Clase final :(

mat-151

Algoritmos glotones

- **No** hay manera de garantizar que un algoritmo glotón resuelva un problema de optimización dado, pero hay dos ingredientes fundamentales:

Algoritmos glotones

- **No** hay manera de garantizar que un algoritmo glotón resuelva un problema de optimización dado, pero hay dos ingredientes fundamentales:
  - Propiedad de la elección glotona.

Algoritmos glotones

- **No** hay manera de garantizar que un algoritmo glotón resuelva un problema de optimización dado, pero hay dos ingredientes fundamentales:
  - Propiedad de la elección glotona.
  - Subestructura optima.

Algoritmos glotones: **propiedad de la elección glotona**

- Se puede llegar a una solución óptima global haciendo solamente elecciones óptimas localmente.

Algoritmos glotones: **propiedad de la elección glotona**

- Se puede llegar a una solución óptima global haciendo solamente elecciones óptimas localmente.
- Al considerar que elección hacer, se realiza la mejor para el subproblema actual sin considerar los resultados de subproblemas anteriores.

Algoritmos glotones: **propiedad de la elección glotona**

- Se puede llegar a una solución óptima global haciendo solamente elecciones óptimas localmente.
- Al considerar que elección hacer, se realiza la mejor para el subproblema actual sin considerar los resultados de subproblemas anteriores.
- La propiedad de la elección glotona nos hace ser más eficientes al realizar las elecciones para los subproblemas.

Algoritmos glotones: **propiedad de la elección glotona**

- Se puede llegar a una solución óptima global haciendo solamente elecciones óptimas localmente.
- Al considerar que elección hacer, se realiza la mejor para el subproblema actual sin considerar los resultados de subproblemas anteriores.
- La propiedad de la elección glotona nos hace ser más eficientes al realizar las elecciones para los subproblemas.
- Es frecuente que preprocesando la información (por ejemplo, ordenar los datos) o utilizando una estructura de datos apropiada (generalmete una cola de prioridad) podamos hacer la elección glotona muy rápidamente, lo que resulta en un algoritmo eficiente.

Algoritmos glotones: **subestructura óptima**

- Cuando la solución óptima al problema contiene las soluciones óptimas a subproblemas.

Algoritmos glotones: **subestructura óptima**

- Cuando la solución óptima al problema contiene las soluciones óptimas a subproblemas.
- Propiedad fundamental para algoritmos de programación dinámica y glotones.

Algoritmos glotones: **subestructura óptima**

- Cuando la solución óptima al problema contiene las soluciones óptimas a subproblemas.
- Propiedad fundamental para algoritmos de programación dinámica y glotones.
- Ejemplo: Si la solución óptima al subproblema S_{ij} incluye una actividad a_k, entonces debe contener también las soluciones óptimas a los subproblemas S_{ik} y S_{kj}.

Algoritmos glotones: **subestructura óptima**

- Cuando la solución óptima al problema contiene las soluciones óptimas a subproblemas.
- Propiedad fundamental para algoritmos de programación dinámica y glotones.
- Ejemplo: Si la solución óptima al subproblema S_{ij} incluye una actividad a_k, entonces debe contener también las soluciones óptimas a los subproblemas S_{ik} y S_{kj}.
- En los algoritmos glotones se puede argumentar además que la solución óptima se encuentra combinando la elección glotona y la solución óptima a un solo subproblema.

Algoritmos glotones: **subestructura óptima**

- Cuando la solución óptima al problema contiene las soluciones óptimas a subproblemas.
- Propiedad fundamental para algoritmos de programación dinámica y glotones.
- Ejemplo: Si la solución óptima al subproblema S_{ij} incluye una actividad a_k, entonces debe contener también las soluciones óptimas a los subproblemas S_{ik} y S_{kj}.
- En los algoritmos glotones se puede argumentar además que la solución óptima se encuentra combinando la elección glotona y la solución óptima a un solo subproblema.
- Por inducción se debe mostrar que la elección glotona en cada subproblema nos lleva a la solución globalmente óptima.

Algoritmos glotones: **problema de la mochila**

- Regresamos al problema de la mochila:

Algoritmos glotones: **problema de la mochila**

- Regresamos al problema de la mochila:
  - problema 0-1 de la mochila (0-1 knapsack problem)

Algoritmos glotones: **problema de la mochila**

- Regresamos al problema de la mochila:
  - problema 0-1 de la mochila (0-1 knapsack problem)
    - un ladrón robando una tienda encuentra n objetos; el i-ésimo objeto vale v_i pesos y pesa w_i kilogramos (v_i y w_i generalmente son enteros). Se busca determinar los objetos que debe llevar el ladrón para maximizar el valor de su carga mientras mantiene la restricción de peso W. 0-1 porque los objetos se toman o se dejan.

Algoritmos glotones: **problema de la mochila**

- Regresamos al problema de la mochila:
  - problema 0-1 de la mochila (0-1 knapsack problem)
    - un ladrón robando una tienda encuentra n objetos; el i-ésimo objeto vale v_i pesos y pesa w_i kilogramos (v_i y w_i generalmente son enteros). Se busca determinar los objetos que debe llevar el ladrón para maximizar el valor de su carga mientras mantiene la restricción de peso W. 0-1 porque los objetos se toman o se dejan.
  - problema fraccionario de la mochila (fractional knapsack problem)

Algoritmos glotones: **problema de la mochila**

- Regresamos al problema de la mochila:
  - problema 0-1 de la mochila (0-1 knapsack problem)
    - un ladrón robando una tienda encuentra n objetos; el i-ésimo objeto vale v_i pesos y pesa w_i kilogramos (v_i y w_i generalmente son enteros). Se busca determinar los objetos que debe llevar el ladrón para maximizar el valor de su carga mientras mantiene la restricción de peso W. 0-1 porque los objetos se toman o se dejan.
  - problema fraccionario de la mochila (fractional knapsack problem)
    - mismo escenario excepto que el ladrón puede tomar fracciones de los objetos.

Algoritmos glotones: **problema de la mochila**

- Ambos problemas tienen subestructura optima,

Algoritmos glotones: **problema de la mochila**

- Ambos problemas tienen subestructura optima,
  - para el problema 0-1, consideramos la carga más valiosa que cumpla la restricción de peso W. Si quitamos el elemento j de la carga, la carga restante tiene un peso de a lo más $W-w_j$ para llenar con n-1 objetos.

Algoritmos glotones: **problema de la mochila**

- Ambos problemas tienen subestructura optima,
  - para el problema 0-1, consideramos la carga más valiosa que cumpla la restricción de peso W. Si quitamos el elemento j de la carga, la carga restante tiene un peso de a lo más $W-w_j$ para llenar con n-1 objetos.
  - para el problema fraccionario, si se retira un peso w del objeto j de la carga óptima, la carga restante debe ser la más valiosa tal que pese a lo más W-w y que el ladrón pueda llevar de los n-1 objetos.

Algoritmos glotones: **problema de la mochila**

- Ambos problemas tienen subestructura optima,
  - para el problema 0-1, consideramos la carga más valiosa que cumpla la restricción de peso W. Si quitamos el elemento j de la carga, la carga restante tiene un peso de a lo más $W-w_j$ para llenar con n-1 objetos.
  - para el problema fraccionario, si se retira un peso w del objeto j de la carga óptima, la carga restante debe ser la más valiosa tal que pese a lo más W-w y que el ladrón pueda llevar de los n-1 objetos.
- Aunque los problemas 0-1 y fraccionario son muy parecidos, el segundo se puede resolver óptimamente con una estrategia glotona y el primero no.

Algoritmos glotones: **problema de la mochila**

Algoritmos glotones: **problema de la mochila**

- Para resolver el problema fraccionario, se calcula primero el valor por kilo v_i/w_i para cada objeto.

Algoritmos glotones: **problema de la mochila**

- Para resolver el problema fraccionario, se calcula primero el valor por kilo v_i/w_i para cada objeto.
- Siguiendo una estrategia glotona,

Algoritmos glotones: **problema de la mochila**

- Para resolver el problema fraccionario, se calcula primero el valor por kilo v_i/w_i para cada objeto.
- Siguiendo una estrategia glotona,
  1. el ladrón pone en la mochila lo máximo posible del objeto con mayor valor por kilo.

Algoritmos glotones: **problema de la mochila**

- Para resolver el problema fraccionario, se calcula primero el valor por kilo v_i/w_i para cada objeto.
- Siguiendo una estrategia glotona,

1. el ladrón pone en la mochila lo máximo posible del objeto con mayor valor por kilo.
2. si se termina este objeto y queda lugar en la mochila toma el segundo objeto con mayor valor por kilo.

Algoritmos glotones: **problema de la mochila**

- Para resolver el problema fraccionario, se calcula primero el valor por kilo v_i/w_i para cada objeto.
- Siguiendo una estrategia glotona,

1. el ladrón pone en la mochila lo máximo posible del objeto con mayor valor por kilo.
2. si se termina este objeto y queda lugar en la mochila toma el segundo objeto con mayor valor por kilo.
3. se repite la estrategia hasta que se llene la mochila.

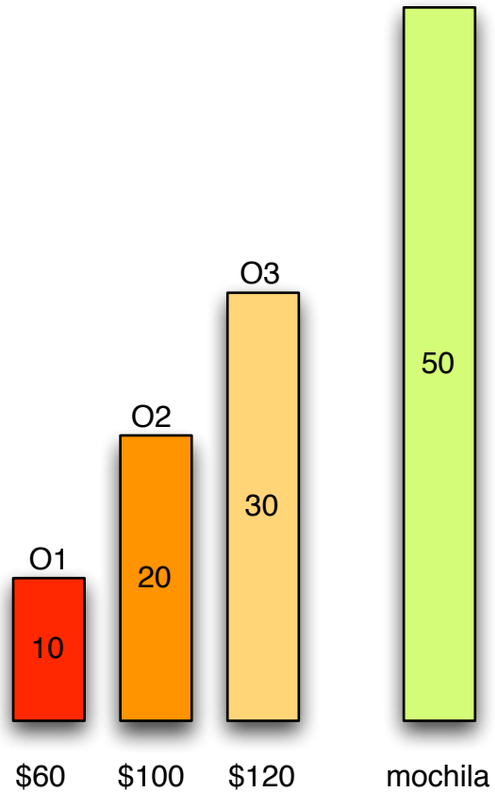
Algoritmos glotones: **problema de la mochila**

- Para resolver el problema fraccionario, se calcula primero el valor por kilo v_i/w_i para cada objeto.
- Siguiendo una estrategia glotona,

  1. el ladrón pone en la mochila lo máximo posible del objeto con mayor valor por kilo.
  2. si se termina este objeto y queda lugar en la mochila toma el segundo objeto con mayor valor por kilo.
  3. se repite la estrategia hasta que se llene la mochila.

