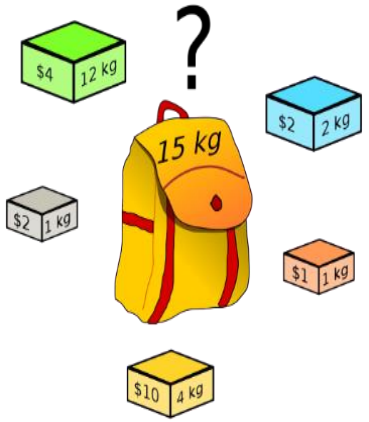
GREEDY METHOD

KNAPSACK PROBLEM

Insights

- Knapsack Problem – An Introduction
- Knapsack Problem – Variants & Its difference.
- Knapsack Problem
- Example:
  - Solution 1 – Random Selection
  - Solution 2 - Decreasing order of Profits
  - Solution 3 – Increasing order of Weights
  - Solution 4 – Decreasing order of Profit per Weight
- Greedy Algorithm for Knapsack Problem
  - Algorithm
  - Time Complexity

Knapsack Problem



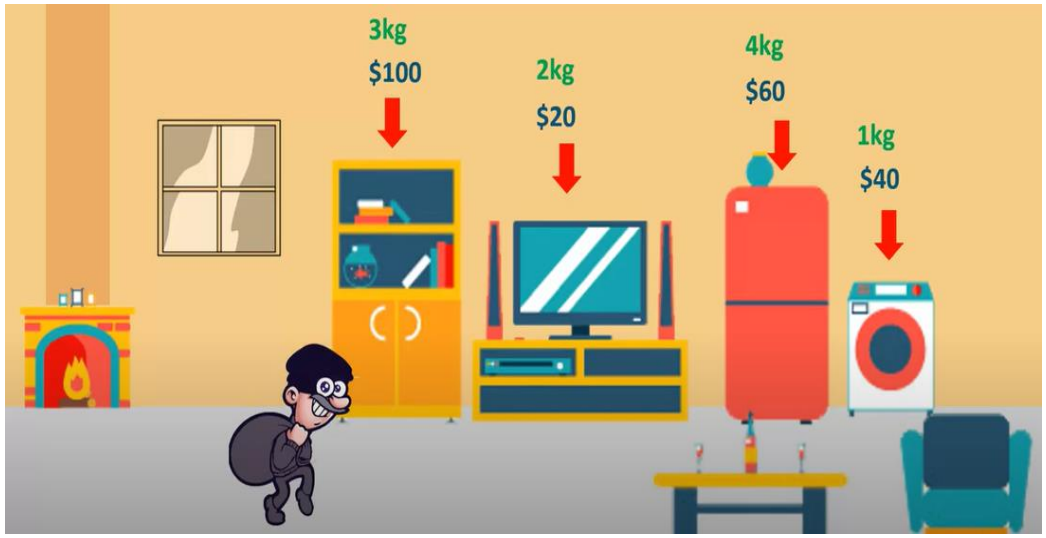
You are given the following-

- A knapsack (kind of shoulder bag) with limited weight capacity.
- Few items each having some weight and value.

The problem states-

Which items should be placed into the knapsack such that-

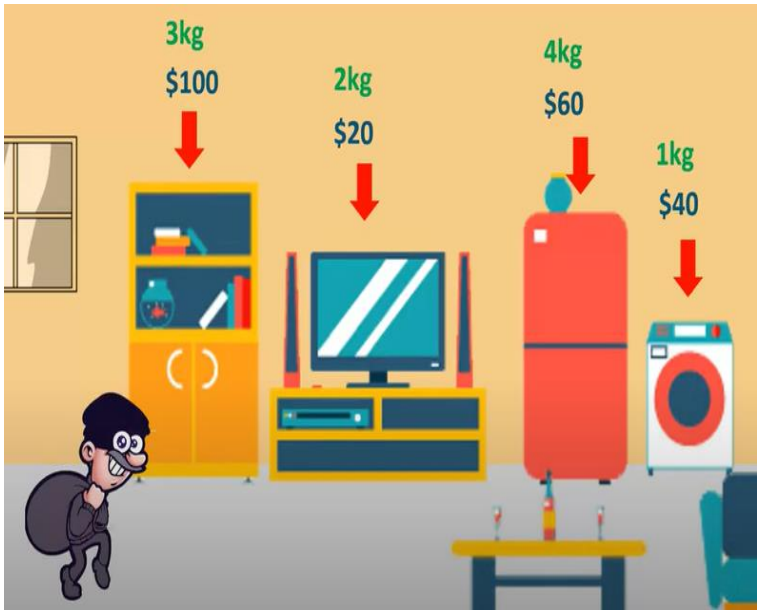
The **value or profit** obtained by putting the items into the knapsack is **maximum.** And the **weight limit** of the knapsack does **not exceed.**



