Caso Integrador:
Problema de la mochila

Análisis y Diseño de Algoritmos

Ing. Román Martínez M.

El problema de la mochila

- Suponer que se tiene un conjunto de objetos, cada uno de los cuales tiene un peso y un valor asociados…
- Suponer también que se tiene una mochila que puede soportar sólo cierto peso en los objetos que guarda, para que no se rompa…

- ***¿Qué objetos se pueden guardar en la mochila de tal manera que se guarde el mayor valor posible, sin exceder la capacidad de la mochila?***

Formalmente...

- Sea $S = \{ obj_1, obj_2, \ldots , obj_n \}$
- Sea p_i el peso del objeto i y v_i el valor del objeto i.
- Sea P el peso máximo que la mochila soporta.
- El problema de la mochila consiste en encontrar un subconjunto A de S tal que:
  - La sumatoria de los v_i de los objetos de A son el máximo posible,
  - siempre y cuando, la sumatoria de los p_i de los objetos de A es menor o igual a P.

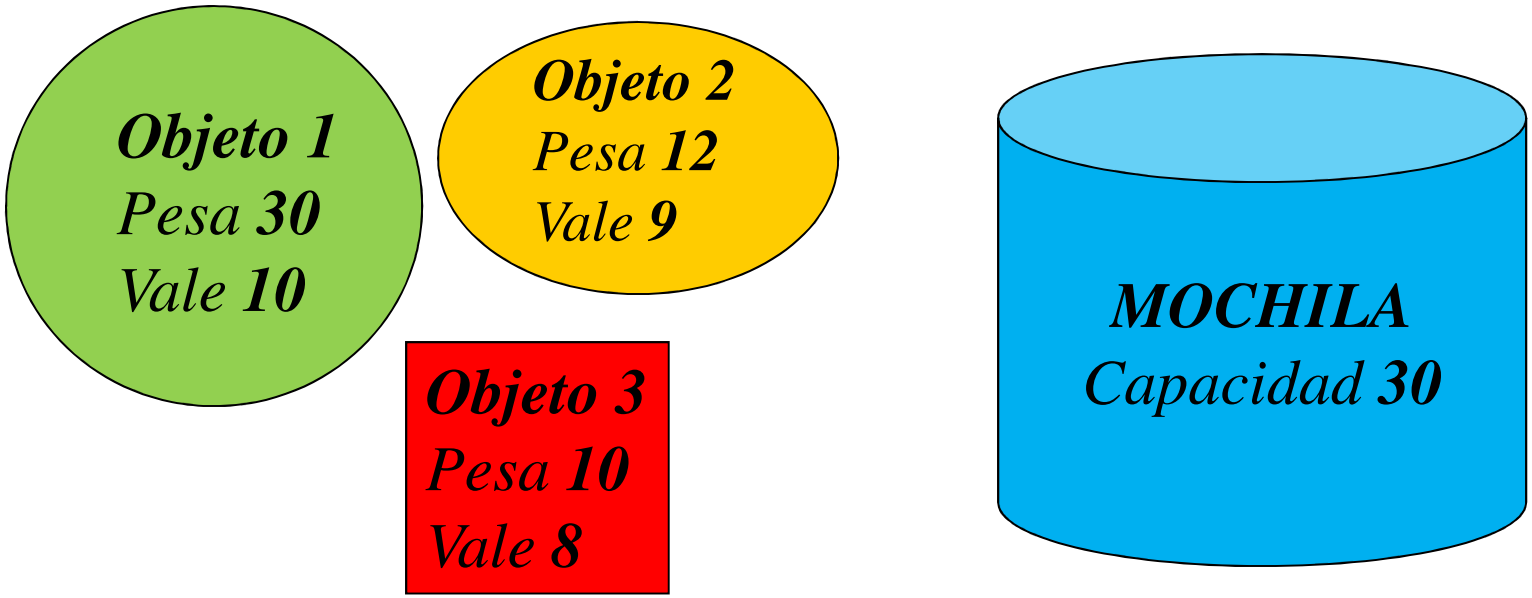
Solución al problema...

