PROJECT - A5

DESIGN AND ANALYSIS OF ALGORITHMS

TINOTENDA MACHIDA

TINOTENDA MACHIDA A5/ PROJECT

TABLE OF CONTENTS

INTRODUCTION

The Bin-Packing problem is a problem whereby we are given several objects, each with its own size. We are also given fixed size containers or so-called bins. The bins have fixed capacity and we would like to fit in as many of the objects in the bins while using the smallest number of bins possible. We would like to minimize the number of bins we use.

This is what's called the Bin-Packing problem. To put this in mathematical terms, we are given a set S, of fixed size objects, each with each own fixed weight S_1, S_2, S_3....S_n. We are also given an infinitely large number of bins which all have the same fixed size capacity C.

Our goal is to make sure that we determine the number of bins that it takes to fit all the objects in the bins and we also find the best way of doing so using as few bins as possible. This problem is shown to be a NP- hard problem.

There are several heuristics we could use to solve the problem, but different methods give different answers. An example would be that we choose the number of objects which fit best (also know as best fit), or as we shall focus on in this project, we shall be particularly looking at "first fit".

This means is that we would be putting our objects in the very first bin we find them to fit them. If we don't have space or if we don't have any suitable bins at hand, we go ahead and get a new one. This will be our approach through and through till we run out of all the figures we want to put into bins.

HARDWARE/ SOFTWARE ENVIRONMENT REQUIREMENTS

Through the development of the project, it was top priority to ensure the most minimum software and hardware requirements to run the project. The algorithm for this project was developed in JAVA and runs through a command line interface to get it running. It should run on most modern-day computers easily, if the input values for the algorithm are not too large. Nonetheless I will just go through some of the requirements needed into to run the actual project

HARDWARE REQUIREMENTS

For you to be able to run and install JAVA you are going to need at least the following:

RAM: 128 MB; 64 MB for Windows XP (32-bit)

Disk space: 124 MB

Browsers: Internet Explorer 7.0 and above, Firefox 3.6 and above

SOFTWARE REQUIREMENTS

Since the project is developed in JAVA, the host machine will need to be JAVA compliant in terms of software the most compatible operating systems are listed below

If you are running any of the following operating systems:

- Windows 10 (8u51 and above)
- Windows 8.x (Desktop)

- Windows 7 SP1
- Windows Vista SP2
- Windows Server 2008 R2 SP1 (64-bit)
- Windows Server 2012 and 2012 R2 (64-bit)
- Intel-based Mac running Mac OS X 10.8.3+, 10.9+
  - Oracle Linux 5.5+1
  - Oracle Linux 6.x (32-bit), 6.x (64-bit)2
  - Oracle Linux 7.x (64-bit)2 (8u20 and above)
  - Red Hat Enterprise Linux 5.5+1, 6.x (32-bit), 6.x (64-bit)2
- Red Hat Enterprise Linux 7.x (64-bit)2 (8u20 and above)
- Suse Linux Enterprise Server 10 SP2+, 11.x
- Suse Linux Enterprise Server 12.x (64-bit)2 (8u31 and above)
- Ubuntu Linux 12.04 LTS, 13.x
- Ubuntu Linux 14.x (8u25 and above)
- Ubuntu Linux 15.04 (8u45 and above)
- Ubuntu Linux 15.10 (8u65 and above)

Then you should find yourself more than equipped to run the project. If you don't have JAVA on your machine already, here's a heads up. You are also going to need a you are also going to need JAVA SDK or JRE 1.6 or higher if you don't have it installed already.

HARDWARE/SOFTWARE USED TO DEVELOP MY PROJECT

HARDWARE

The project was developed on a Lenovo Y530 laptop computer.

This should not have any significant changes to the performance of the program

The computer runs a X64 based Intel® Core ™ i-8750H CPU @ 2.20 2.21 GHz.

The computer also has 16GB memory

SOFTWARE

The computer operating system that was ran during the development process was Windows- 10 Home x64.

Visual Studio Code was used as the IDE to implement the programming of the algorithm.

And both JRE and SDK 8 were installed in order to run java

THE DESIGN OF THE ALGORITHM

The design of the algorithm will be discussed under the Bin-Packing question Part F. As these are closely related it only makes sense to address them together as one for the sake of limiting redundancy.

PROBLEM 35-1

PROBLEM 35-1

A

(Omitted)

B

Suppose we have a n items that we would need to pack into bins of capacity C each. Suppose each object have variable sizes s_1, s_2, s_3... s_n. This implies that S = $\sum_{i=1}^{n} S_i$ for the objects we need to pack, whereby S is the total size of the objects combined.
Let Z be the total available space in our bins. Let the space that remains unused for each bin be (1-C) = d,
whereby D = $\sum_{i=1}^{n} d_i$ .And suppose we are trying to pack the objects into k bins.
We can the go forward to say
k = Z/C
= (Z + (1-C))/C and since 1-C = d
= Z+d and since S = Z,
= S+d
And this would imply that S is at least such that S≤K, and we know that 0≤d, an integer. This would mean that $\lceil S \rceil$≤ k. An example would be
Suppose S = 1.5 and C = 1 and d = 0.5, we would need at least $\lceil S \rceil = \lceil 1.5 \rceil = 2$ bins for us to place the items in. In other words, logically, the number of bins must at least be $\lceil S \rceil$, otherwise the objects would not fit.

C

The first fit heuristics aims to place the object in the first bin that it fits in before moving on to the next object. It aims to find the first bin that has available capacity such that it doesn't exceed the maximum capacity of that bin. We can argue that suppose we have two bins of capacity 1 and Both these bins have not reached maximum capacity and are both halves filled. Suppose we would like to fit an object of capacity 0.5. We would still use the first fitting bin of the two bins if the capacity of the object we want to place does not exceed 1.