- Ordenando los objetos respecto a su valor por kilo, el algoritmo glotón se ejecuta en $O(n \log n)$.

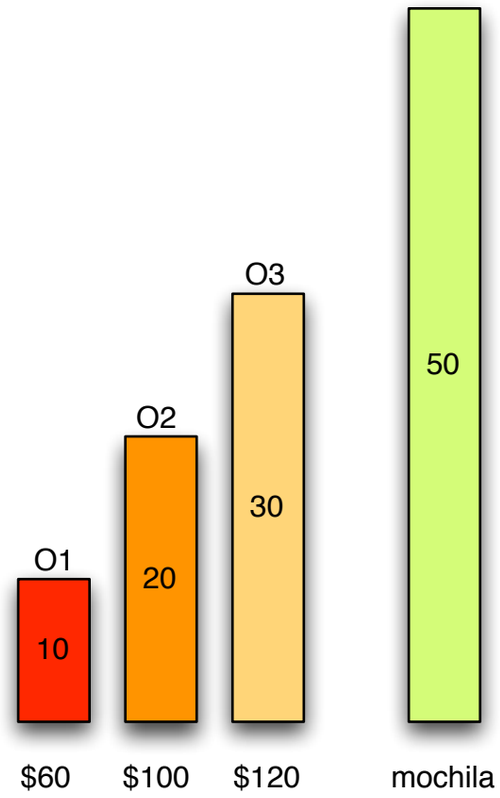
Algoritmos glotones: **problema de la mochila**



Problema 0-1

Algoritmos glotones: **problema de la mochila**

- ¿Funciona esta estrategia para el problema 0-1?¿por que?

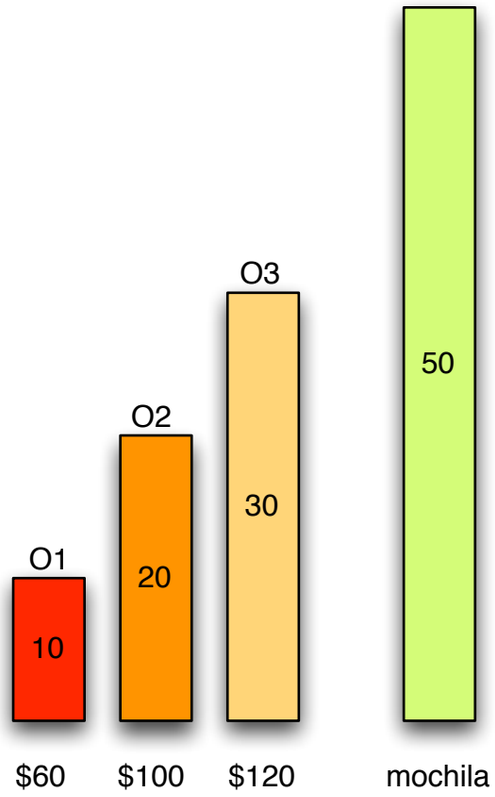


Problema 0-1

Algoritmos glotones: **problema de la mochila**

- ¿Funciona esta estrategia para el problema 0-1?¿por que?

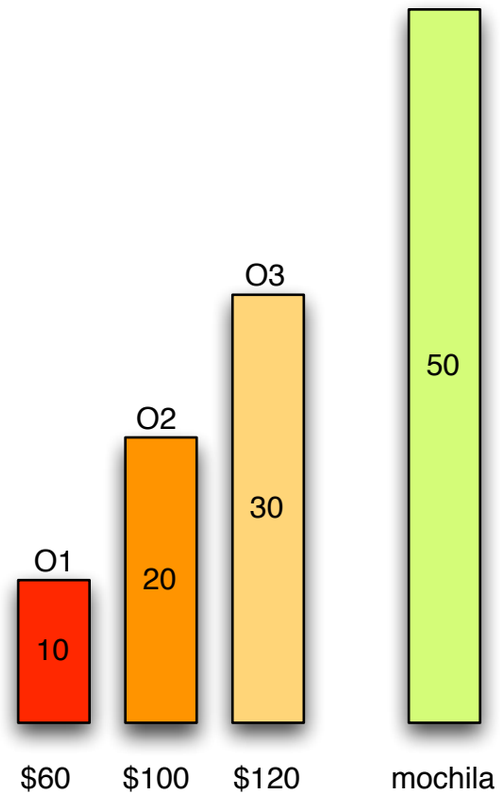
$$\frac{v_1}{w_1} = \frac{60}{10} = 6\ pesos/kilo$$



Problema 0-1

Algoritmos glotones: **problema de la mochila**

- ¿Funciona esta estrategia para el problema 0-1?¿por que?



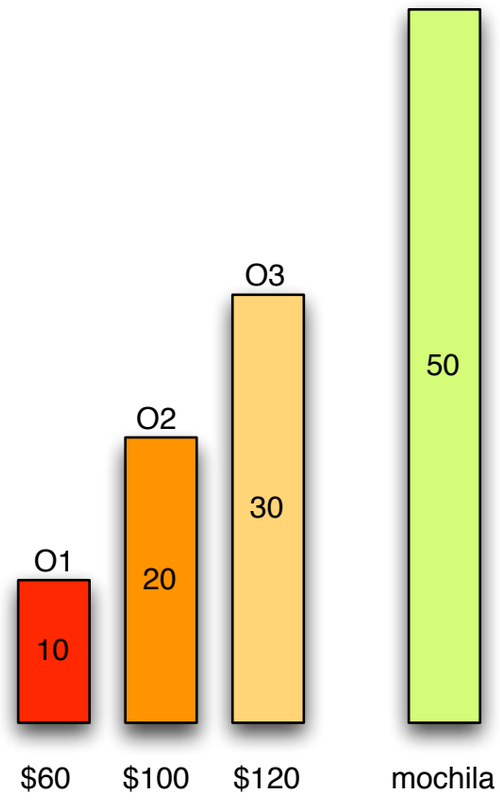
$$\frac{v_1}{w_1} = \frac{60}{10} = 6\ pesos/kilo$$

$$\frac{v_2}{w_2} = \frac{100}{20} = 5\ pesos/kilo$$

Problema 0-1

Algoritmos glotones: **problema de la mochila**

- ¿Funciona esta estrategia para el problema 0-1?¿por que?



$$\frac{v_1}{w_1} = \frac{60}{10} = 6\ pesos/kilo$$

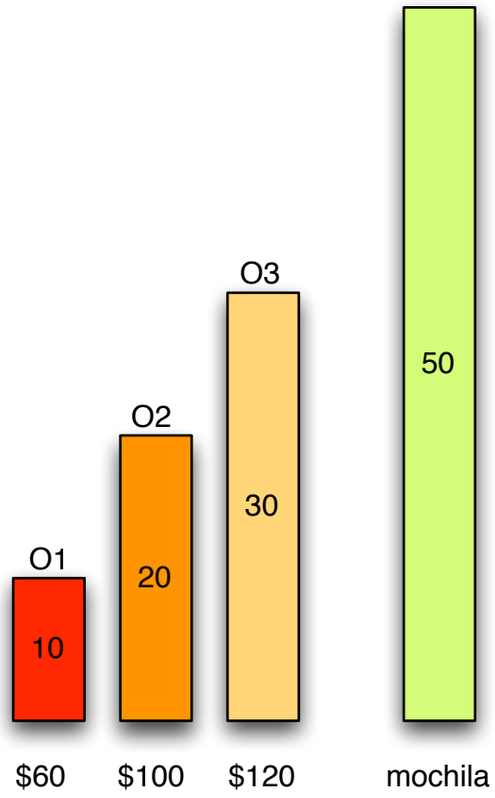
$$\frac{v_2}{w_2} = \frac{100}{20} = 5\ pesos/kilo$$

$$\frac{v_3}{w_3} = \frac{120}{30} = 4\ pesos/kilo$$

Problema 0-1

Algoritmos glotones: **problema de la mochila**

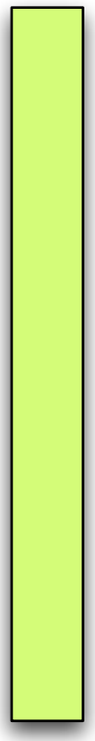
- ¿Funciona esta estrategia para el problema 0-1?¿por que?



$$\frac{v_1}{w_1} = \frac{60}{10} = 6\ pesos/kilo$$

$$\frac{v_2}{w_2} = \frac{100}{20} = 5\ pesos/kilo$$

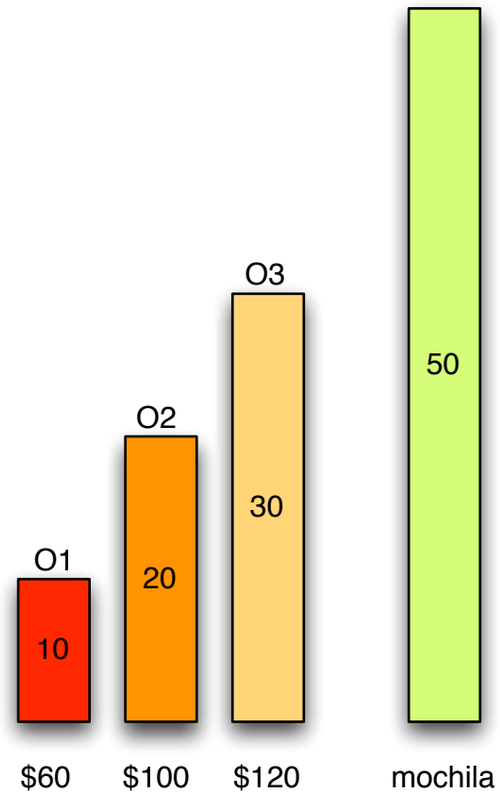
$$\frac{v_3}{w_3} = \frac{120}{30} = 4\ pesos/kilo$$



Problema 0-1

Algoritmos glotones: **problema de la mochila**

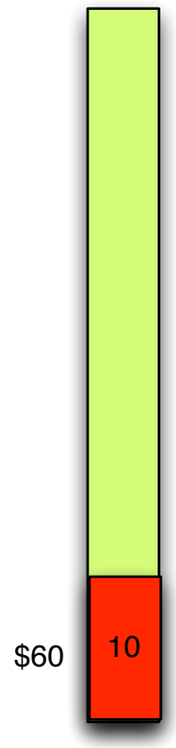
- ¿Funciona esta estrategia para el problema 0-1?¿por que?



