Pranav Prabhu

# Problem 1

## A. Table of Values:

**Recursive Runtime:**

| n | time(sec) |
|---|---|
| 10 | 0.001767873764038086 |
| 15 | 0.07301712036132812 |
| 20 | 0.29126572608947754 |
| 25 | 3.064086437225342 |
| 30 | 17.834840774536133 |
| 35 | 41.00116300582886 |
| 40 | 90.78732419013977 |

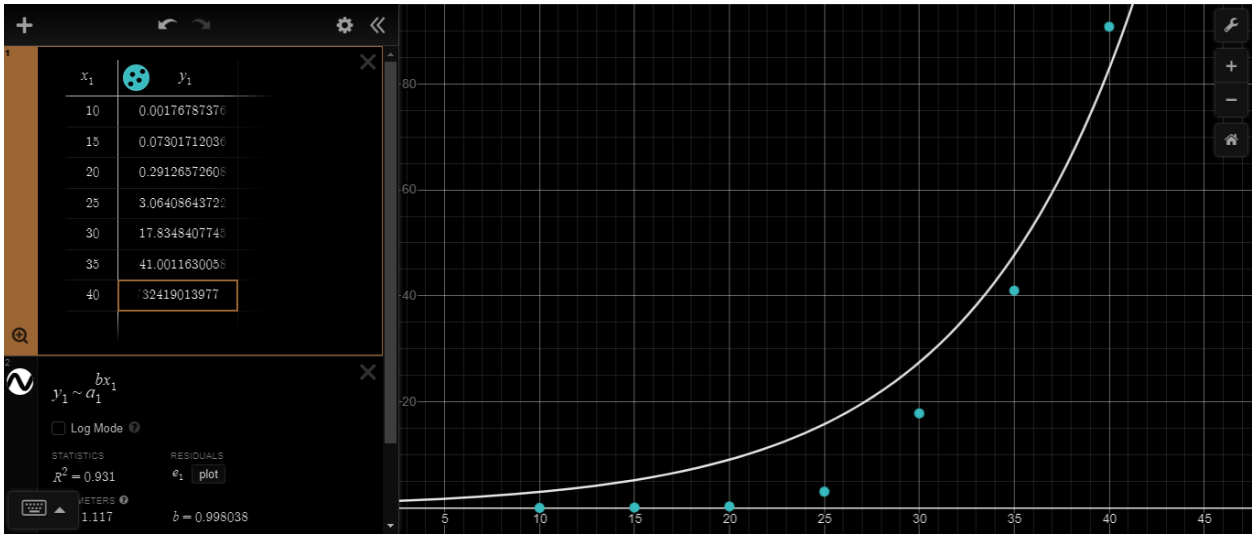
**Dynamic Programming Runtime:**

| n | time (sec) |
|---|---|
| 10 | 0.0019495487213134766 |
| 15 | 0.0029630661010742188 |
| 20 | 0.0038340091705322266 |
| 25 | 0.004757404327392578 |
| 30 | 0.006201505661010742 |
| 35 | 0.0066606998443603516 |
| 40 | 0.007135868072509766 |

