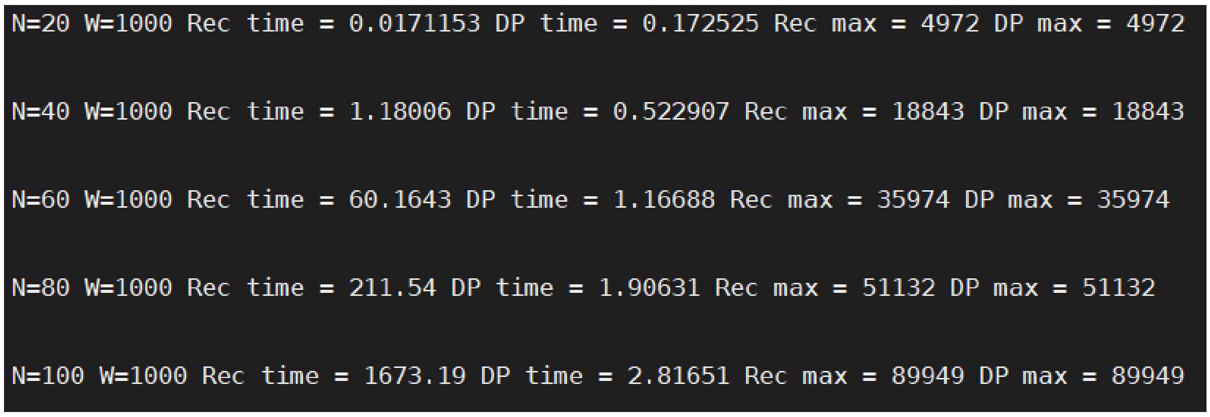
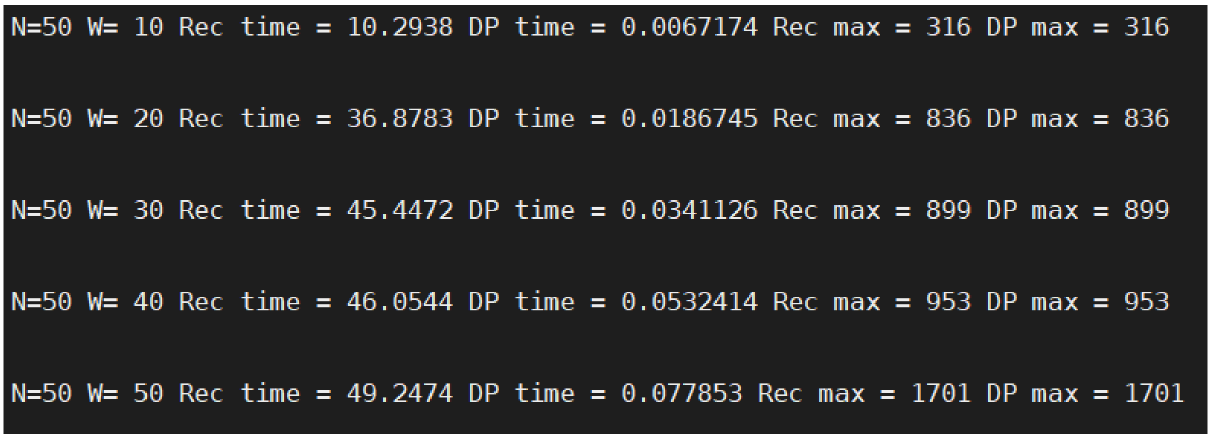
Problem 1:

a)



```
N=20 W=1000 Rec time = 0.0171153 DP time = 0.172525 Rec max = 4972 DP max = 4972

N=40 W=1000 Rec time = 1.18006 DP time = 0.522907 Rec max = 18843 DP max = 18843

N=60 W=1000 Rec time = 60.1643 DP time = 1.16688 Rec max = 35974 DP max = 35974

N=80 W=1000 Rec time = 211.54 DP time = 1.90631 Rec max = 51132 DP max = 51132

N=100 W=1000 Rec time = 1673.19 DP time = 2.81651 Rec max = 89949 DP max = 89949
```