- FUERZA BRUTA:
  - Encontrar todos los posibles subconjuntos que se pueden formar de S y seleccionar el que tiene la sumatoria mayor de los valores de los objetos.
  - Comportamiento **$O(2^n)$**
- Aplicando técnicas de diseño de algoritmos:
  - Algoritmo voraz?
  - Programación dinámica?
  - Divide y vencerás?

Solución con el enfoque de un algoritmo voraz...

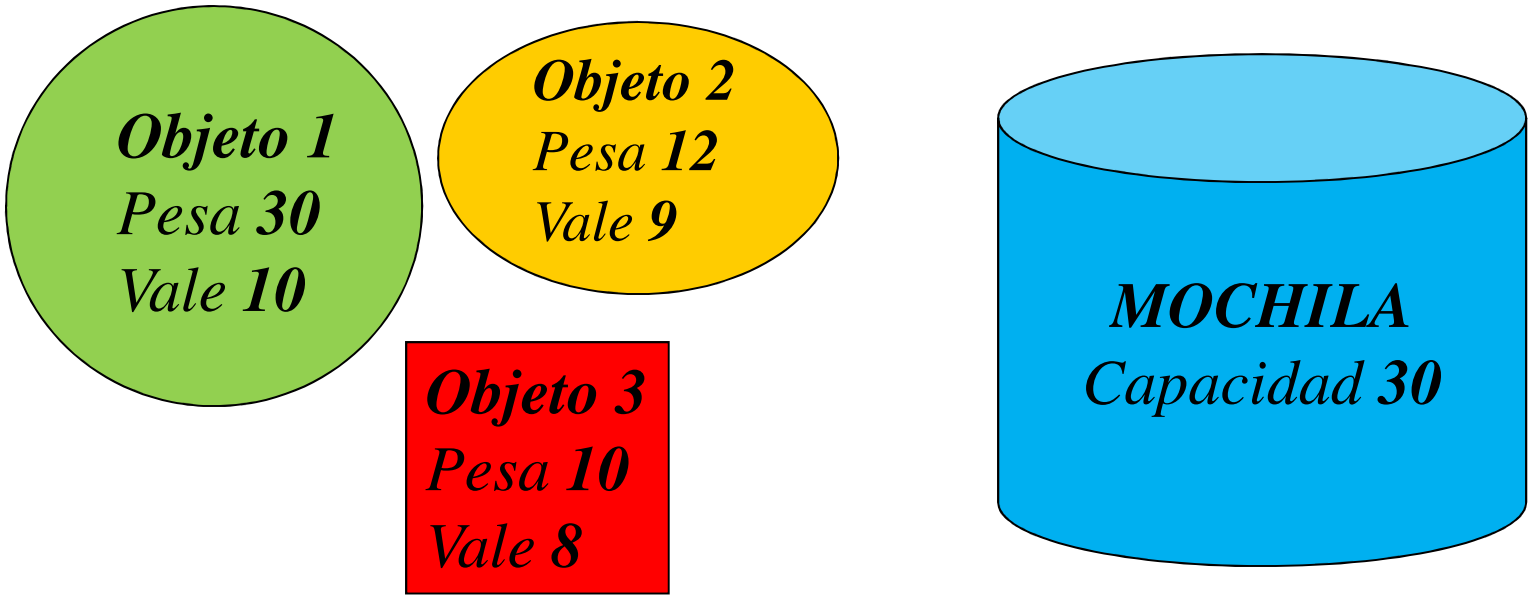
- Intuitivamente,
  ¿cómo llenarías la mochila?
  - ✗ Seleccionando primero a los de **mayor peso**?
  - ✗ Seleccionando primero a los de **menor valor**?
  - ✓ Seleccionando primero a los de **mayor valor**?
  - ✓ Seleccionando primero a los de **menor peso**?

