Junhyeok Jeong

CS 325 – 400 F2019

December 4, 2019

Homework 8

1.
a)
Pseudo code of First-Fit:

```
def first_fit (bin_capacity, item_wegihts []):
#Initialize bin array with bin capacity
    bin_ar = [bin_cacpacity]
    for (index = 0 to the number of items) {
        j = 0
        while (j =0 < size of bin_ar):
            if bin_ar[j] – item_weights[index] >= 0:
                bin_ar[j] = bin_ar[j] – item_weights[index]
                break
            else:
                if size of bin_ar – 1 == j:
                    bin_ar.append(bin_capacity)
            j += 1
    return len(bin_ar)
```

The running time of First-Fit algorithm:
There is a nested loop (outer loop: for, inner loop: while), so the worst case of running time is $O(n^2)$ when input values are close to maximum. Therefore, running time is $O(n^2)$ or $\Theta(n^2)$.

Pseudo code of First-Fit-Decreasing:

```
#define merge_sort for making descending ordered item array
def merge_sort(ar):
    if size of ar > 1:
        mid = size of ar // 2
        left = ar[:mid]
        right = ar[mid:]

        merge_sort(left)
        merge_sort(right)
        i = j = k

        while i < size of len and j < size of right:
            if left[i] > right[j]:
```

```
                ar[k] = left[i]
                i += 1
            else:
                ar[k] = right[j]
                j += 1
            j += 1
        while i < len(left):
            ar[k] = left[i]
            k += 1
            i += 1
        while i < len(right):
            ar[k] = right[j]
            k += 1
            j += 1

def first_fit_decreasing (bin_capacity, item_wegihts []):
    sorted_items = []
        for i in item_weights:
            sorted_items.append(i)
    merge_sort(sorted_items)

    #After sorting the item_weights in descending order, then do same algorithm with the
First_Fit
    return first_fit(bin_capacity, sorted_items)
```

The running time of First-Fit-Decreasing algorithm:
At first, the merge sort has O(nlogn) average running time and the above First-Fit algorithm has $O(n^2)$ running time because of a nested loop. Therefore, the running time of First-Fit-Decreasing algorithm is $O(nlogn) + O(n^2) = O(n^2)$.

Pseudo code of Best-Fit:

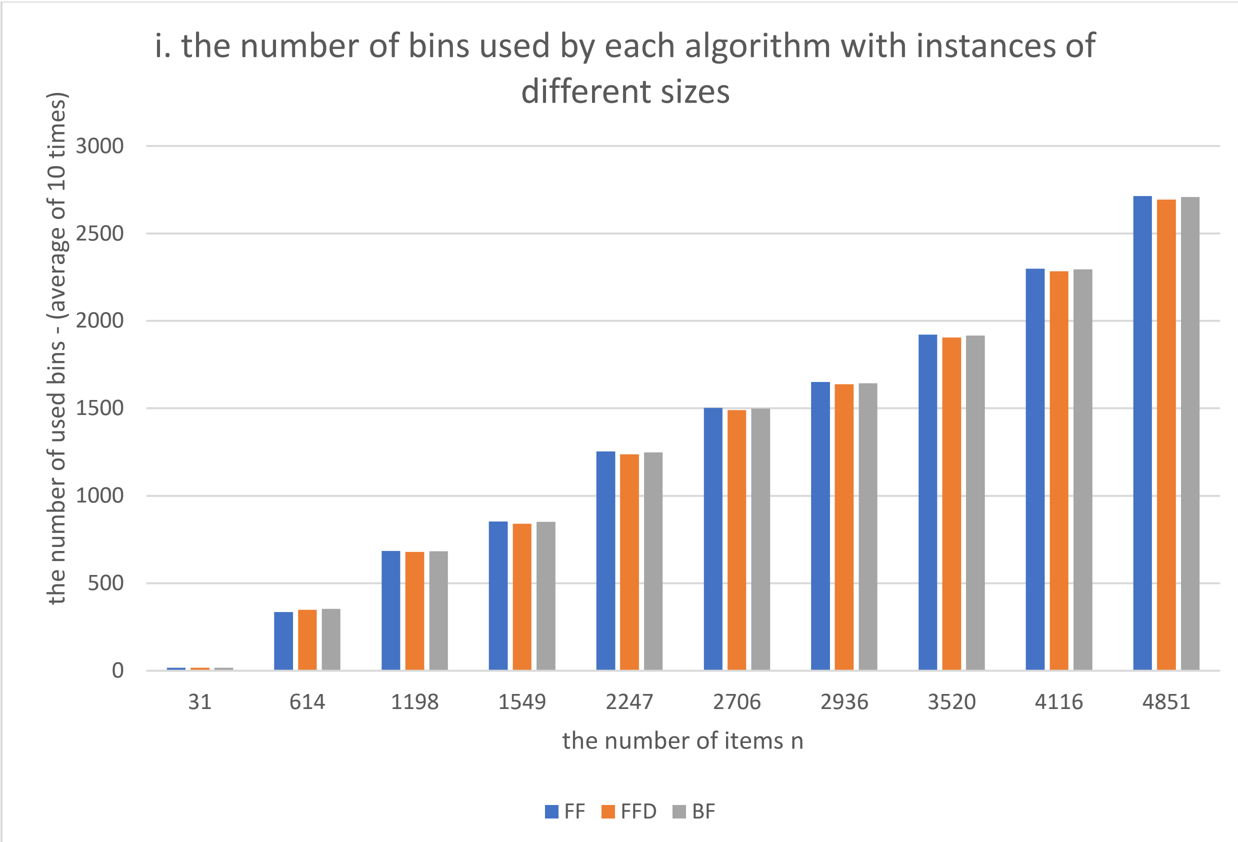
```
def best_fit (bin_capacity, item_weights []):
#Initialize bin array with bin capacity as much as the number of items
    bin_ar = [bin_cacpacity] * size of item_weights
#set result value which means the number of used bins in the bin_ar
    Num_nonempty_bin = 0

for (i to size of item_weights):
    bin_index = 0  #when loop finds the minimum-leftover space bin, use this index for
deducting space as much as weight of the indexed item
    minimum = bin_cacapcity            #indicator for minimum space
    j = 0

    while j < num_nonempty_bin:
        if bin_ar[j] – item_weights[i] >= 0 and bin_ar[j] – item_weights[i] < minimum:
            minimum = bin_ar[j] – item_weights[i]
```

```
                bin_index = j
            j += 1

# a case for there is no enough space for indexed item (go to new bin)
    if minimum == bin_capacity:
        bin_ar[num_nonempty_bin] -= item_weights[i]
        num_nonempty_bin += 1

# if there is enough space for indexed item, then deduction
    else:
        bin_ar[bin_index] -= intem_weights[i]
return num_nonempty_bin
```

The running time of Best-Fit algorithm:
Like the above First-Fit algorithm, I used a nested loop (outer loop: for, inner loop: while). However, I approached differently because I initialized the bin array with the possible maximum number (the number of items) of bins and then outputted the number of used bins from the bin array because of looping to find minimum space bins. Anyway, the running time is $O(n^2)$ or $\Theta(n^2)$ because of the 2 for loops.

c)
To solve these problems, I used the rand library in Python for generating random number. I set the range of each item weights from 1 to 10 randomly because bin's capacity is 10. Furthermore, I set the range of the number of items (n) is from 20 to 5000 randomly.
i) The number of used bins is the y-axis and the number of items is the x-axis