D

The first-fit heuristic always leaves at most one bin less than half full and happens if each object is just over size 0.5 capacity in size. Then only then can we allocate a new bin and each one goes in a separate bin. In these cases, since we haven't filled the rest of the bin almost half of each bin is not fully utilized hence goes to waste. An example would suppose we have each object have variable sizes s_1, s_2, s_3… s_n such that S = $\sum_{i=1}^{n} S_{\ i}$. And the total amount that we have packed into our bins being (1-C), where C is the capacity. This would imply that S>0.5(C-1). Rearranging this would imply that 2S+1>C. By contradiction, if we look at what we have established in part C this would imply that by contradiction:
if $\lceil 2S \rceil$<C
then C < 2S+1≤$\lceil 2S+1 \rceil$≤ C

E

Using part b – the optimal number of bins is at least $\lceil S \rceil$

Using part d - First fit will have us at most $\lceil 2S \rceil\ bins$

We can therefore say that the factor is $\frac{\lceil 2S \rceil}{\lceil S \rceil} \leq 2$ which an approximation ratio of 2.

F

```
firstFitHeuristicBin(n,S)
bincount =0
  //Initialize data structures

     for i =1 to n do
     Let the first bin that can fit object i with size S i (*) be
bin j
```

```
          If bin j exists, then
               Insert i at the list L[i] (list of objects in bin
          j)
               Update datastructures1

          Else//Object could not fit
               b = b+1
               Insert I at the list L[b]
               Update datastrutures2
          end if
     end for
Update data strucutures1: B[j] = b[j]-S i
Update data structures2: B[b] = 1-S i
```

ANALYSIS OF THE BIN PACKING ALGORITHM

TESTING INPUT DATA AND RESULTS

INVALID DATA INPUTS

BIN CAPACITY

The program will not accept any negative values for the BIN CAPCITY.

The program will not also accept BIN CAPACITY = 0

The program will not accept non integer values for BIN CAPACITY.

NUMBER OF OBJECTS

The program will not accept any negative values for NUMBER OF OBJECTS.

The program will not also accept NUMBER OF OBJECTS = 0

OBJECT VALUE/SIZE

The program will not accept any negative values for **OBJECT VALUE/SIZE.**

The program will not accept **OBJECT VALUE/SIZE > BIN CAPACITY** whereby the value/size of the object is bigger than the BIN CAPACITY.

The program will not accept invalid data inputs. Hence for the sake of reducing redundancy these will not be tested. Only valid data entry's will be accepted. For the program's response on invalid data please refer to the manual

VALID DATA INPUTS

SMALL VALUE TESTS

TEST 1

BIN CAPACITY= 10

NUMBER OF OBJECTS =5

OBJECT VALUES/SIZES = [2.0, 6.0, 4.0, 8.0, 3.0]

RESULTS:

Total Bins Used =3

Total time spent =17527100[M/S]

Bin 1 Contains values:

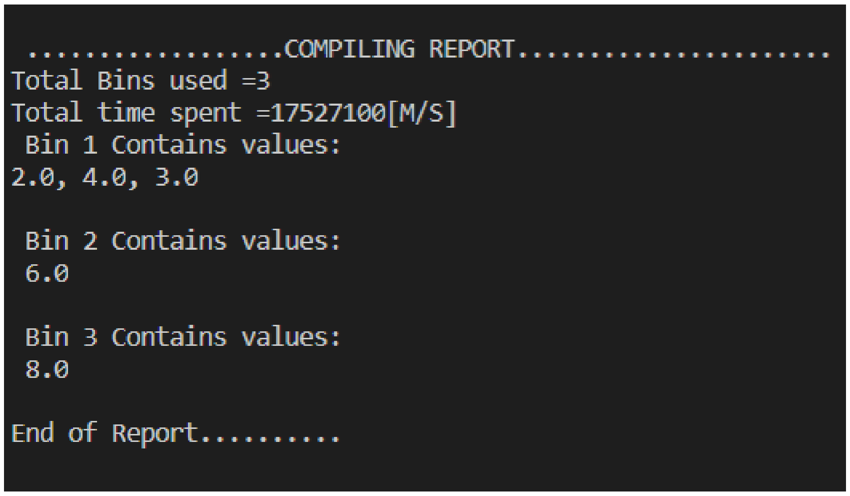
2.0, 4.0, 3.0

Bin 2 Contains values:

6.0

Bin 3 Contains values:

8.0


```
 ..................COMPILING REPORT......................
Total Bins used =3
Total time spent =17527100[M/S]
 Bin 1 Contains values:
2.0, 4.0, 3.0

 Bin 2 Contains values:
 6.0

 Bin 3 Contains values:
 8.0

End of Report..........
```

TEST 2

BIN CAPACITY= 10

NUMBER OF OBJECTS =10

OBJECT VALUES/SIZES = [2.0, 6.0, 4.0, 8.0, 3.0, 2.0, 6.0, 4.0, 8.0, 3.0]

RESULTS:

Total Bins used = 5

Total time spent =16747800[M/S]

Bin 1 Contains values:

2.0, 6.0, 2.0

Bin 2 Contains values:

4.0, 3.0, 3.0

Bin 3 Contains values:

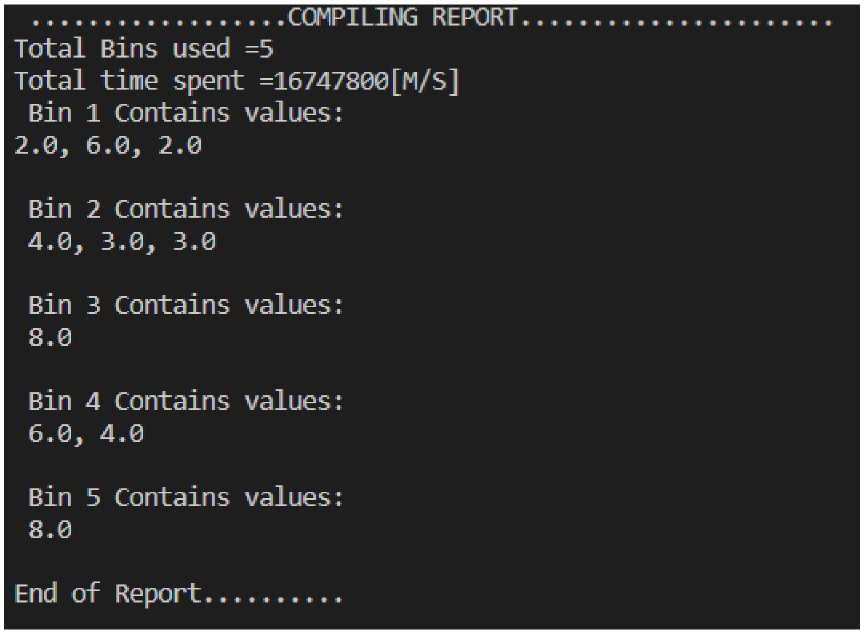
8.0

Bin 4 Contains values:

6.0, 4.0

Bin 5 Contains values

8.0



```
 ..................COMPILING REPORT......................
Total Bins used =5
Total time spent =16747800[M/S]
 Bin 1 Contains values:
2.0, 6.0, 2.0

 Bin 2 Contains values:
 4.0, 3.0, 3.0

 Bin 3 Contains values:
 8.0

 Bin 4 Contains values:
 6.0, 4.0

 Bin 5 Contains values:
 8.0

End of Report.........
```

MEDIUM VALUE TESTS

TEST 3

BIN-CAPCITY: 50

NUMBER-OF-OBJECTS: 40

OBJECT SIZES:

[43.356959730380346, 7.752365902945349, 22.54231069423211, 40.20361462769413, 29.225023326221994, 32.86029126527136, 18.34782203478329, 11.231063965763566, 32.427570739921116, 32.8556172736819, 43.943154824584326, 16.651904576759254, 33.78546384160614, 7.242887813693926, 37.9050970682197, 27.163090248578484, 0.7325614180513673, 14.057863514390222, 34.450685248288075, 26.384398118686327, 28.89322738069873, 24.184367620667928, 18.579059066026876, 47.75845667644176, 36.19775137287294, 48.97049687451174, 47.99778476731891, 48.03431128935811, 0.9292170319424591, 17.783686938906325, 37.12260824864424, 38.64211094462654, 40.38778361622036, 43.54435470065102, 4.560113000438336, 10.419796165394951, 12.856936742895135, 38.63424942675273, 17.609528231827394, 20.96225952663634]

RESULTS:

Total Bins used =25

Total time spent =49738000[M/S]

Bin 1 Contains values:

43.356959730380346, 0.7325614180513673, 0.9292170319424591, 4.560113000438336

Bin 2 Contains values:

7.752365902945349, 22.54231069423211, 18.34782203478329

Bin 3 Contains values:

40.20361462769413, 7.242887813693926

Bin 4 Contains values:

29.225023326221994, 11.231063965763566

Bin 5 Contains values:

32.86029126527136, 16.651904576759254

Bin 6 Contains values:

32.427570739921116, 14.057863514390222

Bin 7 Contains values:

32.8556172736819, 10.419796165394951

Bin 8 Contains values:

43.943154824584326

Bin 9 Contains values:

33.78546384160614, 12.856936742895135

Bin 10 Contains values:

37.9050970682197

Bin 11 Contains values:

27.163090248578484, 18.579059066026876

Bin 12 Contains values:

34.450685248288075

Bin 13 Contains values:

26.384398118686327, 17.783686938906325

Bin 14 Contains values:

28.89322738069873, 17.609528231827394

Bin 15 Contains values:

24.184367620667928, 20.96225952663634

Bin 16 Contains values:

47.75845667644176

Bin 17 Contains values:

36.19775137287294

Bin 18 Contains values:

48.97049687451174

Bin 19 Contains values:

47.99778476731891

Bin 20 Contains values:

48.03431128935811

Bin 21 Contains values:

37.12260824864424

Bin 22 Contains values:

38.64211094462654

Bin 23 Contains values:

40.38778361622036

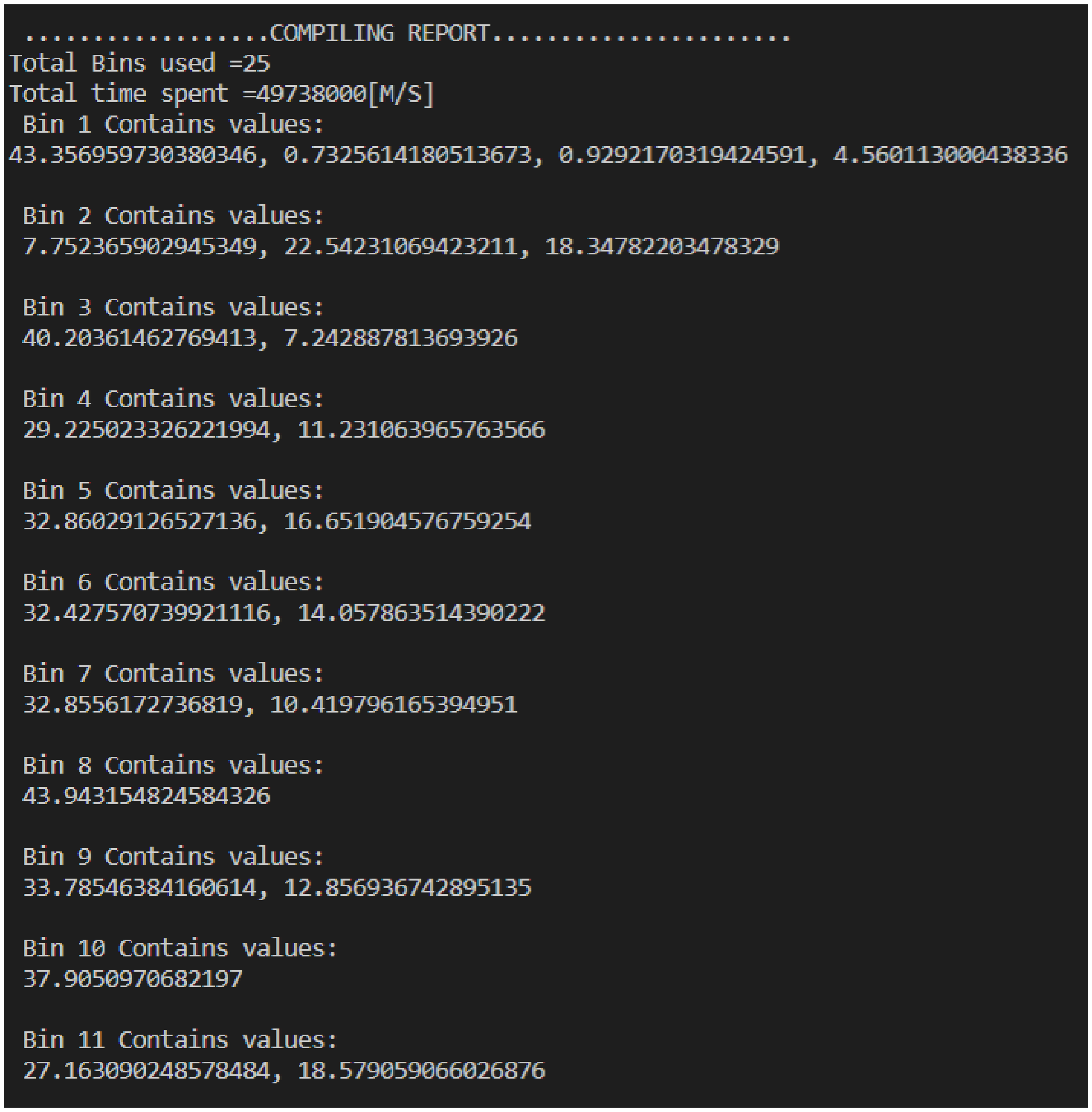
Bin 24 Contains values:

43.54435470065102

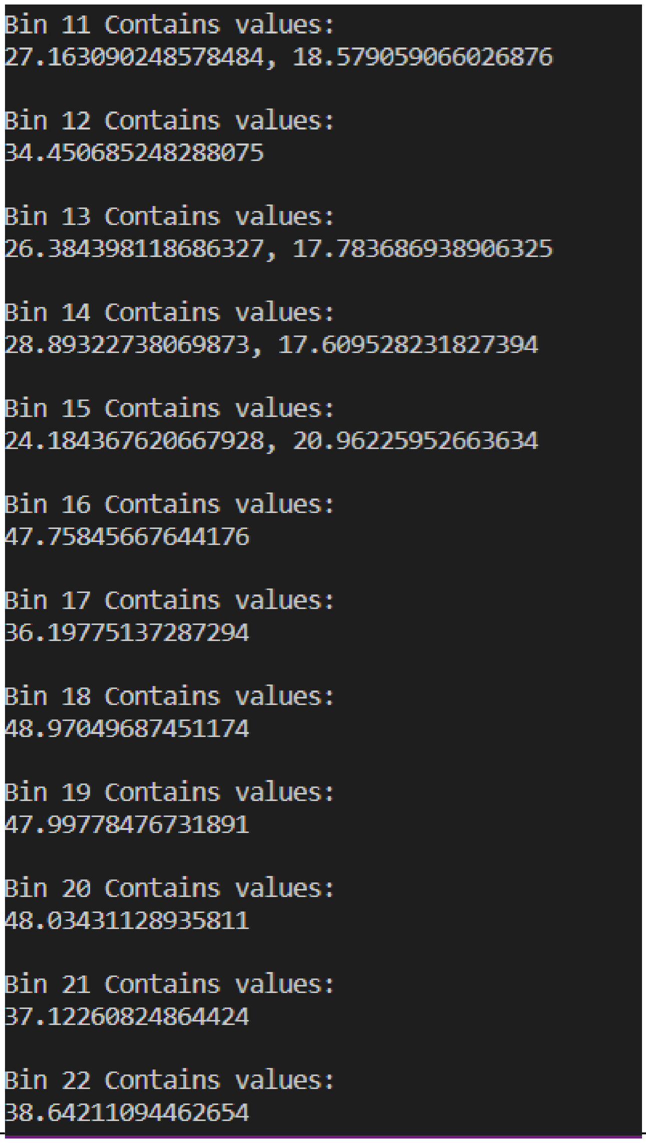
Bin 25 Contains values:

38.63424942675273

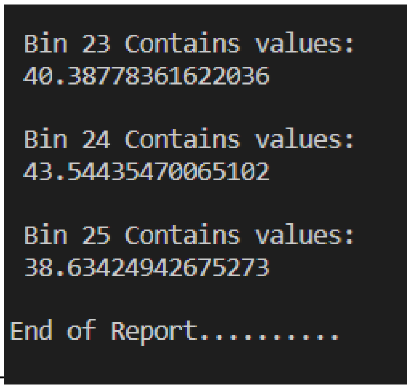
:



```
 ..................COMPILING REPORT.....................
Total Bins used =25
Total time spent =49738000[M/S]
 Bin 1 Contains values:
43.356959730380346, 0.7325614180513673, 0.9292170319424591, 4.560113000438336

 Bin 2 Contains values:
 7.752365902945349, 22.54231069423211, 18.34782203478329

 Bin 3 Contains values:
 40.20361462769413, 7.242887813693926

 Bin 4 Contains values:
 29.225023326221994, 11.231063965763566

 Bin 5 Contains values:
 32.86029126527136, 16.651904576759254

 Bin 6 Contains values:
 32.427570739921116, 14.057863514390222

 Bin 7 Contains values:
 32.8556172736819, 10.419796165394951

 Bin 8 Contains values:
 43.943154824584326

 Bin 9 Contains values:
 33.78546384160614, 12.856936742895135

 Bin 10 Contains values:
 37.9050970682197

 Bin 11 Contains values:
 27.163090248578484, 18.579059066026876
```



```
Bin 11 Contains values:
27.163090248578484, 18.579059066026876

Bin 12 Contains values:
34.450685248288075

Bin 13 Contains values:
26.384398118686327, 17.783686938906325

Bin 14 Contains values:
28.89322738069873, 17.609528231827394

Bin 15 Contains values:
24.184367620667928, 20.96225952663634

Bin 16 Contains values:
47.75845667644176

Bin 17 Contains values:
36.19775137287294

Bin 18 Contains values:
48.97049687451174

Bin 19 Contains values:
47.99778476731891

Bin 20 Contains values:
48.03431128935811

Bin 21 Contains values:
37.12260824864424

Bin 22 Contains values:
38.64211094462654
```



```
Bin 23 Contains values:
40.38778361622036

Bin 24 Contains values:
43.54435470065102

Bin 25 Contains values:
38.63424942675273

End of Report..........
```