Seleccionando el valor mayor...



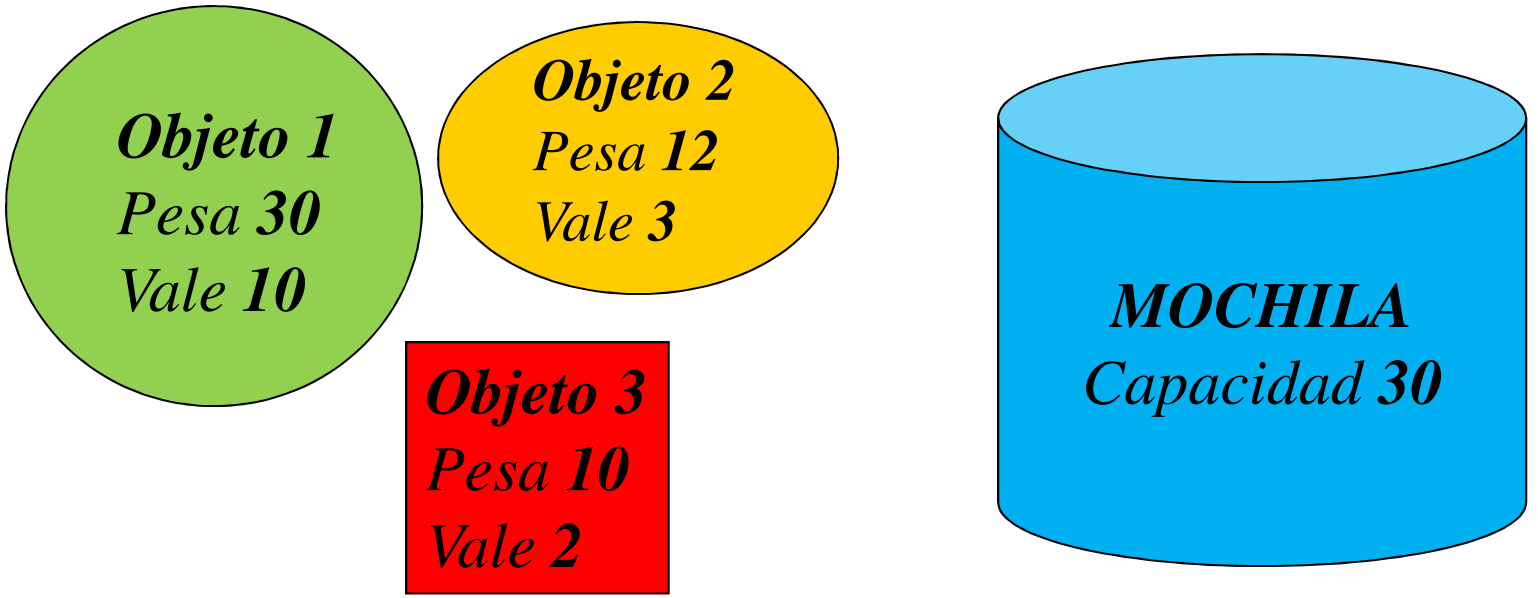
- Solución: *Objeto 1*, acumula **10**
- Solución óptima: *Objetos 2* y *3*, acumulan **17**

Seleccionando el peso menor...



