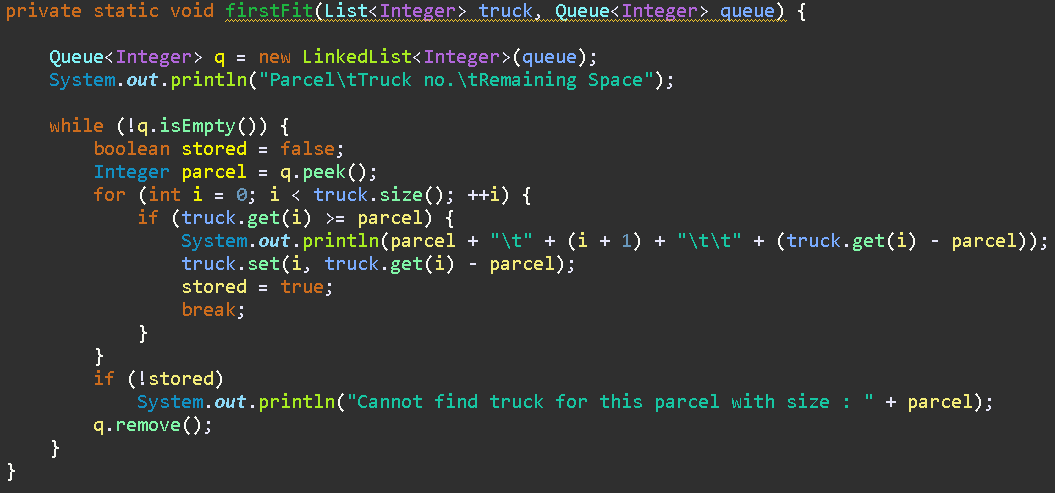
**I. Proposed Solution**

**First-Fit Algorithm**

In the bin-packing problem, the first-fit algorithm packs the parcels into the bins in the order of arrival. It searches for the largest first free bin for the process and allocates the parcel to the coming process. In short, we search from a large number of bins arranged from left to right and pack into the left-most free bin available.

First fit is a quick algorithm as it searches as few as possible, but it has the disadvantage of wastage of memory as the large size of the partition is allocated to the job.



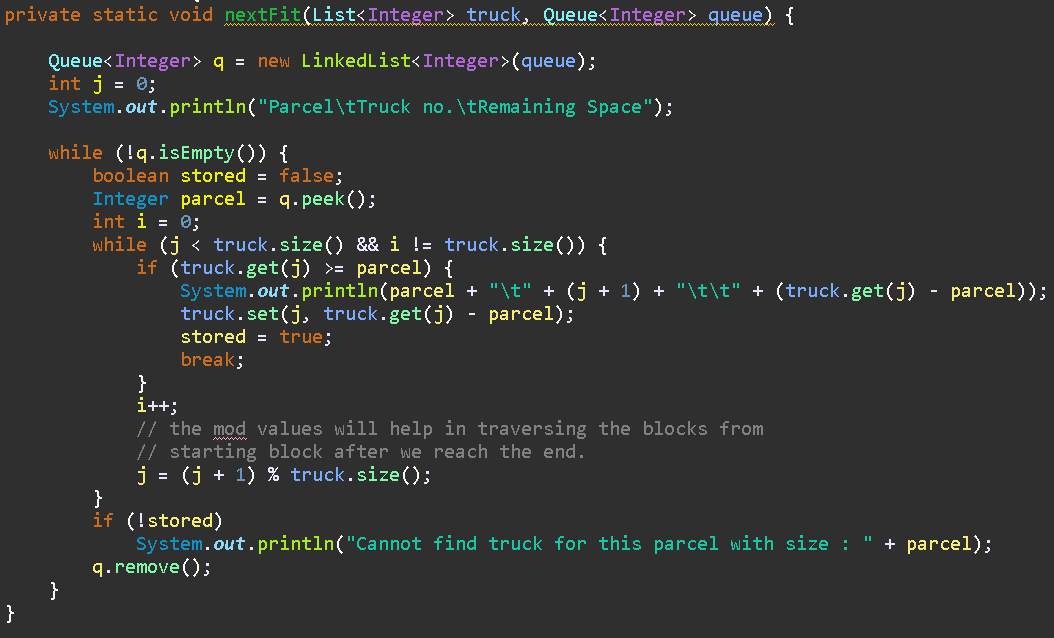
From the figure above, the class *firstFit* will obtain a list of integers from the *truck* and a queue of integers from the *queue*. The interfaces and abstract classes of list and queue are implemented based on the java collection.

