Problem 1: (20 pts) In the bin packing problem, items of different weights (or sizes) must be packed into a finite number of bins each with the capacity C in a way that minimizes the number of bins used. The decision version of the bin packing problem (deciding if objects will fit into <= k bins) is NP-complete. There is no known polynomial time algorithm to solve the optimization version of the bin packing problem. In this homework you will be examining three greedy approximation algorithms to solve the bin packing problem.

- First-Fit: Put each item as you come to it into the first (earliest opened) bin into which it fits. If there is no available bin then open a new bin.

a) Give pseudo code and the running time for each of the approximation algorithms.

```
Binpack(list of items, C)
Create list to hold list of bins
    For each item in list:
        If there is a bin available that can fit item place into first bin where it fits
        Else make new bin and place item in
```

In worse case when everything is larger than half of C we would run the for loop N times but we would possibly check N bins within this loop.

Running time: O(n^2)

- First-Fit-Decreasing: First sort the items in decreasing order by size, then use First-Fit on the resulting list.

a) Give pseudo code and the running time for each of the approximation algorithms.

```
Binpack(list of items, C)
Create list to hold list of bins
Sort list of items to be in decreasing order by size
    For each item in list:
        If there is a bin available that can fit item place into first bin where it fits
        Else make new bin and place item in
```

Running time: O(n^2)

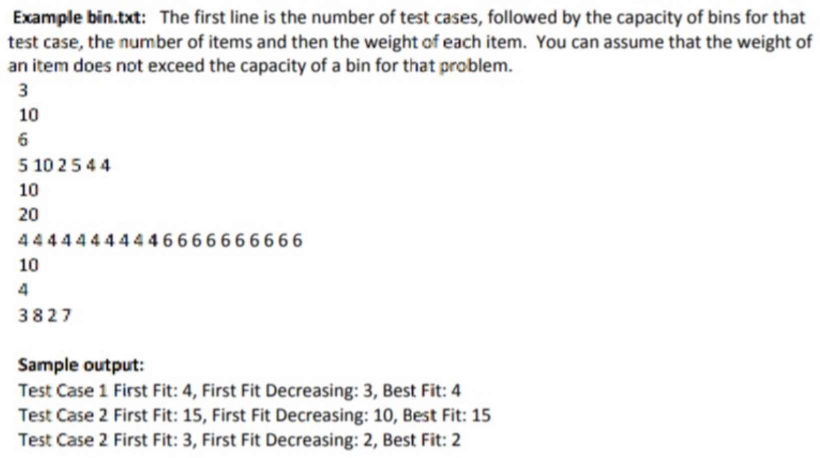
• Best Fit: Place the items in the order in which they arrive. Place the next item into the bin which will leave the least room left over after the item is placed in the bin. If it does not fit in any bin, start a new bin.

a) Give pseudo code and the running time for each of the approximation algorithms.

```
Binpack(list of items, C)
Create list to hold list of bins
    For each item in list:
        -If there is one or more bin available that can fit item place into bin where doing so will
        leave the least room left over after the item is placed in the bin
        -Else make new bin and place item in
```

Running time: O(n^2)

b) Implement the algorithms in Python, C++ or C. Your program named binpack should read in a text file named bin.txt with multiple test cases as explained below and output to the terminal the number of bins each algorithm calculated for each test case. Submit a README file and your program to TEACH.



Example bin.txt: The first line is the number of test cases, followed by the capacity of bins for that test case, the number of items and then the weight of each item. You can assume that the weight of an item does not exceed the capacity of a bin for that problem.

```
3
10
6
5 10 2 5 4 4
10
20
4 4 4 4 4 4 4 4 4 4 6 6 6 6 6 6 6 6 6 6
10
4
3 8 2 7
```

Sample output:

```
Test Case 1 First Fit: 4, First Fit Decreasing: 3, Best Fit: 4
Test Case 2 First Fit: 15, First Fit Decreasing: 10, Best Fit: 15
Test Case 2 First Fit: 3, First Fit Decreasing: 2, Best Fit: 2
```