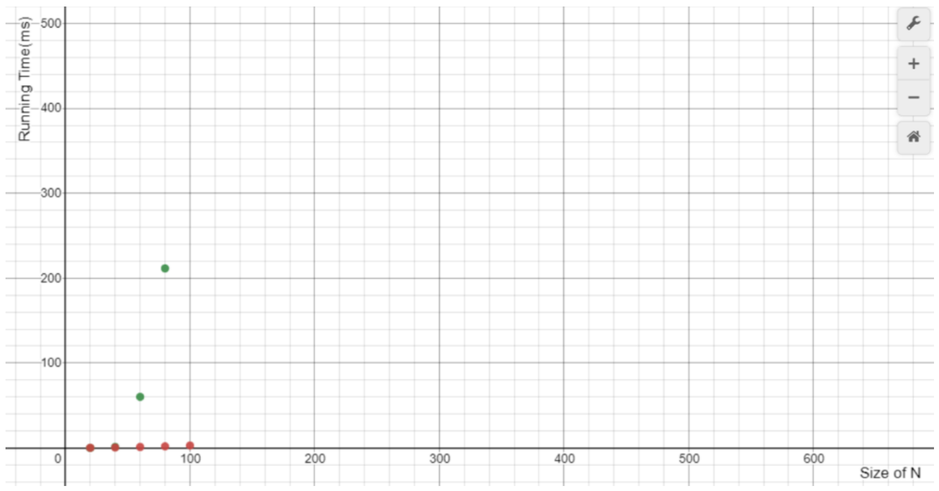
```
N=50 W= 10 Rec time = 10.2938 DP time = 0.0067174 Rec max = 316 DP max = 316

N=50 W= 20 Rec time = 36.8783 DP time = 0.0186745 Rec max = 836 DP max = 836

N=50 W= 30 Rec time = 45.4472 DP time = 0.0341126 Rec max = 899 DP max = 899

N=50 W= 40 Rec time = 46.0544 DP time = 0.0532414 Rec max = 953 DP max = 953

N=50 W= 50 Rec time = 49.2474 DP time = 0.077853 Rec max = 1701 DP max = 1701
```

(Note: The time of unit is millisecond. The program runs 10 times for each data set, and averages each results.)

b)

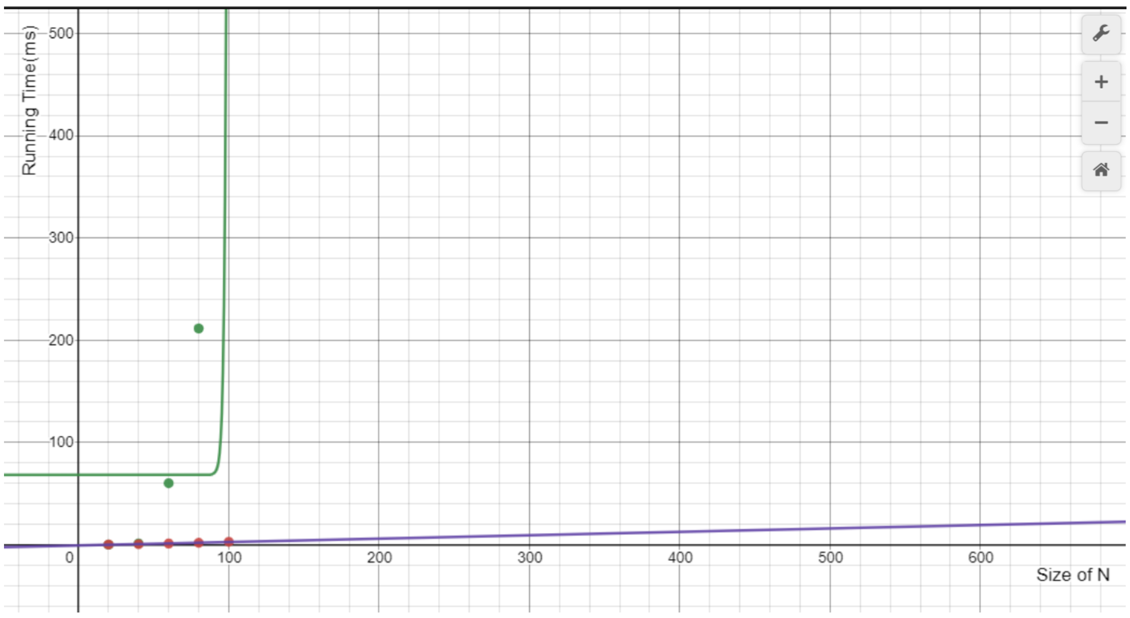
1. DP vs Recursion (Fixed W(=1000), varying N(20~100))