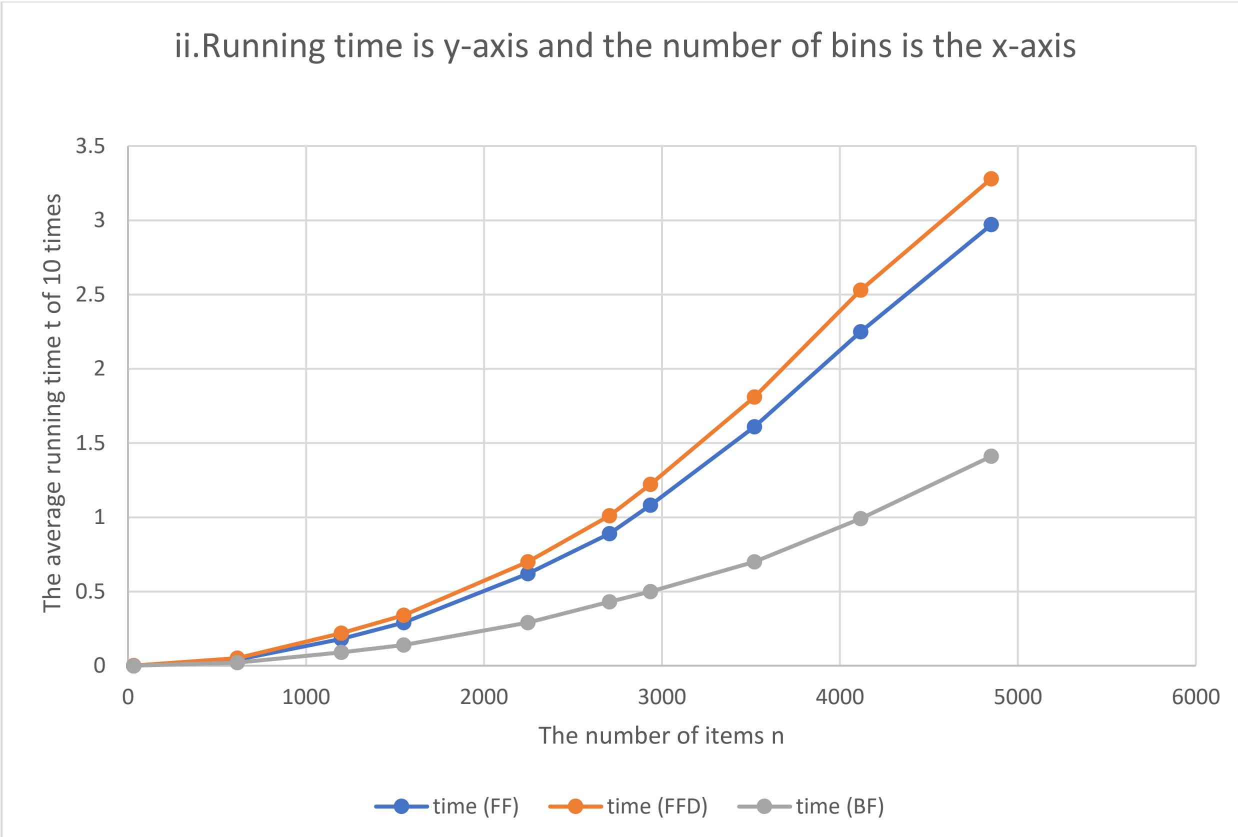
The number of used bins (average of 10 times run)				
num	FF	FFD	BF	
31	18	17	18	
614	335	349	353	
1198	685	680	683	
1549	853	841	852	
2247	1254	1238	1249	
2706	1503	1489	1499	
2936	1650	1638	1644	
3520	1922	1905	1916	
4116	2298	2284	2295	
4851	2714	2694	2708	



When I see the above chart, First-Fit-Decreasing showed the best performance among the three algorithms because FFD usually has the least used bins, even though the differences are small.

ii) Running time is y-axis and the number of bins is the x-axis

num	time (FF)	time (FFD)	time (BF)
31	0.00011	0.00023	0.000078
614	0.04	0.05	0.02
1198	0.18	0.22	0.09
1549	0.29	0.34	0.14
2247	0.62	0.7	0.29
2706	0.89	1.01	0.43
2936	1.08	1.22	0.5
3520	1.61	1.81	0.7
4116	2.25	2.53	0.99
4851	2.97	3.28	1.41



Based on the above chart and data, the Best-Fit algorithm showed the best performance among the three algorithms because it usually performed in the half time of the other algorithms. Most of all, the graph shapes of all three algorithms indicate that they have the similar running time $O(n^2)$ or $O(n \log n)$.

2. Software: LINDO

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

LINDO - [C:\USERS\15419\ONEDRIVE\DESKTOP\HW8 2-a.ltx]

File Edit Solve Reports Window Help

```
MIN y1+y2+y3+y4+y5+y6
ST
        y1 + y2 + y3 + y4 + y5 + y6 >= 1
        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
        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

END
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
INT y1
INT y2
INT y3
INT y4
INT y5
INT y6
```

