Experimentally Comparing the Performance of Various Approximation Algorithms for the Bin Packing Problem

Johnny Ceja
COMPSCI 165
University of California, Irvine

Prologue

Introduction

Here we experimentally measure the efficacy of various approximation algorithms for the bin packing problem. We use the *waste* function to determine the efficacy of an approximation algorithm. The *waste* of an approximation algorithm is defined as the number of bins used for an input minus the total space used among all of these bins. We experimentally derive a function for the waste for each of the algorithms used to determine the "best" algorithm. The five approximation algorithms used for the bin packing problem are Next Fit (NF), First Fit (FF), Best Fit (BF), First Fit Decreasing (FFD), and Best Fit Decreasing (BFD). We test these algorithms on varying input sizes, reaching a size of up to 200 thousand elements. The waste for each algorithm for each input size was calculated a total of 64 times and averaged to minimize variation.

The bin packing problem involves fitting an array of N elements within the range of real numbers [0, 1] into bins of size precisely 1. The goal of the problem is to minimize the amount of bins used while making sure every one of the N elements is in a bin. This problem is NP-hard, that is, the only known fully correct solution (non-approximation algorithm) is the naive algorithm. The naive algorithm exhaustively tests every possible configuration to find the minimum number of bins needed, as such, the time complexity of the naive algorithm is $O(2^n)$

where n is the size of the input array. The naive algorithm is impractical for anything other than very small input sizes, which is why approximation algorithms are used.

Test Machine and Environment

The algorithms have been executed on a shared server owned by the University of California, Irvine. The specific machine used is named “Odin” and has a four 16-core AMD Opteron 6378 CPUs @ 2.93 GHZ, for a total of 64-cores on the server. It has 512 GB of RAM and is running on a 64-bit CentOS Linux distribution. Due to the nature of this experiment, the time it takes for one of the algorithms to run to completion is not very important. Nonetheless, each algorithm was ran on its own dedicated core.