LARGE VALUE TESTS

TEST 4

LARGE TEST ONLY INPUT AND RESULT SUMMARY SHOWN

BIN-CAPCITY: 1000

NUMBER-OF-OBJECTS: 50

BIN-CAPCITY: 1000

NUMBER-OF-OBJECTS: 50

OBJECT SIZES:

[367.15391328846346, 117.10034306566648, 249.2855775661342, 569.6083527335829, 46.63054294849669, 665.6650069579864, 261.19877234716347, 83.34847985611248, 825.5993821646211, 380.72270256350345, 668.5125629463146, 847.0631042997227, 477.66898299834395, 509.18685999241, 942.8297734340216, 522.8002785431058, 618.2663409072778, 423.46781220868587, 245.91275241531673, 106.53842934626807, 149.18440600721928, 532.7098709810153, 615.4811449318569, 465.691373126888, 49.99704344727098, 645.8593984336679, 857.4899308041546, 692.054566141536, 289.9934209184176, 318.0023604231881, 907.3226513740793, 693.5666092654373, 441.59768591862934, 627.8666958223364, 8.720051117481649, 755.862730658582, 174.04720885534857, 781.8837562062596, 913.8440357089357, 639.5437494633852, 181.2044893718032, 914.1471026362578, 455.85008933330755, 147.37435846344107, 255.37214546982153, 464.7946004056547, 257.50847380667756, 995.4589598823522, 155.67041627941435, 555.5696730157213]

RESULT:

Total Bins used =27

Total time spent =399027500[M/S]

TEST 5

LARGE TEST ONLY INPUT SUMMARY AND RESULT SUMMARY SHOWN

BIN-CAPCITY: 10000

NUMBER-OF-OBJECTS: 1000

RESULT:

Total Bins used =553

Total time spent =1583577600[M/S]

TEST 6

LARGE TEST ONLY INPUT SUMMARY AND RESULT SUMMARY SHOWN

BIN-CAPCITY: 100000

NUMBER-OF-OBJECTS: 10000

RESULT:

Total Bins used =5117

Total time spent =6471505900 [M/S]

COMPLEXITY ANALYSIS

The complexity of the algorithm used in this project was based off the size of the input data and basic operations that needed to be done on that set of input data.

We will always be looking at the worst case as it gives a more reliable and appropriate bound for our project. If we are a considering that the object goes through all the of the bins and cannot fit into anyone of the bins available this will give us O(binCount) where these are the bins which are available before a new bin is created for that particular object which did not fit.

Also, for each object to be inserted into the bin we do a comparison to the bin. For this step we can conclude our running time to be O(NumObjects).

If we put these steps together, we get a time complexity of O(binCount*NumObjects) which is approximately $O(N^2)$.

Hence, we can conclude our running time to be $O(N^2)$.

INSTRUCTION MANUAL

1.SETTING UP

PRE-LIMINARY REQUIREMENTS

Before running the project there are a few things needed in the set-up process. You will need to make sure you have the following.

- **RAM: 128 MB;**
- **Disk space: 124 MB**
- **Browsers: Internet Explorer 7.0 and above, Firefox 3.6 and above**
- **JAVA SDK or JRE 1.6 or higher**

DOWNLOADING THE FILE

Once you have the setup complete, you may now proceed to download the file named **A5.Java**

Make sure that you download the file to a disk location that you have administrative rights to run the file. This should be just one file. Once you have downloaded the file, you may proceed to the execution stage.

2.GETTING TO THE FILE

To execute the file, you are going to need a command line interface. Here we will explain how to navigate to the file after downloading it.

WINDOWS USERS

1. Look for the button with the **windows logo** on your keyboard. It's usually located in the lower left corner of you keyboard. ⊞ Win
2. Once you have found the windows logo. Press it.
3. A **start menu** should immediately show up
4. Once the start menu shows up you should see a search bar right at the bottom of the **Start Menu.**
5. In the bar type in **CMD**. You should see the program, with an icon like this C:_ .
6. Press enter or right click on the icon.
7. This should open your Command Line Interface, in this case **Command Prompt.**
8. In the command prompt, use **basic commands** to navigate to the folder location where you downloaded the file.

MAC USERS

1. Click the spotlight icon, alternatively you can enter ⌘ + space.
2. Once the search bar is open type in "***Terminal***" in the search box
3. You should see a terminal icon such as the following Terminal.
4. Double-click on the icon. This should open the Command Line Interface for MAC.
5. Use basic terminal commands to navigate to the location where you downloaded A5.java to.

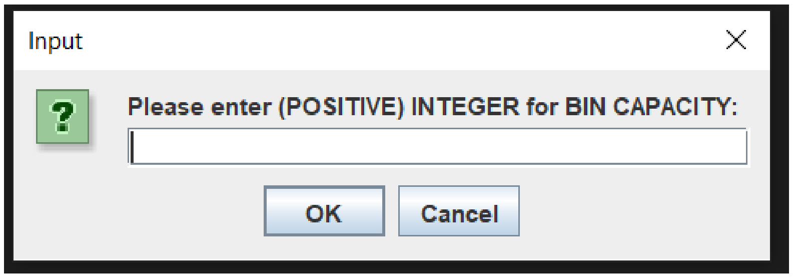
3.EXECUTION

1. Once you have reached the location type in **javac A5.java** to compile the file.
2. After a successful compilation you may type in **java A5** to run the file.

   Once you type this in the program should immediately start running.

1.BIN CAPACITY

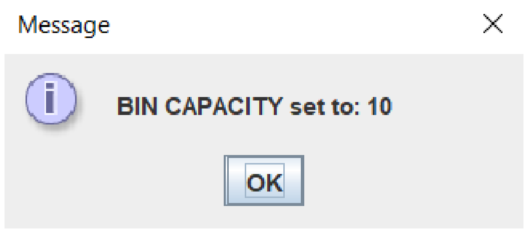
Once the program is running. It will prompt you to enter BIN CAPACITY.