$$\frac{v_1}{w_1} = \frac{60}{10} = 6\ pesos/kilo$$

$$\frac{v_2}{w_2} = \frac{100}{20} = 5\ pesos/kilo$$

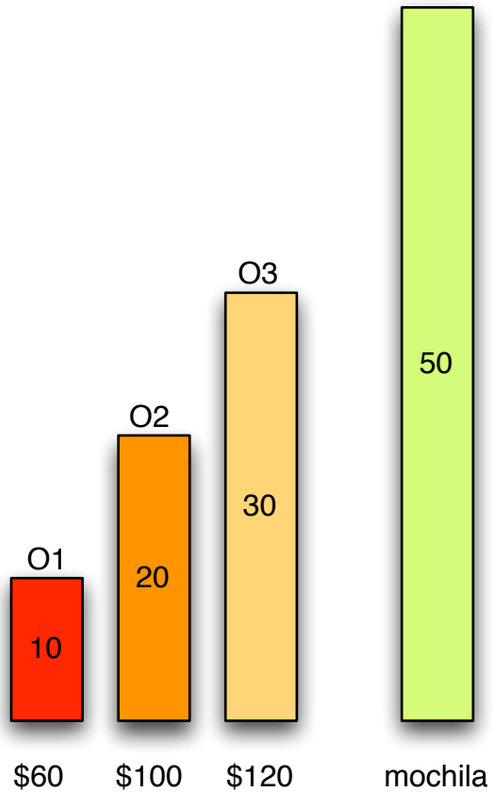
$$\frac{v_3}{w_3} = \frac{120}{30} = 4\ pesos/kilo$$



Problema 0-1

Algoritmos glotones: **problema de la mochila**

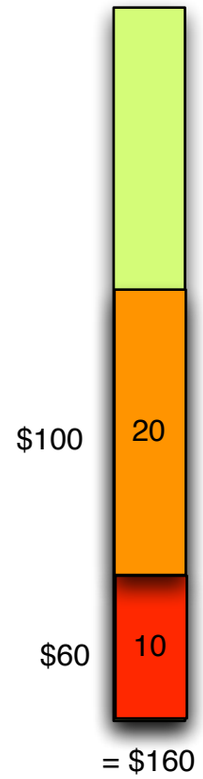
- ¿Funciona esta estrategia para el problema 0-1?¿por que?



$$\frac{v_1}{w_1} = \frac{60}{10} = 6\ pesos/kilo$$

$$\frac{v_2}{w_2} = \frac{100}{20} = 5\ pesos/kilo$$

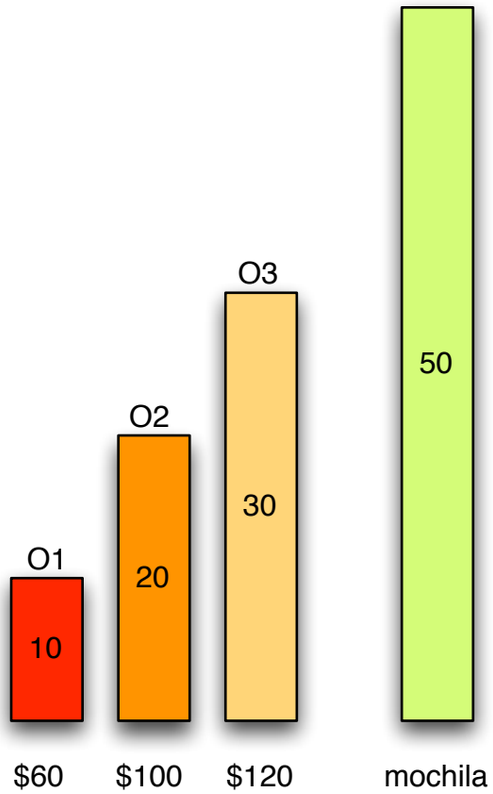
$$\frac{v_3}{w_3} = \frac{120}{30} = 4\ pesos/kilo$$



Problema 0-1

Algoritmos glotones: **problema de la mochila**

- ¿Funciona esta estrategia para el problema 0-1?¿por que?



$$\frac{v_1}{w_1} = \frac{60}{10} = 6\ pesos/kilo$$

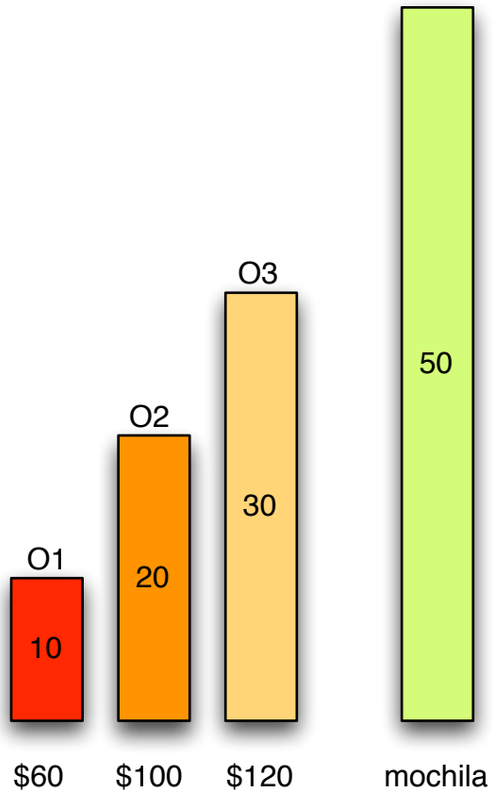
$$\frac{v_2}{w_2} = \frac{100}{20} = 5\ pesos/kilo$$

$$\frac{v_3}{w_3} = \frac{120}{30} = 4\ pesos/kilo$$

Problema 0-1

Algoritmos glotones: **problema de la mochila**

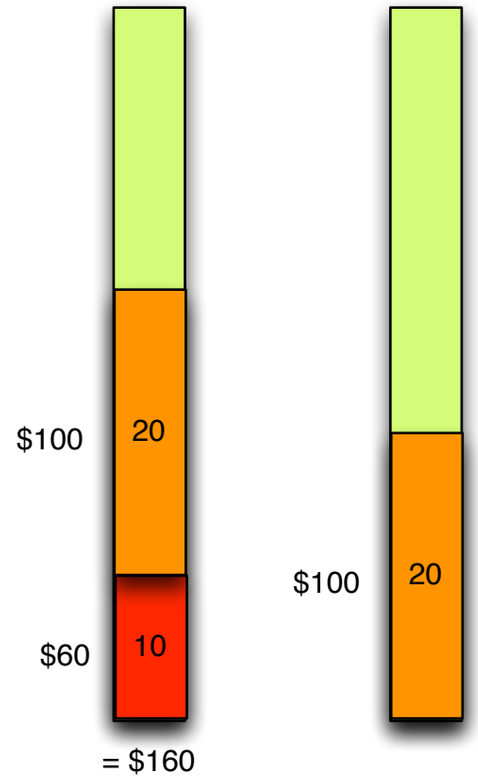
- ¿Funciona esta estrategia para el problema 0-1?¿por que?



$$\frac{v_1}{w_1} = \frac{60}{10} = 6\ pesos/kilo$$

$$\frac{v_2}{w_2} = \frac{100}{20} = 5\ pesos/kilo$$

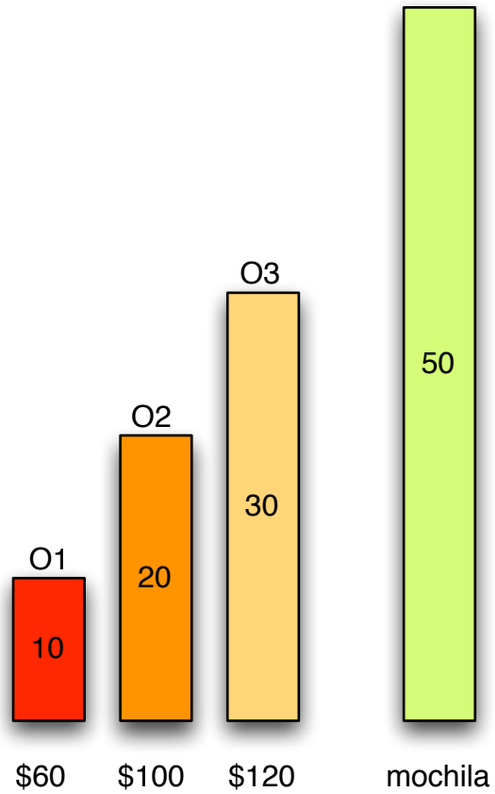
$$\frac{v_3}{w_3} = \frac{120}{30} = 4\ pesos/kilo$$



Problema 0-1

Algoritmos glotones: **problema de la mochila**

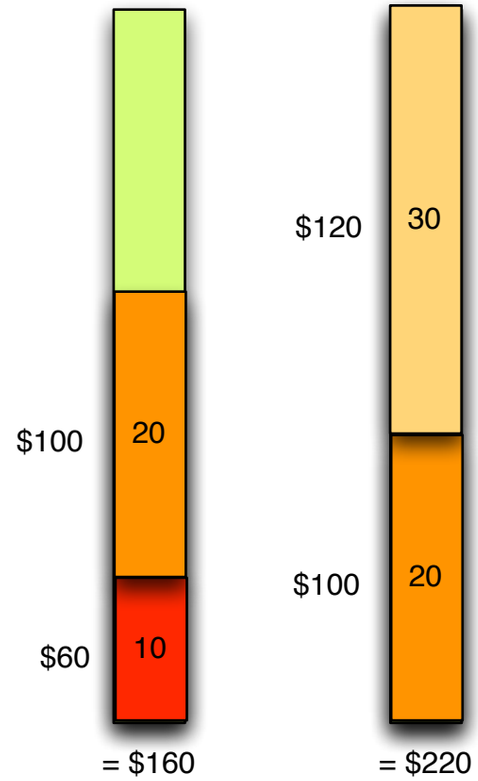
- ¿Funciona esta estrategia para el problema 0-1?¿por que?



$$\frac{v_1}{w_1} = \frac{60}{10} = 6\ pesos/kilo$$

$$\frac{v_2}{w_2} = \frac{100}{20} = 5\ pesos/kilo$$

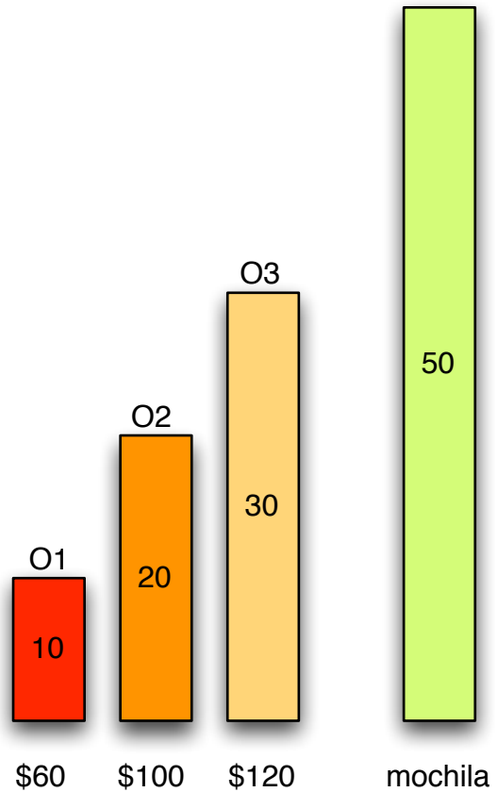
$$\frac{v_3}{w_3} = \frac{120}{30} = 4\ pesos/kilo$$



