each of the items into the bins and ensuring it does not exceed the bins’ capacity, and the minimum number of bins is used at the same time.

For First Fit Algorithm:

It’s one of the methods to solve the warehouse bin packing problem by simply placing the item into the first bin that has enough space.

The steps below demonstrate how the First Fit Algorithm works:

1. Initialised the bin is empty at first.
2. Add the items into the bins based on their original listed order in the Excel file, based on their ID one by one.
3. Store the items in the first available bins (Checking the availability from left/beginning to right).
4. If the item fits the bin, store it in the bins; else, move to the next bin
5. If there is no available bin to fit the item, open a new bin.
6. Repeat steps 2 to 5 until all the items are placed into bins.

By applying the First Fit algorithm, the warehouse can easily manage the items due to its simplicity. This is because this algorithm does not need extra sorting before the item allocation. We can just place the items into the bins based on their arrival according to the list. This increases the time efficiency indirectly.

However, there are some limitations on this algorithm itself. The First Fit Algorithm does not always ensure the minimum usage of bins. This means that it might not be the best overall solution in our case, since it might not always fulfill our goals. Furthermore, it might leave some remaining available space in some bins, which the space could be filled more effectively. The other limitation of this algorithm would be that it does not reconsider the placement once it’s placed in a particular bin, which might prevent better allocation afterwards.

The next solution we proposed to solve the warehouse bin packing problem is the First Fit Decreasing Algorithm. This is considered an improved version of the First Fit Algorithm since it does sorting before the allocation.

The steps below demonstrate how the First Fit Decreasing Algorithm works:

1. Initialised the bin is empty at first.
2. Sort the items in the list in descending order (From biggest to smallest)
3. Add the items into the bins based on their original listed order in the Excel file, based on their ID one by one.
4. Store the items in the first available bins (Checking the availability from left/beginning to right).
5. If the item fits the bin, store it in the bins; else, move to the next bin
6. If there is no available bin to fit the item, open a new bin.
7. Repeat steps 2 to 5 until all the items are placed into bins.

By implementing FFD, we reduce the storage space wasted as it places the items into bins from the biggest to the smallest. This is said so because bigger items are always harder to fit. Thus, by using FFD, it can more effectively pack the items into the bins compared to First Fit.

Although this algorithm is more efficient than First Fit, it does bring disadvantages. This is said so because FFD takes time and resources to do sorting at the beginning. However, this is acceptable in the real-world situation as it improved the overall result and saved storage spaces and the number of bins. Besides, the FFD algorithm may still not guarantee a completely optimal solution in all scenarios.

In conclusion, both data structures suit themselves in providing solutions for the warehouse bin packing problem. In terms of simplicity and time efficiency, First Fit algorithms would be a better choice to solve the problems. However, in terms of space efficiency, the First Fit Decreasing algorithm will be a better solution to solve the problem.

**Data Structures**

We proposed 2 data structures to be used in both of the algorithms. In our case, our ultimate goal is to efficiently allocate objects of different sizes and volumes into bins with fixed capacity while minimizing the number of bins at the same time. Thus, we proposed that **Queue and LinkedHashMap** would be suitable for this case and align with our goals.

As we know, Queue applies the First In First Out(FIFO) manner. Thus, Queue will be suitable for handling items in order. And this is aligned with both of the algorithms we proposed due to the following reasons:

* **Fits Real World Operation:** This is said so because in real-world operation, the warehouse will also process their stocks like Queue where the items will be picked or arrive based on their order.
* **Help to process the Item in Fixed Order:** For both FF and FFD algorithms, they process the items based on the order, one by one. The only difference between these 2 algorithm here is that FF uses the arrival order for insertion, but FFD uses the sorting order. And how Queue can help in this case is due to its FIFO manner. Queue processed items based on the sequence of items added into bins. This may reduce confusion and errors that might happen in accessing the items.
* **Sequential Bin Checking of Both Algorithms:** From what we explained above, we know that both of the algorithms check the remaining space of the bins sequentially. Both algorithms go through one by one without jumping to check the first available space of bins to fit the item. And how Queue can help in this case is that again of its FIFO manner. Queue provides us the ability to keep track of bins in the order in which items were added, as Queue always checks the bins starting from the leftmost(Start) to the rightmost(End).