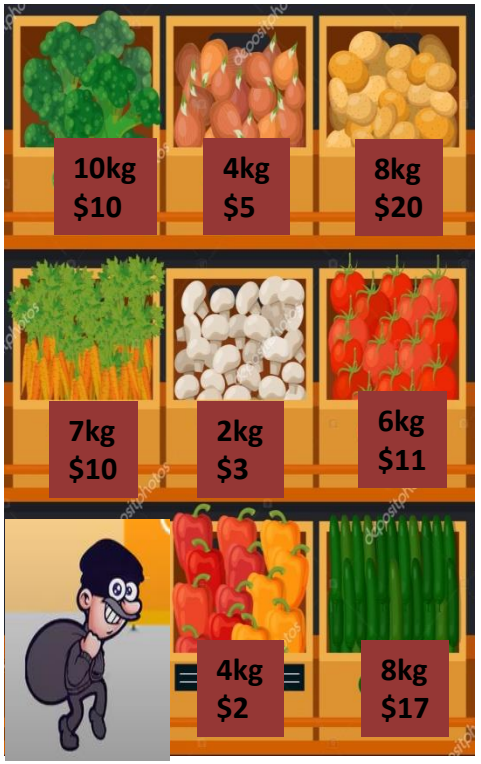
Knapsack Problem Variants & its differences

Variants:

- 0/1 Knapsack.
  - not allowed to break items. We either take the whole item or don't take it.



- Fractional Knapsack
  - can break items for maximizing the total value of knapsack



Knapsack Problem

N -number of objects

M -knapsack or bag Capacity

p_i - a positive profit

w_i - a positive weight

x_i – Selection Vector [0 - 1](allows fractional values)

Goal: Choose items which will maximums the profit and Total weight must be less than or equal to M.

$$maximize\ "Profit\ of\ the\ selected\ Object"\ (\sum_{i=1}^{N} p_i x_i) \quad \rightarrow 1$$

$$with\ subject\ to\ "Weight\ of\ Selected\ Objects"\ (\sum_{i=1}^{N} w_i x_i <= M\ (maximum\ capacity\ of\ the\ knapsack) \quad \rightarrow 2$$

$$where\ 0 <= xi \leq 1\ and\ 1 \leq i \leq N \quad \rightarrow 3$$

Knapsack Problem – Solution: Random Selection

Example:

Consider the following instance of the knapsack problem

n= 3, m=20,(p_1,p_2,p_3) = (25,24,15) and (w_1,w_2,w_3) = (18,15,10)

Objects = {A, B, C}

Maximum Weight that Knapsack can hold (m) = 20

Objects	Profit (p_i)	Weight (w_i)
A	25	18
B	24	15
C	15	10
Maximum Weight that Knapsack can hold (m) = 20		

Solution:1 **Randomly selected**

Selection Vector (x_1,x_2,x_3) : (1/2, 1/3, 1/4)

Weight of Selected Objects ($\sum_{i=1}^{n} w_i x_i$) = (1/2 *18 + 1/3 *15 + 1/4*10) = (9+5+2.5) = 16.5

$Profit\ of\ the\ selected\ Object\ (\sum_{i=1}^{n} \rho_i x_i)$ = (1/2 *25 + 1/3 *24 + 1/4*15) = (12.5+8+3.75) = 24.25