- Scatter plot



(Green Dot: Recursion, Orange Dot: Dynamic Programming)

- Graph

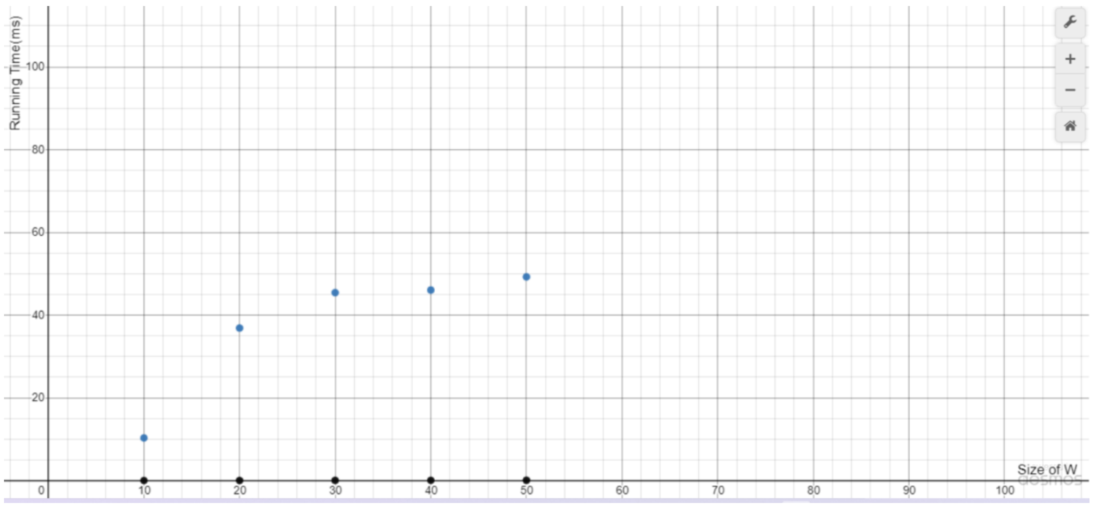


(Green graph(Recursion): $y = 1.2661 * 10^{-29} * x2^{x} + 68.225$

Purple graph(Dynamic Programming): $y = 0.0333569x - 0.684386$)