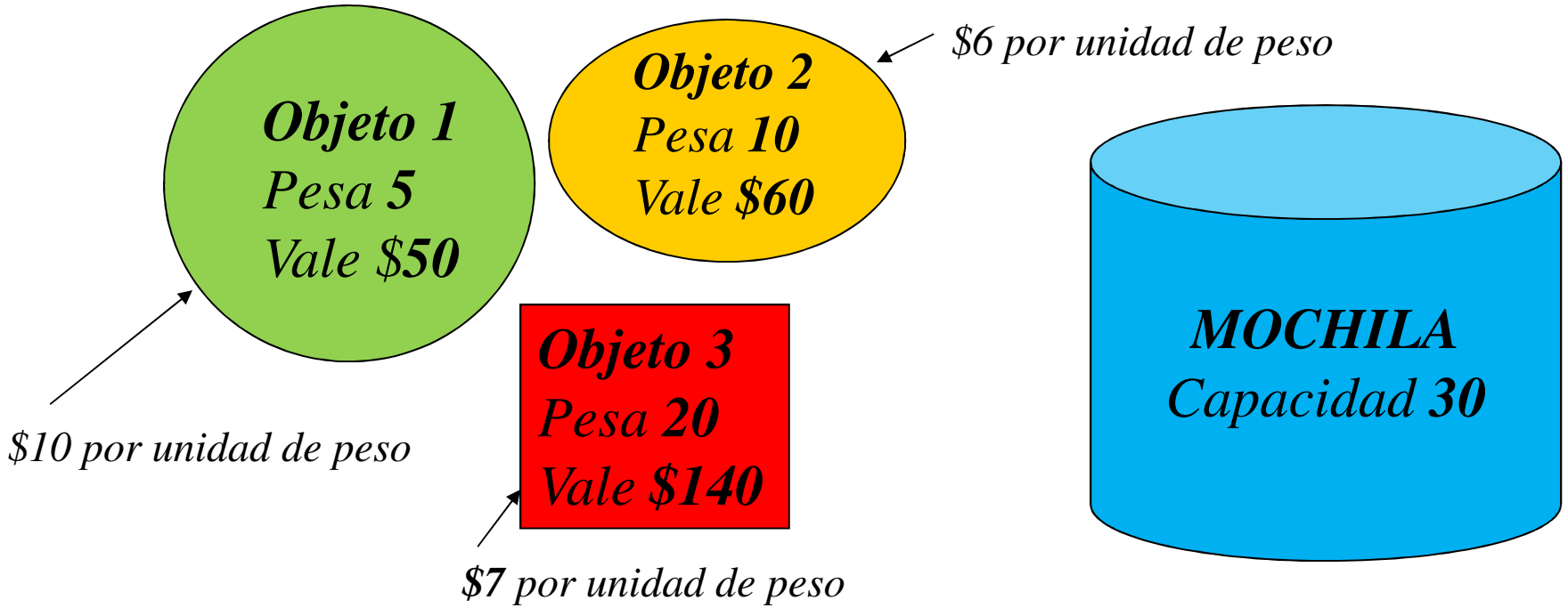
- Solución: *Objetos 2* y *3*, acumulan **17**
- Solución óptima: *Objetos 2* y *3*, acumulan **17**

Seleccionando el peso menor...



- Solución: *Objetos 2* y *3*, acumulan **5**
- Solución óptima: *Objeto 1*, acumula **10**

NUEVA PROPUESTA: Seleccionar el valor mayor por unidad de peso



- Solución: *Objetos 1* y *3*, acumulan **$190**
- Solución óptima: *Objetos 2* y *3*, acumulan **$200**

Conclusión (parcial)...

- No hay una solución "greedy" que resuelva el problema... (por contraejemplo se demostró)...
- ***¿Qué pasaría si los objetos se pueden fraccionar?***
- La mochila se puede llenar en forma óptima, utilizando la última propuesta de selección (valor mayor por unidad de peso), y llenando la mochila con la fracción correspondiente al siguiente objeto...
- En este caso el algoritmo voraz funciona... y tiene un comportamiento de $\mathbf{O(n \log_2 n)}$.

Solución con Programación dinámica

- ***¿Aplica el principio de optimalidad?***
- Sea **A** el subconjunto de objetos que maximiza el valor acumulado en la mochila (solución al problema).
- Si **A** NO contiene al *objeto*$_n$, **A** es igual a la solución óptima si el problema tuviera sólo a los primeros *n-1* objetos.
- Si **A** contiene al *objeto*$_n$, **A** el valor total acumulado es igual a v_n más el valor óptimo del subconjunto formado por los primeros *n-1* objetos, con la restricción de que el peso no exceda a $P-p_n$.

