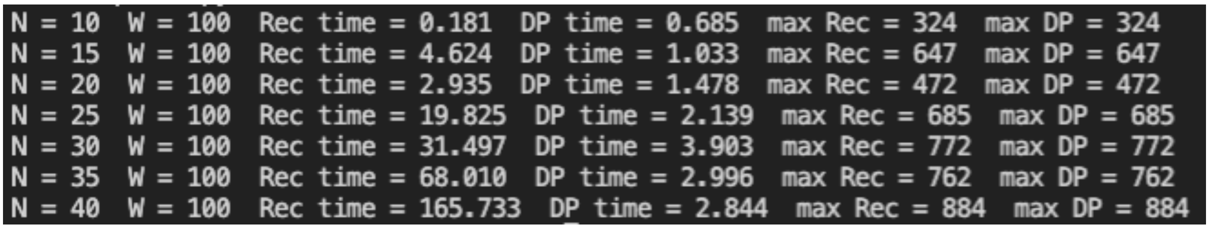
CS325-Homework2

Problem1:

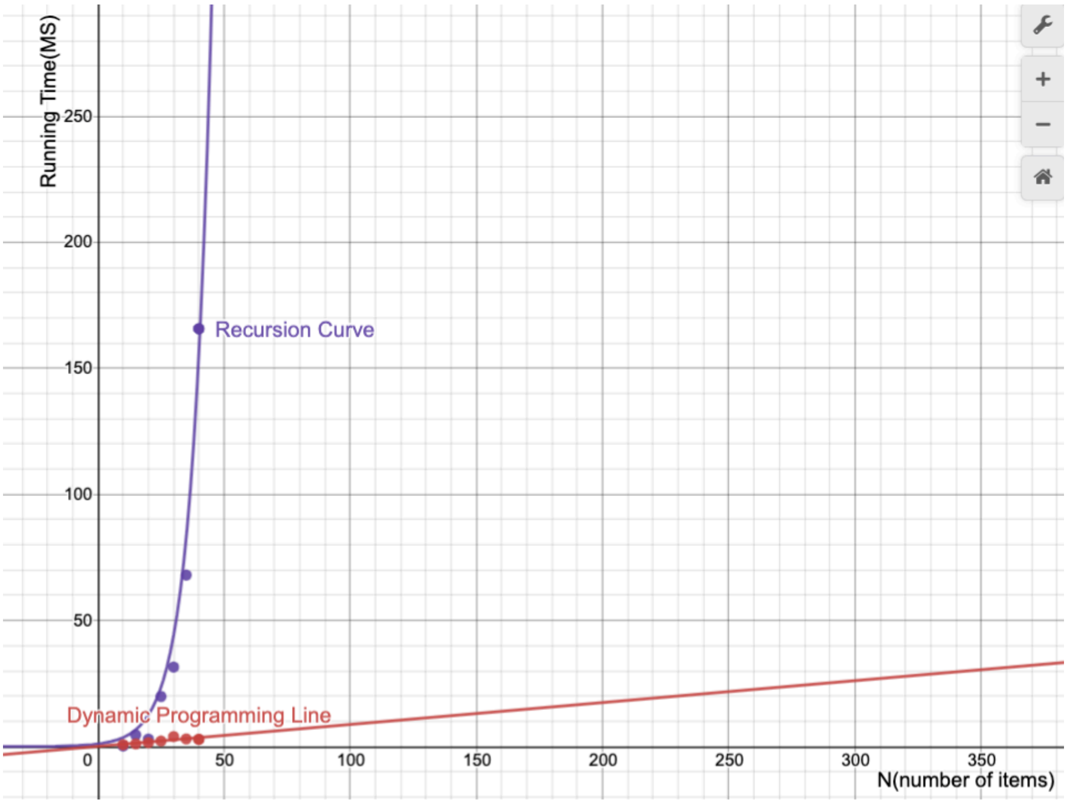
CASE1: Data: Constant W = 100, N = 10, 15, 20 …