A linked list, *q* is defined to store the queue of parcels from the user input. The first parcel size, *parcel* from the *q* is obtained and pack into the ith *truck*. Whenever the parcel is stored in any *truck*, the *stored* will return *true*, otherwise, it will send the message to show the user that none of the *truck* can fit the size of the following *parcel*. The algorithm will run through the following *parcel* while checking for the availability of *truck* at the beginning of the last *truck*

**Next-Fit Algorithm**

In the bin-packing problem, the next-fit algorithm is a modified version of the first-fit. It starts just like the first-fit to pack the parcel to the bin from left to right. When the next parcel is called, it continues searching from the bin allocated by the previous parcel, instead of starting from the beginning.

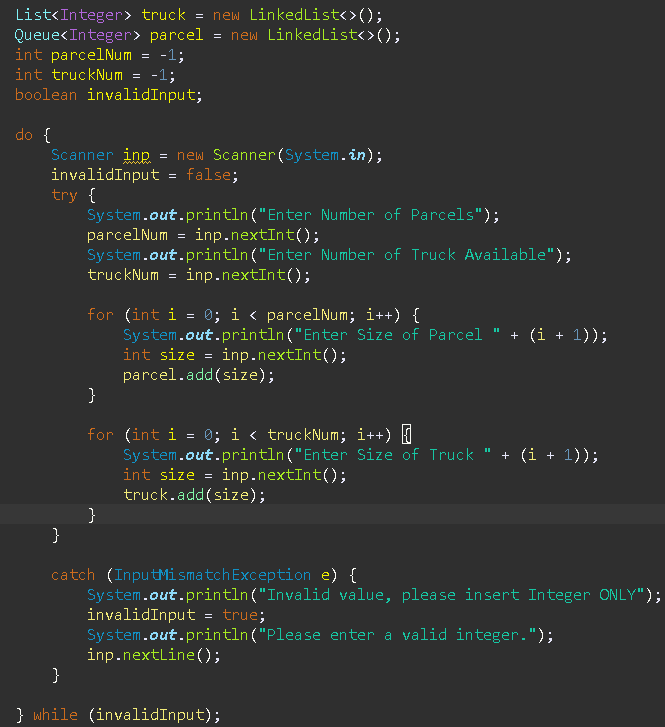
Next-fit has a faster searching than first-fit, but sometimes it will not achieve the optimality compares to first-fit.