Knapsack Problem – Solution: Decreasing Order of Profits

decreasing order of **profits** : A, B, C

Objects	Profit (p_i)	Weight (w_i)
A	25	18
B	24	15
C	15	10

Maximum Weight that Knapsack can hold (m) = 20

Selection Vector (x_1,x_2,x_3) : (1,2/15,0)

Knapsack Weight = $\sum w_i x_i$ = (1*18 + 2/15*15 + 0*10) = (18+2+0) = 20

Profit = $\sum p_i x_i$ = (1 *25+ 2/15 *24+0*15) = (25+3.2+0) = 28.2

Knapsack Problem – Solution: Increasing Order of Weights

Objects are arranged in **increasing order** of **weights**

: C, B, A

Objects	Profit (p_i)	Weight (w_i)
C	15	10
B	24	15
A	25	18
Maximum Weight that Knapsack can hold (m) = 20		

Selection Vector (x_1,x_2,x_3) : (0,2/3,1)

Knapsack Weight = $\sum w_i x_i$ = (0*18+ 2/3*15+1*10) = (0+10+10) = 20

Profit = $\sum p_i x_i$ = (0 *25+ 2/3 *24+1*15) = (0+16+15) = 31

Knapsack Problem – Solution: Decreasing Order of Profits per Weight

Objects are arranged in **Decreasing order of p_i/w_i** : B, C, A

Objects	Profit (p_i)	Weight (w_i)	P_i / w_i
B	24	15	1.6
C	15	10	1.5
A	25	18	1.4

Maximum Weight that Knapsack can hold (m) = 20

Selection Vector (x_1,x_2,x_3) : $(0,1,1/2)$

Knapsack Weight = $\sum w_i x_i$ = (0*18+ 1*15+1/2*10) = (0+15+5) = 20

Profit = $\sum p_i x_i$ =(0 *25+ 1 *24+1/2*15) = (0+24+7.5)= 31.5

GREEDY ALGORITHM FOR THE FRACTIONAL KNAPSACK PROBLEM

Algorithm GreedyKnapsack(m, n)

//P[1:n] and w[1:n] contain the profits and weights respectively of the n objects

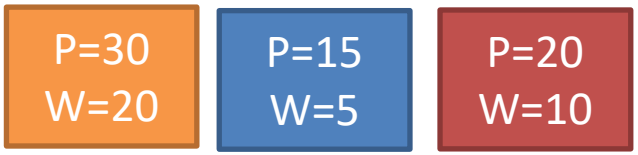
// ordered such that p[i]/w[i] >= p[i+1]/w[i+1].

//m is the knapsack size and x[1:n] is the solution vector.

```
{
        for i :=1 to n  do
            x[i] := 0.0;          // Initialize x.
        U := m;                   //Knapsack Size
        for i := 1 to n do  {
            if ( w[i] > U ) then
                break;
            x[i] := 1
            U := U - w[i];
        }
        if ( i <= n) then
            x[i] := U/w[i];
}
```

GREEDY ALGORITHM FOR THE FRACTIONAL KNAPSACK PROBLEM

```
Algorithm GreedyKnapsack(m, n)
{
        for i :=1 to n  do
            x[i] := 0.0;
        U := m;
        for i := 1 to n do  {
            if ( w[i] > U ) then
                break;
            x[i] := 1
            U := U - w[i];
        }
        if ( i <= n) then
           x[i] := U/w[i];
}
```



GREEDY ALGORITHM FOR THE FRACTIONAL KNAPSACK PROBLEM

```
Algorithm GreedyKnapsack(m, n)
{
        for i :=1 to n  do
            x[i] := 0.0;
        U := m;
        for i := 1 to n do  {
            if ( w[i] > U ) then
                break;
            x[i] := 1
            U := U - w[i];
        }
        if ( i <= n) then
            x[i] := U/w[i];
}
```