Problema 0-1

Algoritmos glotones: **problema de la mochila**

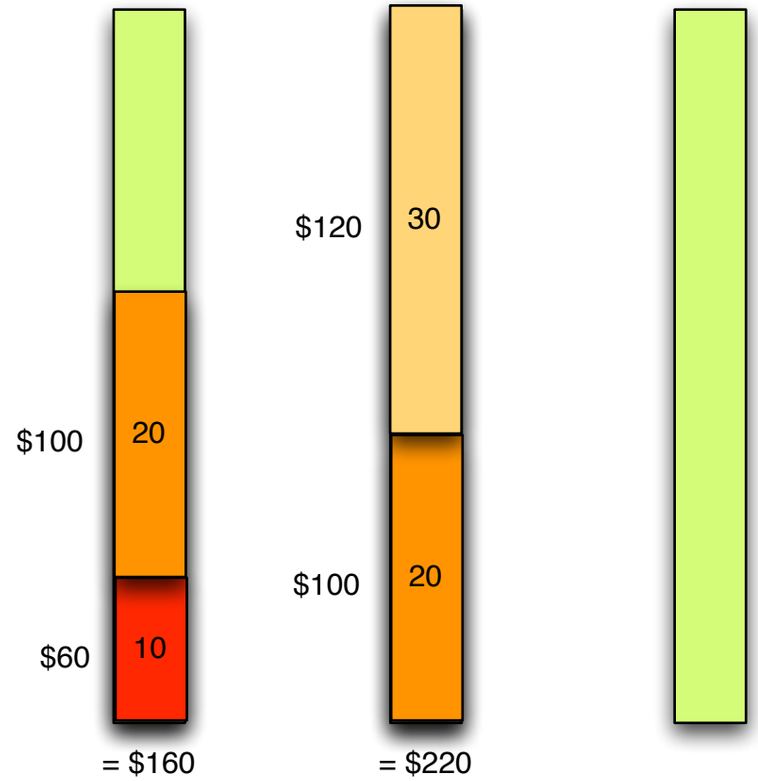
- ¿Funciona esta estrategia para el problema 0-1?¿por que?



$$\frac{v_1}{w_1} = \frac{60}{10} = 6\ pesos/kilo$$

$$\frac{v_2}{w_2} = \frac{100}{20} = 5\ pesos/kilo$$

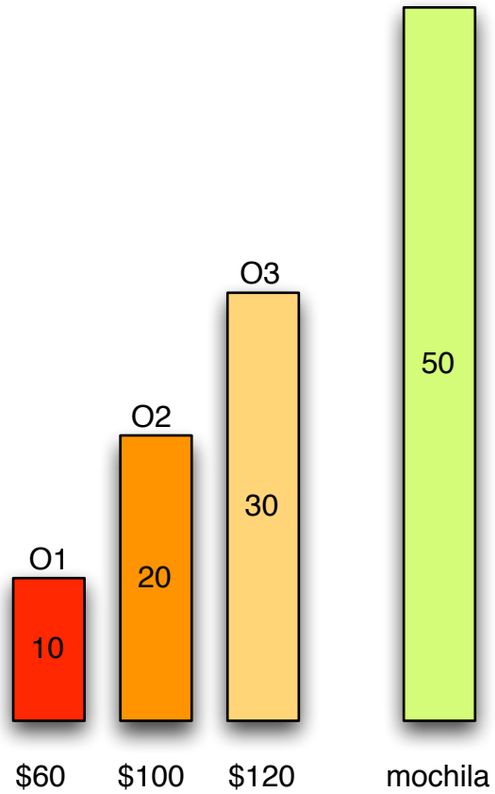
$$\frac{v_3}{w_3} = \frac{120}{30} = 4\ pesos/kilo$$



Problema 0-1

Algoritmos glotones: **problema de la mochila**

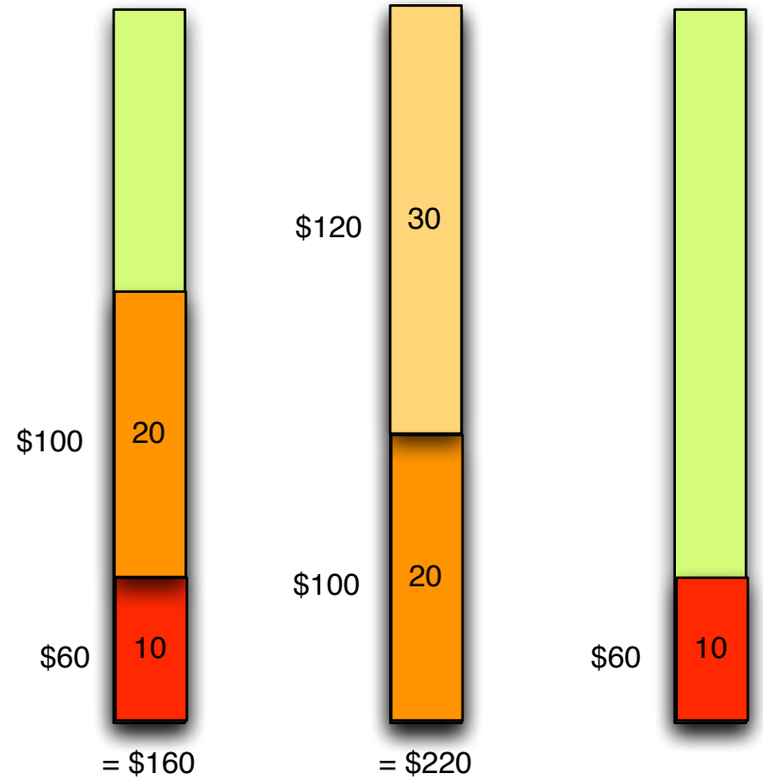
- ¿Funciona esta estrategia para el problema 0-1?¿por que?



$$\frac{v_1}{w_1} = \frac{60}{10} = 6\ pesos/kilo$$

$$\frac{v_2}{w_2} = \frac{100}{20} = 5\ pesos/kilo$$

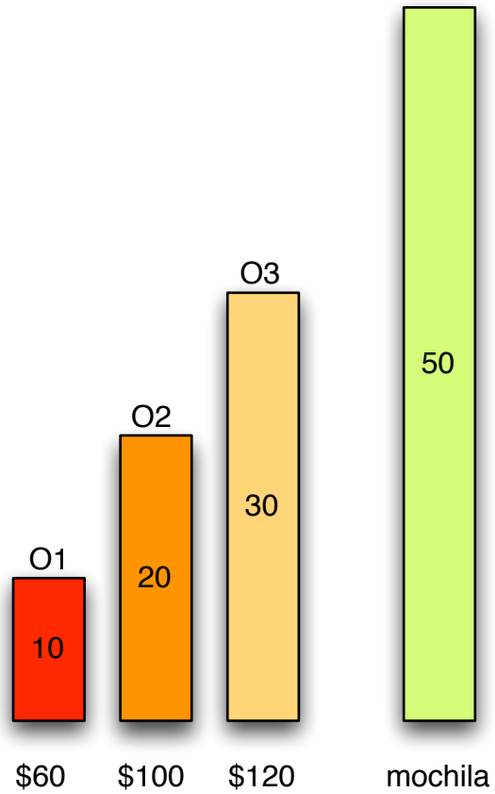
$$\frac{v_3}{w_3} = \frac{120}{30} = 4\ pesos/kilo$$



Problema 0-1

Algoritmos glotones: **problema de la mochila**

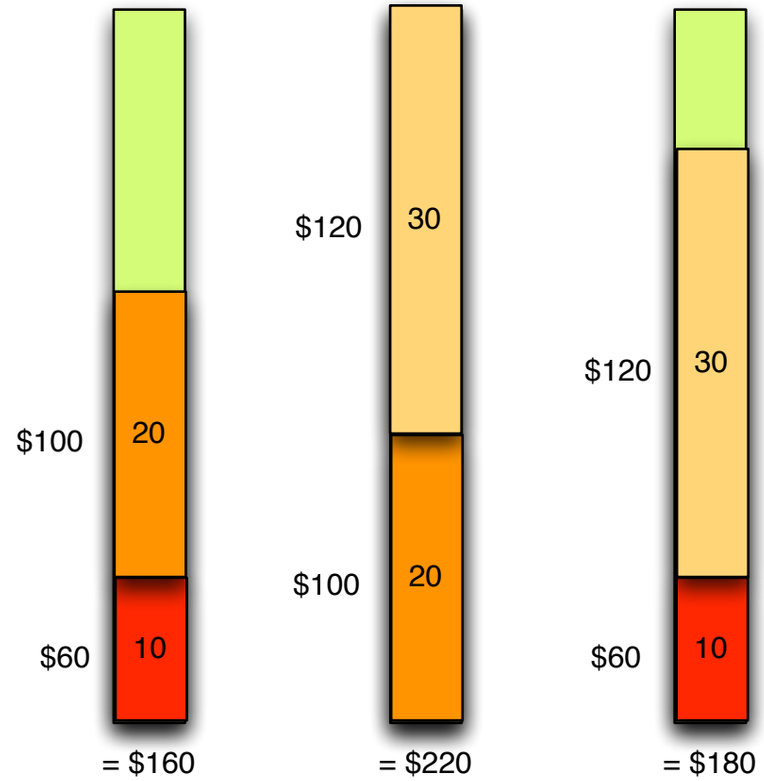
- ¿Funciona esta estrategia para el problema 0-1?¿por que?