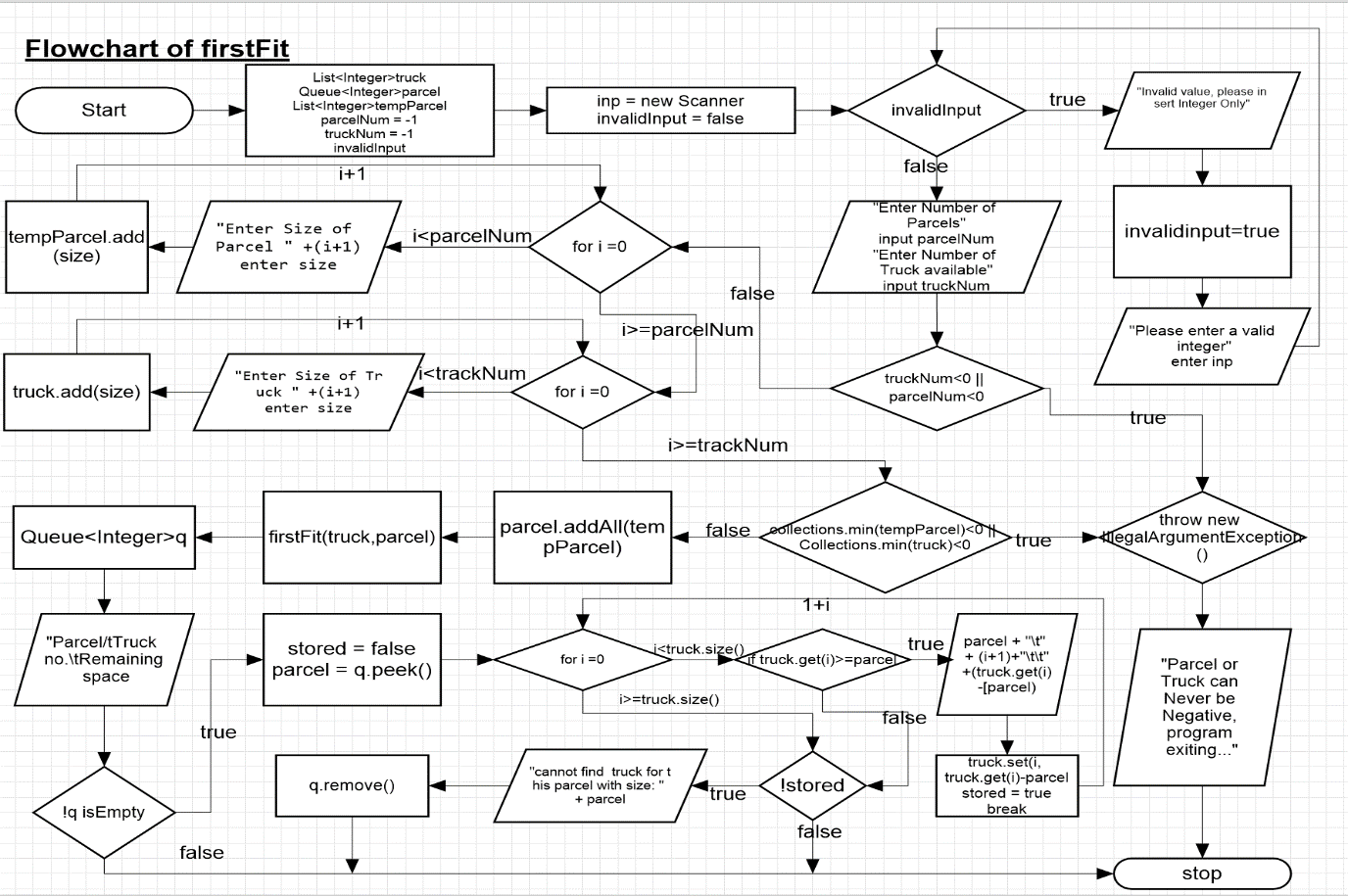
From the figure above, the class *nextFit* will obtain a list of integers from *truck* and a queue of integers from *queue*. The interfaces and abstract classes of list and queue are implemented based on the java collection.

A linked list, *q* is defined to store the queue of parcels from the user input. The first parcel size, *parcel* from the *q* is obtained and pack into the ith *truck*. Whenever the parcel is stored in any *truck*, the *stored* will return *true*, otherwise, it will send the message to show the user that none of the *truck* can fit the size of the following *parcel*. The algorithm will run through the following *parcel*, while checking for the availability of *truck* at the beginning until the last *truck*.