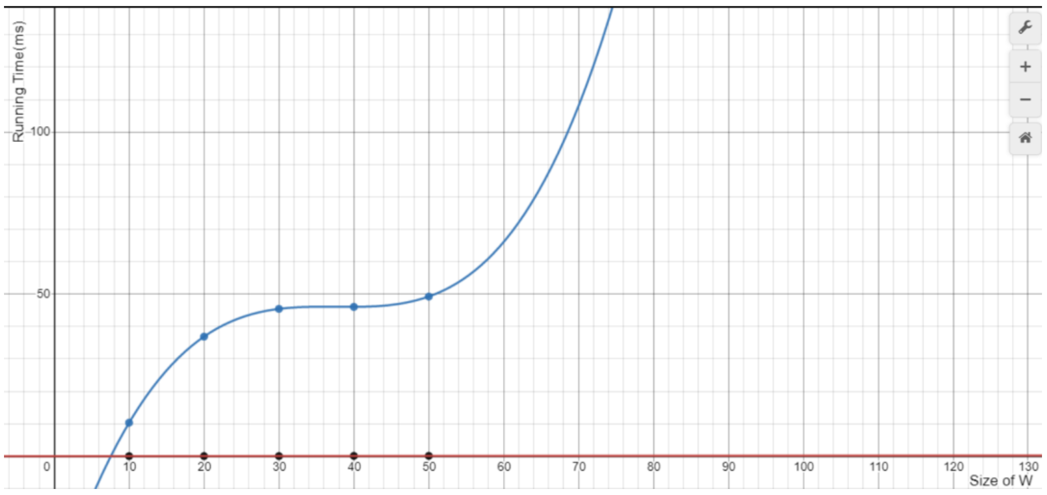
2. DP vs Recursion(Fixed N(=50), varying W(100 ~ 500))

- Scatter plot



(Blue Dots: Recursive, Black Dots: Dynamic Programming)

- Graph



(Blue Graph(Recursion) :

$$y = 0.0000205667x^4 - 0.00146998x^3 - 0.183419x^2 + 7.10117x - 43.8666$$

Red Graph(Dynamic Programming): $y = 0.0000353676x - 0.0149316$)

c)

In recursive knapsack, the program keeps searching all cases in order to maximize the value in range of W. However, in dynamic programming knapsack, the program keeps searching cases similar to recursive knapsack, and it tracked the past values instead of searching all cases.

In the process of experiment, each experiment increased one variable with fixed other variable. Both experiments showed that recursive algorithm is extremely slower than dynamic programming algorithm.

I ran each algorithm 10 times for each data set, and then averaged those results. The reason is that the running time of each data set was too small to check the general feature of each algorithm.

The size of W influenced running time of both algorithms. In case of dynamic programming, the algorithm showed linear (polynomial) running time. Thus, this means that the running time of dynamic programming is pseudo-polynomial. Furthermore, in case of recursive algorithm, the algorithm showed polynomial running time which was at least quartic polynomial. Thus, W has correlation between the running time of the algorithm.

Problem 2:

a)