Please enter (POSITIVE) INTEGER for BIN CAPACITY:

The **BIN CAPACITY** is the total amount of value/size that the bin can accommodate before creating a new bin.

Note: 0, and non discreet positive integers are not accepted.
You may only enter positive, integers.
Entering an invalid value will result in the program showing you the following.

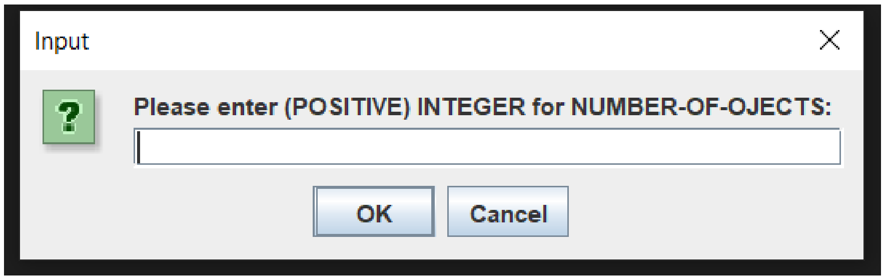
Once the BIN CAPACITY value has been accepted it should show the following


BIN CAPACITY set to: 10

Note: *If you intend to run a quick test. It's advised to not use large bin capacities with randomized input and large bin capacities with small object values.*

2.NUMBER OF OBJECTS

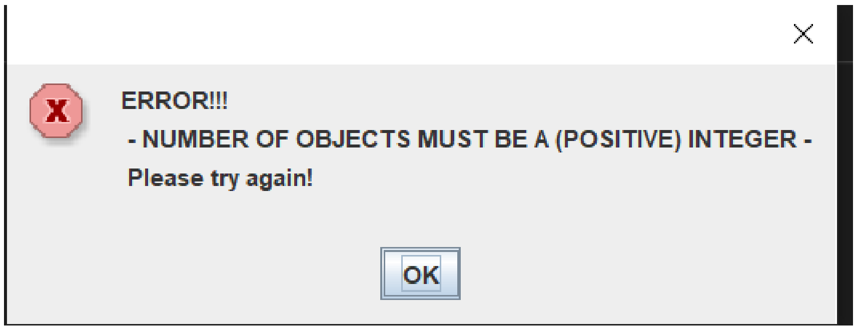
After you have entered the Capacity of your Bin and the program has accepted it. It will prompt you to enter the number of objects you wish to work with. For TESTS smaller values are recommended especially if you wish to enter the number of objects manually.



Please enter (POSITIVE) INTEGER for NUMBER-OF-OJECTS:

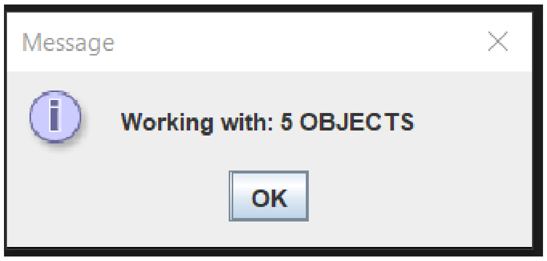
Note: You may only enter ***positive, integers***.
0, and non discreet positive integers are not accepted.

Entering an invalid value will result in the program showing you the following.



ERROR!!!
- NUMBER OF OBJECTS MUST BE A (POSITIVE) INTEGER -
Please try again!

Once your Number of Objects has been accepted. The program should show you the number of objects you decided to work with like the following (5 Objects selected in this case)

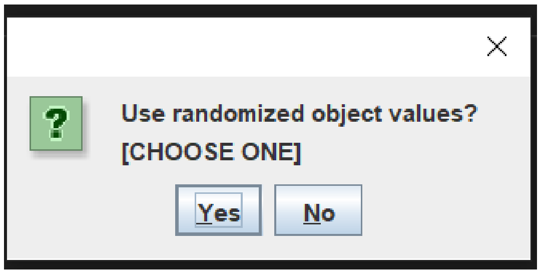

Message

Working with: 5 OBJECTS

Note: *If you intend to run a large test and manually enter your object/size values. It's not advised to select a large set of objects as you would have to manually enter their individual values i.e. If you enter your Number OF Objects to be 100, you will have to manually enter all of the 100 individual values.*

3.OBJECT SIZES/VALUES

The program offers you two ways of inputting the object values/sizes. You can either manually enter the values or you can use randomized values for your object size values. You may choose the one you see most fit for your application.


Use randomized object values? [CHOOSE ONE]

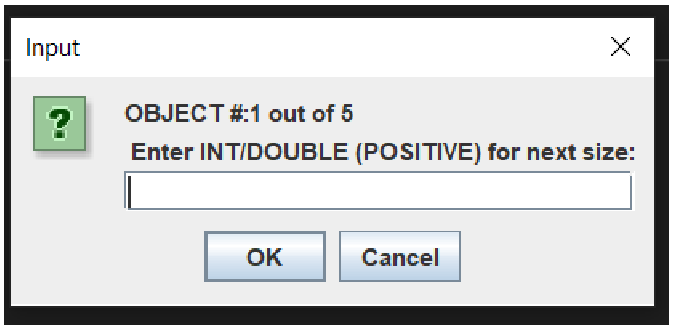
3.1 USING RANDOMIZED VALUES/SIZES

If you decide to use the randomized values, you will not need to worry about entering the object values manually. The program will enter these for you.

This comes handy when running tests for a large set of objects and large bins.

3.2 MANUALLY INPUTTING VALUES/SIZES

If you have chosen to input the values manually the program should show you this



Input

OBJECT #:1 out of 5

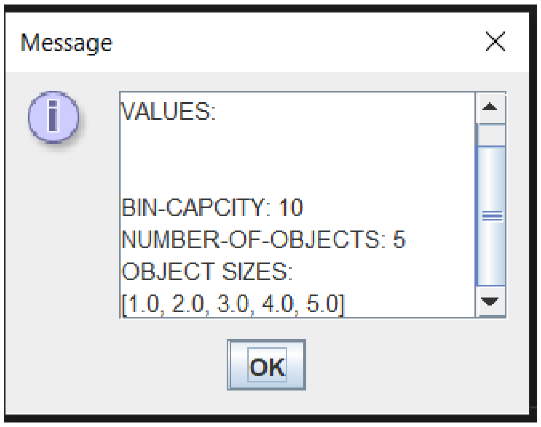
Enter INT/DOUBLE (POSITIVE) for next size:

You may enter in positive values for your sizes. The program will accept both integers and doubles as your values but will not accept 0 or negative values as your input.
It will prompt you continuously up until you have reached the number of objects you requested to work with.