The next data structure we choose is LinkedHashedMap. As we know, LinkedHashMap stores items based on the insertion order. Additionally, LinkedHashMap provides us with the functionality of quick access and searching. And this is aligned with both of the algorithms we proposed due to the following reasons:

* **Same Working Logic:** As we mentioned above, both FF and FFD insert the items based on arrival order and sorting order respectively. Thus, in this case, LinkedHashMap works perfectly due to it can store the items based on the insertion order which is similar to both of our algorithms.
* **Management of Bins and Remaining Capacity:** This is said so because LinkedHashMap operates in the manner of a key and value pair. In this case, the bin number can be key, and the remaining space can be the value, and this allows us to retrieve, update, check and access the items in constant time, which is very efficient in saving time and reuse of bins since we can update the value(remaining space) of bins quickly.
* **Fits Real World Operation:** Thus, based on what are explained, it fits the real world operation since the workers in the warehouse will also go through the bins with the order of bins added. Besides, the key and value pair of LinkedHashMap allow workers for instant access to the bins by using the key would also be the main reason LinkedHashMap is suitable for both algorithms, as it increases the overall efficiency.

**Discussion / Complexity Analysis of the application**

**Time Complexity of First Fit Algorithm**

**Iteration:** The algorithm places the item into the first bin that has available space. The time complexity for the First Fit Algorithm is O(n^2), where n is the number of items to be placed into the bins. This means that the time complexity increases as the items increases.

Thus, the overall time complexity for First Fit will be O(n^2).

**Time Complexity of First Fit Decreasing Algorithm**

The time complexity for the First Fit Decreasing(FFD) Algorithm involves 2 different steps, which are sorting steps followed by iteration steps to place the items into the bins.

**Sorting:** Sort the items from the biggest to the smallest before allocation implement the logarithmic algorithm(O(n log n)).

**Iteration:** The algorithm places the item into the first bin that has available space after sorting. The time complexity for FFD is O(n^2), where n is the number of items to be placed into the bins. This means that the time complexity increases as the items increases.

Thus, the overall time complexity for First Fit Decreasing will be O(n^2).

**Space Complexity of First Fit Algorithm**

The space complexity of the First Fit(FF) Algorithm relies on the way of bin storage is set up, but the space required is directly proportional to the number of items stored in the bins.

Thus, the space complexity of FF is O(n), where n is the number of items to be placed into the bins.

**Space Complexity of First Fit Decreasing Algorithm**  
First Fit Decreasing(FFD) generally need more space than FF since it requires extra space for sorting steps, as it needs some temporary memory. FFD will use more space than FF. However, it just similar to FF, FFD relies on the way of bin storage is set up, but the space required is directly proportional to the number of items stored in the bins.

Thus, the space complexity of FFD is O(n), where n is the number of items to be placed into the bins.

**Speed Efficiency**

The speed efficiency is how fast this algorithm can complete its task.

For FF, it process the items based on the arrival order, and no additional sorting steps are required, it will be faster in execution than FFD.

For FFD, it will be slower due to the additional sorting process. Before insertion into bins, it sort the items based on the decreasing order of size and this takes (O(n log n)) time. Thus, it’s slower.

In short, FF has higher speed efficiency than FFD.

**Space Utilization Efficiency**

The space utilization efficiency is how well the space is being utilized. In our case, it refers to how well we minimize the number of bins used.

For FF, it will have lower space utilization efficiency. This is said so because FF will simply go through bins by bins and place the items in the first available bins. It does not have sorting based on size, thus, it might cause more gaps and space remaining in each bin, and more bins will be used eventually.

For FFD, it has higher space utilization efficiency than FF. This is said so due to FFD do sort the items based on the decreasing order of size. By placing the larger items first, it significantly reduces the chance that big items can’t be fitted into the remaining small space due to small items inside the bins. Thus, it reduces the number of bins used eventually.

In short, FFD has higher space utilization efficiency than FF.

Overall, both the First Fit Algorithm and the First Fit Decreasing Algorithm have the same time complexity and space complexity. However, the First Fit Decreasing Algorithm has slightly better performance than the First Fit Algorithm. In addition, the First Fit Algorithm has higher speed efficiency than the First Fit Decreasing Algorithm. However, the First Fit Algorithm is more efficient in terms of space utilization than the First Fit Algorithm. This is because the First Fit Decreasing Algorithm done the sorting first prior allocation into bins.

In short, First Fit is more efficient when speed and simplicity are the main considerations, while First Fit Decreasing is more efficient when space utilization is the main consideration, especially when the item size varies significantly.

For our case, our optimal goal is to efficiently allocate objects of different sizes and volumes into bins with fixed capacity while minimizing the number of bins at the same time. Thus, the First Fit Decreasing Algorithm would be the better choice for us.