$$\frac{v_1}{w_1} = \frac{60}{10} = 6\ pesos/kilo$$

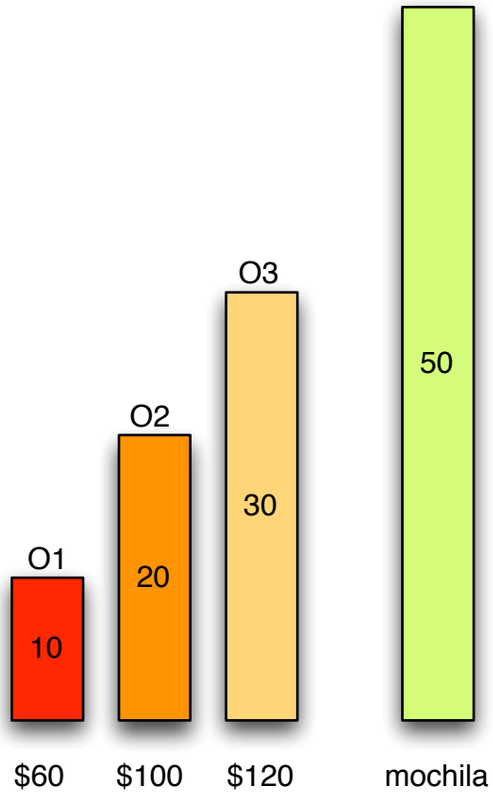
$$\frac{v_2}{w_2} = \frac{100}{20} = 5\ pesos/kilo$$

$$\frac{v_3}{w_3} = \frac{120}{30} = 4\ pesos/kilo$$



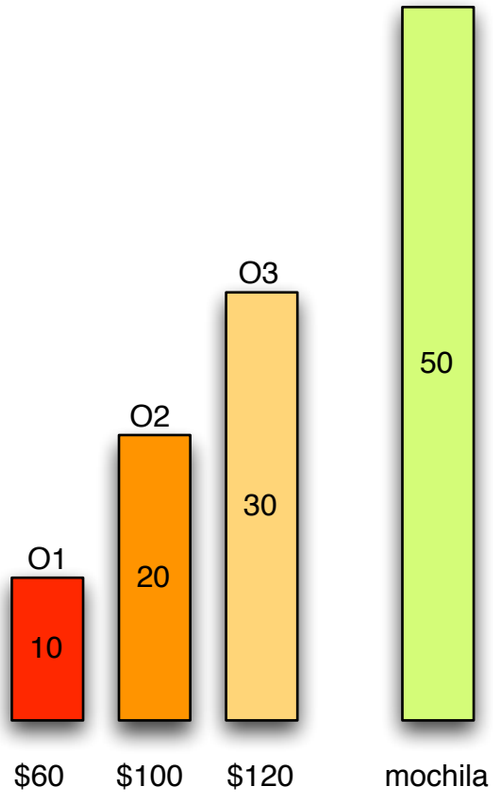
Problema 0-1

Algoritmos glotones: **problema de la mochila**



Problema fraccionario

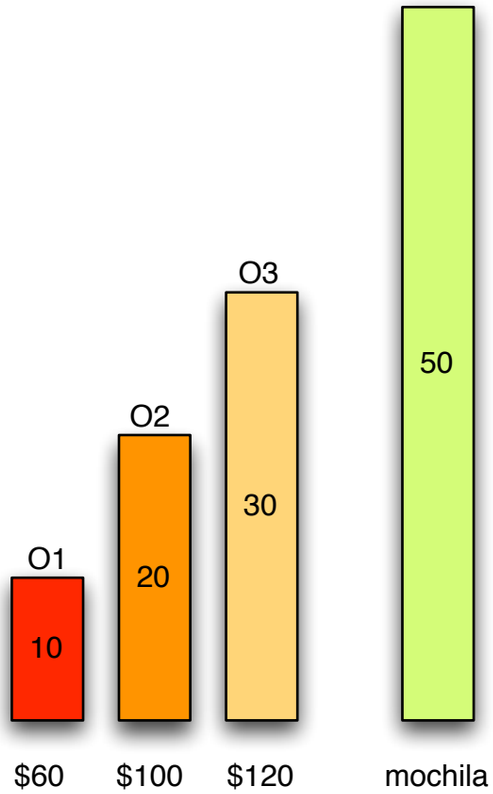
Algoritmos glotones: **problema de la mochila**



$$\frac{v_1}{w_1} = \frac{60}{10} = 6\ pesos/kilo$$

Problema fraccionario

Algoritmos glotones: **problema de la mochila**

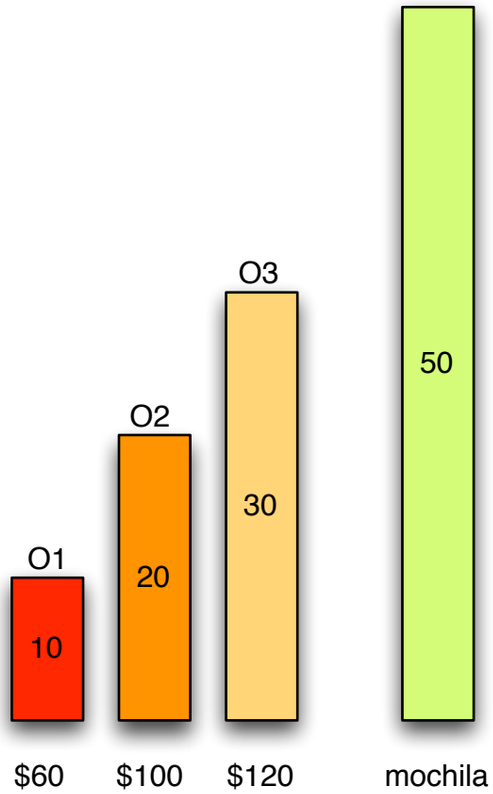


$$\frac{v_1}{w_1} = \frac{60}{10} = 6\ pesos/kilo$$

$$\frac{v_2}{w_2} = \frac{100}{20} = 5\ pesos/kilo$$

Problema fraccionario

Algoritmos glotones: **problema de la mochila**



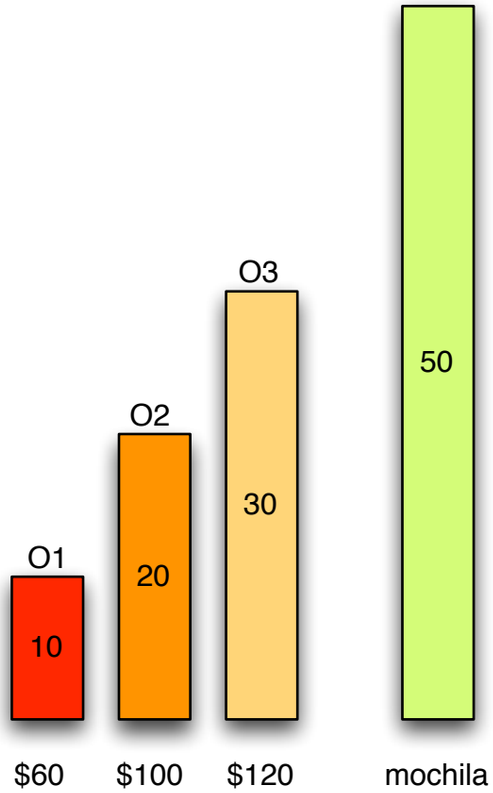
$$\frac{v_1}{w_1} = \frac{60}{10} = 6\ pesos/kilo$$

$$\frac{v_2}{w_2} = \frac{100}{20} = 5\ pesos/kilo$$

$$\frac{v_3}{w_3} = \frac{120}{30} = 4\ pesos/kilo$$

Problema fraccionario

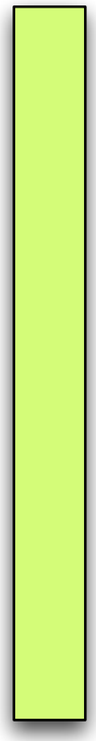
Algoritmos glotones: **problema de la mochila**



$$\frac{v_1}{w_1} = \frac{60}{10} = 6\ pesos/kilo$$

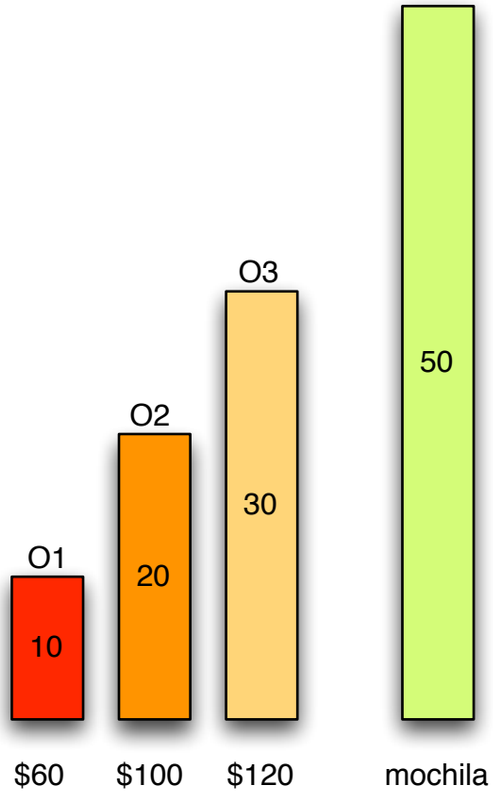
$$\frac{v_2}{w_2} = \frac{100}{20} = 5\ pesos/kilo$$

$$\frac{v_3}{w_3} = \frac{120}{30} = 4\ pesos/kilo$$



Problema fraccionario

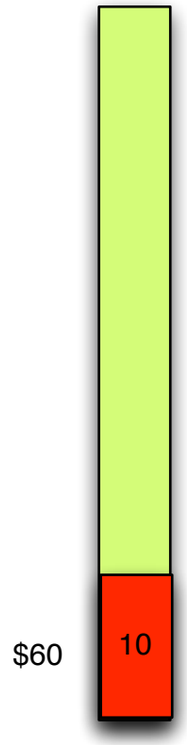
Algoritmos glotones: **problema de la mochila**



$$\frac{v_1}{w_1} = \frac{60}{10} = 6\ pesos/kilo$$

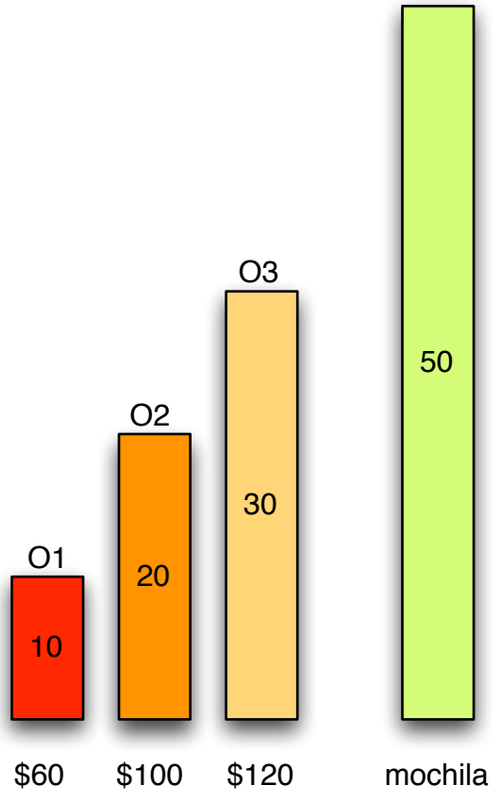
$$\frac{v_2}{w_2} = \frac{100}{20} = 5\ pesos/kilo$$