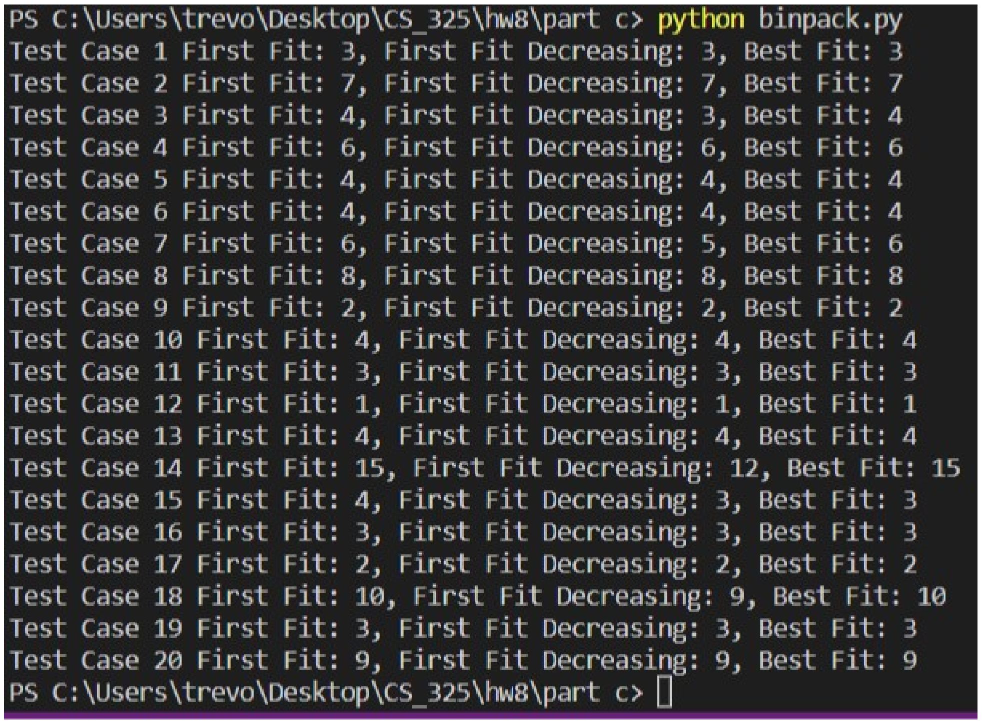
c) Randomly generate at least 20 bin packing instances. Summarize the results for each algorithm. Which algorithm performs better? How often? Note: Submit a description of how the inputs were generated not the code used to produce the random inputs.

Description of how the inputs were generated

I used a random number generator online to produce random numbers in ranges appropriate for each instance's capacity of bins, etc. I then manually put these into bin.txt as I generated them. I then ran my program on this larger txt file. The RNG I used specifically is Google's within Google search.

Summarize the results for each algorithm.

Most test cases resulted in algorithms having same results. Except for 5 instances.



```
PS C:\Users\trevo\Desktop\CS_325\hw8\part c> python binpack.py
Test Case 1 First Fit: 3, First Fit Decreasing: 3, Best Fit: 3
Test Case 2 First Fit: 7, First Fit Decreasing: 7, Best Fit: 7
Test Case 3 First Fit: 4, First Fit Decreasing: 3, Best Fit: 4
Test Case 4 First Fit: 6, First Fit Decreasing: 6, Best Fit: 6
Test Case 5 First Fit: 4, First Fit Decreasing: 4, Best Fit: 4
Test Case 6 First Fit: 4, First Fit Decreasing: 4, Best Fit: 4
Test Case 7 First Fit: 6, First Fit Decreasing: 5, Best Fit: 6
Test Case 8 First Fit: 8, First Fit Decreasing: 8, Best Fit: 8
Test Case 9 First Fit: 2, First Fit Decreasing: 2, Best Fit: 2
Test Case 10 First Fit: 4, First Fit Decreasing: 4, Best Fit: 4
Test Case 11 First Fit: 3, First Fit Decreasing: 3, Best Fit: 3
Test Case 12 First Fit: 1, First Fit Decreasing: 1, Best Fit: 1
Test Case 13 First Fit: 4, First Fit Decreasing: 4, Best Fit: 4
Test Case 14 First Fit: 15, First Fit Decreasing: 12, Best Fit: 15
Test Case 15 First Fit: 4, First Fit Decreasing: 3, Best Fit: 3
Test Case 16 First Fit: 3, First Fit Decreasing: 3, Best Fit: 3
Test Case 17 First Fit: 2, First Fit Decreasing: 2, Best Fit: 2
Test Case 18 First Fit: 10, First Fit Decreasing: 9, Best Fit: 10
Test Case 19 First Fit: 3, First Fit Decreasing: 3, Best Fit: 3
Test Case 20 First Fit: 9, First Fit Decreasing: 9, Best Fit: 9
PS C:\Users\trevo\Desktop\CS_325\hw8\part c> 
```

Which algorithm performs better? How often?

All equal: 15 Test Cases

FF best: 0 Test Cases

FFD best: 4.5 Test Cases

BF best: 0.5 Test Cases

The best algorithm is First Fit Decreasing, one Test Case resulted in BF and FFD having the same better solution compared to FF which is what the .5 represents. Outside of this most Test Cases resulted in algorithms having same results. However, FFD sticks out as the best in that 4.5/20 times it was the best solution.