**‌Flowchart**A diagram of a flowchart

AI-generated content may be incorrect.A diagram of a flowchart

AI-generated content may be incorrect.

A diagram of a flowchart

AI-generated content may be incorrect.A diagram of a flowchart

AI-generated content may be incorrect.

**A diagram of a flowchart

AI-generated content may be incorrect.**

**UML Diagram**

**A diagram of a company

AI-generated content may be incorrect.**

**Sample of Input Data and Test Cases**

**Sample of Input Data:** [Warehouse's Dataset](https://docs.google.com/spreadsheets/u/0/d/1lMeoHOT6utQwQyk5L6K1boew6XOslwz1/edit) **(Follow The Link)**

**Test Case 1**

// Using Queue's First Fit algorithms in queue

**int** binFirstFit\_queue = BinPackingAlgorithm.*firstFit*(queuePackage, queuePackage.size(), 130 \* 115 \* 110);

System.***out***.println("the number of bin used through first fit algorithms in queue is :" + binFirstFit\_queue);

System.***out***.println();

// Using Queue's First Fit Decreasing algorithms in queue

**int** binFFD\_queue = BinPackingAlgorithm.*firstFitDec*(queuePackage, queuePackage.size(), 130 \* 115 \*110);

System.***out***.println("The number of bin used through first fit decreasing algorithms in queue is :" + binFFD\_queue);

System.***out***.println();

// Using linkedHashMap's First Fit algorithm in linkedHasHMap

**int** binFirstFit\_map = BinPackingAlgorithmMap.*firstFit*(linkedHashMapPackage, linkedHashMapPackage.size(), 130 \* 115 \*110);

System.***out***.println("The number of bin used through first fit algorithms in map is :" + binFirstFit\_map);

System.***out***.println();

// Using linkedHashMap's First Fit Decreasing algorithm in linkedHasHMap

**int** binFFD\_map = BinPackingAlgorithmMap.*firstFitDec*(linkedHashMapPackage, linkedHashMapPackage.size(), 130 \* 115 \*110);

System.***out***.println("The number of bin used through first fit decreasing algorithms in map is :" + binFFD\_map);

System.***out***.println();

**A screenshot of a computer code

AI-generated content may be incorrect.**

**Sample Output**

**Test Case 1: First Fit Algorithm in Queue**

**A screenshot of a computer

AI-generated content may be incorrect.**

**A white background with many small dots

AI-generated content may be incorrect.**

**Test Case 1: First Fit Decreasing Algorithm in Queue**

**A close up of a number

AI-generated content may be incorrect.**

**A close up of a number

AI-generated content may be incorrect.**

**Incomplete Packages due to the Limitation of Screenshot**

**Bin 40 = Packages :** [4, 69, 69, 69, 69, 69, 69, 69, 39, 39, 39, 39, 39, 39, 1, 1, 1, 1, 1, 28, 28, 28, 28, 28, 28, 28, 28, 28, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 8]

**Bin 41 = Packages :** [10, 10, 10, 23, 23, 8, 8, 8, 8, 8, 8, 8, 8, 8, 12, 12, 12, 12, 12, 12, 12, 12, 3, 3, 3, 3, 3, 3, 3, 75, 75, 75, 75, 75, 91, 91, 91, 91, 91, 91, 48, 48, 48, 48, 48, 16, 16, 16, 16, 16, 16, 16, 16, 16, 32, 32, 32, 32, 32, 32, 32, 32]

**Test Case 1: First Fit Algorithm in Map**

**A close up of a number

AI-generated content may be incorrect.**

**A close up of a grid

AI-generated content may be incorrect.**

**Test Case 1: First Fit Decreasing Algorithm in Map**

**A black and white image of a graph

AI-generated content may be incorrect.**

**A group of small dots

AI-generated content may be incorrect.**

**Incomplete Packages due to the Limitation of Screenshot**

**Bin 40 = Packages :** [4, 69, 69, 69, 69, 69, 69, 69, 39, 39, 39, 39, 39, 39, 1, 1, 1, 1, 1, 28, 28, 28, 28, 28, 28, 28, 28, 28, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 8]

**Bin 41 = Packages :** [10, 10, 10, 23, 23, 8, 8, 8, 8, 8, 8, 8, 8, 8, 12, 12, 12, 12, 12, 12, 12, 12, 3, 3, 3, 3, 3, 3, 3, 75, 75, 75, 75, 75, 91, 91, 91, 91, 91, 91, 48, 48, 48, 48, 48, 16, 16, 16, 16, 16, 16, 16, 16, 16, 32, 32, 32, 32, 32, 32, 32, 32]

