Daniel Vu

CS 165

Project 2

Bin Packing Report

I – Introduction

Bin packing problem is one of commonly known combinatorial NP-hard problems. The idea is to pack n items of different weights into identical unit capacity bins, assuming that all items have weights smaller than bin capacity.

n items: $w_1, w_2, \ldots, w_n$ such that $0 \leq w_i \leq 1$ for $1 \leq i \leq n$ and bin capacity is 1.

The goal is to pack the items into minimum number of bins in best time-efficient way as possible. There are different algorithms for bin packing, each have their own advantages. These are the list of algorithms:

- Next fit
- First fit / First fit decreasing
- Best fit / Best fit decreasing

There is no known polynomial time algorithm for its solution, and it is conjectured that none exists. They have many applications, such as filling up containers, loading trucks with weight capacity constraints, job scheduling, creating file backups in media and technology, etc.

For this project, we implement five algorithms of bin packing problem and experiment the waste, W(A), as a function of n and as n grows towards infinity, for random items uniformly distributed in the interval (0,1). The waste of each algorithm is difference of the number of bins that the algorithm uses for contains all items and the total weights of all n items.

II – Bin packing Algorithms

1. Next-fit algorithm

We process each item, check if it fits the current bin or not. If so, place the item in the bin and process the next item, otherwise we use a new empty bin to contain the item.

Pseudocode:

```
for item in Items do
    if item is smaller free space of current bin
        add item into current bin
    else
        create new bin
        add item into current bin
```

Running time of next-fit is O(n). Next Fit uses at most 2M bins if M is optimal.

2. First-fit / First-fit decreasing algorithm

We process each item, check if it fits the first bin or not. If so, place the item in the bin and process the next item. Otherwise, keep going to next bin and check again until there is no bins that fits the item that we create a new bin, and place the item in the new bin.

Pseudocode:

```
for item in Items do
    for bin in Bins do
```

if item is smaller than free space of current bin

add item to bin

break

if no bin fits the item

create new bin

add item to bin

Running time of first-fit is $O(n^2)$. Next Fit uses at most 1.7M bins if M is optimal. First-fit decreasing algorithm will sort the items in descending order according to their weights and uses same pseudocode as first-fit.

First-fit and first-fit decreasing algorithm can be implemented with AVL tree that reduces running time to O(nlogn).

3. Best-fit/Best-fit decreasing algorithm

We process each item, check for all the bins that fit the item and choose the bin that will have the minimum free space after placing the item in that bin. If no bins found then create a new bin and place the item in the bin. Process to the next item.

Pseudocode:

for items in Items do

for bin in Bins do

if item is smaller than free space of current bin

record the remaining free space if placing item to current bin

if item can be placed in any bin

put the item in the bin with minimum remaining free space

else

create a new bin

add item to bin

Running time of best-fit is $O(n^2)$. Best-fit uses at most 1.7M bins if M is optimal. Best-fit decreasing algorithm will sort the items in descending order according to their weights and uses same pseudocode as best-fit.

Best-fit and best-fit decreasing algorithm can be implemented with AVL tree that reduces running time to O(nlogn).

III – Description of inputs and outputs

1. Random uniformly distributed algorithm

To create a random uniformly distributed set of input items given the size, we used Mersenne Twister pseudorandom number generator (mt19937) to get items with weights is double/float value in the interval (0,1).

2. Input Size

The minimum input size is 1 and the maximum input size is 10^7 (assuming it is infinity). In total, there are 8 input sizes: 1, 10, 100, 1000, 10000, 100000, 1000000 and 10000000.

3. Times running for algorithms

I run total of **five** times for each input and get the average waste W(A) of each algorithm for each input sizes. In general, there are 200 tests (8 input sizes * 5 times/size * 5 algorithms) and it takes around 26 hours in total to get final result.

4. Output

Next_fit	
Size	Waste
1	0.5724186
10	2.1414082
100	15.20004
1000	168.8362
10000	1677.294
100000	16623.64
1000000	166621
10000000	1667430