**Main Method**



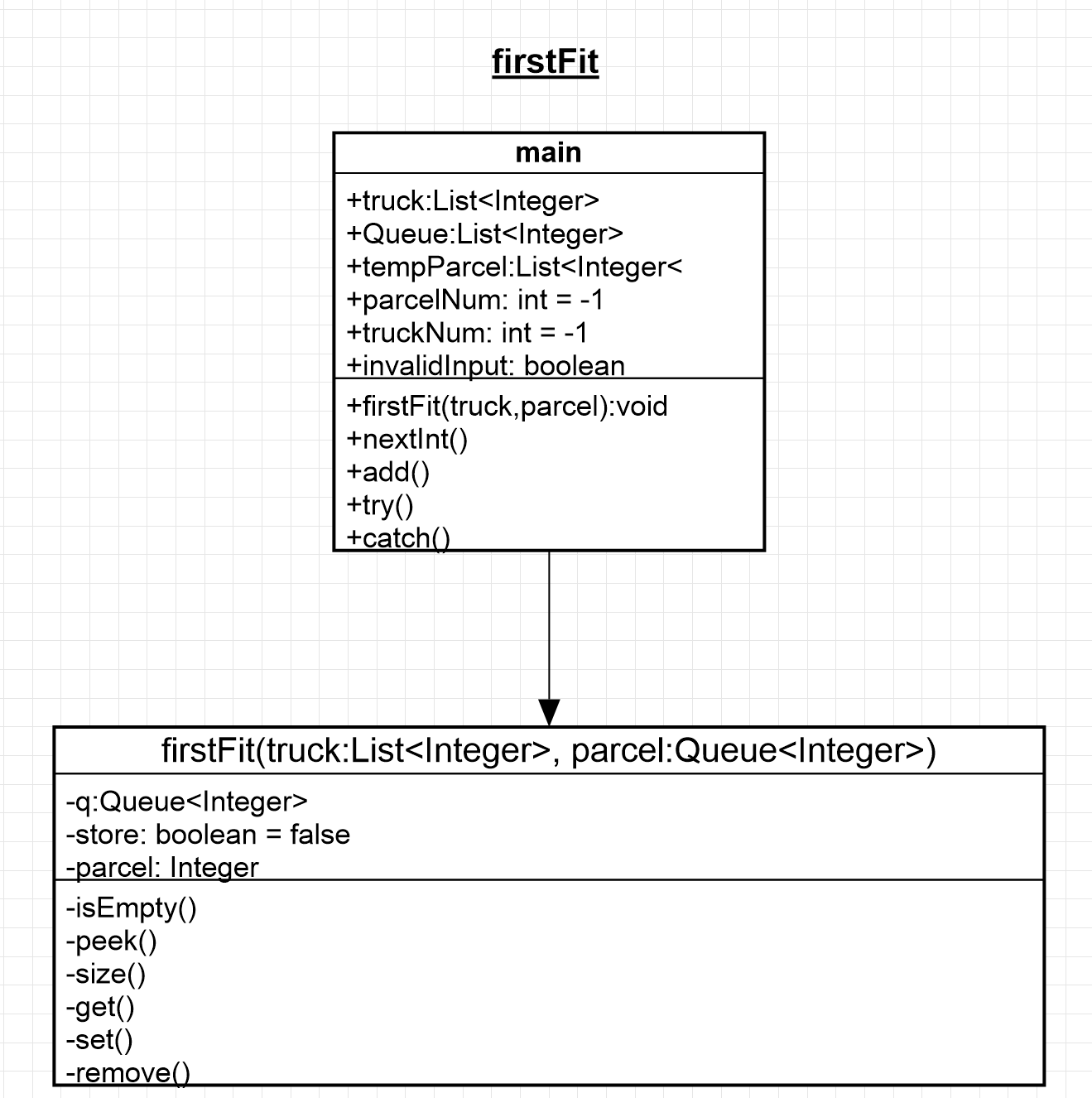
In the main method, a list of integers, *truck*, and a queue of integers, *parcel* is defined as linked list interface. Next, the user is asked to enter the number of parcels, *parcelNum*, and the number of truck(s) available, *truckNum*. For each of the *parcelNum* and *truckNum* are required to set the size from the user input afterward. Alternatively, if the user input is not an integer, it will throw an exception error and ask the user to reenter the value. Hence, the following algorithms will demonstrate the steps of storing each parcel into the bins.