In some of the 4.5 cases FFD was two less or three less than the other(s).

Problem 2: (10 pts) An exact solution to the bin packing optimization problem can be found using 0-1 integer programming (IP) see the format on the Wikipedia page.

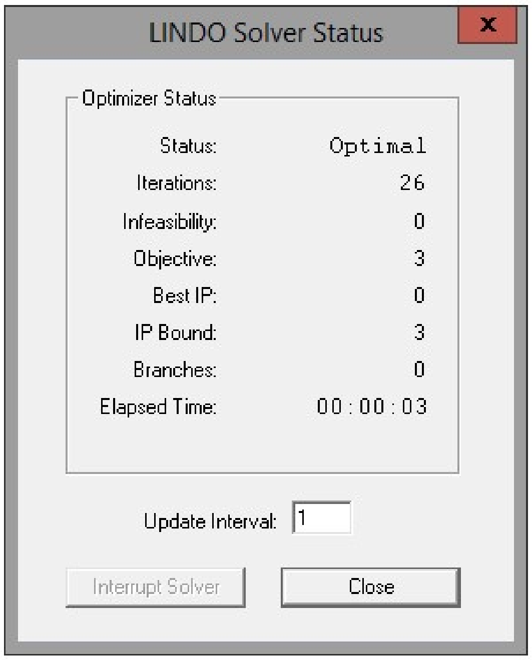
Write an integer program for each of the following instances of bin packing and solve with the software of your choice. Submit a copy of the code and interpret the results.

a) Six items S = { 4, 4, 4, 6, 6, 6} and bin capacity of 10

```
MIN y1 + y2 + y3 + y4 + y5 + y6
ST
y1 + y2 + y3 + y4 + y5 + y6 > 0

4x11 + 4x12 + 4x13 + 6x14 + 6x15 + 6x16 - 10y1 <= 0
4x21 + 4x22 + 4x23 + 6x24 + 6x25 + 6x26 - 10y2 <= 0
4x31 + 4x32 + 4x33 + 6x34 + 6x35 + 6x36 - 10y3 <= 0
4x41 + 4x42 + 4x43 + 6x44 + 6x45 + 6x46 - 10y4 <= 0
4x51 + 4x52 + 4x53 + 6x54 + 6x55 + 6x56 - 10y5 <= 0
4x61 + 4x62 + 4x63 + 6x64 + 6x65 + 6x66 - 10y6 <= 0

x11 + x21 + x31 + x41 + x51 + x61 = 1
x12 + x22 + x32 + x42 + x52 + x62 = 1
x13 + x23 + x33 + x43 + x53 + x63 = 1
x14 + x24 + x34 + x44 + x54 + x64 = 1
x15 + x25 + x35 + x45 + x55 + x65 = 1
x16 + x26 + x36 + x46 + x56 + x66 = 1
END
INT y1
INT y2
INT y3
INT y4
INT y5
INT y6
INT x11
INT x12
INT x13
INT x14
INT x15
INT x16
INT x21
INT x22
INT x23
INT x24
INT x25
INT x26
INT x31
INT x32
INT x33
INT x34
INT x35
INT x36
INT x41
INT x42
INT x43
INT x44
INT x45
INT x46
INT x51
INT x52
INT x53
INT x54
INT x55
INT x56
INT x61
INT x62
INT x63
INT x64
INT x65
INT x66
```



