1. What is the time/space efficiency of each of your algorithms?

Time Complexity for brute force algorithm(Exhaustive Search) is O(N*2^n)

Besides the memory used to store the values and weights of all items, this algorithm requires a two one dimensional arrays

$$\sum_{i=1}^{2^n}\left[\sum_{j=n}^{1}+\sum_{k=1}^{n}\right]=\sum_{i=1}^{2^n}[\{1+..+1\}(\text{n times})+\{1+..+1\}(\text{n times})]$$

$$=(2n)*[1+1+1.....+1]\ (2^n\text{ times})$$
$$=O(2n*2^n)$$
$$=O(n*2^n)$$

Time Complexity for dynamic programming is O(N*Capacity)

In terms of memory, Dynamic Programming requires a two dimensional array with rows equal to the number of items and columns equal to the capacity of the knapsack

$$\sum_{i=0}^{N}\sum_{j=0}^{\text{Capacity}}1=\sum_{i=0}^{N}[1+1+1+......+1]\ (\text{Capacity times})$$

$$=\text{Capacity}*[1+1+1+......+1]\ (\text{N times})$$
$$=\text{Capacity}*N$$
$$=O\ (N*\text{Capacity})$$

Time Complexity for Greedy Algorithm is O(NlogN)

In terms of memory, this algorithm only requires a one dimensional array to record the solution string.

Complexity for Sorting by non-increasing order O(NlogN)
Complexity for

$$.\sum_{i=0}^{N}1=[1+1+1....1]\ (\text{N times})=N\approx O(N)$$

Total : O(NlogN) + O(N) = O(NlogN)

2. Do all three of your solutions provide an optimal solution? Why or Why not?

Both the Exhaustive Search and dynamic programming provide an optimal solution fro the given example . However, they don’t include fraction during the calculate. In this case, Only the Greedy Algorithm provide the optimal solution.
For the given example, both the Exhaustive Search and dynamic programming provide 198 but Greedy Algorithm provide 226 which is calculate by the multiplication of following to array. Noticed it has 0.8.
[1, 1, 0.8, 0, 0]
[120, 74, 40, 25, 4]
optimal value: 226.0

For my own example 1, Only the Greedy Algorithm provide the optimal solution.

3. Create and supply two other knapsack examples to test your 3 functions.
example 1

```
 input file: input2.txt is:
   The Capacity: 60
   Weights of each item: ['40', '10', '20', '24']
   Valuesc of each item:  ['280', '100', '120', '120']
   Number of items: 4

Testing Brute Force
optimal value: 340
Time use: 2.6941299438476562e-05 seconds.

Testing Dynamic Programming
optimal value: 400
Time use: 0.00036716461181640625 seconds.

Testing Greedy Algorithm
optimal value: 440.0
Time use: 0.0001201629638671875 seconds.


example 2

Solution  for  Knapsack Problem
 input file: input3.txt is:
   The Capacity: 5
   Weights of each item: ['4', '2', '3']
   Valuesc of each item:  ['10', '4', '7']
   Number of items: 3

Testing Brute Force
optimal value: 11
Time use: 2.09808349609375e-05 seconds.

Testing Dynamic Programming
optimal value: 11
Time use: 2.7179718017578125e-05 seconds.
```

```
Testing Greedy Algorithm
optimal value: 12.3333333333
Time use: 0.000114917755126953125 seconds.
```

4. Which is the better knapsack solution and why? Is this true for all knapsack examples?

The Greedy Algorithm is a better solution because it consider the fraction and give the most optimize solution. Dynamic programming is probably one of the easiest to implement because it does not require the use of any additional structures.Since the complexity of Exhaustive Search grows exponentially, it can only be used for small instances of the KP. In conclusion, for small value, all three solution works well. For later value, the greedy algorithm is the best solution.