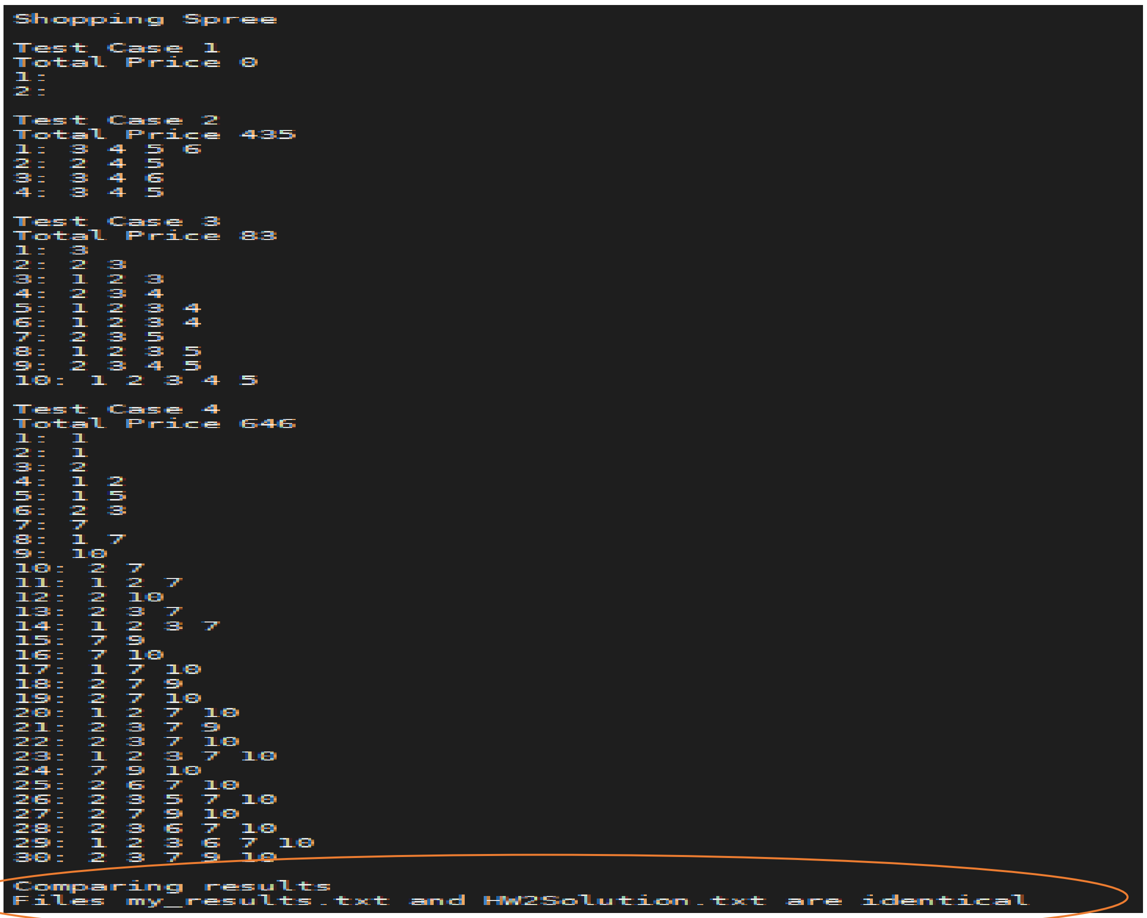
<Verbal Description>

Frist, the program reads the amount of test cases, and number of items. Second, the program iterates reading each data set as the number of items. Third, the program reads the number of family members. Fourth, the program iterates reading the capacity of knapsack and solving knapsack problem as the number of family members. Fifth, in the process of solving knapsack problem, the program fills the 2D array with the index of items selected by each family member, and adds up the maximum value packed by each family member. Sixth, the program prints out the total price and the list of select item.

<Pseudo-Code>

```
I.   Reads 'T'
II.  Read 'N'
III. Fill value array and weight array with each of 'P' and 'W'
IV.  Read 'F'
V.   Solve knapsack problem with given each 'M' & Fills 2D-array with selected items
     – iterates as 'F'
VI.  Print out maximum total price & list of selected items by each family members.
VII. Iterate step II ~ VI as number of 'T'.
```

b)



```
Shopping Spree

Test Case 1
Total Price 0
1:
2:

Test Case 2
Total Price 435
1: 3 4 5 6
2: 2 4 5
3: 3 4 6
4: 3 4 5

Test Case 3
Total Price 83
1: 3
2: 2 3
3: 1 2 3
4: 2 3 4
5: 1 2 3 4
6: 1 2 3 4
7: 2 3 5
8: 1 2 3 5
9: 2 3 4 5
10: 1 2 3 4 5

Test Case 4
Total Price 646
1: 1
2: 1
3: 2
4: 1 2
5: 1 5
6: 2 3
7: 7
8: 1 7
9: 10
10: 2 7
11: 1 2 7
12: 2 10
13: 2 3 7
14: 1 2 3 7
15: 7 9
16: 7 10
17: 1 7 10
18: 2 7 9
19: 2 7 10
20: 1 2 7 10
21: 2 3 7 9
22: 2 3 7 10
23: 1 2 3 7 10
24: 7 9 10
25: 2 6 7 10
26: 2 3 5 7 10
27: 2 7 9 10
28: 2 3 6 7 10
29: 1 2 3 6 7 10
30: 2 3 7 9 10

Comparing results
Files my_results.txt and HW2Solution.txt are identical
```

The program satisfies the specifications.