$$\frac{v_3}{w_3} = \frac{120}{30} = 4\ pesos/kilo$$



Problema fraccionario

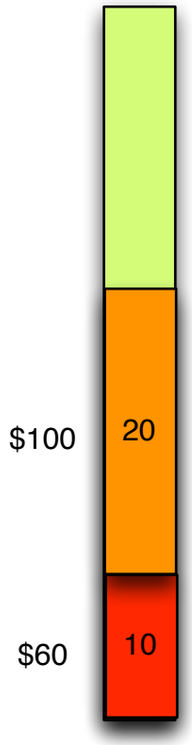
Algoritmos glotones: **problema de la mochila**



$$\frac{v_1}{w_1} = \frac{60}{10} = 6\ pesos/kilo$$

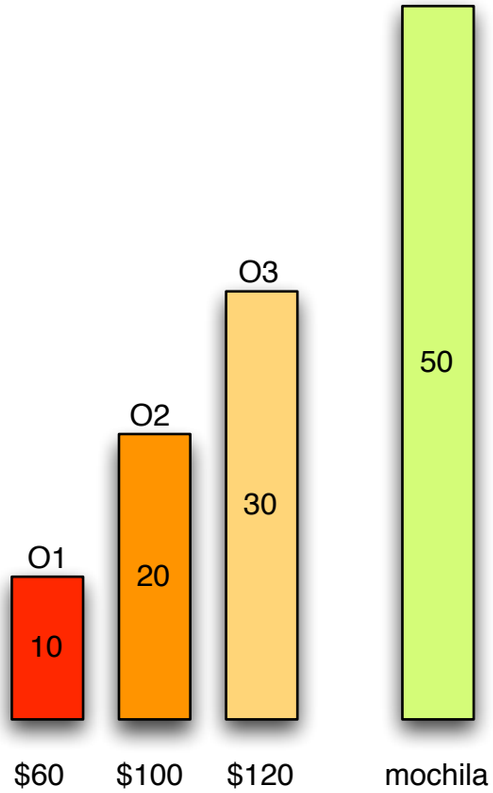
$$\frac{v_2}{w_2} = \frac{100}{20} = 5\ pesos/kilo$$

$$\frac{v_3}{w_3} = \frac{120}{30} = 4\ pesos/kilo$$



Problema fraccionario

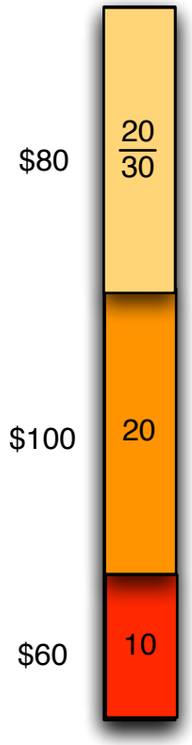
Algoritmos glotones: **problema de la mochila**



$$\frac{v_1}{w_1} = \frac{60}{10} = 6\ pesos/kilo$$

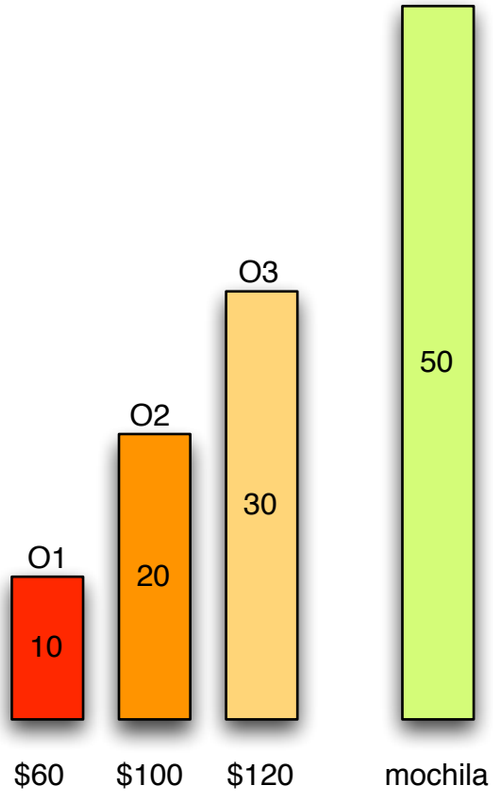
$$\frac{v_2}{w_2} = \frac{100}{20} = 5\ pesos/kilo$$

$$\frac{v_3}{w_3} = \frac{120}{30} = 4\ pesos/kilo$$



Problema fraccionario

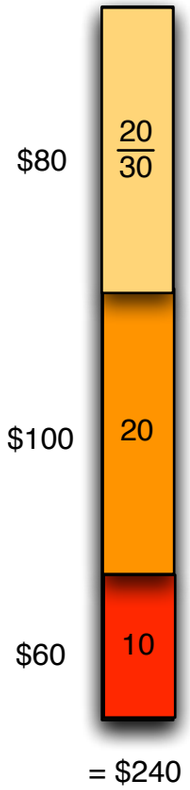
Algoritmos glotones: **problema de la mochila**



$$\frac{v_1}{w_1} = \frac{60}{10} = 6\ pesos/kilo$$

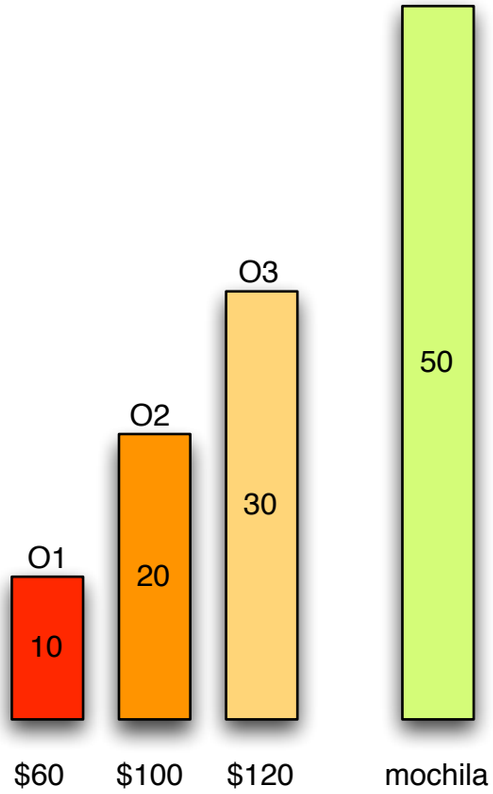
$$\frac{v_2}{w_2} = \frac{100}{20} = 5\ pesos/kilo$$

$$\frac{v_3}{w_3} = \frac{120}{30} = 4\ pesos/kilo$$



Problema fraccionario

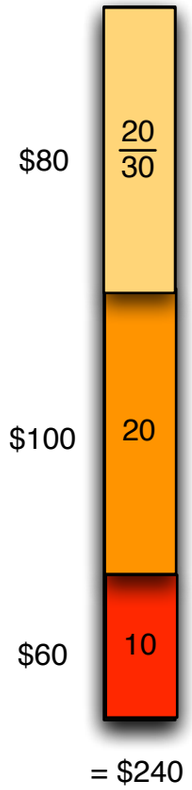
Algoritmos glotones: **problema de la mochila**



$$\frac{v_1}{w_1} = \frac{60}{10} = 6\ pesos/kilo$$

$$\frac{v_2}{w_2} = \frac{100}{20} = 5\ pesos/kilo$$

$$\frac{v_3}{w_3} = \frac{120}{30} = 4\ pesos/kilo$$



Problema fraccionario

- En el problema 0-1 se deben comparar la solución al subproblema donde el objeto es incluido y la solución cuando el objeto no ha sido incluido.

Estrategias para resolver recurrencias

- El *master theorem* para resolver recurrencias,

Estrategias para resolver recurrencias

- El *master theorem* para resolver recurrencias,

$$T(n) = a\,T\left(\frac{n}{b}\right) + f(n) \quad \text{where } a \geq 1,\, b > 1$$

Estrategias para resolver recurrencias

- El *master theorem* para resolver recurrencias,

$$T(n) = a\,T\left(\frac{n}{b}\right) + f(n) \quad \text{where } a \geq 1,\ b > 1,\ \text{and assuming } T(1) = 1$$

Estrategias para resolver recurrencias

- El *master theorem* para resolver recurrencias,

$$T(n) = a\,T\left(\frac{n}{b}\right) + f(n) \quad \text{where } a \geq 1,\ b > 1,\ \text{and assuming } T(1) = 1$$

Caso 1

Estrategias para resolver recurrencias

- El *master theorem* para resolver recurrencias,

$$T(n) = a\,T\left(\frac{n}{b}\right) + f(n) \quad \text{where } a \geq 1,\ b > 1,\ \text{and assuming } T(1) = 1$$

Caso 1

If $f(n) = O\left(n^{\log_b(a)-\epsilon}\right)$ for some constant $\epsilon > 0$

it follows that:

$$T(n) = \Theta\left(n^{\log_b a}\right)$$

Estrategias para resolver recurrencias

- El *master theorem* para resolver recurrencias,

$$T(n) = a\,T\left(\frac{n}{b}\right) + f(n) \quad \text{where } a \geq 1,\ b > 1,\ \text{and assuming } T(1) = 1$$

Caso 1

If $f(n) = O\left(n^{\log_b(a)-\epsilon}\right)$ for some constant $\epsilon > 0$

it follows that:

$$T(n) = \Theta\left(n^{\log_b a}\right)$$

Caso 2

Estrategias para resolver recurrencias

- El *master theorem* para resolver recurrencias,

$$T(n) = a\,T\left(\frac{n}{b}\right) + f(n) \quad \text{where } a \geq 1,\ b > 1,\ \text{and assuming } T(1) = 1$$

Caso 1

If $f(n) = O\left(n^{\log_b(a)-\epsilon}\right)$ for some constant $\epsilon > 0$

it follows that:

$$T(n) = \Theta\left(n^{\log_b a}\right)$$

Caso 2

If it is true, for some constant $k \geq 0$, that:

$$f(n) = \Theta\left(n^{\log_b a} \log^k n\right)$$

it follows that:

$$T(n) = \Theta\left(n^{\log_b a} \log^{k+1} n\right)$$

Estrategias para resolver recurrencias

- El *master theorem* para resolver recurrencias,

$$T(n) = a\,T\left(\frac{n}{b}\right) + f(n) \quad \text{where } a \geq 1,\, b > 1$$

