Lachlan Sinclair

8/16/2019

Homework 8

Problem 1:

a)

Psuedo code:

First fit:

```
Struct bin(properties: int capacity, int currentWeight)
Weights[]: array containing the weights of items
Bins[]: array of open bin structs
Int number of open bins
For i:0 to number of items
        For j:0 to number of open bins
                If weight[i] +bins[j].currentWeight<=bins[j].capacity
                        bins[j].currentWeight+= weight[i]
                else if j=number of open bins -1
                        open new bin (add bin to bins array)
return number of open bins
```

First fit decreasing:

```
Struct bin(properties: int capacity, int currentWeight)
Weights[]: array containing the weights of items
Mergesort Weights by decreasing weight
Bins[]: array of open bin structs
Int number of open bins
For i:0 to number of items
        For j:0 to number of open bins
                If weight[i] +bins[j].currentWeight<=bins[j].capacity
                        bins[j].currentWeight+= weight[i]
                else if j=number of open bins -1
```

```
                    open new bin (add bin to bins array)
return number of open bins
```

Best fit:

```
Struct bin(properties: int capacity, int currentWeight)
Weights[]: array containing the weights of items
Bins[]: array of open bin structs
Int best bin = -1
Int best cap = 0
For i:0 to number of items
    For j:0 to number of open bins
        If weight[i] +bins[j].currentWeight<=bins[j].capacity
            If weight[i] + bins[j].currentWeight > best cap
                Bestcap = weight[i] + bins[j].currentWeight
                Best bin = j
    if best bin = -1
        open new bin (add bin to bins array)
        bins[num of open bins].currentWeight = weight[i]
    else
        bins[bestbin].currentWeight+= weight[i]
    best bin = -1
    best cap =0
return number of open bins
```

Runtimes:

First Fit: $O(n^2)$ runtime. The worse case is when every item fills an entire bin, when this occurs every item will need to search through a number of bins equal to all previously added items which causes a total of n^2 operations.

First fit decreasing: $O(n^2)$ runtime. Merge sort take nlgn time. After we add the runtime for first fit, which as previously discussed for first fit is n^2 which dominates the nlgn runtime of merge sot.

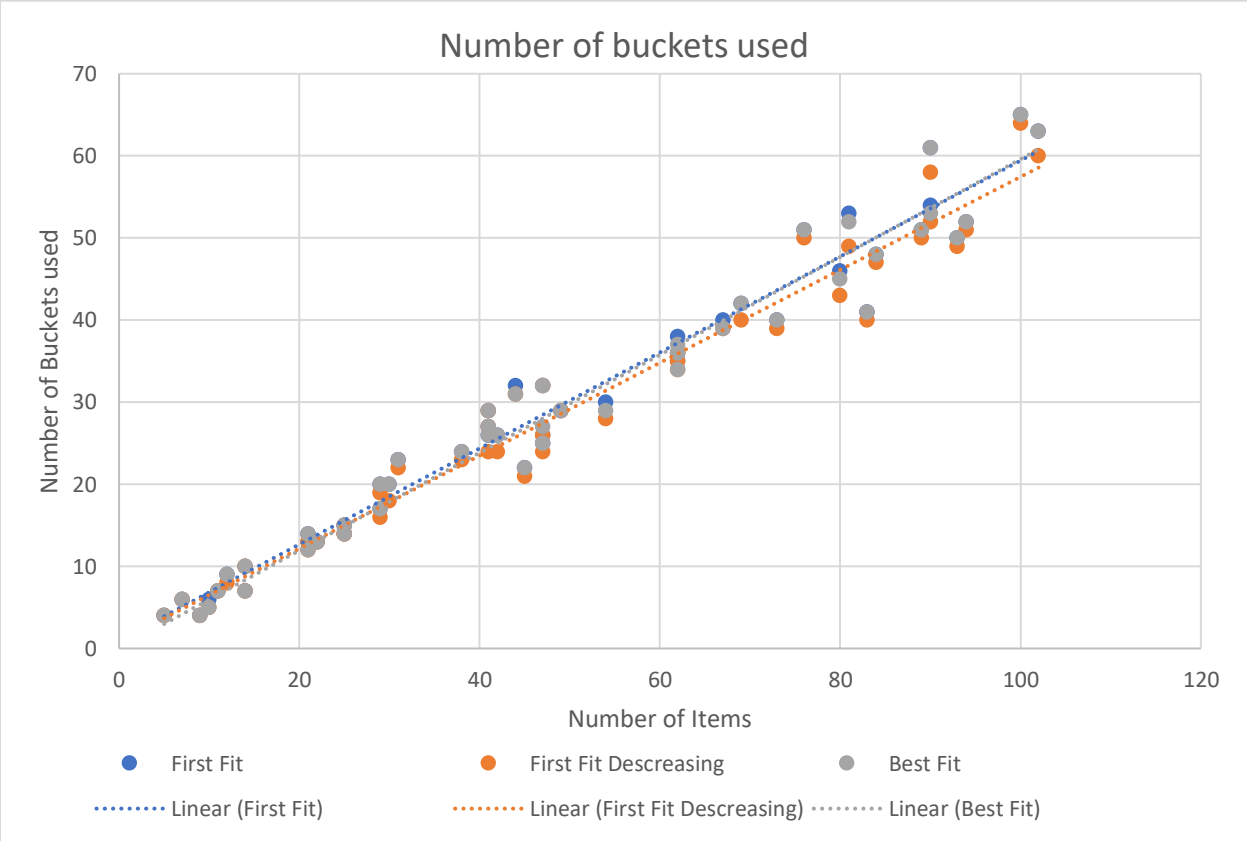
Best fit: O(n^2) runtime. Similar to first fit this is bound by the case if which every item fills an entire bin, which causes n^2 operations. Best fit always checks every bin, but this has no impact on its worse case runtime.

