**CSC 2023 Algorithm Design and Analysis- Assignment 2: Truck Loading Problem**

Overview

This is an algorithm to solve the Truck Loading Problem, a more complex version of the Bin Loading problem. It is implemented in the java programming language, using 3 classes, Box, Truck and Search. The Box class holds the height and width of the box, and the Truck class holds a truck’s height, width and capacity, as well as a running total of how many boxes are on each truck. Each method has modified ‘toString’ methods in order to neatly print out this information to the console.

In the Search class, algorithms are provided to solve the Truck Loading Problem using both next fit and first fit. The next fit algorithm checks if a box will fit in a truck and creates a new truck if not, while first fit keeps an array of all the trucks and checks each of them to see if there is space for a given box. There is also a method to generate an array of a specified number of boxes, which uses Random.nextInt to populate an array of boxes with random heights and widths.

The following pseudocode documents my implementation of these two algorithms. There is very little difference between this pseudocode and the actual source code, however certain details are implied, such as initialising the tw and th running totals to the height and width of the supplied truck. Also the first fit algorithm doesn’t take an array of trucks as stated below, rather it is supplied a single truck and the algorithm itself makes an array of said truck.

**Pseudocode**

**Algorithm** NFTLP

**Inputs** B: Array of Boxes; t: truck

**Variables** I,itemsOnTruck,tw,th: int

**Returns** truckCount:int

**Begin**

**for** i=0 **to** i= B.length **do**

Box b:= B[i]

**if** b.width<=tw **&** b.height <= th**&** itemsontruck< t.capacity **then**

itemsOnTruck++

tw= tw – b.width

th= th – b.height

**continue**

**else**

truckCount++

tw= t.width

th=t.height

itemsOnTruck=0

i--

**fi**

**od**

**End­­­­­**

**Algorithm** FFTLP

**Inputs** B: Array of Boxes; T: Array of Trucks

**Variables** i,n: int

**Returns** truckCount: int

**Begin**

**for** i=0 **to** i=B.length **do**

Box b := B[i]

**for** n=0 **to** n=T.length **do**

Truck t := T[i]

**if** b.width>=t.width **&** b.height>=t.height **&**

t.boxesOnTruck < t.capacity **then**

t.boxesOnTruck++

t.height = t.height- b.height

t.width = t.width-b.width

**break**

**fi**

**od**

**if** t.boxesOnTruck != 0 **do**

truckCount++

**od**

**od**

**End**

**Results**

NFTLP

|  |  |  |
| --- | --- | --- |
| Number of Boxes | Number of Trucks | Time taken (Nanoseconds) |
| 50 | 3 | 6368 |
| 100 | 6 | 7230 |
| 500 | 29 | 32693 |
| 1000 | 56 | 39740 |
| 5000 | 276 | 55418 |
| 10000 | 558 | 96649 |

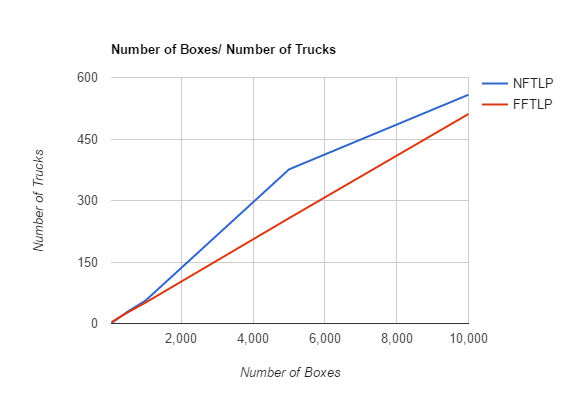
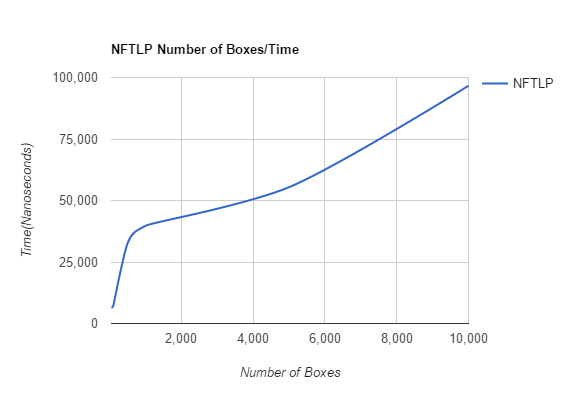
FFTLP

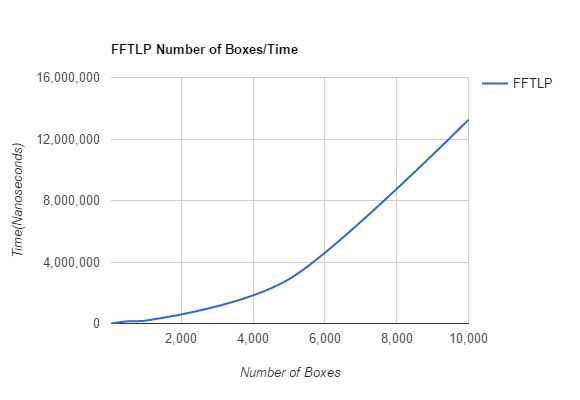
|  |  |  |
| --- | --- | --- |
| Number of Boxes | Number of Trucks | Time taken (Nanoseconds) |
| 50 | 3 | 10741 |
| 100 | 6 | 21318 |
| 500 | 27 | 142497 |
| 1000 | 51 | 187010 |
| 5000 | 257 | 2887096 |
| 10000 | 511 | 13252507 |

Notes on test taking

* Each test was performed using a truck measuring 5000x5000 and with a capacity of 50 boxes
* Each box had a randomly generated height and width between 1 and 500
* Each test was run 1000 times and an average was obtained
* Generation of the boxes is not included in the time taken- the array of boxes was generated prior to the tests taking place and kept the same to ensure reliable results

**Results**





**Evaluation**

In performing the tests, I found that FFTLP performed better in terms of least amount of trucks used, whereas NFTLP used more trucks but performed faster. However, the two algorithms differ greatly in terms of efficiency, for example when sorting 10000 boxes NFTLP used 9% more boxes, yet completed nearly 100% quicker. When sorting smaller numbers of boxes, the two algorithms use the same number of trucks, up until around 1000 boxes where NFTLP starts to use more. However after this both algorithms have a fairly constant gradient, meaning that the amount of trucks used increases proportionally to the number of boxes. At the end of the Boxes/Trucks graph the algorithms have a difference of about 50 trucks, which isn’t that many in relation to the trucks already used, but may represent a company going over budget on a particular job. Therefore, if a large firm is sorting a large number of boxes then they should use NFTLP since the expenditure of a few more trucks is more than made up for in time taken, meaning they can sort more boxes in the same amount of time. However, if a small business is unable to afford the costs of purchasing extra trucks, then it may be advantageous for them to use FFTLP and try to maximise the amount of boxes placed in each truck. The gradient of the NFTLP Boxes/Time graph is a lot shallower than its FFTLP equivalent, which shows that NFTLP is much more efficient in its time taken. Furthermore, by the time FFTLP has sorted 500 boxes, NFTLP has already sorted 10000, showing just how much faster it is.

If I were to do these tests again, I would focus on optimising the FFTLP algorithm and making it more efficient.