Caso 3

If it is true that:

$$f(n) = \Omega\left(n^{\log_b(a)+\epsilon}\right) \text{ for some constant } \epsilon > 0$$

and if it is also true that:

$$af\left(\frac{n}{b}\right) \leq cf(n) \text{ for some constant } c < 1 \text{ and sufficiently large } n$$

it follows that:

$$T(n) = \Theta\left(f(n)\right)$$

Ejemplos

Algorithm	Recurrence Relationship	Run time	Comment
Binary search	$T(n) = T\left(\frac{n}{2}\right) + O(1)$	$O(\log n)$	Apply Master theorem case 2 $c = \log_b a$, where $a = 1, b = 2, c = 0, k = 0$[3]
Merge Sort	$T(n) = 2T\left(\frac{n}{2}\right) + O(n)$	$O(n \log n)$	Apply Master theorem case 2 $c = \log_b a$, where $a = 2, b = 2, c = 1, k = 0$

Ejemplos

Algorithm	Recurrence Relationship	Run time	Comment
Binary search	$T(n) = T\left(\frac{n}{2}\right) + O(1)$	$O(\log n)$	Apply Master theorem case 2 $c = \log_b a$, where $a = 1, b = 2, c = 0, k = 0$[3]
Merge Sort	$T(n) = 2T\left(\frac{n}{2}\right) + O(n)$	$O(n \log n)$	Apply Master theorem case 2 $c = \log_b a$, where $a = 2, b = 2, c = 1, k = 0$

Algoritmo de Strassen para multiplicación de matrices

- **Pensando de manera recursiva:**

Let *A*, *B* be two square matrices over a ring *R*. We want to calculate the matrix product *C* as

$$\mathbf{C} = \mathbf{AB} \qquad \mathbf{A}, \mathbf{B}, \mathbf{C} \in R^{2^n \times 2^n}$$

If the matrices *A*, *B* are not of type 2^n x 2^n we fill the missing rows and columns with zeros.

We partition *A*, *B* and *C* into equally sized block matrices

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_{1,1} & \mathbf{A}_{1,2} \\ \mathbf{A}_{2,1} & \mathbf{A}_{2,2} \end{bmatrix}, \mathbf{B} = \begin{bmatrix} \mathbf{B}_{1,1} & \mathbf{B}_{1,2} \\ \mathbf{B}_{2,1} & \mathbf{B}_{2,2} \end{bmatrix}, \mathbf{C} = \begin{bmatrix} \mathbf{C}_{1,1} & \mathbf{C}_{1,2} \\ \mathbf{C}_{2,1} & \mathbf{C}_{2,2} \end{bmatrix}$$

with

$$\mathbf{A}_{i,j}, \mathbf{B}_{i,j}, \mathbf{C}_{i,j} \in R^{2^{n-1} \times 2^{n-1}}$$

Algoritmo de Strassen para multiplicación de matrices

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_{1,1} & \mathbf{A}_{1,2} \\ \mathbf{A}_{2,1} & \mathbf{A}_{2,2} \end{bmatrix}, \mathbf{B} = \begin{bmatrix} \mathbf{B}_{1,1} & \mathbf{B}_{1,2} \\ \mathbf{B}_{2,1} & \mathbf{B}_{2,2} \end{bmatrix}, \mathbf{C} = \begin{bmatrix} \mathbf{C}_{1,1} & \mathbf{C}_{1,2} \\ \mathbf{C}_{2,1} & \mathbf{C}_{2,2} \end{bmatrix}$$

with

$$\mathbf{A}_{i,j}, \mathbf{B}_{i,j}, \mathbf{C}_{i,j} \in R^{2^{n-1} \times 2^{n-1}}$$

then

$$\mathbf{C}_{1,1} = \mathbf{A}_{1,1}\mathbf{B}_{1,1} + \mathbf{A}_{1,2}\mathbf{B}_{2,1}$$
$$\mathbf{C}_{1,2} = \mathbf{A}_{1,1}\mathbf{B}_{1,2} + \mathbf{A}_{1,2}\mathbf{B}_{2,2}$$
$$\mathbf{C}_{2,1} = \mathbf{A}_{2,1}\mathbf{B}_{1,1} + \mathbf{A}_{2,2}\mathbf{B}_{2,1}$$
$$\mathbf{C}_{2,2} = \mathbf{A}_{2,1}\mathbf{B}_{1,2} + \mathbf{A}_{2,2}\mathbf{B}_{2,2}$$

Algoritmo de Strassen para multiplicación de matrices

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_{1,1} & \mathbf{A}_{1,2} \\ \mathbf{A}_{2,1} & \mathbf{A}_{2,2} \end{bmatrix}, \mathbf{B} = \begin{bmatrix} \mathbf{B}_{1,1} & \mathbf{B}_{1,2} \\ \mathbf{B}_{2,1} & \mathbf{B}_{2,2} \end{bmatrix}, \mathbf{C} = \begin{bmatrix} \mathbf{C}_{1,1} & \mathbf{C}_{1,2} \\ \mathbf{C}_{2,1} & \mathbf{C}_{2,2} \end{bmatrix}$$

with

$$\mathbf{A}_{i,j}, \mathbf{B}_{i,j}, \mathbf{C}_{i,j} \in R^{2^{n-1} \times 2^{n-1}}$$

then

$$\mathbf{C}_{1,1} = \mathbf{A}_{1,1}\mathbf{B}_{1,1} + \mathbf{A}_{1,2}\mathbf{B}_{2,1}$$
$$\mathbf{C}_{1,2} = \mathbf{A}_{1,1}\mathbf{B}_{1,2} + \mathbf{A}_{1,2}\mathbf{B}_{2,2}$$
$$\mathbf{C}_{2,1} = \mathbf{A}_{2,1}\mathbf{B}_{1,1} + \mathbf{A}_{2,2}\mathbf{B}_{2,1}$$
$$\mathbf{C}_{2,2} = \mathbf{A}_{2,1}\mathbf{B}_{1,2} + \mathbf{A}_{2,2}\mathbf{B}_{2,2}$$

- La estrategia anterior nos lleva a la recurrencia de complejidad siguiente:

Algoritmo de Strassen para multiplicación de matrices

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_{1,1} & \mathbf{A}_{1,2} \\ \mathbf{A}_{2,1} & \mathbf{A}_{2,2} \end{bmatrix}, \mathbf{B} = \begin{bmatrix} \mathbf{B}_{1,1} & \mathbf{B}_{1,2} \\ \mathbf{B}_{2,1} & \mathbf{B}_{2,2} \end{bmatrix}, \mathbf{C} = \begin{bmatrix} \mathbf{C}_{1,1} & \mathbf{C}_{1,2} \\ \mathbf{C}_{2,1} & \mathbf{C}_{2,2} \end{bmatrix}$$

with

$$\mathbf{A}_{i,j}, \mathbf{B}_{i,j}, \mathbf{C}_{i,j} \in R^{2^{n-1} \times 2^{n-1}}$$

then

$$\mathbf{C}_{1,1} = \mathbf{A}_{1,1}\mathbf{B}_{1,1} + \mathbf{A}_{1,2}\mathbf{B}_{2,1}$$
$$\mathbf{C}_{1,2} = \mathbf{A}_{1,1}\mathbf{B}_{1,2} + \mathbf{A}_{1,2}\mathbf{B}_{2,2}$$
$$\mathbf{C}_{2,1} = \mathbf{A}_{2,1}\mathbf{B}_{1,1} + \mathbf{A}_{2,2}\mathbf{B}_{2,1}$$
$$\mathbf{C}_{2,2} = \mathbf{A}_{2,1}\mathbf{B}_{1,2} + \mathbf{A}_{2,2}\mathbf{B}_{2,2}$$

- La estrategia anterior nos lleva a la recurrencia de complejidad siguiente:
- $T(2^n) = 8*T(2^{n-1}) + O((2^n)^2)$ (tenemos que llenar todas las entradas de ***C***)

Algoritmo de Strassen para multiplicación de matrices

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_{1,1} & \mathbf{A}_{1,2} \\ \mathbf{A}_{2,1} & \mathbf{A}_{2,2} \end{bmatrix}, \mathbf{B} = \begin{bmatrix} \mathbf{B}_{1,1} & \mathbf{B}_{1,2} \\ \mathbf{B}_{2,1} & \mathbf{B}_{2,2} \end{bmatrix}, \mathbf{C} = \begin{bmatrix} \mathbf{C}_{1,1} & \mathbf{C}_{1,2} \\ \mathbf{C}_{2,1} & \mathbf{C}_{2,2} \end{bmatrix}$$

with

$$\mathbf{A}_{i,j}, \mathbf{B}_{i,j}, \mathbf{C}_{i,j} \in R^{2^{n-1} \times 2^{n-1}}$$

then

$$\mathbf{C}_{1,1} = \mathbf{A}_{1,1}\mathbf{B}_{1,1} + \mathbf{A}_{1,2}\mathbf{B}_{2,1}$$
$$\mathbf{C}_{1,2} = \mathbf{A}_{1,1}\mathbf{B}_{1,2} + \mathbf{A}_{1,2}\mathbf{B}_{2,2}$$
$$\mathbf{C}_{2,1} = \mathbf{A}_{2,1}\mathbf{B}_{1,1} + \mathbf{A}_{2,2}\mathbf{B}_{2,1}$$
$$\mathbf{C}_{2,2} = \mathbf{A}_{2,1}\mathbf{B}_{1,2} + \mathbf{A}_{2,2}\mathbf{B}_{2,2}$$

- La estrategia anterior nos lleva a la recurrencia de complejidad siguiente:
- $T(2^n) = 8*T(2^{n-1}) + O((2^n)^2)$ (tenemos que llenar todas las entradas de ***C***)
- Si tomamos como *N* la dimensión del problema $T(N) = 8*T(N/2) + O(N^2)$

Algoritmo de Strassen para multiplicación de matrices

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_{1,1} & \mathbf{A}_{1,2} \\ \mathbf{A}_{2,1} & \mathbf{A}_{2,2} \end{bmatrix}, \mathbf{B} = \begin{bmatrix} \mathbf{B}_{1,1} & \mathbf{B}_{1,2} \\ \mathbf{B}_{2,1} & \mathbf{B}_{2,2} \end{bmatrix}, \mathbf{C} = \begin{bmatrix} \mathbf{C}_{1,1} & \mathbf{C}_{1,2} \\ \mathbf{C}_{2,1} & \mathbf{C}_{2,2} \end{bmatrix}$$

with

$$\mathbf{A}_{i,j}, \mathbf{B}_{i,j}, \mathbf{C}_{i,j} \in R^{2^{n-1} \times 2^{n-1}}$$

then

$$\mathbf{C}_{1,1} = \mathbf{A}_{1,1}\mathbf{B}_{1,1} + \mathbf{A}_{1,2}\mathbf{B}_{2,1}$$
$$\mathbf{C}_{1,2} = \mathbf{A}_{1,1}\mathbf{B}_{1,2} + \mathbf{A}_{1,2}\mathbf{B}_{2,2}$$
$$\mathbf{C}_{2,1} = \mathbf{A}_{2,1}\mathbf{B}_{1,1} + \mathbf{A}_{2,2}\mathbf{B}_{2,1}$$
$$\mathbf{C}_{2,2} = \mathbf{A}_{2,1}\mathbf{B}_{1,2} + \mathbf{A}_{2,2}\mathbf{B}_{2,2}$$

- La estrategia anterior nos lleva a la recurrencia de complejidad siguiente:
- $T(2^n) = 8*T(2^{n-1}) + O((2^n)^2)$ (tenemos que llenar todas las entradas de ***C***)
- Si tomamos como *N* la dimensión del problema $T(N) = 8*T(N/2) + O(N^2)$
- lo cual lleva a $T(N) = O(N^3)$ a partir del master theorem, caso 1.