Note: You may only enter ***positive, integers*** and ***doubles***.
0, and negative values are not accepted.

3.3 VALUE SUMMARY

Before you proceed to execute the program. It will show you a summary of the parameters selected.
If you have chosen random values these will be shown to you here as well.

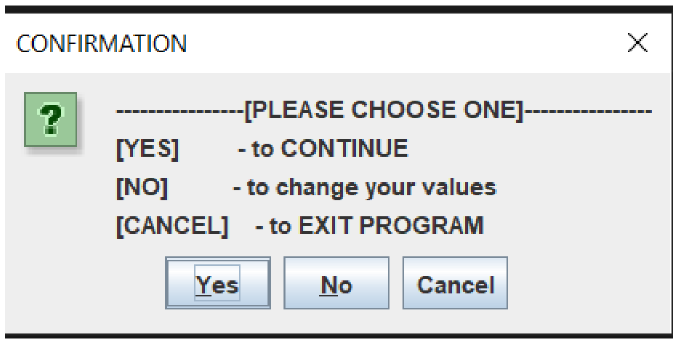


```
VALUES:


BIN-CAPCITY: 10
NUMBER-OF-OBJECTS: 5
OBJECT SIZES:
[1.0, 2.0, 3.0, 4.0, 5.0]
```

Click **OK** to proceed to the next menu

3.4. VALUE CONFIRMATION

Before you run the algorithm, it will give you a chance to change your param-eters.



```
----------------[PLEASE CHOOSE ONE]----------------
[YES]        - to CONTINUE
[NO]         - to change your values
[CANCEL]     - to EXIT PROGRAM
```

NO- If you are not happy with the values you may click NO and this will take you back to selecting the BIN CAPACITY

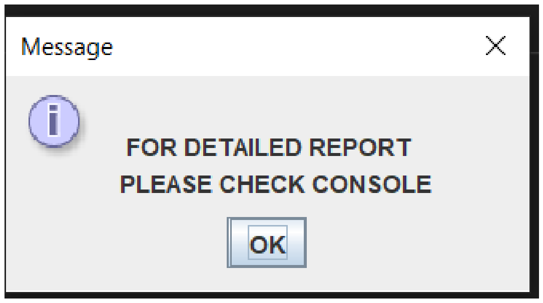
CANCEL – This will exit the program

YES – If you are happy with the parameters you may click and the Algorithm will start running.

4.INTERPRETING RESULTS

4.1 SUCCESSFUL EXECUTION

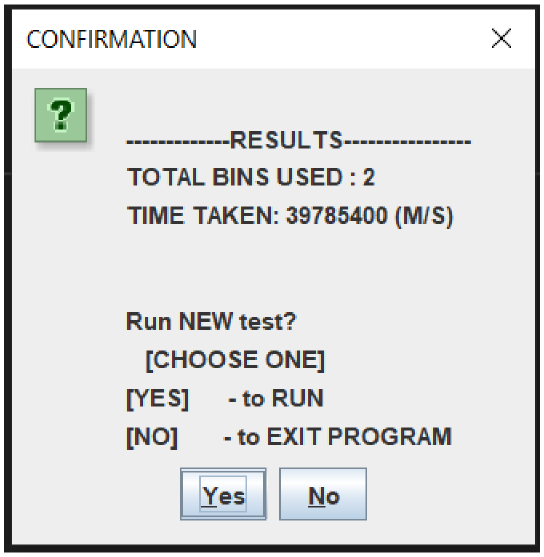
Once the program has finished running it should show you this message to indicate that it ran successfully. You may click **OK** to see the summary



FOR DETAILED REPORT
PLEASE CHECK CONSOLE

4.2 RESULT SUMMARY

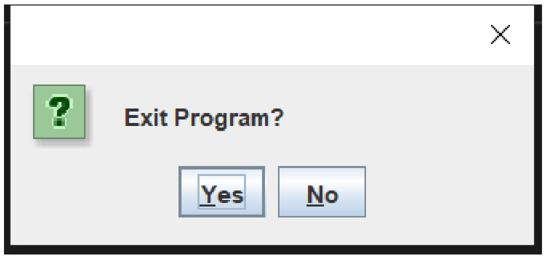
A successful run should show you a brief summary which shows the time and the number of bins used for the Algorithm



```
-------------RESULTS----------------
TOTAL BINS USED : 2
TIME TAKEN: 39785400 (M/S)


Run NEW test?
   [CHOOSE ONE]
[YES]     - to RUN
[NO]      - to EXIT PROGRAM
```

The program gives you an option to run a new set of values. The detailed results to the execution will be shown in your Command Line Interface.

- Selecting **YES** will take you to the BIN CAPACITY selection again.
- Selecting **NO** will exit the program and display the following



Exit Program?