Program result:



```
N = 10  W = 100  Rec time = 0.181  DP time = 0.685  max Rec = 324  max DP = 324
N = 15  W = 100  Rec time = 4.624  DP time = 1.033  max Rec = 647  max DP = 647
N = 20  W = 100  Rec time = 2.935  DP time = 1.478  max Rec = 472  max DP = 472
N = 25  W = 100  Rec time = 19.825  DP time = 2.139  max Rec = 685  max DP = 685
N = 30  W = 100  Rec time = 31.497  DP time = 3.903  max Rec = 772  max DP = 772
N = 35  W = 100  Rec time = 68.010  DP time = 2.996  max Rec = 762  max DP = 762
N = 40  W = 100  Rec time = 165.733  DP time = 2.844  max Rec = 884  max DP = 884
```

Plot: DP: Time – N and Rec: Time-N

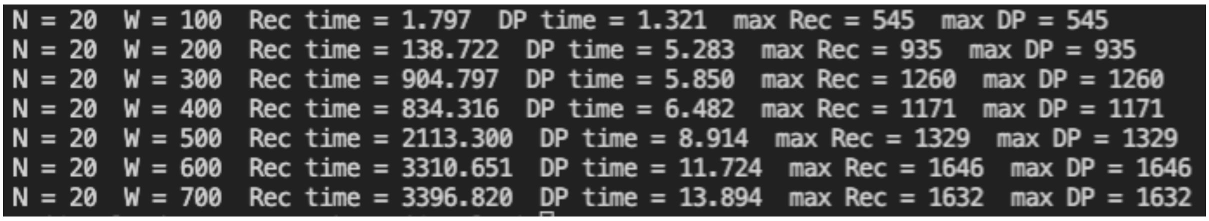


DP: y = 0.00087x*100

Rec: y = $2^{0.182x}$

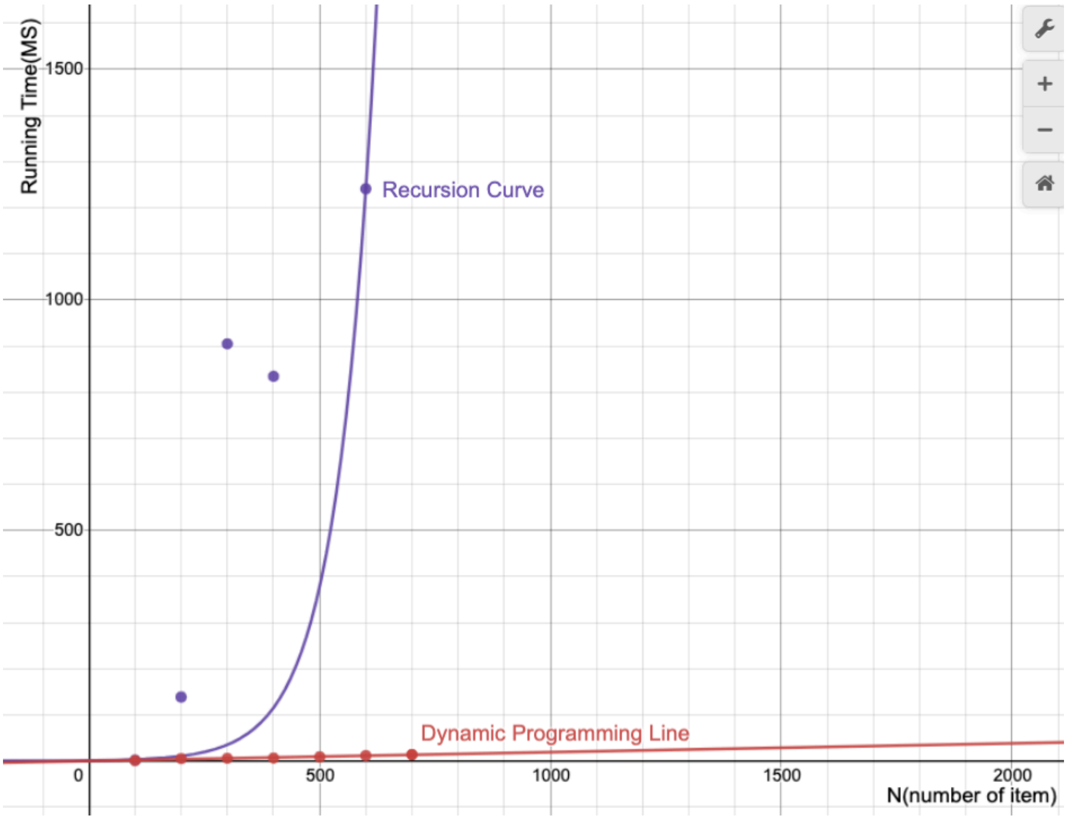
CASE2: Data: Constant N = 20, W = 100, 200, 300 …