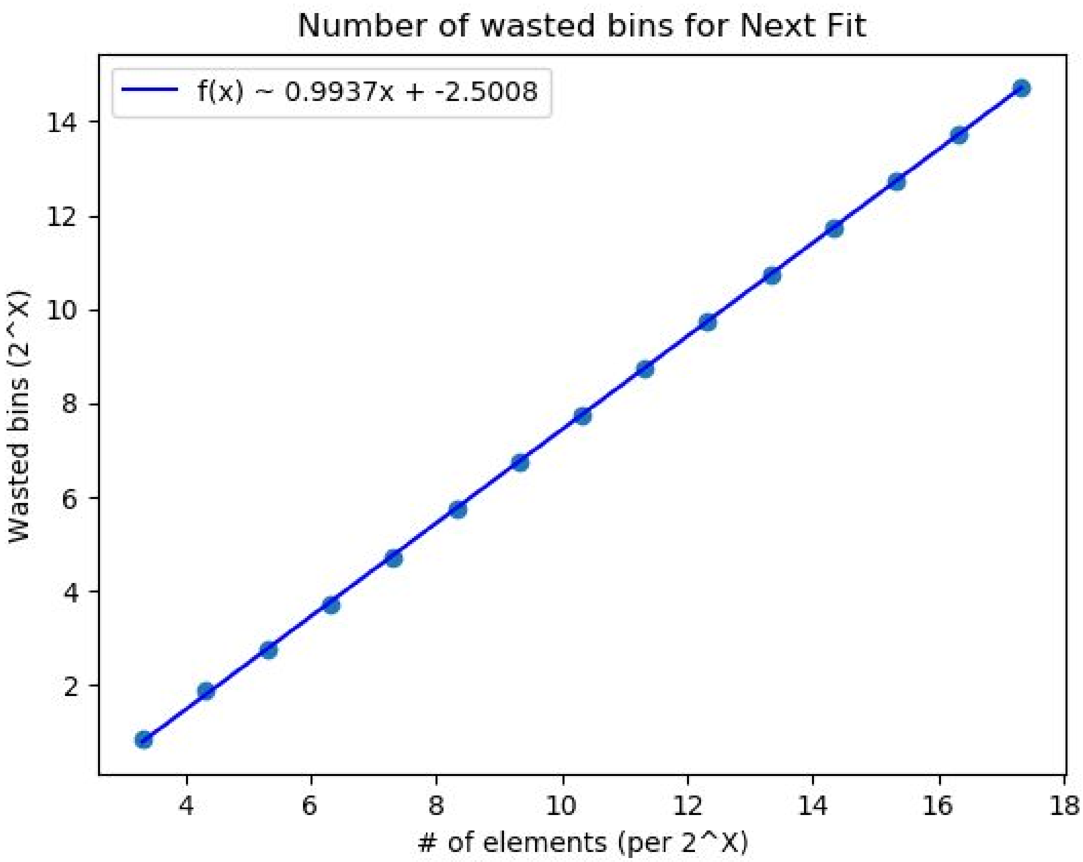
Results

Next Fit (NF)

```
func nextFit(input):
    free_space = []
    pos = 0
    for every element in input:
        for every space in free_space starting at pos:
            find first occurrence of free space for element
            insert element into free space
            pos = position inserted into
        if no free space found. insert at the end of free_space
            pos = end of free_space
    return free_space
```

The Next Fit approximation algorithm works by traversing the array looking for a bin that fits the element in question starting at the same position that the most recent bin insertion was.

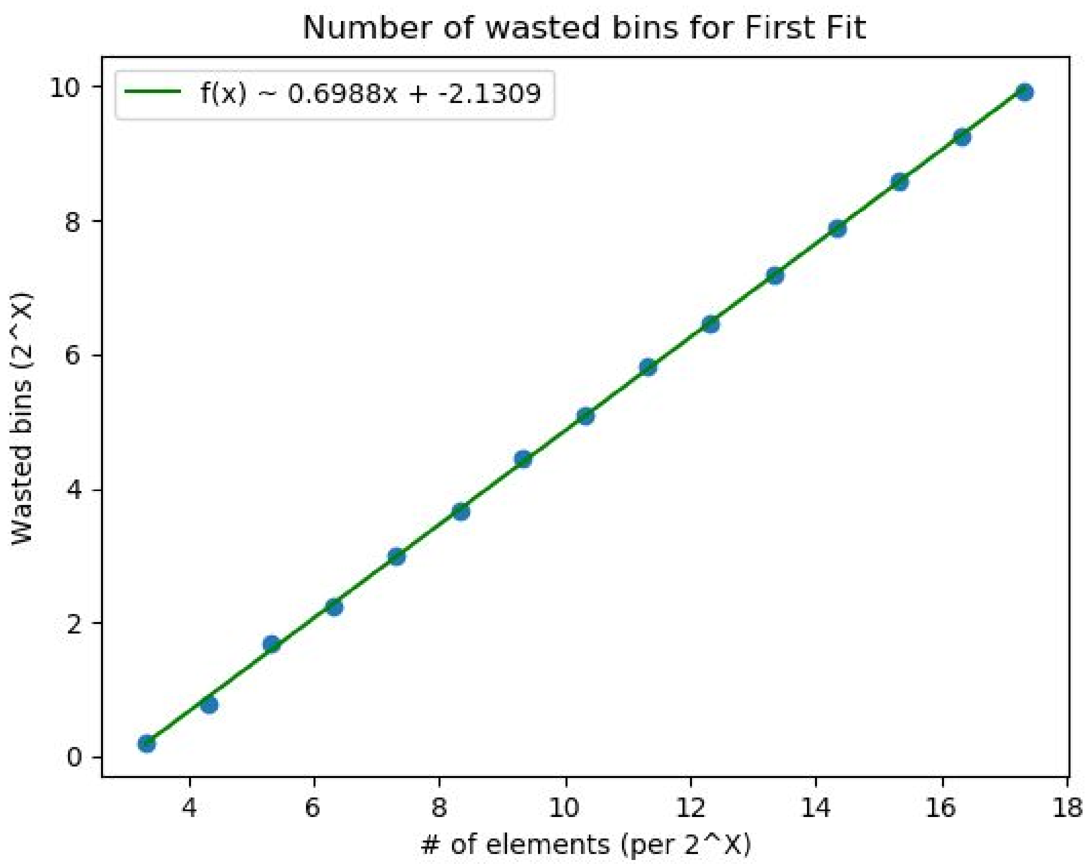
The number of wasted bins as a function of input size on a log-log scale is shown below:



First Fit (FF)

func firstFit(input):
free_space = []
for every element in input:
for every space in free_space:
find first occurrence of a free space for element
insert element into free space
if no free space found, insert at the end of free_space
return free_space

The First Fit approximation algorithm works by finding the first bin (from left to right) that is able to fit the element in question. This is done for each element in the input array. Note that the given implementation is the naive approach which takes $O(n^2)$. There exist a $O(n\ log(n))$ approach that uses a balanced search tree. The number of wasted bins as a function of input size on a log-log scale is shown below:

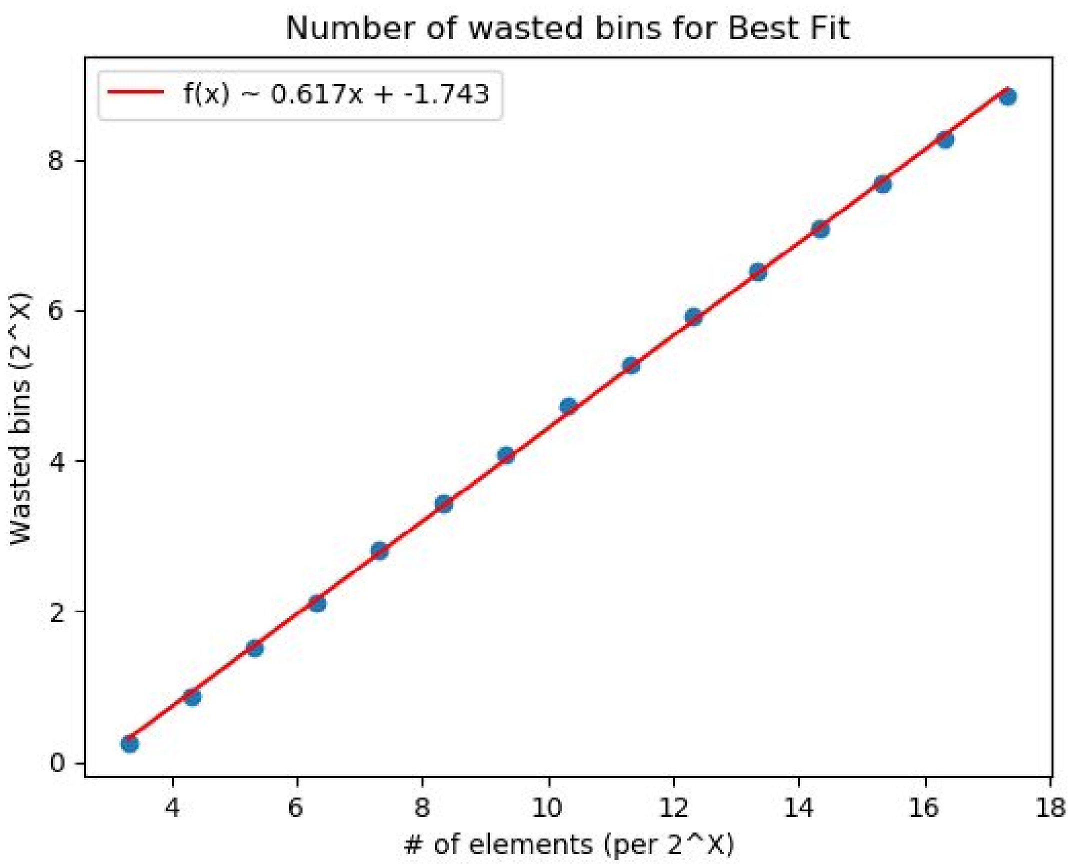


Best Fit (BF)