Solución con Programación dinámica

- ***¿Cuál es el planteamiento de solución recursiva?***
- Sea $V_{i,p}$ el valor acumulado óptimo para los objetos de 1 a i, sin exceder al peso p.
- La solución a encontrar es $V_{n,P}$... la cual puede obtenerse de la siguiente forma:

$$V_{n,P} = V_{n-1,P} \; si \; p_n > P$$

$$V_{n,P} = máximo \; entre \; V_{n-1,P} \;\; y \;\; v_n + V_{n-1,P-p_n} \; si \; p_n <= P$$

- A su vez esto se generaliza para cualquier i y cualquier p.

Implementación

- Definir una matriz **V** de 0 a n renglones y de 0 a P columnas…
- El renglón 0 y la columa 0 se incializan con 0…
- El cálculo se realiza renglón por renglón de acuerdo a la siguiente fórmula:

$$V[i,p] = V[i-1,p] \ si \ p_i > p$$

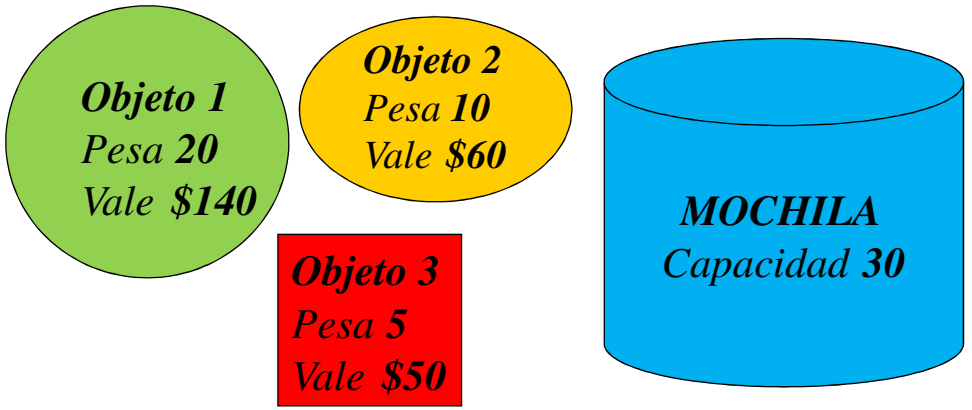
$$V[i,p] = máximo(V[i-1,p] \ , \ v_i + V[i-1,p-p_i]) \ si \ p_i <= p$$

- Comportamiento del algoritmo: **O(nP)**.

Análisis del algoritmo

- Para el caso de 3 objetos y una mochila con capacidad de 30 unidades, se tendría una matriz de 3 X 30 para encontrar el elemento V[3,30]…
- Cuando el valor de P sea muy grande, la eficiencia del algoritmo se verá afectada… y pudiera llegar a ser mejor la solución de la fuerza bruta!!
- Analizando la definición recursiva, el cálculo de un valor requiere sólo ciertos valores del renglón anterior, por lo que una mejora al algoritmo, implica sólo calcular los valores que se utilizarán en el siguiente renglón...