Program result:



```
N = 20  W = 100  Rec time = 1.797  DP time = 1.321  max Rec = 545  max DP = 545
N = 20  W = 200  Rec time = 138.722  DP time = 5.283  max Rec = 935  max DP = 935
N = 20  W = 300  Rec time = 904.797  DP time = 5.850  max Rec = 1260  max DP = 1260
N = 20  W = 400  Rec time = 834.316  DP time = 6.482  max Rec = 1171  max DP = 1171
N = 20  W = 500  Rec time = 2113.300  DP time = 8.914  max Rec = 1329  max DP = 1329
N = 20  W = 600  Rec time = 3310.651  DP time = 11.724  max Rec = 1646  max DP = 1646
N = 20  W = 700  Rec time = 3396.820  DP time = 13.894  max Rec = 1632  max DP = 1632
```

Plot: DP: Time-W and Rec: Time-W

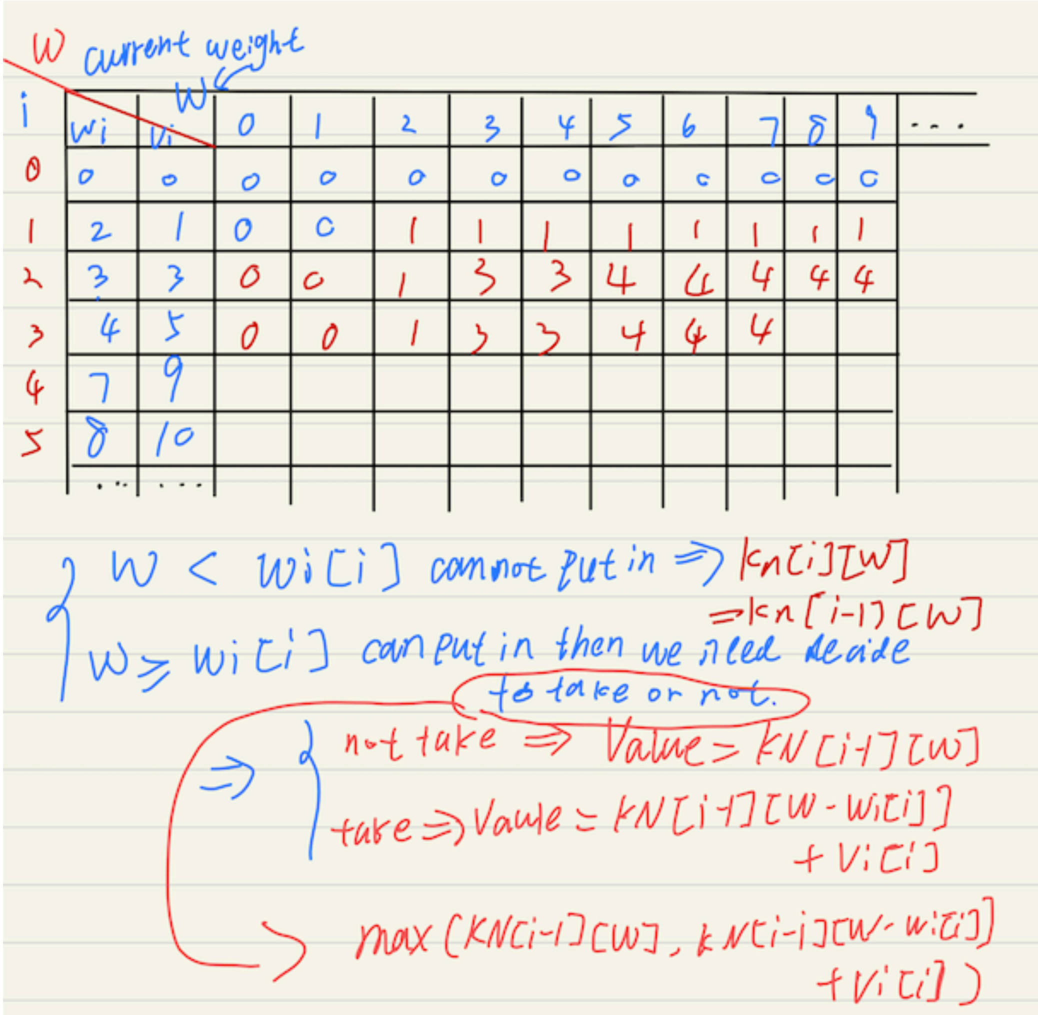


DP: y = 0.00096x*20

Rec: y = $2^{0.017x}$

C) For the code:

DP implementation:



W current weight

i	wi	vi	0	1	2	3	4	5	6	7	8	9	...
0	0	0	0	0	0	0	0	0	0	0	0	0	
1	2	1	0	0	1	1	1	1	1	1	1	1	
2	3	3	0	0	1	3	3	4	4	4	4	4	
3	4	5	0	0	1	3	3	4	4	4			
4	7	9											
5	8	10											
	...	...											

$W < wi[i]$ cannot put in => kn[i][W] => kn[i-1][W]

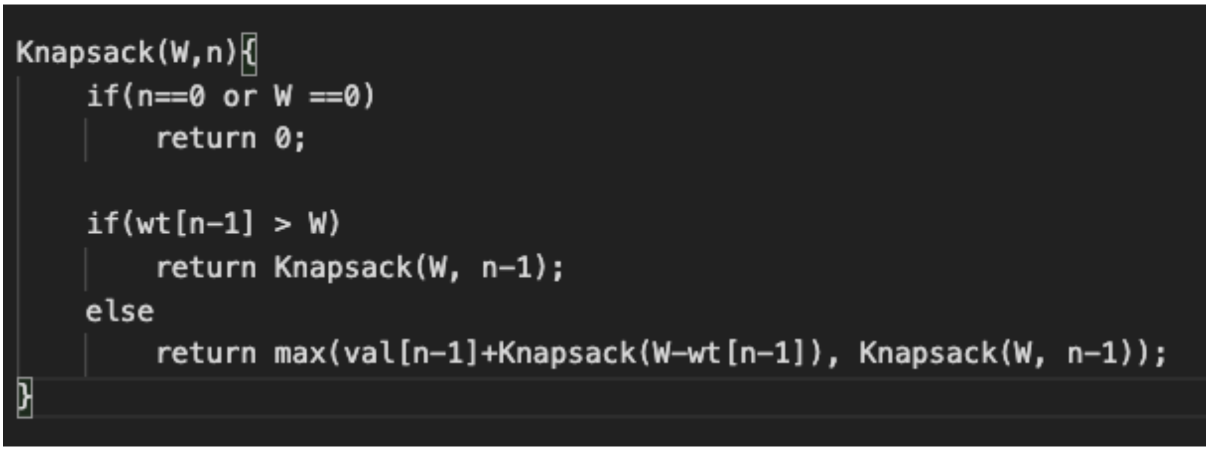
$W \geq wi[i]$ can put in then we need decide to take or not.

not take => Value = KN[i-1][W]

take => Vaule = KN[i-1][W-wi[i]] + Vi[i]

=> max(KN[i-1][W], kN[i-1][W-wi[i]] + Vi[i])

Rec implementation:



```
Knapsack(W,n){
    if(n==0 or W ==0)
        return 0;

    if(wt[n-1] > W)
        return Knapsack(W, n-1);
    else
        return max(val[n-1]+Knapsack(W-wt[n-1]), Knapsack(W, n-1));
}
```

For the val and wt list, I just use the function to generate n list with range 50-150 for val.

And n list with range 1-100 for wt. Then I just put the two sets of data into the two algorithms to start collecting time. I will repeat this process 7 times, but at the end of each loop I will judge the user's needs to decide whether to superimpose W or n.