func bestFit(input):
free_space = []
pos = 0
for every element in input:
bestSpace = INFINITY
for every space in free_space:
if space has enough size for element:
bestSpace = minimum(bestSpace, space)
if bestSpace < INFINITY:
insert element into bestSpace
else insert element into new bin at end of free_space

return free_space

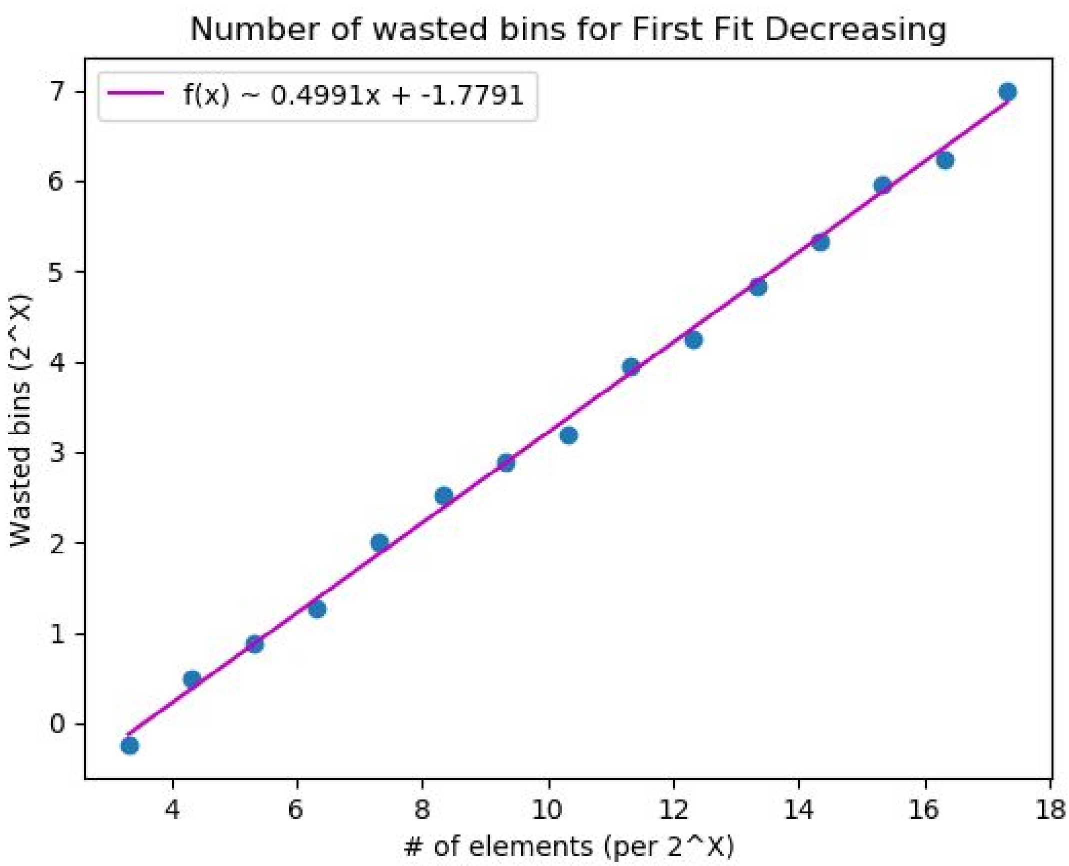
The Best Fit approximation algorithm works by finding the bin that minimizes the free space available on that bin should the element in question be inserted. The number of wasted bins as a function of input size on a log-log scale is shown below:



First Fit Decreasing

func firstFitDecreasing(input):
sort input in decreasing order
return firstFit(input)

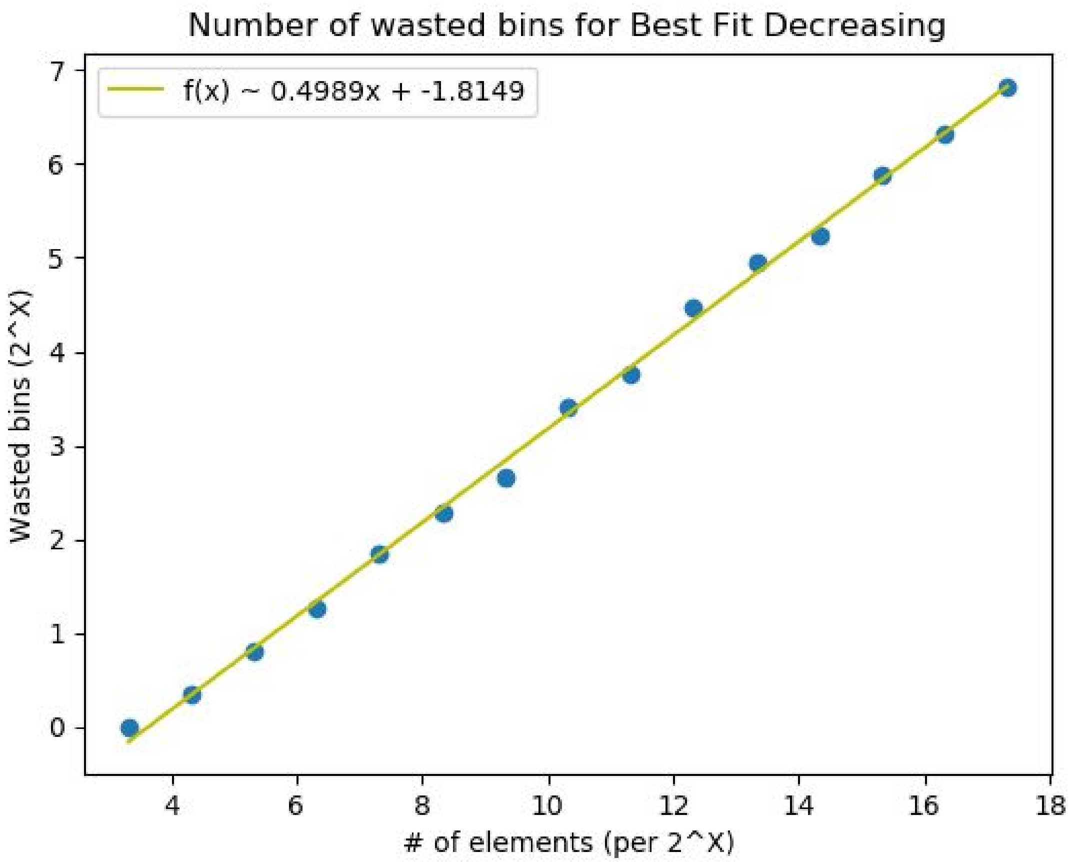
The First Fit Decreasing approximation algorithm works by first sorting the input array in decreasing order and then passing this into the First Fit approximation algorithm as an input. The number of wasted bins as a function of input size on a log-log scale is shown below:



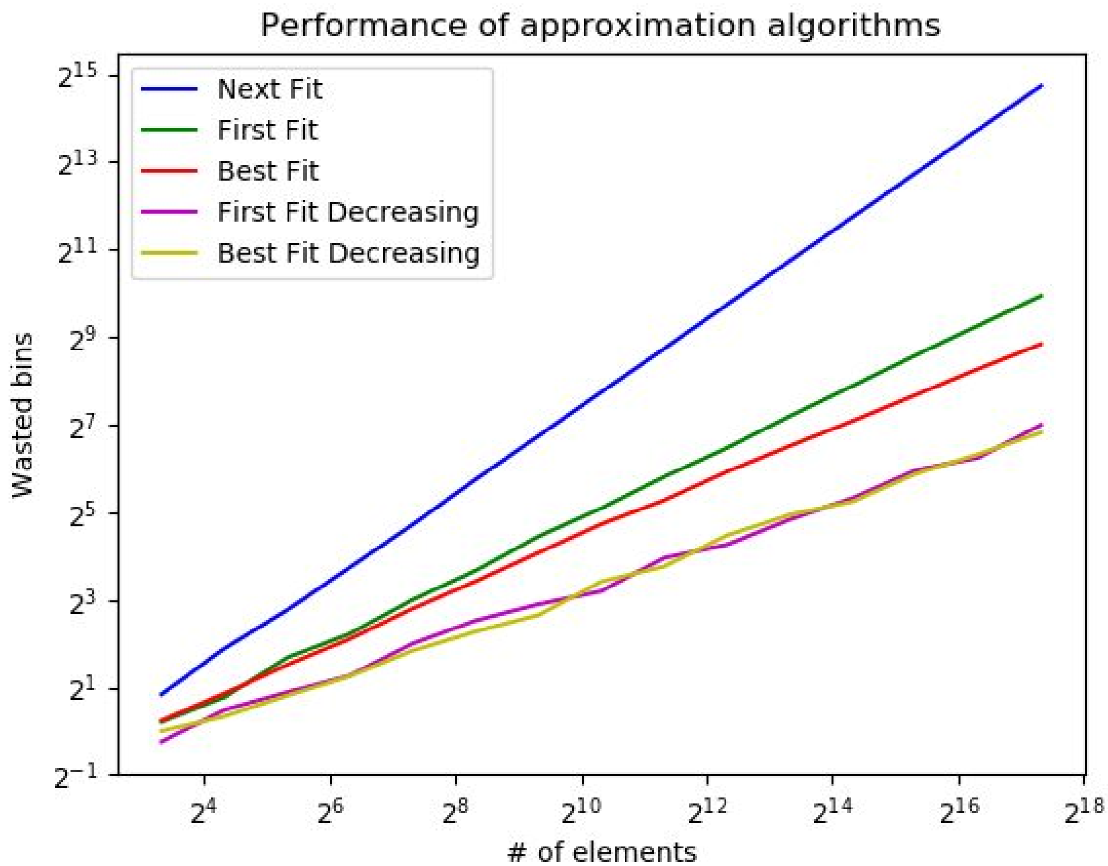
Best Fit Decreasing

func bestFitDecreasing(input):
    sort input in decreasing order
    return bestFit(input)

The Best Fit Decreasing algorithm works by first sorting the input array in decreasing order and then passing this into the Best Fit approximation algorithm as input. The number of wasted bins as a function of input size on a log-log scale is shown below. Upon visual inspection, the graph does roughly appear to be a linear line.



Conclusion



The *Best Fit Decreasing* and *First Fit Decreasing* approximation algorithms seem to be the **best** of the ones tested. The regression line slope coefficient for these algorithms are 0.4989 and 0.4991 respectively. What these numbers mean is that an input size n times larger than some baseline input size, will make the corresponding increase in the number of wasted bins be equal to $c * n$ where c is the slope coefficient. A lower c coefficient is ideal as this determines the rate of growth of the waste function as a function of input size. In the worst case, a new bin is created for every element in the input vector, this corresponds with a c coefficient of 1 which is close to coefficient of the Next Fit approximation algorithm, 0.9937. This means that the **worst** approximation algorithm as determined by the waste function is *Next Fit* as this is very close to the worst case scenario mentioned above.