LINDO Solver Status

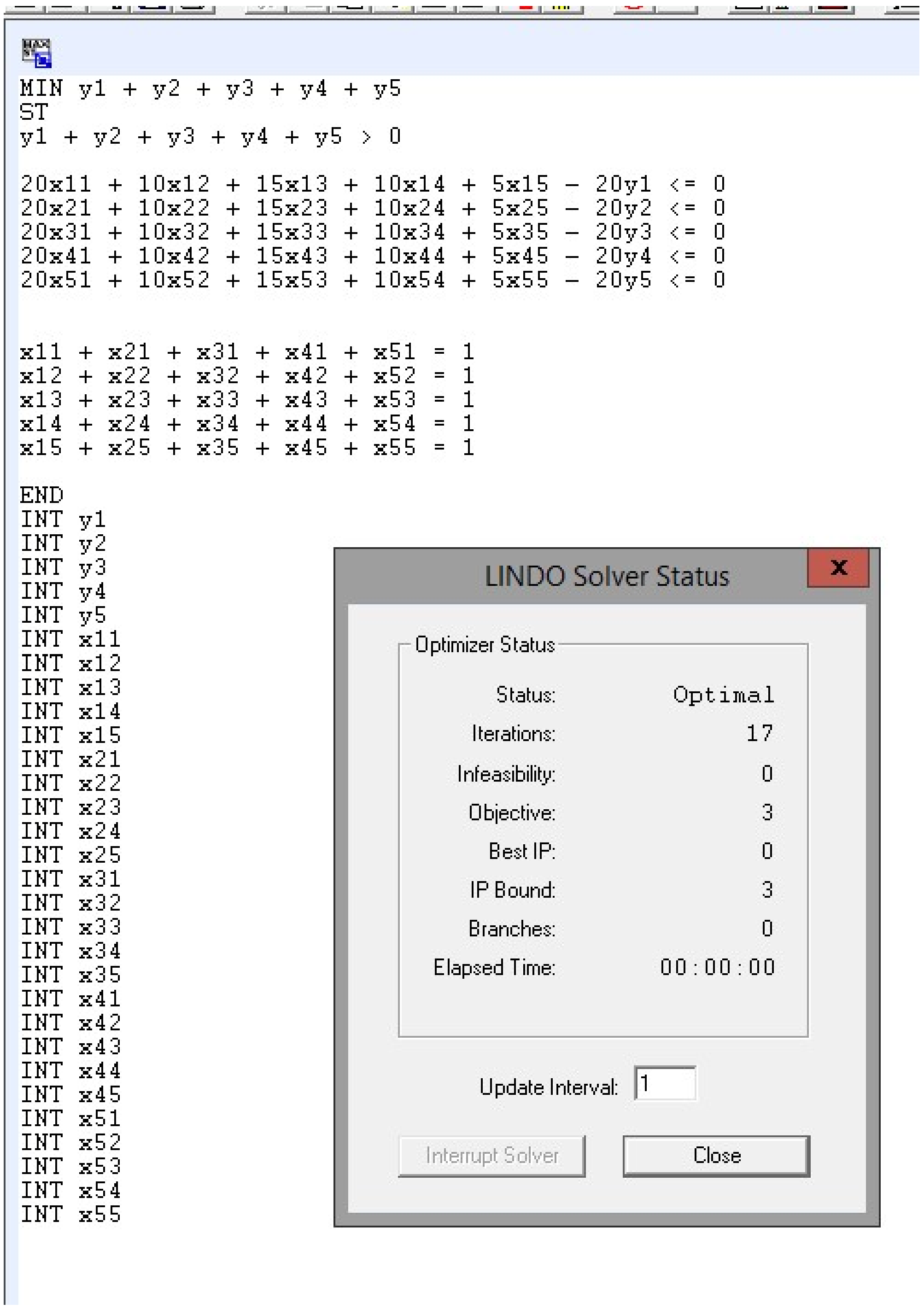
Optimizer Status

```
Status:          Optimal
Iterations:      26
Infeasibility:   0
Objective:       3
Best IP:         0
IP Bound:        3
Branches:        0
Elapsed Time:    00:00:03
```

Update Interval: 1

This means the optimal solution was found, and found to be 3 bins.

b) Five items S = { 20, 10, 15, 10, 5} and bin capacity of 20



```
MIN y1 + y2 + y3 + y4 + y5
ST
y1 + y2 + y3 + y4 + y5 > 0

20x11 + 10x12 + 15x13 + 10x14 + 5x15 - 20y1 <= 0
20x21 + 10x22 + 15x23 + 10x24 + 5x25 - 20y2 <= 0
20x31 + 10x32 + 15x33 + 10x34 + 5x35 - 20y3 <= 0
20x41 + 10x42 + 15x43 + 10x44 + 5x45 - 20y4 <= 0
20x51 + 10x52 + 15x53 + 10x54 + 5x55 - 20y5 <= 0


x11 + x21 + x31 + x41 + x51 = 1
x12 + x22 + x32 + x42 + x52 = 1
x13 + x23 + x33 + x43 + x53 = 1
x14 + x24 + x34 + x44 + x54 = 1
x15 + x25 + x35 + x45 + x55 = 1

END
INT y1
INT y2
INT y3
INT y4
INT y5
INT x11
INT x12
INT x13
INT x14
INT x15
INT x21
INT x22
INT x23
INT x24
INT x25
INT x31
INT x32
INT x33
INT x34
INT x35
INT x41
INT x42
INT x43
INT x44
INT x45
INT x51
INT x52
INT x53
INT x54
INT x55
```

LINDO Solver Status

Optimizer Status

Status:	Optimal
Iterations:	17
Infeasibility:	0
Objective:	3
Best IP:	0
IP Bound:	3
Branches:	0
Elapsed Time:	00:00:00

Update Interval: 1

This means the optimal solution was found, and found to be 3 bins.

Note: The version of LINDO that you have access to on the OSU server has a limit of 50 integer variables. Therefore, LINDO will only be able to solve problems with at most 6 items.