Ejemplo

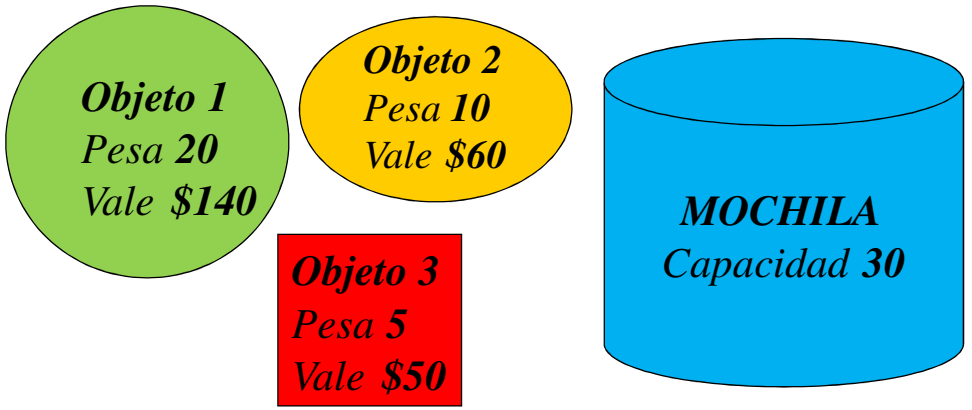


- La solución se encuentra en V[3,30]…

$V[3,30] = \mathbf{V[2,30]}$ *si* $5 > 30$

$V[3,30] = máximo(\mathbf{V[2,30]}, \$50 + \mathbf{V[2,25]})$ *si* $5 <= 30$

Ejemplo



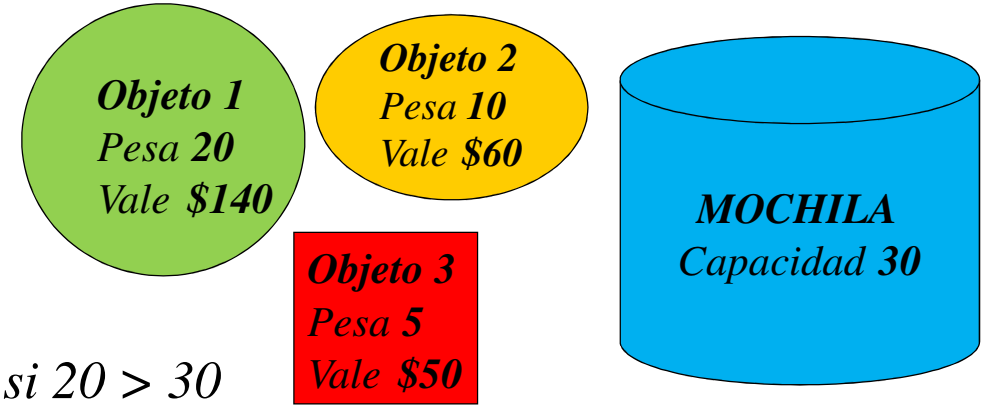
*V[2,30] = **V[1,30]** si 10 > 30*

*V[2,30] = máximo(**V[1,30]** , $60 + **V[1,20]**) si 10 <= 30*

*V[2,25] = **V[1,25]** si 10 > 25*

*V[2,25] = máximo(**V[1,25]** , $60 + **V[1,15]**) si 10 <= 25*

Ejemplo



$V[1,30] = \mathbf{V[0,30]}$ si $20 > 30$

$V[1,30] = máximo(\mathbf{V[0,30]}, \$140 + \mathbf{V[0,10]})$ si $20 <= 30$

$V[1,25] = \mathbf{V[0,25]}$ si $20 > 25$

$V[1,25] = máximo(\mathbf{V[0,25]}, \$140 + \mathbf{V[0,5]})$ si $20 <= 25$

$V[1,20] = \mathbf{V[0,20]}$ si $20 > 20$

$V[1,20] = máximo(\mathbf{V[0,20]}, \$140 + \mathbf{V[0,0]})$ si $20 <= 20$

$V[1,15] = \mathbf{V[0,15]}$ si $20 > 15$

$V[1,15] = máximo(\mathbf{V[0,15]}, \$140 + \mathbf{V[0,-5]})$ si $20 <= 15$

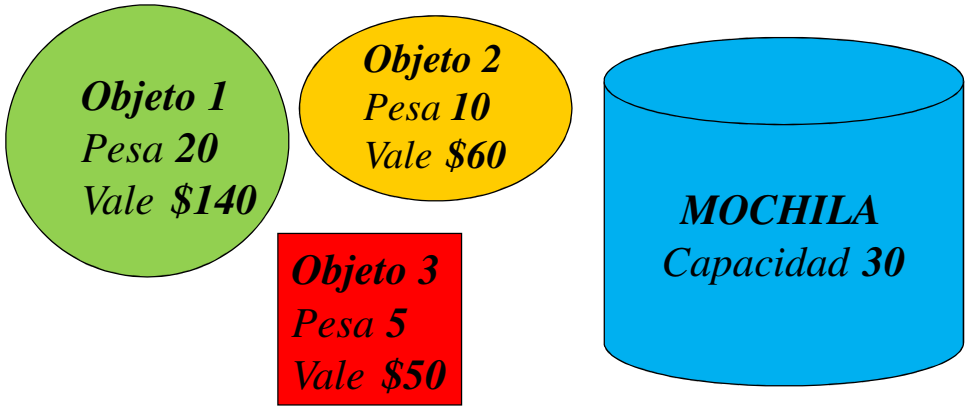
V[1,30] = 140

V[1,25] = 140

V[1,20] = 140

V[1,15] = 0

Ejemplo



V[1,30] = 140