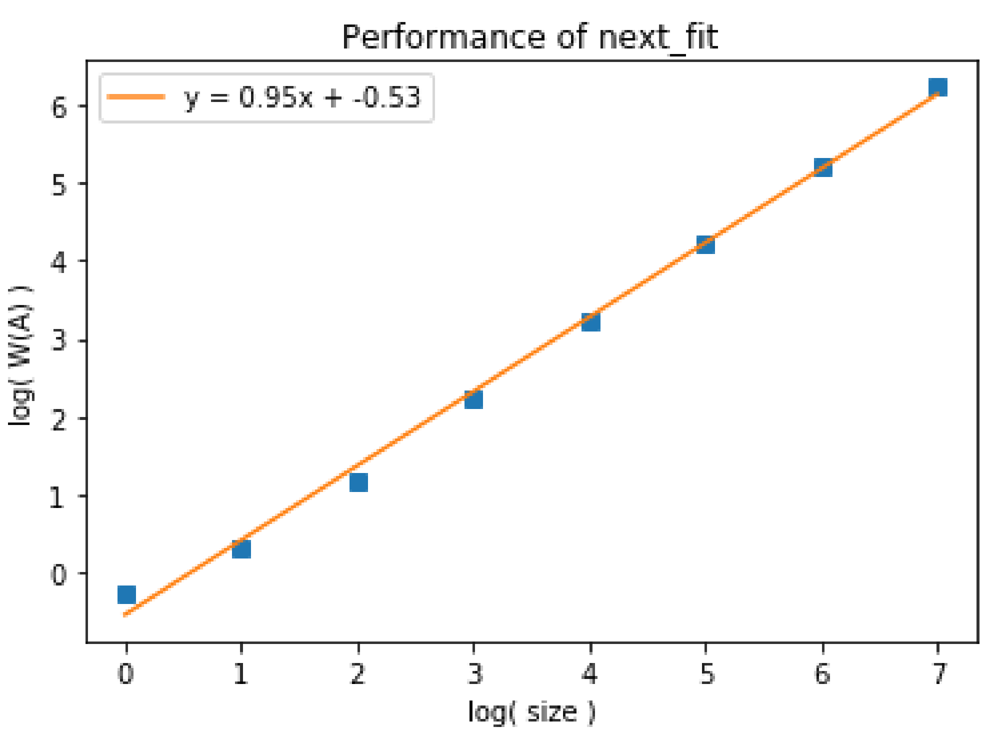
First_fit	
Size	Waste
1	0.5724186
10	1.1414094
100	5.00003
1000	30.63614
10000	143.6832
100000	688.8538
1000000	3432.93
10000000	14602.12

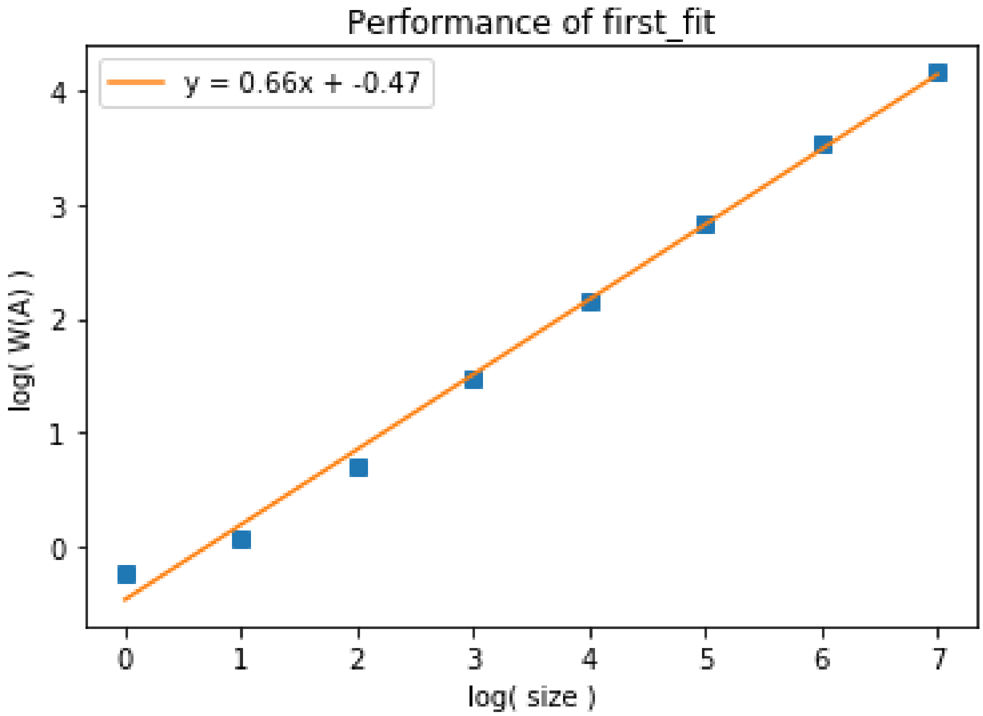
Best_fit	
Size	Waste
1	0.5724186
10	1.1414094
100	4.60003
1000	23.23614
10000	87.29356
100000	345.4538
1000000	1331.93
10000000	4370.8818

First_fit_decreasing	
Size	Waste
1	0.5724816
10	0.9414098
100	2.40003
1000	10.236146
10000	33.29366
100000	46.65376
1000000	165.117
10000000	457.73517

