1. The exhaustive search algorithm takes $O(n * 2^n)$ time total, because it adds up all the items, taking $O(n)$ time, for each of the 2^n items.

   The dynamic algorithm takes $\theta(n * W)$ time total, where n is the number of items and W is the capacity of the knapsack:
   - $\theta(n * W)$ time to fill in the c table: $(n + 1) * (W + 1)$ entries, each requiring $\theta(1)$ time to compute.
   - $\theta(n)$ time to trace the optimal path (since it starts in row n of the table and moves up one row at each step).

   My method of choice was greedy algorithm. This has a time efficiency of $O(n \log n)$, because the list items gets sorted by ratio in $O(n \log n)$ time.

   The exhaustive search algorithm has a space efficiency of $O(2^n)$, because the power set of a list of n elements is 2^n elements long.

   The dynamic algorithm has a space efficiency of $O(n * W)$, because it creates a matrix with $n * W$ elements

   The greedy algorithm has a space efficiency of $O(n)$, because it duplicates the existing item list.

2. No, the greedy algorithm doesn't provide an algorithm because it does not check all possibilities and is sometimes wrong. The exhaustive search and dynamic algorithm provide optimal solutions, but not in polynomial time.
3.

Example 1	**Exhaustive Search**	**Dynamic Programming Method**	**Greedy Method**
Time (s)	3.2000e-05	0.00013824	8.10600e-06

Capacity: 65		
Item	**Weight**	**Value**
1	12	3
2	50	100
3	24	29
4	6	38

Best Value: 138		
Item	**Weight**	**Value**
1	6	38
2	50	100

Example 2	**Exhaustive Search**	**Dynamic Programming Method**	**Greedy Method**
Time (s)	3.0720e-05	3.4987e-05	8.1060e-06

Capacity: 8		
Item	**Weight**	**Value**
1	1	15
2	5	10
3	3	9
4	4	5

Best Value: 29		
Item	**Weight**	**Value**
1	4	5
2	3	9
3	1	15

Example 3	Exhaustive Search	Dynamic Programming Method	Greedy Method
Time (s)	1242.08159	0.0065015	4.1813e-05

Capacity: 400		
Item	**Weight**	**Value**
1	4	26
2	35	36
3	61	93
4	2	65
5	14	17
6	13	85
7	61	38
8	82	81
9	33	79
10	93	71
11	60	19
12	11	79
13	41	72
14	31	3
15	11	28
16	18	51
17	47	93
18	12	47
19	58	43
20	48	53
21	30	4
22	52	71
23	38	69
24	3	94
25	49	89
26	87	48
27	79	15
28	82	2
29	8	30
30	22	21

Best Value: 1053		
Item	**Weight**	**Value**
1	4	26
2	61	93
3	2	65
4	13	85
5	33	79
6	11	79
7	41	72
8	11	28
9	18	51
10	47	93
11	12	47
12	48	53
13	38	49
14	3	94
15	49	89
16	8	30
17	12	47

4. The greedy algorithm is the fastest algorithm, but isn't correct for all item sets. For item sets where the greedy algorithm is correct it is the best solution. For other item sets, the dynamic programming method is better.