Assignment 2 Knapsack

Instructions on how to run code

- Programming Language: **Python**
- Python Runtime Engine/Version: **3.6.5**

Dependencies

Make sure to install the following packages before running

- matplotlib
- numpy

you should be able to install the dependencies via the following commands
python -m pip install --user numpy scipy matplotlib ipython jupyter pandas sympy nose

Usage

Download the Knapsack.py file

execute the Knapsack.py file

python Knapsack.py

Writeup

Description of Recursive Solution

Algorithm

For the Recursion I crawl through all of the possible options, and if the blocks align it is extremely fast. It checks by adding the item to the first knapsack, then the second, then discards that particular item, and tries the next

Description of memorizing

The memorizing utilizes the same recursion as the knapsack, but generates a cache key for the potential outcome. If the solution is true, then it returns directly to prevent any delay. If the result is false, then it stores the cache value if it is called again.

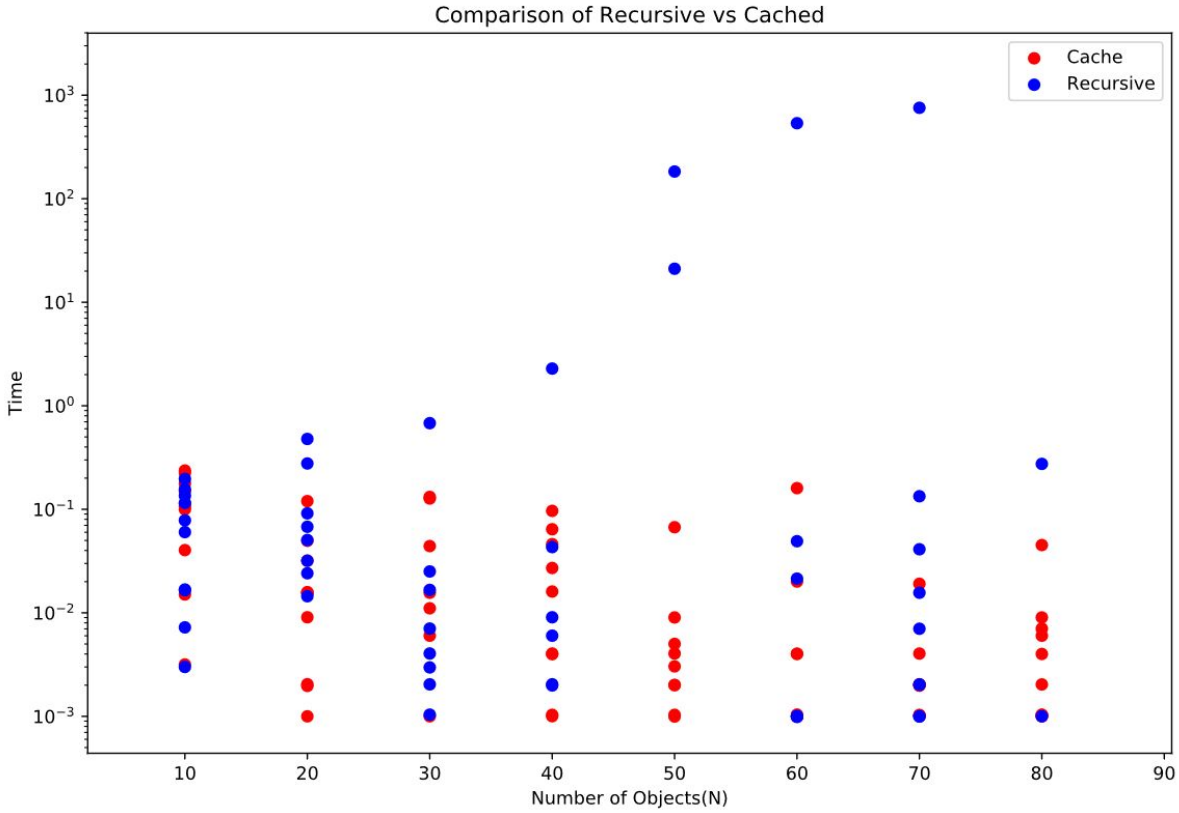
Cache

The cache I used was a Dictionary Approach by creating a hash.

I take the 3 inputs n, I1, I2 and concatinate there values into a string, for example with n=4, I1=23, I2=83 the string would be '4_23_83'.

Empirical Study

When running the empirical study the caching approach starts out slower, but with more objects, it tends to be faster at finding the solution since it can run through more possible solutions to return before attempting the deeply in depth recursion.



When running the empirical study the caching approach starts out slower, but it tends to be near constant at finding the solution since it calls on the same parameters so many times.
When running the KnapRecursive() it tends to be fast, but once in a while it will get a set that will cause it to take a very log time.

Generate a table of runtime vs. the number of objects. At what object count does it become infeasible to run the algorithm? Around 60 it may it a corner case where it runs through all possible options, and it takes a very long time for it try all possible options.

When you average it out. The cache implementation grows to be N^N time faster than the recursion.

When experimenting with larger data sets the larger L1 and L2 are with larger size items slow down the performance.
Here are the averages of the number of objects for Cache and the averages for the Recursion.

Cache			
Objects	Time(Seconds	Function Calls	CacheHitCount
10	0.14896	6963.5	5432.2
20	0.01064	407.7	88.2
30	0.01597	653.1	206.4
40	0.03084	1365.2	695.4
50	0.00261	107.4	9.3
60	0.03339	1476.2	742.5
70	0.03028	1273.2	858.2
80	0.00411	150.9	2.5
90	0.00722	258.4	25.7
100	0.00331	126.4	1.8
110	0.00942	355.1	14.6
120	0.00211	84.5	0.1
130	0.00286	95.1	0.1
140	0.00231	83.3	0
150	0.00447	155.4	0.3
160	0.00501	187.2	8.3
170	0.00346	107	0.1
180	0.00546	196.1	7.6
190	0.00281	103.9	0
Recursion			
10	0.10503	11406.8	0
20	0.07029	8293.4	0
30	0.02788	15477.6	0
40	0.01175	715369.6	0
50	0.00276	285.4	0
60	0.00737	791.6	0
70	0.00175	168.6	0
80	0.00190	176.1	0
90	0.00411	431.5	0
100	0.00762	863.2	0
110	0.00301	322.9	0
120	0.00451	507.5	0
130	0.00090	74.1	0
140	0.00095	70.5	0
150	5+ hrs	1,976,663,974.00	0