**Sample of Input Data and Test Cases**

**Test Case 2**

// Another test case of different bin volume

**int** binFirstFit\_queue = BinPackingAlgorithm.*firstFit*(queuePackage, queuePackage.size(), 135 \* 110 \* 100);

System.***out***.println("the number of bin used through first fit algorithms in queue is :" + binFirstFit\_queue);

System.***out***.println();

// Using Queue's First Fit Decreasing algorithms in queue

**int** binFFD\_queue = BinPackingAlgorithm.*firstFitDec*(queuePackage, queuePackage.size(), 135 \* 110 \* 100);

System.***out***.println("The number of bin used through first fit decreasing algorithms in queue is :" + binFFD\_queue);

System.***out***.println();

// Using linkedHashMap's First Fit algorithm in linkedHasHMap

**int** binFirstFit\_map = BinPackingAlgorithmMap.*firstFit*(linkedHashMapPackage, linkedHashMapPackage.size(), 135 \* 110 \* 100);

System.***out***.println("The number of bin used through first fit algorithms in map is :" + binFirstFit\_map);

System.***out***.println();

// Using linkedHashMap's First Fit Decreasing algorithm in linkedHasHMap

**int** binFFD\_map = BinPackingAlgorithmMap.*firstFitDec*(linkedHashMapPackage, linkedHashMapPackage.size(), 135 \* 110 \* 100);

System.***out***.println("The number of bin used through first fit decreasing algorithms in map is :" + binFFD\_map);

System.***out***.println();

**A screenshot of a computer code

AI-generated content may be incorrect.**

**Sample Output**

**Test Case 2: First Fit Algorithm in Queue**

**A screenshot of a computer

AI-generated content may be incorrect.**

**Test Case 2: First Fit Decreasing Algorithm in Queue**

**A close up of a screen

AI-generated content may be incorrect.**

**Incomplete Packages due to the Limitation of Screenshot**

**Bin 35 = Packages :** [55, 55, 22, 22, 22, 22, 22, 22, 22, 22, 22, 58, 58, 58, 58, 58, 58, 58, 58, 58, 58, 58, 66, 66, 66, 66, 66, 66, 66, 66, 66, 66, 66, 44, 44, 44, 44, 44, 44, 44]

**Bin 36 = Packages :** [44, 44, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 69, 69, 69, 69, 69, 69, 69, 39, 39, 39, 39, 39, 39, 1, 1, 1, 1, 1, 28, 28, 28, 28, 28, 28, 28, 28, 10, 10, 10, 10, 10]

**Bin 37 = Packages :** [10, 10, 10, 10, 10, 10, 10, 10, 23, 23, 8, 8, 8, 8, 8, 8, 8, 8, 8, 12, 12, 12, 12, 12, 12, 3, 3, 3, 3, 3, 3, 3, 75, 75, 75, 75, 75, 75, 91, 91, 91, 91, 91, 91, 48, 48, 48, 48, 48, 48, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 32, 32, 32, 32, 32, 32, 32, 32]

**Test Case 2: First Fit Algorithm in Map**

**A white background with many small black and white dots

AI-generated content may be incorrect.**

**Test Case 2: First Fit Decreasing Algorithm in Queue**

**A line of dots in a triangle

AI-generated content may be incorrect.**

**Incomplete Packages due to the Limitation of Screenshot**

**Bin 35 = Packages :** [55, 55, 22, 22, 22, 22, 22, 22, 22, 22, 22, 58, 58, 58, 58, 58, 58, 58, 58, 58, 58, 58, 66, 66, 66, 66, 66, 66, 66, 66, 66, 66, 66, 44, 44, 44, 44, 44, 44, 44]

**Bin 36 = Packages :** [44, 44, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 69, 69, 69, 69, 69, 69, 69, 39, 39, 39, 39, 39, 39, 1, 1, 1, 1, 1, 28, 28, 28, 28, 28, 28, 28, 28, 10, 10, 10, 10, 10]

**Bin 37 = Packages :** [10, 10, 10, 10, 10, 10, 10, 10, 23, 23, 8, 8, 8, 8, 8, 8, 8, 8, 8, 12, 12, 12, 12, 12, 12, 3, 3, 3, 3, 3, 3, 3, 75, 75, 75, 75, 75, 75, 91, 91, 91, 91, 91, 91, 48, 48, 48, 48, 48, 48, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 32, 32, 32, 32, 32, 32, 32, 32]