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)

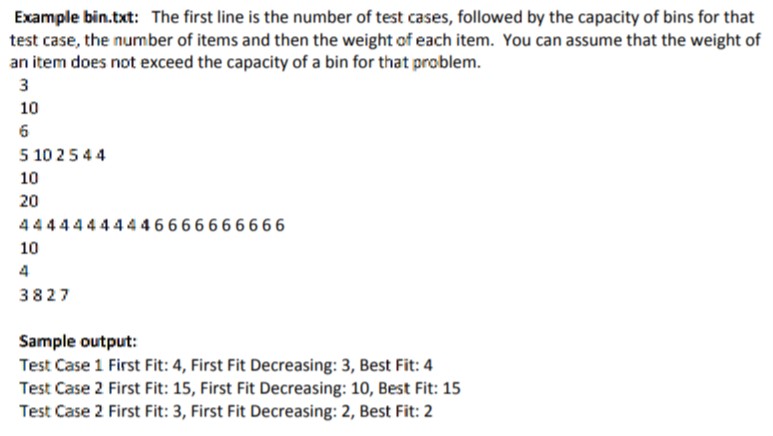
Create list to hold list of bins

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.

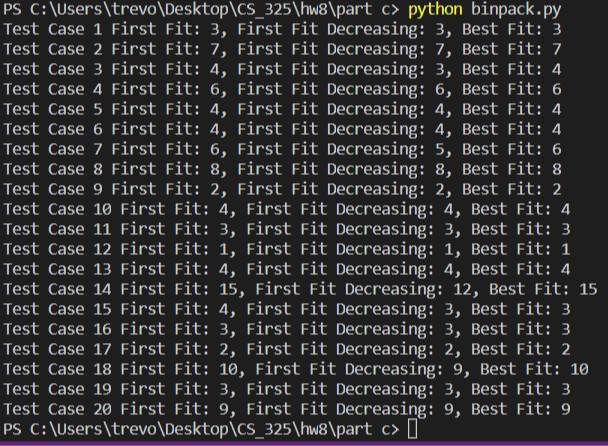
c) Randomly generate at least 20 bin packing instances. Summarize the results for each algorithm. Which algorithm performs better? How often? Note: Submit a description of how the inputs were generated not the code used to produce the random inputs.

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.



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

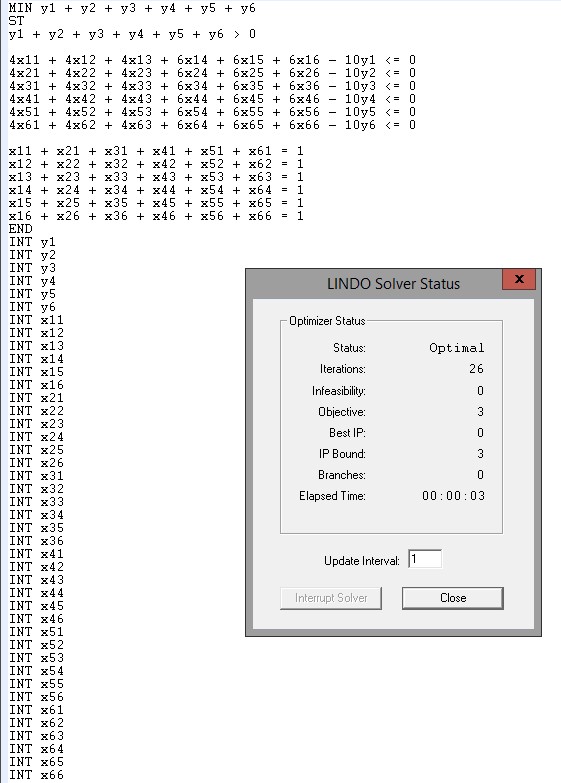
The best algorithm is First Fit Decreasing, one Test Case resulted in BF and FFD having the same better solution compared to FF which is what the .5 represents. Outside of this most Test Cases resulted in algorithms having same results. However, FFD sticks out as the best in that 4.5/20 times it was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

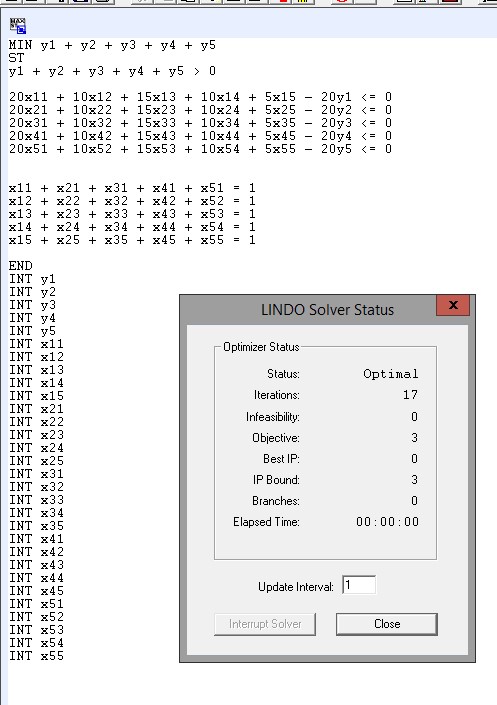
Write an integer program for each of the following instances of bin packing and solve with the software of your choice. Submit a copy of the code and interpret the results.

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



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



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.