LINDO

File Edit Solve Reports Window Help

Reports Window

```
LP OPTIMUM FOUND AT STEP     43
OBJECTIVE VALUE =    3.00000000


NEW INTEGER SOLUTION OF     3.00000000      AT BRANCH      0 PIVOT      43
RE-INSTALLING BEST SOLUTION...

       OBJECTIVE FUNCTION VALUE

       1)      3.000000

 VARIABLE        VALUE          REDUCED COST
      X11         0.000000          0.000000
      X12         0.000000          0.000000
      X13         0.000000          0.000000
      X14         0.000000          0.000000
      X15         0.000000          0.000000
      X16         0.000000          0.000000
      X21         0.000000          0.000000
      X22         0.000000          0.000000
      X23         0.000000          0.000000
      X24         0.000000          0.000000
      X25         0.000000          0.000000
      X26         0.000000          0.000000
      X31         0.000000          0.000000
      X32         1.000000          0.000000
      X33         0.000000          0.000000
      X34         0.000000          0.000000
      X35         1.000000          0.000000
      X36         0.000000          0.000000
      X41         0.000000          0.000000
      X42         0.000000          0.000000
      X43         0.000000          0.000000
      X44         0.000000          0.000000
      X45         0.000000          0.000000
      X46         0.000000          0.000000
      X51         1.000000          0.000000
      X52         0.000000          0.000000
      X53         0.000000          0.000000
      X54         1.000000          0.000000
      X55         0.000000          0.000000
      X56         0.000000          0.000000
      X61         0.000000          0.000000
      X62         0.000000          0.000000
      X63         1.000000          0.000000
      X64         0.000000          0.000000
      X65         0.000000          0.000000
      X66         1.000000          0.000000
       Y1         0.000000          1.000000
       Y2         0.000000          1.000000
       Y3         1.000000          1.000000
       Y4         0.000000          1.000000
       Y5         1.000000          1.000000
       Y6         1.000000          1.000000
```

- Based on the above LINDO code and report screenshots, the number of used bins is 3 for item list S = {4, 4, 4, 6, 6, 6} when the bin capacity is 10 and lower bound is 4+4+4+6+6+6 = 30. The item filled the bin like this: bin 1: {S1, S4}, bin 2: {S2, S5}, and bin 3: {S3, S6} with 43 iterations for optimization.

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

LINDO - [C:\USERS\15419\ONEDRIVE\DESKTOP\HW8 2-b.ltx]

File Edit Solve Reports Window Help

```
MIN y1+y2+y3+y4+y5
ST
        y1 + y2 + y3 + y4 + y5 >= 1
        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
        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

END
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
INT y1
INT y2
INT y3
INT y4
INT y5
```

LINDO - [Reports Window]

File Edit Solve Reports Window Help

```
LP OPTIMUM FOUND AT STEP      22
OBJECTIVE VALUE =    3.00000000


NEW INTEGER SOLUTION OF    3.00000000     AT BRANCH      0 PIVOT      22
RE-INSTALLING BEST SOLUTION...

       OBJECTIVE FUNCTION VALUE

       1)      3.000000

 VARIABLE        VALUE          REDUCED COST
      X11         0.000000          0.000000
      X12         1.000000          0.000000
      X13         0.000000          0.000000
      X14         1.000000          0.000000
      X15         0.000000          0.000000
      X21         0.000000          0.000000
      X22         0.000000          0.000000
      X23         1.000000          0.000000
      X24         0.000000          0.000000
      X25         1.000000          0.000000
      X31         1.000000          0.000000
      X32         0.000000          0.000000
      X33         0.000000          0.000000
      X34         0.000000          0.000000
      X35         0.000000          0.000000
      X41         0.000000          0.000000
      X42         0.000000          0.000000
      X43         0.000000          0.000000
      X44         0.000000          0.000000
      X45         0.000000          0.000000
      X51         0.000000          0.000000
      X52         0.000000          0.000000
      X53         0.000000          0.000000
      X54         0.000000          0.000000
      X55         0.000000          0.000000
       Y1         1.000000          1.000000
       Y2         1.000000          1.000000
       Y3         1.000000          1.000000
       Y4         0.000000          1.000000
       Y5         0.000000          1.000000


      ROW   SLACK OR SURPLUS     DUAL PRICES
       2)         2.000000          0.000000
       3)         0.000000          0.000000
       4)         0.000000          0.000000
       5)         0.000000          0.000000
       6)         0.000000          0.000000
       7)         0.000000          0.000000
       8)         0.000000          0.000000
       9)         0.000000          0.000000
      10)         0.000000          0.000000
      11)         0.000000          0.000000
      12)         0.000000          0.000000

NO. ITERATIONS=      22
BRANCHES=    0 DETERM.=  1.000E    0
```

- Based on the above LINDO code and report screenshots, the number of used bins is 3 for the item list S = {20, 10, 15, 10, 5} when bin capacity is 20 and the lower bound is 20 + 10 + 15 + 10 + 5 = 60. The items filled like this: bin 1: {S1}, bin 2: {S2, S4} and bin 3: {S3, S5} with 22 iterations for optimization.