```python
# 2.4.1
import numpy as np

# Input format example:
# 103
# 10 100 20 1
# 5 40 15 2
# Where first line is the weight,
# second is the items' profits
# and third is the items' weights

# Reading input
w = int(input())
Ps = [int(p) for p in input().split()]
Ws = [int(w) for w in input().split()]
n = len(Ps)

# To make the code faster
Ps = np.array(Ps)
Ws = np.array(Ws)

T = np.zeros(w+1)

# Build vector T
for i in range(w+1):
    for j in range(n):
        # If we can insert the item
        if (Ws[j] <= i):
            # Keep the item with max profit which is either
            # the previous item (with lower weight) or the
            # current item (with heigher weight)
            T[i] = max(T[i], T[i - Ws[j]] + Ps[j])

# Output example:
# 241.0
# Where 241.0 is the optimal knapsack value given the inputs
print(T[-1])
```

```python
# 2.4.2
import numpy as np

# Reading input
# Input format example:
# 375
# 60 100 130 10
# 10 30 20 5
# Where first line is the profit,
# second is the items' profits
# and third is the items' weights
p = int(input())
Ps = [int(p) for p in input().split()]
Ws = [int(w) for w in input().split()]
n = len(Ps)

# Create p/w ratio
Rs = []
for i in range(n):
    Rs.append(Ps[i] / Ws[i])

# Create tuple structure (item, profit, weight, ratio)
T = []
for i in range(n):
    T.append((i, Ps[i], Ws[i], Rs[i]))

# Sort by ratio
T = sorted(T, key=lambda x: x[3], reverse=True)
curr_p = 0
items = np.zeros(n)

# Add the highest p/w item ratio taking into account the item profit
# in order to not surpass the desired input value
while curr_p < p:
    for i in range(n):
        item_p = T[i][1]
        if item_p + curr_p <= p:
            curr_p += item_p
            items[T[i][0]] += 1
            break
    else:
        break

# Output example:
# Item 0 was added 1 times.
# Item 1 was added 0 times.
# Item 2 was added 2 times.
# Item 3 was added 5 times.
for i in range(n):
    print(f'Item {i} was added {int(items[i])} times.')
```

```python
# 2.4.3
import numpy as np

# Reading input
# 8
# 1 2 5 6
# 2 3 4 5
# Where first line is the weight,
# second is the items' profits
# and third is the items' weights

w = int(input())
Ps = [int(p) for p in input().split()]
Ws = [int(w) for w in input().split()]
n = len(Ps)

# To make the code more readable
Ps.insert(0, 0)
Ws.insert(0, 0)

# To make the code faster
Ps = np.array(Ps)
Ws = np.array(Ws)

T = np.zeros((n+1, w+1))

# Build matrix T
for i in range(n+1):
    for j in range(w+1):
        if i == 0 or w == 0:
            T[i, j] = 0
        # If it's worth to add one more item or to
        # leave the itens we had before
        elif Ws[i] <= j:
            T[i, j] = max(Ps[i] + T[i-1, j-Ws[i]], T[i-1, j])
        # Keep the item we had before
        else:
            T[i, j] = T[i-1, j]

# Output example:
# 8.0
# Where 8.0 is the optimal knapsack value given the inputs
print(T[-1][-1])
```

```python
# 2.4.4
import numpy as np

# Reading input
# Input format example:
# 374
# 60 100 130 10 200 5
# 10 30 20 5 40 20
# Where first line is the profit,
# second is the items' profits
# and third is the items' weights
p = int(input())
Ps = [int(p) for p in input().split()]
Ws = [int(w) for w in input().split()]
n = len(Ps)

# Create p/w ratio
Rs = []
for i in range(n):
    Rs.append(Ps[i] / Ws[i])

# Create tuple structure (item, profit, weight, ratio)
T = []
for i in range(n):
    T.append([i, Ps[i], Ws[i], Rs[i]])

# Sort by ratio
T = sorted(T, key=lambda x: x[3], reverse=True)
curr_p = 0
items = np.zeros(n)

# Add the highest p/w item ratio taking into account the item profit
# in order to not surpass the desired input value
while curr_p < p:
    for i in range(n):
        item_p = T[i][1]
        if item_p != 0:
            item_p_frac = (p - curr_p) / item_p
            if item_p + curr_p <= p:
                curr_p += item_p
                items[T[i][0]] += 1
                # Item's profit is now 0 so that it's not selected more than once
                T[i][1] = 0
                break
            elif item_p_frac < 1:
                curr_p += item_p_frac * T[i][1]
                items[T[i][0]] += item_p_frac
                break
        else:
            break

# Output example:
# Item 0 was added 1.0 times.
# Item 1 was added 0.0 times.
# Item 2 was added 1.0 times.
# Item 3 was added 0.0 times.
# Item 4 was added 0.92 times.
# Item 5 was added 0.0 times.
for i in range(n):
    print(f'Item {i} was added {items[i]} times.')
```