Algoritmo de Strassen para multiplicación de matrices

- Pero si definimos:

- Y lo usamos para calcular

Algoritmo de Strassen para multiplicación de matrices

- Pero si definimos:

$$
\begin{aligned}
\mathbf{M}_1 &:= (\mathbf{A}_{1,1} + \mathbf{A}_{2,2})(\mathbf{B}_{1,1} + \mathbf{B}_{2,2}) \\
\mathbf{M}_2 &:= (\mathbf{A}_{2,1} + \mathbf{A}_{2,2})\mathbf{B}_{1,1} \\
\mathbf{M}_3 &:= \mathbf{A}_{1,1}(\mathbf{B}_{1,2} - \mathbf{B}_{2,2}) \\
\mathbf{M}_4 &:= \mathbf{A}_{2,2}(\mathbf{B}_{2,1} - \mathbf{B}_{1,1}) \\
\mathbf{M}_5 &:= (\mathbf{A}_{1,1} + \mathbf{A}_{1,2})\mathbf{B}_{2,2} \\
\mathbf{M}_6 &:= (\mathbf{A}_{2,1} - \mathbf{A}_{1,1})(\mathbf{B}_{1,1} + \mathbf{B}_{1,2}) \\
\mathbf{M}_7 &:= (\mathbf{A}_{1,2} - \mathbf{A}_{2,2})(\mathbf{B}_{2,1} + \mathbf{B}_{2,2})
\end{aligned}
$$

- Y lo usamos para calcular

Algoritmo de Strassen para multiplicación de matrices

- Pero si definimos:

$$
\begin{aligned}
\mathbf{M}_1 &:= (\mathbf{A}_{1,1} + \mathbf{A}_{2,2})(\mathbf{B}_{1,1} + \mathbf{B}_{2,2}) \\
\mathbf{M}_2 &:= (\mathbf{A}_{2,1} + \mathbf{A}_{2,2})\mathbf{B}_{1,1} \\
\mathbf{M}_3 &:= \mathbf{A}_{1,1}(\mathbf{B}_{1,2} - \mathbf{B}_{2,2}) \\
\mathbf{M}_4 &:= \mathbf{A}_{2,2}(\mathbf{B}_{2,1} - \mathbf{B}_{1,1}) \\
\mathbf{M}_5 &:= (\mathbf{A}_{1,1} + \mathbf{A}_{1,2})\mathbf{B}_{2,2} \\
\mathbf{M}_6 &:= (\mathbf{A}_{2,1} - \mathbf{A}_{1,1})(\mathbf{B}_{1,1} + \mathbf{B}_{1,2}) \\
\mathbf{M}_7 &:= (\mathbf{A}_{1,2} - \mathbf{A}_{2,2})(\mathbf{B}_{2,1} + \mathbf{B}_{2,2})
\end{aligned}
$$

- Y lo usamos para calcular

$$
\begin{aligned}
\mathbf{C}_{1,1} &= \mathbf{M}_1 + \mathbf{M}_4 - \mathbf{M}_5 + \mathbf{M}_7 \\
\mathbf{C}_{1,2} &= \mathbf{M}_3 + \mathbf{M}_5 \\
\mathbf{C}_{2,1} &= \mathbf{M}_2 + \mathbf{M}_4 \\
\mathbf{C}_{2,2} &= \mathbf{M}_1 - \mathbf{M}_2 + \mathbf{M}_3 + \mathbf{M}_6
\end{aligned}
$$

Algoritmo de Strassen para multiplicación de matrices

- Pero si definimos:

$$\mathbf{M}_1 := (\mathbf{A}_{1,1} + \mathbf{A}_{2,2})(\mathbf{B}_{1,1} + \mathbf{B}_{2,2})$$
$$\mathbf{M}_2 := (\mathbf{A}_{2,1} + \mathbf{A}_{2,2})\mathbf{B}_{1,1}$$
$$\mathbf{M}_3 := \mathbf{A}_{1,1}(\mathbf{B}_{1,2} - \mathbf{B}_{2,2})$$
$$\mathbf{M}_4 := \mathbf{A}_{2,2}(\mathbf{B}_{2,1} - \mathbf{B}_{1,1})$$
$$\mathbf{M}_5 := (\mathbf{A}_{1,1} + \mathbf{A}_{1,2})\mathbf{B}_{2,2}$$
$$\mathbf{M}_6 := (\mathbf{A}_{2,1} - \mathbf{A}_{1,1})(\mathbf{B}_{1,1} + \mathbf{B}_{1,2})$$
$$\mathbf{M}_7 := (\mathbf{A}_{1,2} - \mathbf{A}_{2,2})(\mathbf{B}_{2,1} + \mathbf{B}_{2,2})$$

7 multiplicaciones recursivas de dimension n/2

- Y lo usamos para calcular

$$\mathbf{C}_{1,1} = \mathbf{M}_1 + \mathbf{M}_4 - \mathbf{M}_5 + \mathbf{M}_7$$
$$\mathbf{C}_{1,2} = \mathbf{M}_3 + \mathbf{M}_5$$
$$\mathbf{C}_{2,1} = \mathbf{M}_2 + \mathbf{M}_4$$
$$\mathbf{C}_{2,2} = \mathbf{M}_1 - \mathbf{M}_2 + \mathbf{M}_3 + \mathbf{M}_6$$

Algoritmo de Strassen para multiplicación de matrices

- Pero si definimos:

$$\mathbf{M}_1 := (\mathbf{A}_{1,1} + \mathbf{A}_{2,2})(\mathbf{B}_{1,1} + \mathbf{B}_{2,2})$$
$$\mathbf{M}_2 := (\mathbf{A}_{2,1} + \mathbf{A}_{2,2})\mathbf{B}_{1,1}$$
$$\mathbf{M}_3 := \mathbf{A}_{1,1}(\mathbf{B}_{1,2} - \mathbf{B}_{2,2})$$
$$\mathbf{M}_4 := \mathbf{A}_{2,2}(\mathbf{B}_{2,1} - \mathbf{B}_{1,1})$$
$$\mathbf{M}_5 := (\mathbf{A}_{1,1} + \mathbf{A}_{1,2})\mathbf{B}_{2,2}$$
$$\mathbf{M}_6 := (\mathbf{A}_{2,1} - \mathbf{A}_{1,1})(\mathbf{B}_{1,1} + \mathbf{B}_{1,2})$$
$$\mathbf{M}_7 := (\mathbf{A}_{1,2} - \mathbf{A}_{2,2})(\mathbf{B}_{2,1} + \mathbf{B}_{2,2})$$

7 multiplicaciones recursivas de dimension n/2

- Y lo usamos para calcular

$$\mathbf{C}_{1,1} = \mathbf{M}_1 + \mathbf{M}_4 - \mathbf{M}_5 + \mathbf{M}_7$$
$$\mathbf{C}_{1,2} = \mathbf{M}_3 + \mathbf{M}_5$$
$$\mathbf{C}_{2,1} = \mathbf{M}_2 + \mathbf{M}_4$$
$$\mathbf{C}_{2,2} = \mathbf{M}_1 - \mathbf{M}_2 + \mathbf{M}_3 + \mathbf{M}_6$$

sumas y restas de orden $O((n/2)^2)$

Algoritmo de Strassen para multiplicación de matrices

Algoritmo de Strassen para multiplicación de matrices

- Lo anterior nos lleva a una recursion del tipo

Algoritmo de Strassen para multiplicación de matrices

- Lo anterior nos lleva a una recursion del tipo
    - $T(2^n) = 7*T(2^{n-1}) + O((2^n)^2)$

Algoritmo de Strassen para multiplicación de matrices

- Lo anterior nos lleva a una recursion del tipo
    - $T(2^n) = 7*T(2^{n-1}) + O((2^n)^2)$

Algoritmo de Strassen para multiplicación de matrices

- Lo anterior nos lleva a una recursion del tipo
    - $T(2^n) = 7*T(2^{n-1}) + O((2^n)^2)$

- Si tomamos como *N* la dimensión del problema $T(N) = 7*T(N/2) + O(N^2)$

Algoritmo de Strassen para multiplicación de matrices

- Lo anterior nos lleva a una recursion del tipo
    - $T(2^n) = 7*T(2^{n-1}) + O((2^n)^2)$

- Si tomamos como *N* la dimensión del problema $T(N) = 7*T(N/2) + O(N^2)$
- lo cual lleva a $T(N) = O(N^{\log_2 7}) = O(N^{2.807})$ a partir del *master theorem*, caso 1.

Algoritmo de Strassen para multiplicación de matrices

Algoritmo de Strassen para multiplicación de matrices

- Desventajas

Algoritmo de Strassen para multiplicación de matrices

- Desventajas
  - El factor no indicado de $O(n^{2.807})$ hace más lento el algoritmo de lo aparente.

Algoritmo de Strassen para multiplicación de matrices

- Desventajas
  - El factor no indicado de $O(n^{2.807})$ hace más lento el algoritmo de lo aparente.
  - malo para matrices ralas, hay mejores métodos

Algoritmo de Strassen para multiplicación de matrices

- Desventajas
  - El factor no indicado de $O(n^{2.807})$ hace más lento el algoritmo de lo aparente.
  - malo para matrices ralas, hay mejores métodos
  - las sub-matrices consumen espacio que se adminstra independiente

- Y finalmente ....


The end

- Examen Teórico: martes 5 Junio 9:00-12:00,

- Examen Teórico: martes 5 Junio 9:00-12:00,
- Examen práctico: martes 12:00-3:00pm .

- Examen Teórico: martes 5 Junio 9:00-12:00,
- Examen práctico: martes 12:00-3:00pm .
- Entrega de proyectos lunes 11, a las 9:30am se pone en la carpeta de DRIVE.