Knapsack Problem

1. 請問兩種 Knapsack Problem 定義為何？

   說明（Knapsack Problem）：

   將一群物品儘量塞進背包裡面，令背包裡面的物品總價值最高。背包沒有容量限制，無論物品是什麼形狀大小，都能塞進背包；但是背包有重量限制，如果物品太重，就會撐破背包。

   a、 0/1 Knapsack Problem

   定義：每種物品只會放進背包零個或一個。一個物品不是「整個不放進背包」就是「整個放進背包」。物品無法切割。

   b、 Fractional Knapsack Problem

   Fractional 是「分數」的意思。一個物品可以切下一部分、只取幾分之幾放進背包。

2. 請寫出兩種 knapsack problem 的演算法，並且寫出其對應設計的技巧。（設計的技巧的一例為：Kruskal's algorithm –-> greedy algorithm）

   a. 0/1 Knapsack Problem → Dynamic Programming

   令物品編號為: 1，2，...，n 號，

   令 c[i][w]為 在背包重量 W 單位的限制下，1~i 號可以拿到的最高價值。

   則 `c[i][w]` =

   (1) `c[i-1][w]`，如果不拿第 i 號物品(再加上 i 號物品，重量會大於 W)。

   (2) `c[i-1][w-(i號物品重量)]+ i號物品價值`，如果拿第 i 號物品(再加上第 i 號物品，重量<=W)。

   `c[i][w]`為 max((1), (2,))

   初始值：當 i=0 或 w=0 時，`c[i][w]` = 0

   演算法：

   input:

   v = <v_1, v_2, …, v_n>，為 n 個物品分別的價值

   w = <w_1, w_2, …, w_n>，為 n 個物品分別的重量

n 為物品數

W 為背包可容量重量

```
Dynamic-0-1-knapsack (v, w, n, W)
for w = 0 to W do
   c[0, w] = 0
for i = 1 to n do
   c[i, 0] = 0
   for w = 1 to W do
      if wi ≤ w then
         if vi + c[i-1, w-wi] then
            c[i, w] = vi + c[i-1, w-wi]
         else c[i, w] = c[i-1, w]
      else
         c[i, w] = c[i-1, w]
```

分析：此演算法的主程式有兩個 for 迴圈，因此此演算法的時間複雜度為$\theta(nW)$。

b. Fractional Knapsack Problem → Greedy Algorithm

策略：價值與重量的比值最高的物品，優先放進背包。

令背包的最大容量 W，以及可以放入背包的 n 個物品的重量 w_i 與價格 p_i

步驟：

(1) 將 p_i/w_i 由大至小排序。

(2) 根據此排序來將物品依序盡可能地放入背包中。如果那個物品的重量大於背包剩餘重量，那就只放背包剩餘重量的重量。

演算法：

input:

w[1..n] 為 n 個物品分別的重量

p[1..n] 為 n 個物品分別的價值

（w[1..n], p[1..n]已經依照 p_i/w_i 由大至小排序。）

W 為背包可容納重量

```
Algorithm: Greedy-Fractional-Knapsack (w[1..n], p[1..n], W)
for i = 1 to n
   do x[i] = 0
weight = 0
for i = 1 to n
   if weight + w[i] ≤ W then
      x[i] = 1
      weight = weight + w[i]
   else
      x[i] = (W - weight) / w[i]
      weight = W
      break
return x
```

output: x 中 1 代表全拿，0 代表全不拿，分數代表拿了幾分之幾

分析：演算法中，for 迴圈的複雜度為$\theta(n)$，再加入排序，所以此演算法的時間複雜度為$\theta(nlogn)$

3. 請舉出一個例子說明為何 0/1 Knapsack Problem 不能用 greedy algorithm 解。

策略 1：價值與重量的比值最高的物品，優先放進背包。

反例：

假設背包可承受 60 的重量。

Item	A	B	C
Price	100	280	120
Weight	10	40	20
Ratio	10	7	6

根據上圖的例子，若按照 greedy algorithm 解，那他拿的順序就是 A→B→C，而 A+B 的重量為 10+40=50，A+B+C 的重量為 10+40+20=70>60。因此根據 0/1 Knapsack Problem 的定義，我們只能取 A+B 而他們的價值為 100+280=380。但是如果我們是取 B+C 的話，那他的價值為 280+120=400 重量剛好等於 60，且價值也比拿 A+B 高。

策略 2：價值最高的開始取。

反例：

Example-1

Let us consider that the capacity of the knapsack is W = 25 and the items are as shown in the following table.

Item	A	B	C	D
Profit	24	18	18	10
Weight	24	10	10	7

根據上圖的例子，若按照 Greedy Algorithm 解，那他的順序就是 A→B→C→D，而 A 的重量為 24，A+B 的量為 24+10=34>25。因此根據 0/1 Knapsack Problem 的定義，我們只能取 A 而他的價值為 24。但如果我們是取 B+C 的話，那他的價值為 18+18=36 重量為 20<25，且價值也比只取 A 高。