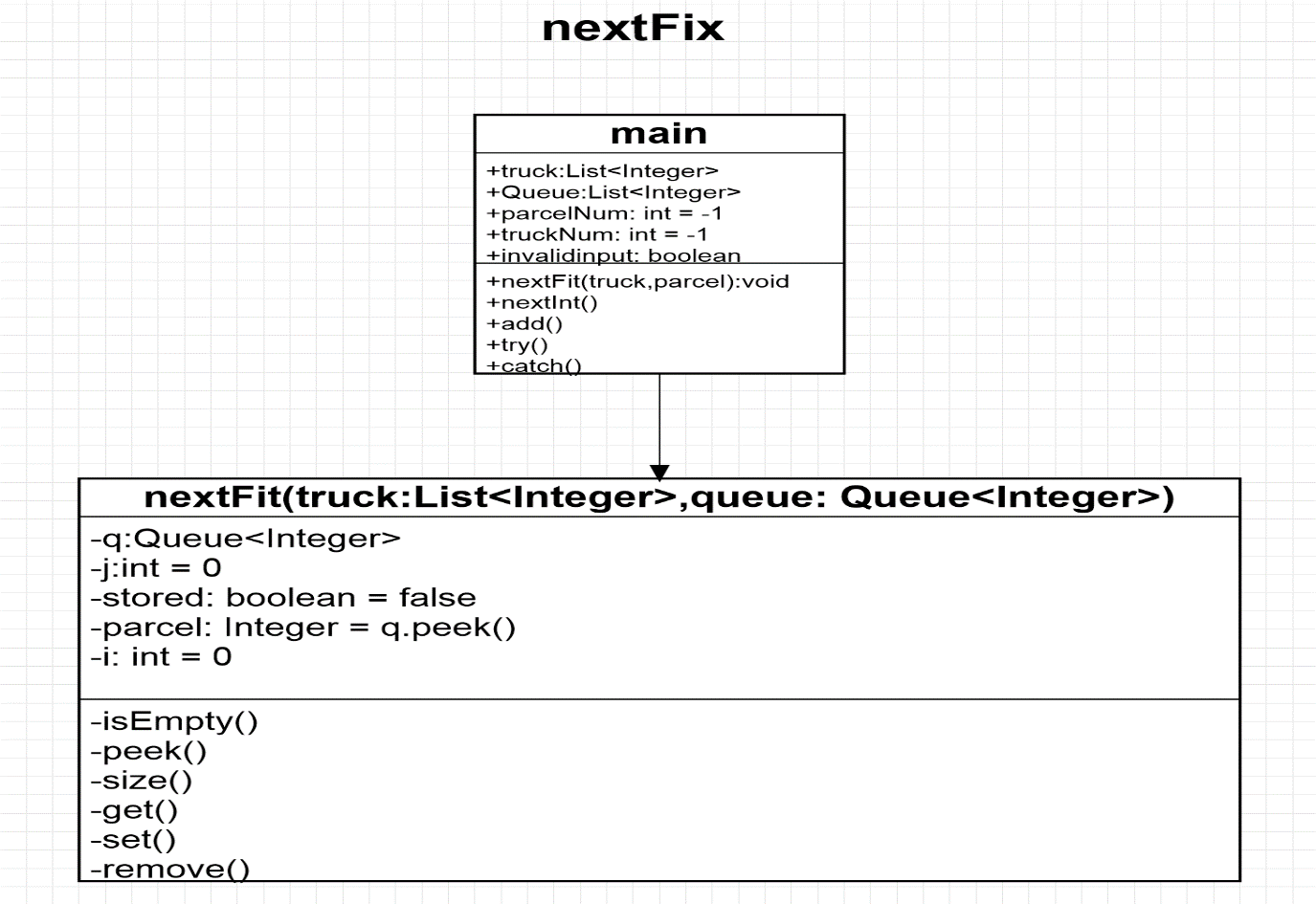
**II. Flowchart for bin-packing algorithms **

**Diagram

Description automatically generated**

**III. UML diagrams for illustrating the design of Java program**

****

****

**IV. Print out of Java program**

**Source Code: First Fit**

**package** java\_assignment;

**import** java.util.LinkedList;

**import** java.util.List;

**import** java.util.Queue;

**import** java.util.Scanner;

**import** java.util.ArrayList;

**import** java.util.Collections;

**import** java.util.InputMismatchException;

**public** **class** firstFit {

**private** **static** **void** firstFit(List<Integer> truck, Queue<Integer> queue) {

Queue<Integer> q = **new** LinkedList<Integer>(queue);

System.***out***.println("Parcel\tTruck no.\tRemaining Space");

**while** (!q.isEmpty()) {

**boolean** stored = **false**;

Integer parcel = q.peek();

**for** (**int** i = 0; i < truck.size(); ++i) {

**if** (truck.get(i) >= parcel) {

System.***out***.println(parcel + "\t" + (i + 1) + "\t\t" + (truck.get(i) - parcel));

truck.set(i, truck.get(i) - parcel);

stored = **true**;

**break**;

}

}

**if** (!stored)

System.***out***.println("Cannot find truck for this parcel with size : " + parcel);

q.remove();

}

}

**public** **static** **void** main(String[] args) {

List<Integer> truck = **new** LinkedList<>();

Queue<Integer> parcel = **new** LinkedList<>();

List<Integer> tempParcel = **new** ArrayList<>();

**int** parcelNum = -1;

**int** truckNum = -1;

**boolean** invalidInput;

**do** {

Scanner inp = **new** Scanner(System.***in***);

invalidInput = **false**;

**try** {

System.***out***.println("Enter Number of Parcels");

parcelNum = inp.nextInt();

System.***out***.println("Enter Number of Truck Available");

truckNum = inp.nextInt();

**if** (truckNum < 0 || parcelNum < 0) {

**throw** **new** IllegalArgumentException();

}

**for** (**int** i = 0; i < parcelNum; i++) {

System.***out***.println("Enter Size of Parcel " + (i + 1));

**int** size = inp.nextInt();

tempParcel.add(size);

}

**for** (**int** i = 0; i < truckNum; i++) {

System.***out***.println("Enter Size of Truck " + (i + 1));