P=15
W=5

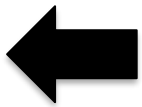
P=20
W=10

P=30
W=20

Knapsack

GREEDY ALGORITHM FOR THE FRACTIONAL KNAPSACK PROBLEM

```
Algorithm GreedyKnapsack(m, n)
{
        for i :=1 to n  do
            x[i] := 0.0;
        U := m;
        for i := 1 to n do  {
            if ( w[i] > U ) then
                break;
            x[i] := 1
            U := U - w[i];
        }
        if ( i <= n) then
           x[i] := U/w[i];
}
```

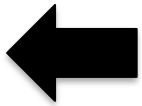


P=15 W=5	P=20 W=10	P=30 W=20

Knapsack
M=15

GREEDY ALGORITHM FOR THE FRACTIONAL KNAPSACK PROBLEM

```
Algorithm GreedyKnapsack(m, n)
{
        for i :=1 to n  do
            x[i] := 0.0;
        U := m;
        for i := 1 to n do  {
            if ( w[i] > U ) then
                break;
            x[i] := 1
            U := U - w[i];
        }
        if ( i <= n) then
           x[i] := U/w[i];
}
```



P=15 W=5	P=20 W=10	P=30 W=20

N=3

X=[0.0 0.0 0.0]

U=15

i=N/A

Knapsack M=15

GREEDY ALGORITHM FOR THE FRACTIONAL KNAPSACK PROBLEM

```
Algorithm GreedyKnapsack(m, n)
{
        for i :=1 to n  do
            x[i] := 0.0;
        U := m;
        for i := 1 to n do  {
            if ( w[i] > U ) then
                break;
            x[i] := 1
            U := U - w[i];
        }
        if ( i <= n) then
           x[i] := U/w[i];
}
```

P=15 W=5	P=20 W=10	P=30 W=20

N=3

X=[1.0 0.0 0.0]

U=10

i=1

Knapsack M=15

GREEDY ALGORITHM FOR THE FRACTIONAL KNAPSACK PROBLEM

```
Algorithm GreedyKnapsack(m, n)
{
        for i :=1 to n  do
            x[i] := 0.0;
        U := m;
        for i := 1 to n do  {
            if ( w[i] > U ) then
                break;
            x[i] := 1
            U := U - w[i];
        }
        if ( i <= n) then
           x[i] := U/w[i];
}
```

P=20
W=10

P=30
W=20

N=3

X=[1.0 1.0 0.0]

U=0

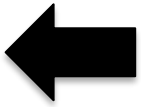
i=2

Knapsack
M=15

P=15
W=5

GREEDY ALGORITHM FOR THE FRACTIONAL KNAPSACK PROBLEM

```
Algorithm GreedyKnapsack(m, n)
{
        for i :=1 to n  do
            x[i] := 0.0;
        U := m;
        for i := 1 to n do  {
            if ( w[i] > U ) then
                break;
            x[i] := 1
            U := U - w[i];
        }
        if ( i <= n) then
           x[i] := U/w[i];
}
```



P=30
W=20

N=3

X=[1.0 1.0 0.0]

U=0

i=3

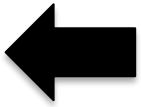
Knapsack
M=15

P=20
W=10

P=15
W=5

GREEDY ALGORITHM FOR THE FRACTIONAL KNAPSACK PROBLEM

```
Algorithm GreedyKnapsack(m, n)
{
        for i :=1 to n  do
            x[i] := 0.0;
        U := m;
        for i := 1 to n do  {
            if ( w[i] > U ) then
                break;
            x[i] := 1
            U := U - w[i];
        }
        if ( i <= n) then
           x[i] := U/w[i];
}
```



P=30
W=20

N=3

X=[1.0 1.0 0.0]

U=0

i=3

Knapsack
M=15

P=20
W=10

P=15
W=5

GREEDY ALGORITHM FOR THE FRACTIONAL KNAPSACK PROBLEM

```
Algorithm GreedyKnapsack(m, n)
{
        for i :=1 to n  do
                x[i] := 0.0;          // Initialize x.
        U := m;                      //Knapsack Size
        for i := 1 to n do  {
             if ( w[i] > U ) then
                  break;
             x[i] := 1
             U := U - w[i];
        }
        if ( i <= n) then
            x[i] := U/w[i];
}
```

Time Complexity
= O(n)