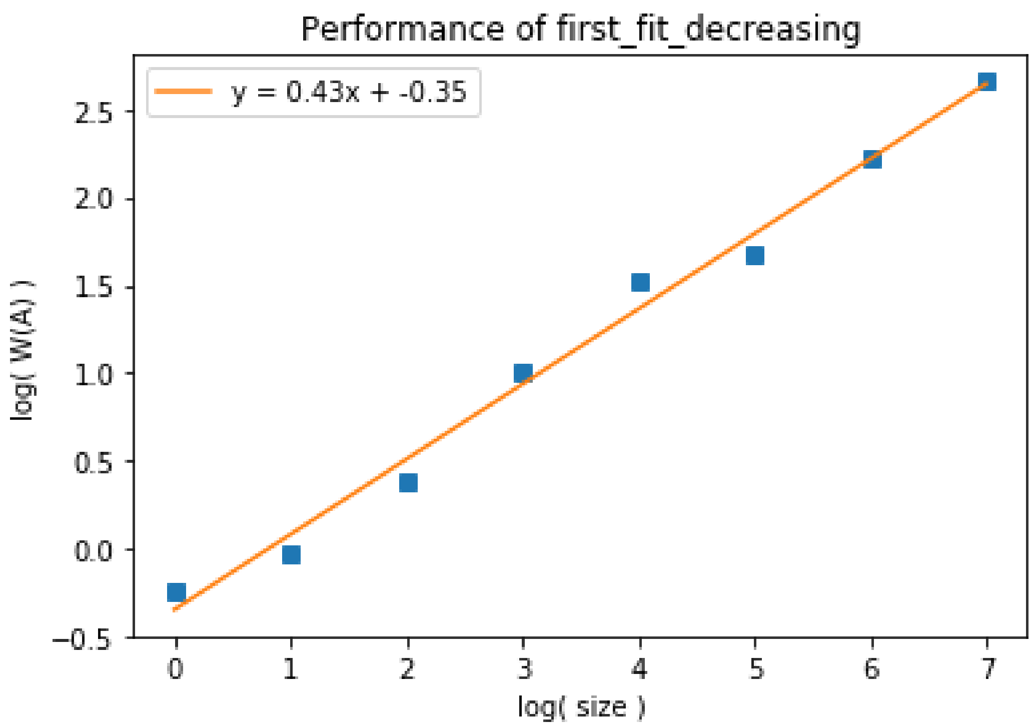
Best_fit_decreasing	
Size	Waste
1	0.5724816
10	0.9414098
100	2.40003
1000	10.236146
10000	33.29366
100000	46.65376
1000000	165.117
10000000	457.73517

These are the log-log plot for each algorithm and their regression slopes included.