**int** size = inp.nextInt();

truck.add(size);

}

**if** (Collections.*min*(tempParcel) < 0 || Collections.*min*(truck) < 0) {

**throw** **new** IllegalArgumentException();

} **else** {

parcel.addAll(tempParcel);

}

}

**catch** (InputMismatchException e) {

System.***out***.println("Invalid value, please insert Integer ONLY");

invalidInput = **true**;

System.***out***.println("Please enter a valid integer.");

inp.nextLine();

} **catch** (IllegalArgumentException e) {

System.***out***.println("Parcel or Truck can Never be Negative, program exiting...");

}

} **while** (invalidInput);

*firstFit*(truck, parcel);

}

}**Source Code: Next Fit**

**package** java\_assignment;

**import** java.util.\*;

**public** **class** nextFit {

**private** **static** **void** nextFit(List<Integer> truck, Queue<Integer> queue) {

Queue<Integer> q = **new** LinkedList<Integer>(queue);

**int** j = 0;

System.***out***.println("Parcel\tTruck no.\tRemaining Space");

**while** (!q.isEmpty()) {

**boolean** stored = **false**;

Integer parcel = q.peek();

**int** i = 0;

**while** (j < truck.size() && i != truck.size()) {

**if** (truck.get(j) >= parcel) {

System.***out***.println(parcel + "\t" + (j + 1) + "\t\t" + (truck.get(j) - parcel));

truck.set(j, truck.get(j) - parcel);

stored = **true**;

**break**;

}

i++;

// the mod values will help in traversing the blocks from

// starting block after we reach the end.

j = (j + 1) % truck.size();

}

**if** (!stored)

System.***out***.println("Cannot find truck for this parcel with size : " + parcel);

q.remove();

}

}

**public** **static** **void** main(String[] args) {

List<Integer> truck = **new** LinkedList<>();

Queue<Integer> parcel = **new** LinkedList<>();

**int** parcelNum = -1;

**int** truckNum = -1;

**boolean** invalidInput;

**do** {

Scanner inp = **new** Scanner(System.***in***);

invalidInput = **false**;

**try** {

System.***out***.println("Enter Number of Parcels");

parcelNum = inp.nextInt();

System.***out***.println("Enter Number of Truck Available");

truckNum = inp.nextInt();

**for** (**int** i = 0; i < parcelNum; i++) {

System.***out***.println("Enter Size of Parcel " + (i + 1));

**int** size = inp.nextInt();

parcel.add(size);

}

**for** (**int** i = 0; i < truckNum; i++) {

System.***out***.println("Enter Size of Truck " + (i + 1));

**int** size = inp.nextInt();

truck.add(size);

}

}

**catch** (InputMismatchException e) {

System.***out***.println("Invalid value, please insert Integer ONLY");

invalidInput = **true**;

System.***out***.println("Please enter a valid integer.");

inp.nextLine();

}

} **while** (invalidInput);

*nextFit*(truck, parcel);

}

}

**V. sample of input data and test cases**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | *parcelNum* | Size of parcel | *truckNum* | Size of truck |
| **Case 1** | 3 | 2, 3, 1 | 3 | 3, 3, 3 |
| **Case 2** | 10  (or more) | 1, 1, 1, 1, 1, 1, 1, 1, 1, 2 | 10  (or more) | 1, 1, 1, 1, 1, 1, 1, 1, 1, 2 |
| **Case 3** | 10  (or more) | 1, 1, 1, 1, 1, 1, 1, 1, 1, 1 | 5  (or more) | 2, 2, 2, 2, 2 |
| **Case 4** | 1 | 2 | 10  (or more) | 1, 1, 1, 1, 1, 1, 1, 1, 1, 1 |
| **Case 5** | 5 | 1, 3, 3, 1, 1 | 5 | 3, 3, 3, 3, 3 |
| **Case 6** | 5 | 2, 3, 3, 1, 5 | 5 | 3, 3, 3, 3, 3 |
| **Case 7** | 5 | 1, 3, 1, 3, 1 | 5 | 3, 3, 3, 3, 3 |
| **Case 8** | Test exception error | | | |