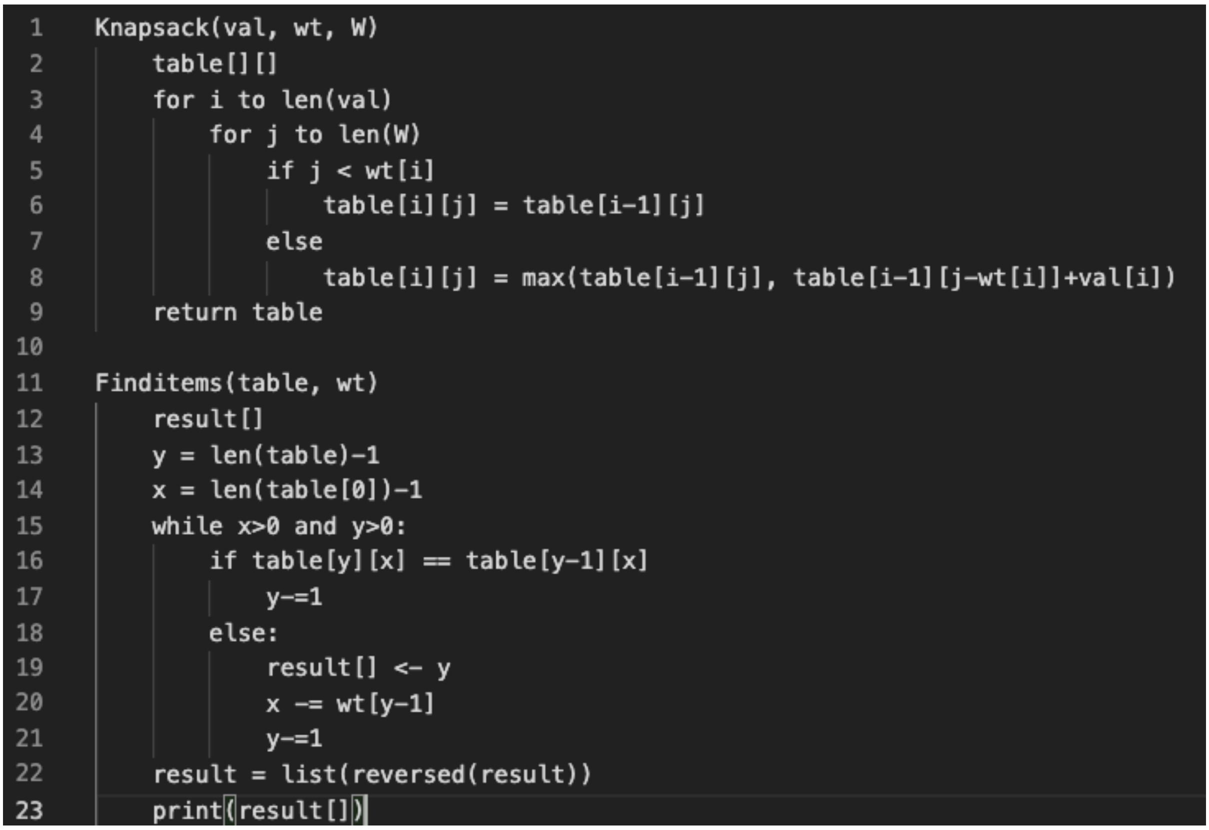
When I need the W is a constant, n is increasing, I can just use “result (7,0)”. 7 for number of results, 0 for n is increasing and vis-a-versa.

As we can see, in the CASE2, when n is a constant, running time is increasing with the increase of W.

Problem2.

