Assignment 1 Part 1

1. Explanation of the problem. **(1 point)**
   The Knapsack problem is an optimization problem. It can be defined as follows:
   There are items and each item has weight and benefit. The task is to decide the optimal quantity of each item to pack into a bag. The aim is to ensure that the sum of weights of the chosen items does not surpass the bag's weight capacity (bag_limit) while maximizing the overall benefit. The type of knapsack we deal in this assignment is the 0-1 knapsack problem, where each item can only be selected once, leading to a binary decision of whether to include the item in the bag or not.

   a. Give the representation of a solution (answer) of the problem, as explained during the course. **(0.5)**
   Item count is n, each item has 2 values weight and benefit. The solution can be represented as binary array. This binary array shows that if item is included in knapsack bag (1) or it is excluded from the knapsack bag (0).
   S = $\{x_1, x_2, x_3 \ldots x_n\}$ solution array
   $S_i \in \{0,1\}, i \in \{1,2 \ldots, n\}$, each x_i represents if the item is included in knapsack or excluded.

   b. Give the equation of the objective function (what we want to maximize) **(0.25)**
   The sum of the benefits is wanted to be maximized:
   $$F_{benefit}(S) = \sum_{i \in s} x_i * b_i$$
   b_i represents the benefit of ith item's benefit. If ith item is selected x_i is 1 and it is added to benefit sum.

   c. Give the equation for the restriction(s) of the problem **(0.25)**
   The sum of the weight should be lesser than the knapsack bag's weight capacity is the restriction of the problem. w_i represents the ith item's weight. If it is included in bag x_i would be 1 and its weight is added to sum.
   $$\sum_{i \in s} x_i * w_i < bag\_limit$$

2. Comparison of the problem. **(1 point)**
   a. Comparison of the time expended by the algorithms. **(0.5)**
   BFS explores the search space by systematically expanding all nodes at the current level before moving to nodes at the next level., DFS explores the search space by traversing as deeply as possible along one branch before backtracking to explore

other branches. Their time complexities are the same because in the worst case both of them searches all of the states. For this question the state number is 2^n.Both algorithm's time complexities are the same O (2^{n}). Both should look for 2^{n} subsets to find optimal solution.

b. Comparison of the space used in memory at a time by the algorithms. (0.5)

BFS search algorithm uses queue to store the states for exploration. The space complexity is determined by the queue's maximum size. BFS stores the nodes in the levels, so the maximum numbered level is the space complexity of the BFS. Space complexity it is $O(b^d)$ where b is branching factor and d is the maximum depth. For this question branching factor(b) is 2 because for each state we have two moves (get item and not get item) and the maximum depth is the n (item count). Therefore, the BFS algorithm's space complexity is $O(2^n)$ for 0-1 knapsack problem.

DFS search algorithm uses stack to store the states for exploration. The space complexity is limited by the depth of the tree. Because it holds only the path of states that it came from the initial state and that can be at most the depth of the search space which is the n (item count) for this question. Therefore, the space complexity is $O(n)$