c)

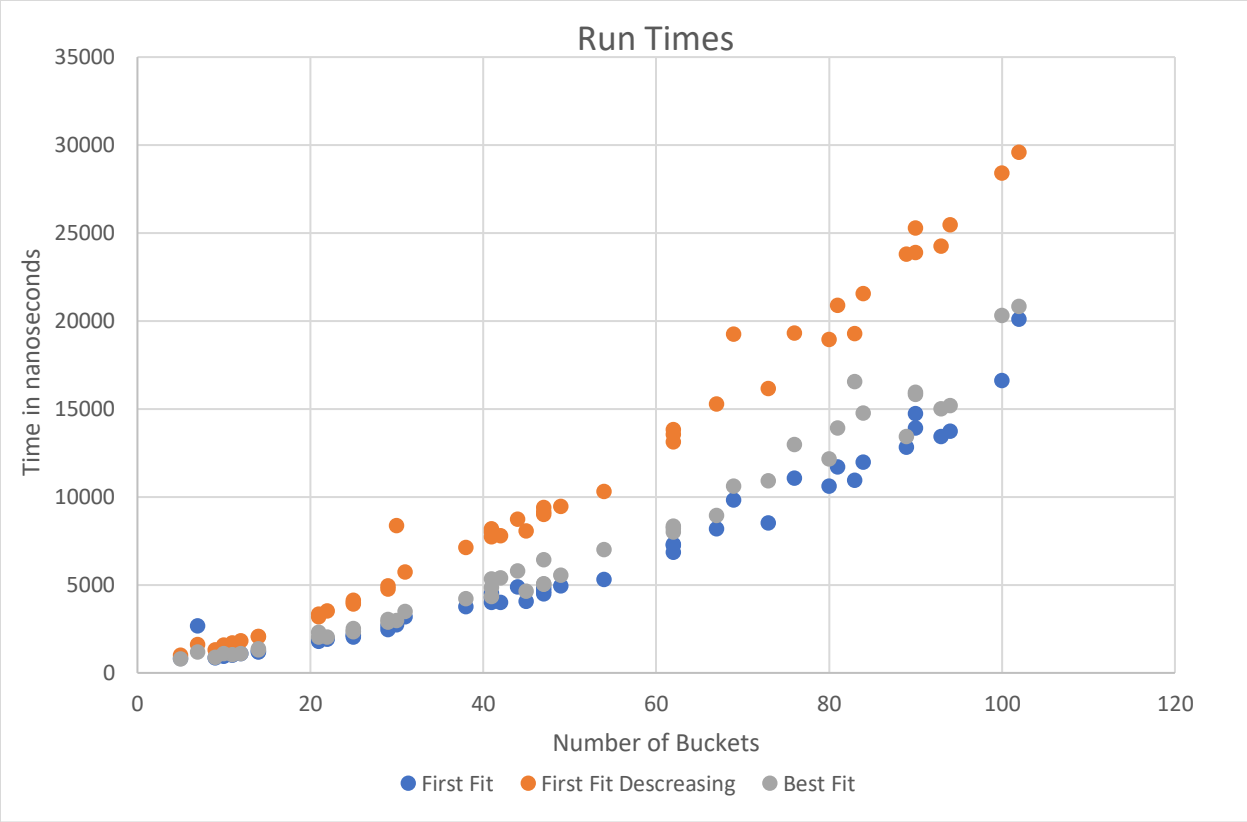
Description:

To generate the random input for the algorithms I created a double pointer to items, this contained 50 pointers to item arrays. I then generated a random number between 5 and 110 to use as the number of items for each of the 50 arrays. I then looped through all of the items for each sub array and assigned random values between 1-10 as the item's weight. I choose to use bins with a weight capacity of 10.

i. First fit decreasing consistently had the least number of buckets required, this by no means was always the case. Certain ordering of the arrays clearly favored specific approximation algorithms.



ii. First Fit performed the best in terms of runtime. It out performs best fit since it doesn't always need to search every bucket. And its quicker than decreasing best fit since it doesn't have to perform a sort.


Run Times
35000
30000
25000
20000
15000
10000
5000
0
Time in nanoseconds
0
20
40
60
80
100
120
Number of Buckets
First Fit
First Fit Descreasing
Best Fit