## B. Graphs:

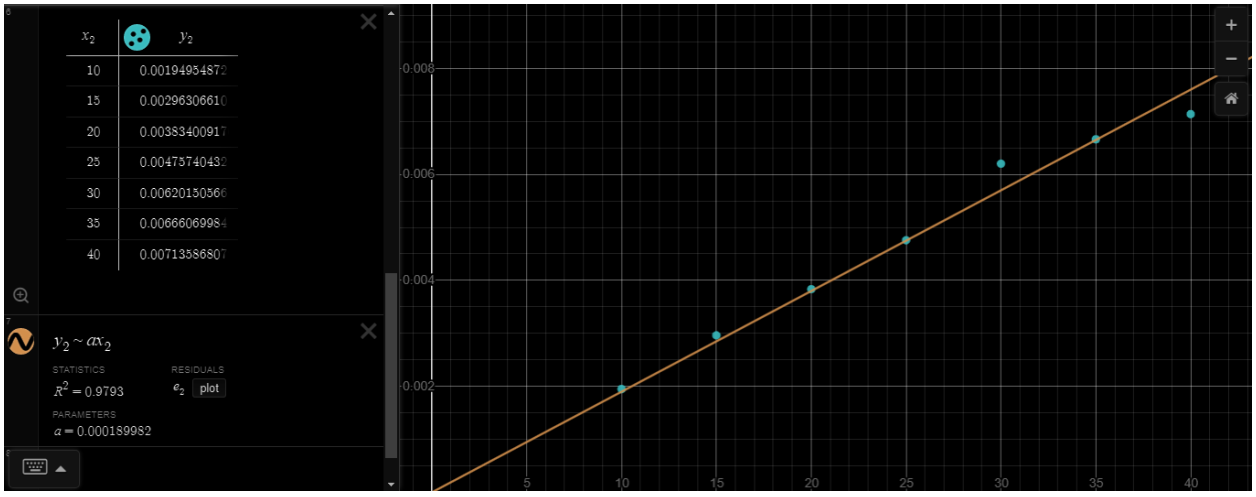
**Recursive Knapsack:**

The graph for recursive knapsack had points very close to the expected graph of $O(2^n)$, as can be seen by the R-value of **0.965** ($R^2 = 0.931$ ). The equation is $y_1 \sim a_1^{bx} 1$, which implies that this was very close to the worst-case because generally, $O(2^n)$ is a worst-case runtime.




**DP Knapsack:**

The graph for DP knapsack had points very close to the expected graph as can be seen by the R-value of **0.989** ($R^2 = 0.979$ ). Generally less than 0.05 error is accepted, and my R value was 0.989. The equation is $y_2 \sim ax_2$, which implies that this was also very close to the worst-case runtime because the runtime complexity for DP is O($N * W$), where "N" is the weight of element and "W" is the capacity. For every weight element, we traverse through all weight capacities 1 <= w <= W, which makes it fall under the worst-case scenario.



# Problem 2

**Description:**

Using the weights and price values of n items, I put them into a knapsack of capacity W to get the maximum total price in the knapsack. So, using 2 arrays namely val and wt, I stored all price and weight values associated with n items in the knapsack. We know the maximum weight is the value of W. So, I found the sum of weights of n items in the knapsack that does not exceed W. So, I first fill up the table (2D array) with 0s and build array[][] in a bottom-top manner. Either the result comes from the top (array[i-1][w]) or from val[i-1] + K[i-1][w-wt[i-1]] as in Knapsack table. If it comes from the latter one, it means the item is included. After the weight is included the value is deducted. I read the whole text file and stored each integer into an array and grabbed the specific indices from the array to read in the correct data values to fill both arrays. Later I make a call to the DP function to return the total weight as well as the path traced back.

**Pseudocode:**

```
shopping(val, wt, W, n):
     array = []

     for i in range(n + 1):
          create empty array[]
          for j in range(W + 1):
               fill array[i] with 0

     for i in range(n + 1):
          for w in range(W + 1):
               if i == 0 or w == 0: array[i][w] = 0
               elif wt[i - 1] <= w:
                    array[i][w] = max(val[i - 1] + array[i -
1][w - wt[i - 1]], array[i - 1][w])
               else: array[i][w] = array[i - 1][w]

     result = array[n][W]
```

```
        w = W

        wt_array = []
        for i in range(n, 0, -1):
            if result <= 0: break

            if result == array[i - 1][w]: continue
            else:
                add i to wt_array
                result = result - val[i - 1]
                w = w - wt[i - 1]

        sort wt_array
        print each element of wt_array
        return array[n][W]

main():
        open and read data_file
        array = []
        store integers in array

        test_cases = array[0]
        num_of_items = array[1]
        test_count = 0

        while test_count <= test_cases:
            num_of_items = array[1]
            total = 0

            read num_items from array
            items = num_items
```

```
read price for each item in array
price_array = item_price

read weight for each item in array
weight_array = item_weight

read num_people in family
read max_weight per person in family
print(test_count)

for member in family:
     total += shopping(price, weight, member,array[1])
print(total)

for i in range(1, fam_mem_count + 1):
     path = shopping(price, weight, i, array[1])
     print(i, ": ", path)
test_count += 1
```