V[1,25] = 140

V[1,20] = 140

V[1,15] = 0

V[2,30] = **V[1,30]** si 10 > 30

V[2,30] = máximo(**V[1,30]** , $60 + **V[1,20]**) si 10 <= 30

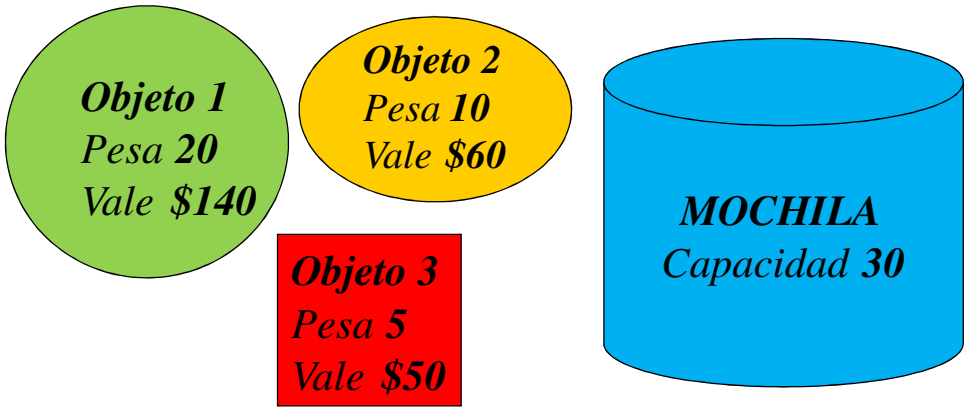
V[2,25] = **V[1,25]** si 10 > 25

V[2,25] = máximo(**V[1,25]** , $60 + **V[1,15]**) si 10 <= 25

V[2,30] = 200

V[2,25] = 140

Ejemplo



V[2,30] = 200

V[2,25] = 140

V[3,30] = ***V[2,30]*** *si 5 > 30*

*V[3,30] = máximo(****V[2,30]*** *, $50 +* ***V[2,25]****) si 5 <= 30*

V[3,30] = 200

7 cálculos vs. 90 cálculos sin la optimización

Por lo tanto, aplicar la optimización cuando 2^n sea menor a nP

Conclusión final…

- El problema de la mochila (sin objetos fraccionados), se puede resolver por medio de la programación dinámica con un comportamiento de **O(minimo(2^n, nP))**…
- Por divide y vencerás tendría un comportamiento de **O(2^n)**…
- Un algoritmo voraz NO resuelve el problema…
- Después de analizará el problema con otras técnicas…
- Pero hasta ahora no se ha demostrado, ni se ha encontrado, un algoritmo que resuelva el problema con un comportamiento mejor a **O(2^n)** para el peor caso…