PRACTICAL ASSIGNMENT - MARKING REPORT

1. PERSONAL DATA

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Group number : 50** | | | | |
| No | Name | ID | Programme | Total Marks |
| 1. | Koh Khai Jeck | 2304740 | SE |  |
| 2. | Leon Siow Yi Hong | 2204403 | SE |  |
| 3. | Quak Jing | 2205378 | SE |  |
| 4. |  |  |  |  |

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 |  |

Table of Contents

[1.0 Proposed Solution 2](#_Toc196929143)

[1.1 People (Items) 2](#_Toc196929144)

[1.2 Elevator (Container) 2](#_Toc196929145)

[1.3 Bin Packing Algorithms (Strategy) 3](#_Toc196929146)

[2.0 Discussion and Complexity Analysis 4](#_Toc196929147)

[2.1 Efficiency of Algorithms 4](#_Toc196929148)

[2.2 Complexity of Algorithms 5](#_Toc196929149)

[3.0 Flowchart of Bin Packing Algorithms 7](#_Toc196929150)

[3.1 First Fit Algorithm 7](#_Toc196929151)

[3.2 Best Fit Algorithm 8](#_Toc196929152)

[3.3 Next Fit Algorithm 9](#_Toc196929153)

[3.4 Worst Fit Algorithm 10](#_Toc196929154)

[4.0 UML Diagram of Java Program 11](#_Toc196929155)

[5.0 Sample of Input Data and Test Cases 12](#_Toc196929156)

[6.0 Sample Output(s) 13](#_Toc196929157)

[7.0 Print Out of Java Program 14](#_Toc196929158)

# 1.0 Proposed Solution

In our daily life, clearing a queue that is using the elevator is a common headache due to unoptimized packing of people in the elevator. For this bin-packing assignment, the main goal is to minimize the number of elevators needed to compensate a queue of people, where people will be optimally packed into elevator. In this case, the people are the items, and the elevator is the container. The solution to achieve said goal are strategies that implements bin-packing algorithms.

## 1.1 People (Items)

A person class declared in “Person.java” outlines the attribute of a person. The attributes of a person are as follows:

|  |  |
| --- | --- |
| **Attribute** | **Representation** |
| area (m2) | The area occupied when standing in an elevator |
| weight (Kg) | The load added on elevator |

A group of persons is identified as people, where the queue data structure declared in “ElevatorQueue.java” is used to store each individual person. The reason for picking queue as the suitable data is to simulate real world scenario where people queue up to enter the elevators on a first come first serve basis.

## 1.2 Elevator (Container)

An elevator class declared in “ElevatorBin.java” outlines the attribute of an elevator. The attributes of an elevator are as follows:

|  |  |
| --- | --- |
| **Attribute** | **Representation** |
| bin | The people that are currently in the elevator |
| FULL\_LOAD (Kg) | The max load of people that the elevator can handle |
| FULL\_AREA (m2) | The dimension of the elevator’s floor to stand on |
| currentLoad (Kg) | The current load of people in the elevator |
| currentArea (m2) | The current area occupied by people in the elevator |

The elevator class is designed like a stack to capture the nature of first in last out when people are entering and exiting the elevator. A group of elevator is identified as elevators, where the array list data structure declared in “ElevatorArrayList.java” is used to store the elevator used. The reason for picking array list as the suitable data structure is because of the constant time indexing operation, which is commonly used in bin-packing algorithms.

## 1.3 Bin Packing Algorithms (Strategy)

To pack the people into the elevators efficiently, four popular approximation bin packing algorithms will be used. The algorithms implemented are:

|  |  |
| --- | --- |
| **Algorithms** | **Strategy** |
| First Fit (FF) | People enter the first elevator that can handle them. |
| Best Fit (BF) | People enter the elevator with the least area available left. |
| Next Fit (NF) | People only enter the current elevator. |
| Worst Fit (WF) | People enter the elevator with the most area available left. |

These algorithms serve as an experimental societal norm of how people should approach entering elevator enter when queuing. Multiple algorithms are implemented to better showcase the difference in efficiency and complexity.

# 2.0 Discussion and Complexity Analysis

## 2.1 Efficiency of Algorithms

Each of the algorithms are tested with the same queue of people. The test is performed three separate times with different number of people in the queue. The documented results are as follows:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Number of People** | **Algorithms** | **First Fit** | **Best Fit** | **Next Fit** | **Worst Fit** |
| 5000 | **Number of Elevators Used** | 461 | 461 | 472 | 462 |
| 50 000 | 4605 | 4603 | 4721 | 4609 |
| 500 000 | 45977 | 45978 | 47128 | 46043 |

As the number of people in the queue increases, the difference in performance is increasingly obvious. As observed from the results, first fit and best fit performed the best with very little difference in performance. The best performing algorithm is followed by worst fit and then next fit.

