CS325

Guangyu Zhang

1.

a).

0-1 knapsack problem in recursive way: For each item, there are two possible actions, which are pick the item or don't pick the item, and then list every possibility. And in dynamic programming way, it deals with a 2d-array by getting the maximum of each step, until I got the value of the arr[W-1][n-1].

Recursion

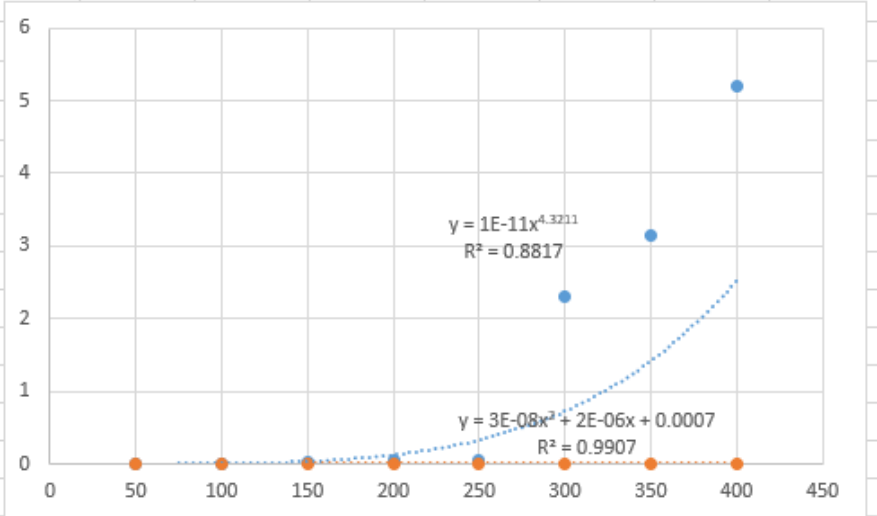
```
if n == 0 or capacity == 0 :

return 0

if weight[n - 1] > capacity:

    return recursion(capacity, weight, value, n-1)

else:

    return max(value[n-1] + recursion(capacity - weight[n-1], weight, value, n-1), recursion(capacity, weight, value, n-1))

K = [[0 for x in range(capacity + 1)] for x in range(n + 1)]

for i in range(n + 1):

    for w in range(capacity + 1):

        if i == 0 or w == 0:

            K[i][w] = 0

        elif weight[i - 1] <= w:

            K[i][w] = max(value[i - 1] + K[i - 1][w - weight[i - 1]], K[i - 1][w])

        else: K[i][w] = K[i - 1][w]

            return K[n][capacity]
```

b). submitted as code

c).

hold N as 30 and vary W.

| N | W | Rec time | DP time |
|---|---|---|---|
| 30 | 50 | 0.000997 | 0.000997 |
| 30 | 100 | 0.002993 | 0.000999 |
| 30 | 150 | 0.028860 | 0.001995 |
| 30 | 200 | 0.045877 | 0.002079 |
| 30 | 250 | 0.054579 | 0.002992 |
| 30 | 300 | 2.293256 | 0.003999 |
| 30 | 350 | 3.134838 | 0.004994 |
| 30 | 400 | 5.190716 | 0.005984 |




d).

The running time increases while the W increases. For recursion algorithm, the possibilities are increasing while the W increases, so the running time increases. For DP algorithm, the W increases since the time complexity is O(nW), running time increases when W increases.

2.

a).

in this case, the number of items is the second line, and next n lines show price and weight of n items. Then, the number next line is number of people in the current family. Next, m line is the weight that person can carry. The core algorithm is similar as knapsack algorithm.

for (i = 0; i <= n; i++)

for (w = 0; w <= W; w++)

if (i == 0 || w == 0) Temp[i][w] = 0;

elif (atoi(weight->data[i - 1]) <= w) Temp[i][w] = change(atoi(value->data[i - 1]) + Temp[i - 1][w - atoi(weight->data[i - 1])], Temp[i - 1][w]);

else Temp[i][w] = Temp[i - 1][w];

b).

the running time should be O(NFW), since for every family member, runs algo once, and each of them is O(NW), so the total time is O(NFW).

c). submitted as code.