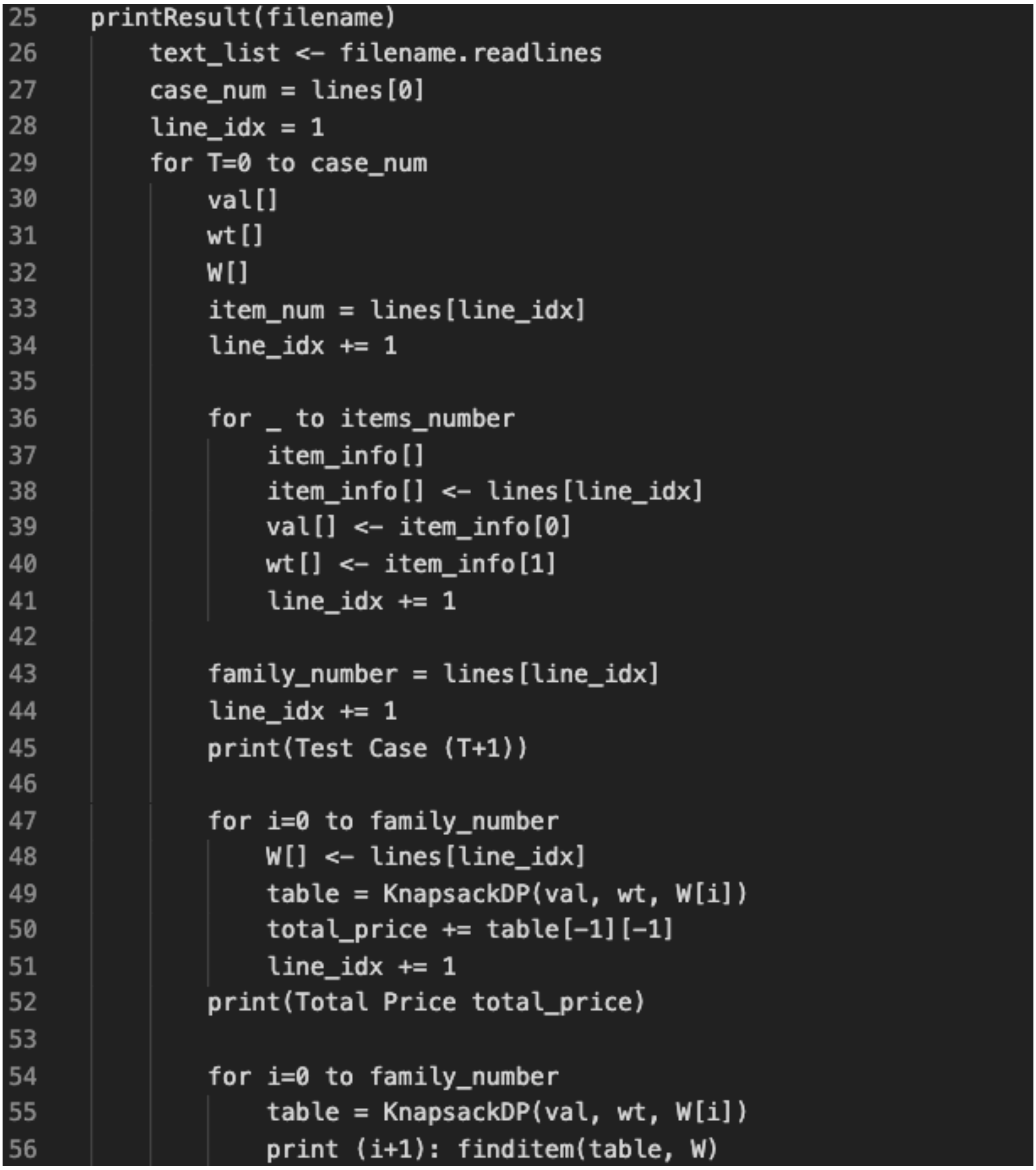
In this problem, I need get all data by reading the file. First, I used “readlines” put all lines in to a list. Second, I keep reading, recording the number of rows, and then loop to store data. Finally, I used two “for” loops to calculate the total value of the items and the number of the items each family member took. Besides, my Knapsack function is return the table which I store optimal values. Because my function “finditems” need this table.

Pseudocode:



```
Knapsack(val, wt, W)
    table[][]
    for i to len(val)
        for j to len(W)
            if j < wt[i]
                table[i][j] = table[i-1][j]
            else
                table[i][j] = max(table[i-1][j], table[i-1][j-wt[i]]+val[i])
    return table

Finditems(table, wt)
    result[]
    y = len(table)-1
    x = len(table[0])-1
    while x>0 and y>0:
        if table[y][x] == table[y-1][x]
            y-=1
        else:
            result[] <- y
            x -= wt[y-1]
            y-=1
    result = list(reversed(result))
    print(result[])
```



```
printResult(filename)
    text_list <- filename.readlines
    case_num = lines[0]
    line_idx = 1
    for T=0 to case_num
        val[]
        wt[]
        W[]
        item_num = lines[line_idx]
        line_idx += 1

        for _ to items_number
            item_info[]
            item_info[] <- lines[line_idx]
            val[] <- item_info[0]
            wt[] <- item_info[1]
            line_idx += 1

        family_number = lines[line_idx]
        line_idx += 1
        print(Test Case (T+1))

        for i=0 to family_number
            W[] <- lines[line_idx]
            table = KnapsackDP(val, wt, W[i])
            total_price += table[-1][-1]
            line_idx += 1
        print(Total Price total_price)

        for i=0 to family_number
            table = KnapsackDP(val, wt, W[i])
            print (i+1): finditem(table, W)
```