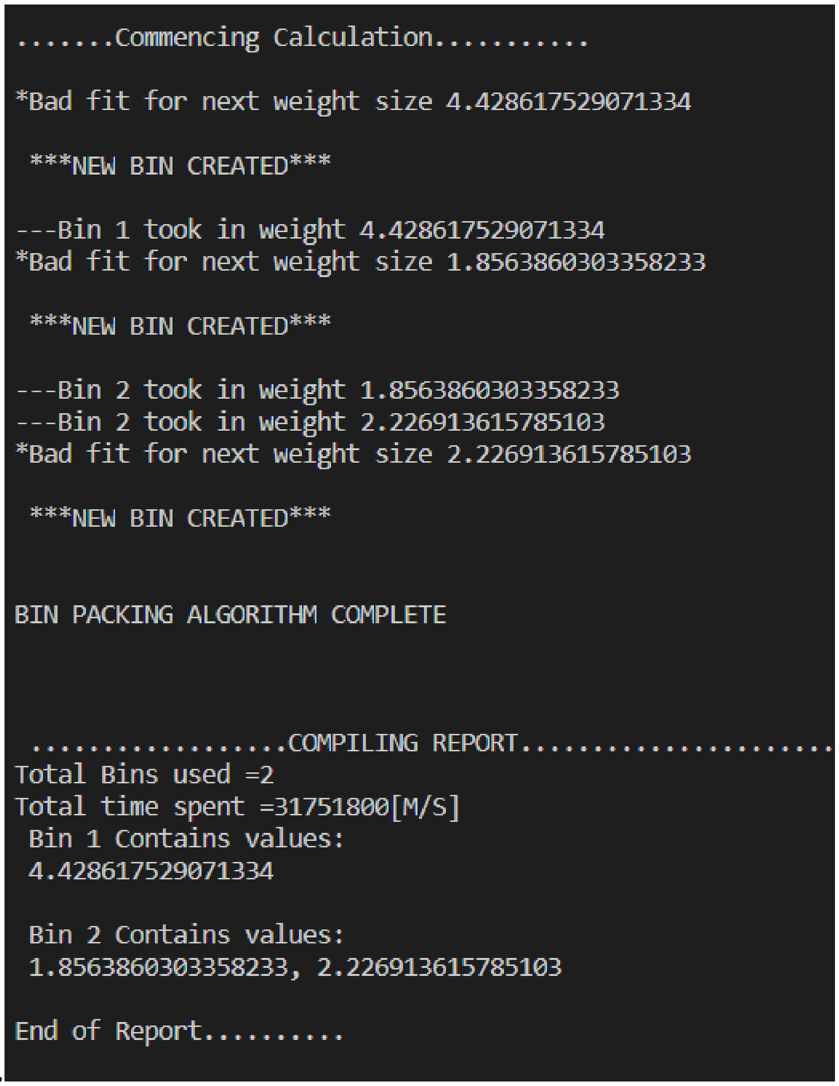
- If you would like to run the program again then select **YES**

4.3 DETAILED REPORT

A detailed report is captured in your Command Line Interface's terminal each run. To see the detailed report. Open your Command Line Interface.

The report will show you the order of events in which the algorithm made its calculations.

It should you something like the following



```
.......Commencing Calculation...........

*Bad fit for next weight size 4.428617529071334

 ***NEW BIN CREATED***

---Bin 1 took in weight 4.428617529071334
*Bad fit for next weight size 1.8563860303358233

 ***NEW BIN CREATED***

---Bin 2 took in weight 1.8563860303358233
---Bin 2 took in weight 2.226913615785103
*Bad fit for next weight size 2.226913615785103

 ***NEW BIN CREATED***


BIN PACKING ALGORITHM COMPLETE



 ..................COMPILING REPORT.....................
Total Bins used =2
Total time spent =31751800[M/S]
 Bin 1 Contains values:
 4.428617529071334

 Bin 2 Contains values:
 1.8563860303358233, 2.226913615785103

End of Report..........
```

.

- **NEW BIN CREATED** – The algorithm creates a new Bin each time an element cannot fit in the current bin it's working in
- **Total Bins Used** – These are the total number of Bins used to pack the elements
- **Total Time Spent** - This is the total time the Algorithm part of the program took to pack the objects

- **Bin X Contains Values**- This shows individual values contained in each bin and how they were packed.

REFLECTIONS

As I was doing my research on the first fit heuristic for the bin-packing algorithm I learnt that there are may other efficient ways that this problem can be approached. I learnt of a few more other ways that we could have approached the problem such as sorted first fit and best fit. I also had a direct interaction with how the choice of the Bin size and the choice of the number of Bin objects can all have an impact on the running time of the Algorithm. I approached the Project by doing the questions on problem 35-1 first and had missed out on some of the importance of the constraints
After running multiple tests with small inputs and larger inputs of objects (e.g. Up to a grand list of 200,000 uniformly distributed object sizes), I noticed that I would get to times where the program would get very large and consume more memory than allocated to it by the computer. This is a direct indication of how the program can grow very large very quickly hence the importance of efficient algorithms always.
Amazingly, for smaller input sizes the program ran very much within acceptable times and did not crash or strain my computer only reconsolidating the importance of how programs can grow very large. Finally, the project helped me consolidate everything I have learnt throughout the course and makes me realise the importance of how appropriate data structures can have a larger than life impact on a program. In this case I was only measuring the algorithm speed and I am guessing that if I was to measure access times for the data structures used my readings would have been greater. I am grateful for such a learning experience and I hope to carry it forward into my Computer Science Career.

REFRENCES

BIBLIOGRAPHY

Coffman, E. G., Csirik, J., Galambos, G., Martello, S., & Vigo, D. (2013). *Bin packing approximation algorithms: Survey and classification*. Retrieved 1 23, 2020, from https://research.vu.nl/en/publications/bin-packing-approximation-algorithms-survey-and-classification

Coffman, E. G., Leung, J. Y.-T., & Ting, D. W. (1978). Bin packing: Maximizing the number of pieces packed. *Acta Informatica, 9*(3), 263-271. Retrieved 1 23, 2020, from https://link.springer.com/article/10.1007/bf00288885

Cormen, T. H. (2002). *Algorithmic Complexity.* CRC Press. Retrieved 1 23, 2020

Csirik, J., Johnson, D. S., & Kenyon, C. (2001). *Better approximation algorithms for bin covering*. Retrieved 1 23, 2020, from https://dl.acm.org/citation.cfm?id=365533

H., T., Cormen, E., C., Leiserson, L., R., Rivest, . . . Stein. (n.d.). *Introduction to Algorithms.* MIT Press / McGraw-Hill. Retrieved 1 23, 2020

Java Downloads for All Operating Systems. (n.d.). Retrieved 1 23, 2020, from Java.com: http://www.java.com/en/download/manual.jsp

Seiden, S. S. (2001). On the Online Bin Packing Problem. *Lecture Notes in Computer Science*, 237-248. Retrieved 1 23, 2020, from https://link.springer.com/chapter/10.1007/3-540-48224-5_20