Kanishk Singh- SE21UCAM005 (CAM)

Assignment 8- 0/1 Knapsack dynamic programming

Functions Used:

1. 0/1 knapsack using dynamic programming:

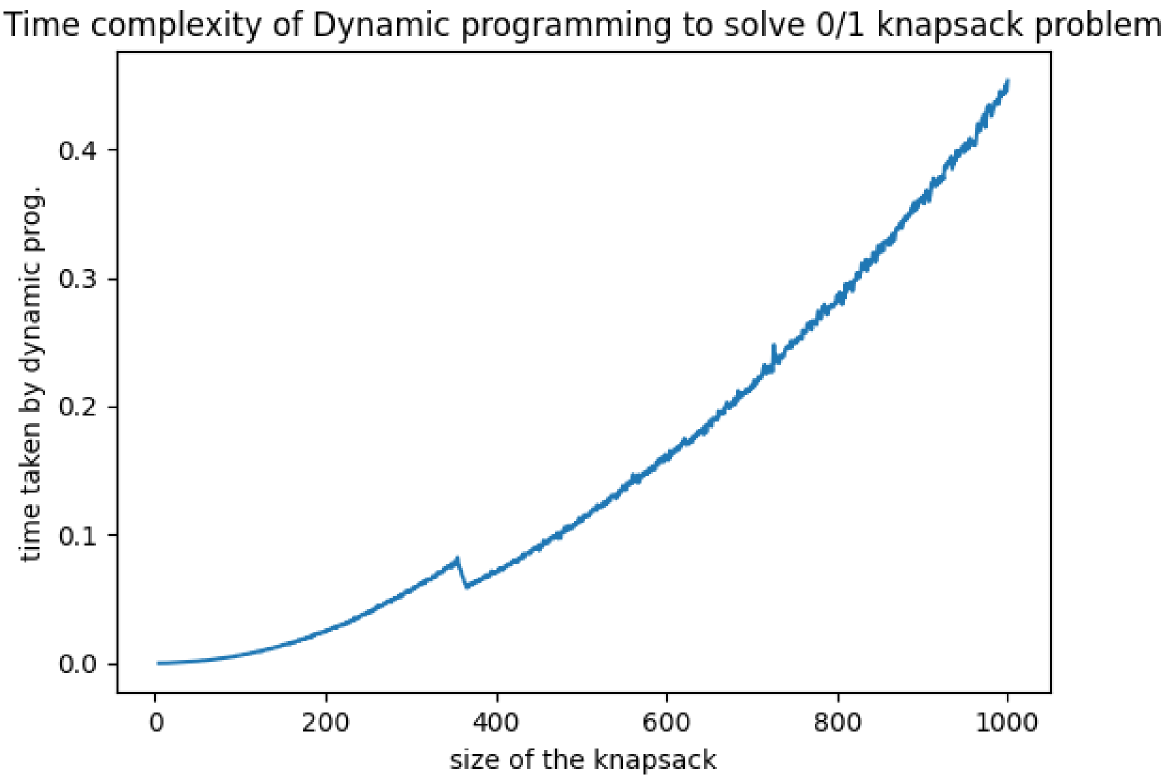
```
def knapsack_01(values, weights, W):
    n = len(values)
    DP = np.zeros((n + 1, W + 1), dtype=int)

    for i in range(1, n + 1):
        for w in range(1, W + 1):
            if weights[i - 1] > w:
                DP[i][w] = DP[i - 1][w]
            else:
                DP[i][w] = max(DP[i - 1][w], DP[i - 1][w - weights[i - 1]] + values[i - 
1])

    selected_items = []
    i, w = n, W
    while i > 0 and w > 0:
        if DP[i][w] != DP[i - 1][w]:
            selected_items.append(i - 1)
            w -= weights[i - 1]
        i -= 1

    return DP[n][W], selected_items
```

2. Random knapsack problem generation

```
def generate_random_knapsack_problem(x):
    items = []
    weights = []
    for _ in range(x):
        weight = random.randint(1, x)
        value = random.randint(1, 2 * x)
        items.append(value)
        weights.append(weight)
    return items, weights
```

Analysing process: we have plotted a graph of time taken by function vs the size of the knapsack. The Theoretical complexity of this ,method is O(n*W), where n is the number of items and W is max weight capacity. In our case we have taken both to be the same so the complexity should be $O(n^2)$

Graph:



Conclusion: We can see the graph is we are getting is an exponential graph of n^2 which matches with our theoretical complexity: $O(n^2)$