From this observation, people should either enter the first elevator that can handle them or enter the elevator with the least area available left to minimize the elevators needed to clear the queue of people. The people can consider entering the elevator with the most area available left if they prefer open spaces as it is on average only 0.002% less inefficient than first fit and best fit. On the contrary, people should avoid only entering the current elevator as it is on average 2.4% less efficient than the other algorithms.

## 2.2 Complexity of Algorithms

By studying the algorithm, the time complexity of each algorithm is identified. The identified time complexity are as follows:

|  |  |  |
| --- | --- | --- |
| **Algorithms** | **Time Complexity** | **Iteration/Loop Analysis** |
| First Fit | O(n2) | Outer Loop: Iterate through each person in people.  Inner Loop (layer 1): Iterate through each elevator in elevators to find the first elevator that can handle the person. |
| Best Fit | O(n2) | Outer Loop: Iterate through each person in people.  Inner Loop 1 (layer 1): Iterate through each elevator in elevators to find all the elevator that can handle the person  Inner Loop 2 (layer 1): Iterate through all the elevator that can handle the person to find the least space remaining |
| Next Fit | O(n) | Outer Loop: Iterate through each person in people. |
| Worst Fit | O(n2) | Outer Loop: Iterate through each person in people.  Inner Loop 1 (layer 1): Iterate through each elevator in elevators to find all the elevator that can handle the person  Inner Loop 2 (layer 1): Iterate through all the elevator that can handle the person to find the most space remaining |

While testing the efficiency of the algorithms, the runtime (ms) of each algorithm was also captured. The documented results are as follows:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Number of People** | **Algorithms** | **First Fit** | **Best Fit** | **Next Fit** | **Worst Fit** |
| 5000 | **Runtime (ms)** | 2 | 2 | 1 | 4 |
| 50 000 | 119 | 457 | 7 | 445 |
| 500 000 | 18005 | 62204 | 15 | 64096 |

As the number of people in the queue increases, the difference in runtime is increasingly obvious. As observed from the results, next fit is the fastest algorithm by a huge margin, followed by first fit. The two of the slowest algorithms are best fit and worst fit with very little difference in runtime.

From the documentation, the complexity of each algorithm can be explained by the number and nesting of loops. Next fit is the fastest as it only iterates through the queue of people once and immediately pack them into the elevator, resulting in linear time. First fit is slower than next fit as its algorithm iterates through the queue of people and has a loop nested to iterate through the elevators to find the first elevator that can handle the current person, resulting in quadratic time.

Best fit and worst fit is the slowest because their algorithm iterates through the queue of people and has a loop nested on layer 1 to iterate through the elevators to find all the elevators that can handle the current person, and another loop nested on layer 1 to find the elevator with the least/most space remaining. This results in quadratic time and longer runtime due to the 2 loops in the first nested layer, in contrast to first fit which only have 1 loop in the first nested layer.

# 3.0 Flowchart of Bin Packing Algorithms

## 3.1 First Fit Algorithm

A diagram of a flowchart

AI-generated content may be incorrect.

## 3.2 Best Fit Algorithm

A diagram of a flowchart

AI-generated content may be incorrect.

## 3.3 Next Fit Algorithm

A diagram of an elevator

AI-generated content may be incorrect.

## 3.4 Worst Fit Algorithm

A diagram of a flowchart

AI-generated content may be incorrect.

# 4.0 UML Diagram of Java Program

# 5.0 Sample of Input Data and Test Cases

## 5.1 Normal Case 1 (Selecting One Algorithm)



A screen shot of a computer

AI-generated content may be incorrect.

A black background with white text

AI-generated content may be incorrect.

## 5.2 Normal Case 2 (Selecting All Algorithms)

  
A black screen with white text

AI-generated content may be incorrect.

Selecting all algorithms does not allow the user to see the details of each elevator bin.

## 5.3 Large Input Case



A screen shot of a computer

AI-generated content may be incorrect.

A black background with white text

AI-generated content may be incorrect.

## 5.4 Invalid Input Case



A black screen with white text

AI-generated content may be incorrect.



# 6.0 Sample Output(s)

## 6.1 Normal Case 1 (Selecting One Algorithm)



A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

## 6.2 Normal Case 2 (Selecting All Algorithms)

A screenshot of a computer program

AI-generated content may be incorrect.

## 6.3 Large Input Case



A black background with white text

AI-generated content may be incorrect.

Users are reminded that the algorithms will take longer time to run when a large number of people is entered.

## 6.4 Invalid Input Case





A black background with white text

AI-generated content may be incorrect.



Users are guided by restrictions and is prompted to re-enter their input until it is valid.

# 7.0 Print Out of Java Program