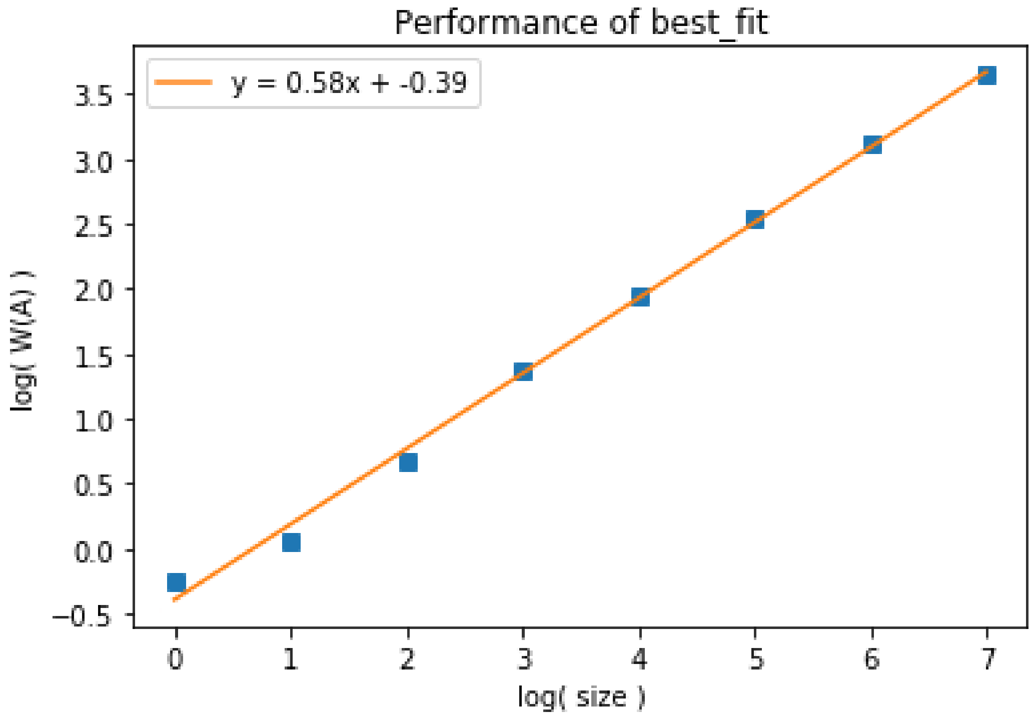


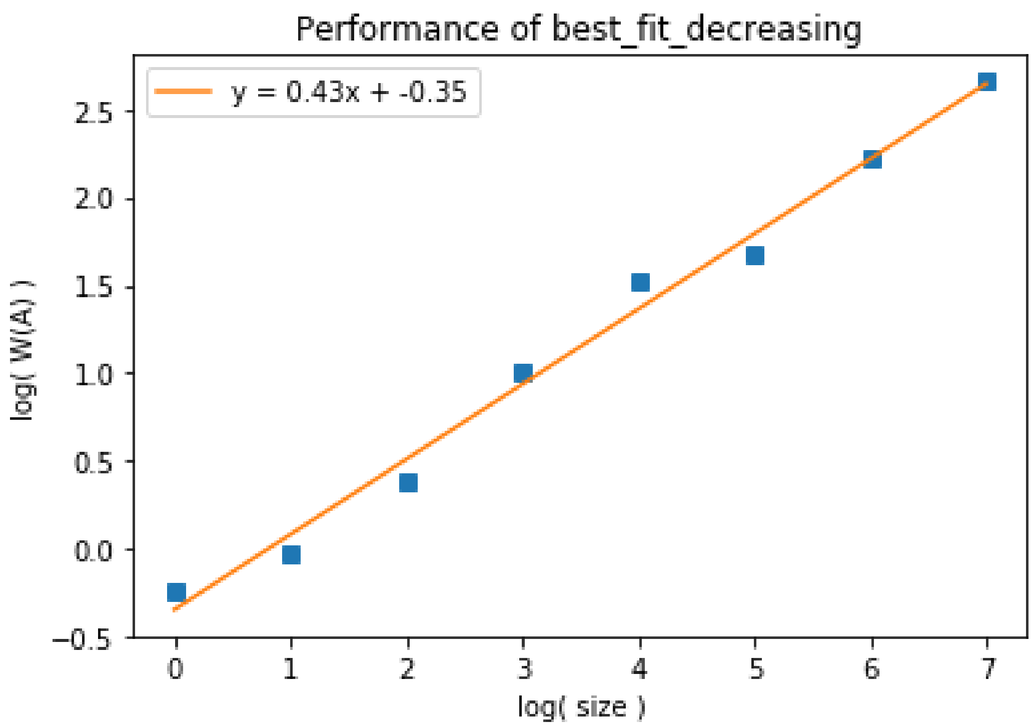


Performance of first_fit
y = 0.66x + -0.47
log(W(A))
log(size)



Performance of first_fit_decreasing
y = 0.43x + -0.35
log(W(A))
log(size)


Performance of best_fit
y = 0.58x + -0.39
log(W(A))
log(size)
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
−0.5
0
1
2
3
4
5
6
7


Performance of best_fit_decreasing
y = 0.43x + -0.35
log(W(A))
log(size)
2.5
2.0
1.5
1.0
0.5
0.0
−0.5
0
1
2
3
4
5
6
7

IV – Conclusion

The reason for doing log-log plot for each algorithm is because we assume that the waste has polynomial growth on increasing numbers of size from 1 to N; therefore, finding a regression line to log-log plot will show the empirical growth of the waste for the purpose of comparing five bin packing problem algorithms.

Regression slopes for each algorithm:

Next-fit:	0.95
First-fit:	0.66
First-fit-decreasing:	0.43
Best-fit:	0.48
Best-fit-decreasing:	0.43

1. **Analysis**

If only comparing the algorithms that are not using the descending-sorting method, next-fit holds the largest regression slope (0.95), while best-fit has the smallest regression slope (0.48). Increasing number of sizes, best-fit has the smaller wastes then next-fit and first-fit.

In the other hand, with descending-sorting method, first-fit-decreasing and best-fit-decreasing both hold the same smallest regression slopes (0.43) compared among all five algorithms. In my experiment, I recognized that with the same set of items, first-fit-decreasing and best-fit-decreasing will create the same waste, number of bins and free space left, but the assignments of items in those bins are different.

2. **Conclusion**

My winner algorithm is best-fit-decreasing algorithm because it has the smallest regression slope in general, and in no sorting category, best-fit still has the smallest regression slope compared to next-fit and first-fit algorithms.