**VI. Sample output(s) of your program**

**Case 1**

|  |  |
| --- | --- |
| First-fit Algorithm | Next-fit Algorithm |
|  |  |

**Case 2**

|  |  |
| --- | --- |
| First-fit Algorithm | Next-fit Algorithm |
|  |  |

**Case 3**

|  |  |
| --- | --- |
| First-fit Algorithm | Next-fit Algorithm |
|  |  |

**Case 4**

|  |  |
| --- | --- |
| First-fit Algorithm | Next-fit Algorithm |
|  |  |

**Case 5**

|  |  |
| --- | --- |
| First-fit Algorithm | Next-fit Algorithm |
|  |  |

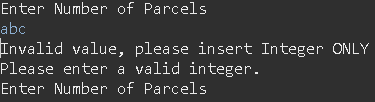
**Case 6**

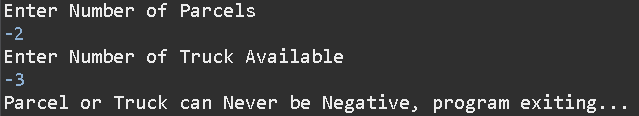
|  |  |
| --- | --- |
| First-fit Algorithm | Next-fit Algorithm |
|  |  |

**Case 7**

|  |  |
| --- | --- |
| First-fit Algorithm | Next-fit Algorithm |
|  |  |

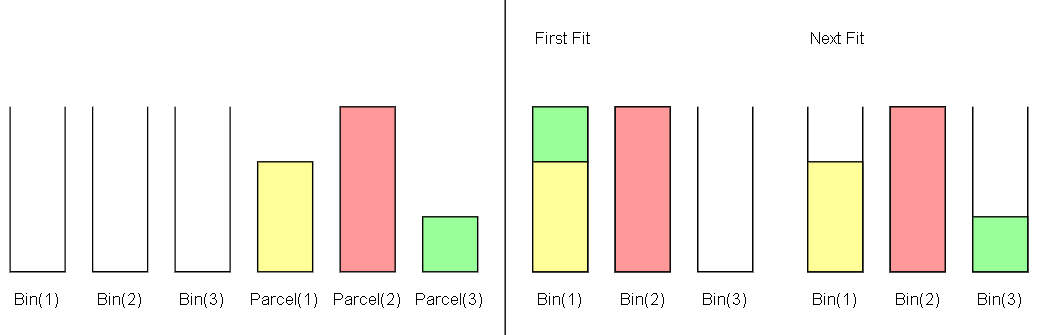
**Case 8**





**VII. Discussion/complexity analysis of the application**

**Worst Case 1**



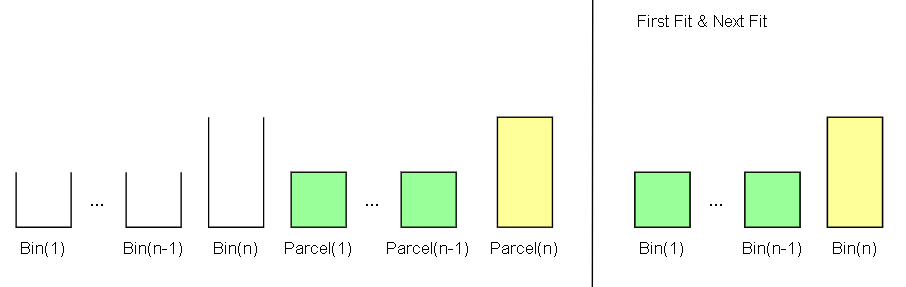
Let

*truckNum* = 3 , *truck.get(i)*  = 3, 3, 3

*parcelNum = 3*, *parcel* = 2, 3, 1

|  |  |  |
| --- | --- | --- |
|  | First-fit Algorithm | Next-fit Algorithm |
| Efficiency | **Optimal** with the least number of *truckNum* used. | **Nonoptimal** as the last *parcel* can be placed in first *truck*. |
| Order of Complexity | O(3) | O(3) |

**Worst Case 2**



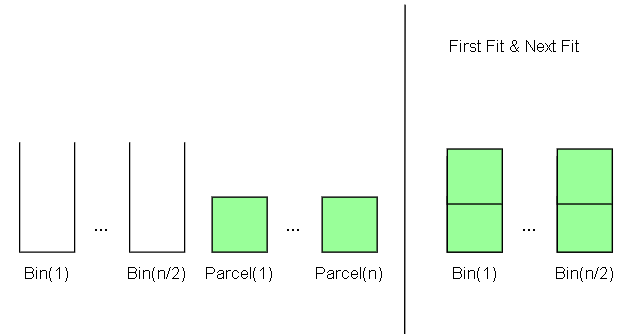
Let

*truckNum* = n , *truck.get(i)*  = 1, … , 1, 2

*parcelNum =* n , *parcel* = 1, … , 1, 2

|  |  |  |
| --- | --- | --- |
|  | First-fit Algorithm | Next-fit Algorithm |
| Efficiency | Not memory effiecient with longer runtime, as the search start all over again. | Memory efficient and short runtime, as it continues searching from previous *truck*. |
| Order of Complexity | O(n2) | O(n) |

**Worst Case 3**



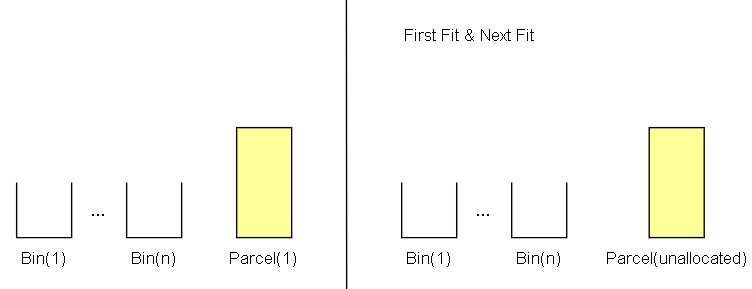
Let

*truckNum* = n , *truck.get(i)*  = 2, … , 2

*parcelNum =* n/2 , *parcel* = 1, … , 1

|  |  |  |
| --- | --- | --- |
|  | First-fit Algorithm | Next-fit Algorithm |
| Efficiency | Not memory effiecient with longer runtime, as the search start all over again. | Memory efficient and short runtime, as it continues searching from previous *truck*. |
| Order of Complexity | O(n2/2) = O(n2) | O(n/2) = O(n) |

**Worst Case 4**



Let

*truckNum* = n , *truck.get(i)*  = 1, … , 1

*parcelNum =* 1 , *parcel* = 2 or *parcelNum =* n , *parcel* = 2, … , 2

|  |  |  |
| --- | --- | --- |
|  | First-fit Algorithm | Next-fit Algorithm |
| Efficiency | Optimal search. | Optimal search. |
| Order of Complexity | O(n),  O(n2) for the additional unallocated parcels. | O(n),  O(n2) for the additional unallocated parcels. |

**Case Study: Truck Loading Problem**

Logistics plays an important role for many industries, and one of the challenges they faced is truck loading problem. A problem where how many parcels to be loaded in the truck so that space is optimized and utilized fully, while minimizing the cost. The more parcels to be placed in the truck, the higher the profits as more parcels are able to be delivered. According to (Reyes, L.C.,2012), one dimensional bin packing problem is a classic combinatorial optimization, where the goal is to determine the smallest amount of bin in which able to pack most items. In a case where the number of bins is unavailable, the problem itself is np-hard, and in relative to the number of bins is known, the problem becomes np-complete where it is solvable within the polynomial time (Lewis, R., 2009).

Bin packing problem can be categorized into Online Heuristic and Offline Heuristic. Online Heuristic is attempted in this assignment where algorithm such as First Fit, Next Fit, Best Fit, Worst Fit etc is attempted. Meanwhile offline heuristic targets the parcels by sorting based on size so that an approximate fitting can be obtained. In real life situation, truck loading problem become complex as more contains are considered, such as weight of parcel, supported weights from truck, height, width, length and etc. The problem complexity increased exponentially and meta heuristic method such as genetic algorithm are proposed (Kanna, Udaiyakumar, Kumar and Lingaraj, 2018).

References

Reyes, L.C., Santillán, C.G., Quiroz, M., Alvim, A., Melin, P., Vanoye, J.R. and Najera, V.L., 2012. Heuristic Algorithms. In: Logistics Management and Optimization through Hybrid Artificial Intelligence Systems. [online] IGI Global.pp.238–267. <https://doi.org/10.4018/978-1-4666-0297-7.ch009>.

Lewis, R., 2009. A general-purpose hill-climbing method for order independent minimum grouping problems: A case study in graph colouring and bin packing. Computers & Operations Research, [online] 36(7), pp.2295–2310. <https://doi.org/10.1016/j.cor.2008.09.004>.

Kanna, S., Udaiyakumar, K., Kumar, S. and Lingaraj, N., 2018. 3D heterogeneous bin packing framework for multi-constrained problems using hybrid genetic approach. *IOP Conference Series: Materials Science and Engineering*, 402, p.012203.