Análisis y Diseño de Algoritmos

Carlos Eduardo Atencio Torres
Universidad Nacional de San Agustín
mailto: catencio@unsa.edu.pe

Adaptado del material de Paulo Feofiloff y aportes de Coelho, Cris y Alair.

2018

Empezamos: Ordenación

$A[1..n]$ es creciente si $A[1] \leq ... \leq A[n]$.

Problema

Ordenar un arreglo $A[1..n]$ de modo que quede en forma creciente.

Empezamos: Ordenación

$A[1..n]$ es creciente si $A[1] \leq ... \leq A[n]$.

Problema

Ordenar un arreglo $A[1..n]$ de modo que quede en forma creciente.

Entra

1										n
33	55	33	44	33	22	11	99	22	55	77

Empezamos: Ordenación

$A[1..n]$ es creciente si $A[1] \leq ... \leq A[n]$.

Problema

Ordenar un arreglo $A[1..n]$ de modo que quede en forma creciente.

Entra

1										n
33	55	33	44	33	22	11	99	22	55	77

Sale

1										n
11	22	22	33	33	33	44	55	55	77	99

Ordenación por Inserción

Clave = 38

1					i	j				n
20	25	35	40	44	55	38	99	10	65	50

Ordenación por Inserción

Clave = 38

1					i	j				n
20	25	35	40	44	55	38	99	10	65	50

1				i		j				n
20	25	35	40	44		55	99	10	65	50

Ordenación por Inserción

Clave = 38

1					i	j				n
20	25	35	40	44	55	38	99	10	65	50

1				i		j				n
20	25	35	40	44		55	99	10	65	50

1			i			j				n
20	25	35	40		44	55	99	10	65	50

Ordenación por Inserción

Clave = 38

1					i	j				n
20	25	35	40	44	55	38	99	10	65	50

1				i		j				n
20	25	35	40	44		55	99	10	65	50

1			i			j				n
20	25	35	40		44	55	99	10	65	50

1		i				j				n
20	25	35		40	44	55	99	10	65	50

Ordenación por Inserción

Clave = 38

1					i	j				n
20	25	35	40	44	55	38	99	10	65	50

1				i		j				n
20	25	35	40	44		55	99	10	65	50

1			i			j				n
20	25	35	40		44	55	99	10	65	50

1		i				j				n
20	25	35		40	44	55	99	10	65	50

1		i				j				n
20	25	35	38	40	44	55	99	10	65	50

Ordenación por Inserción

Clave	1						j			n
99	20	25	35	38	40	44	55	99	10	65	50

Ordenación por Inserción

Clave	1						j			n
99	20	25	35	38	40	44	55	99	10	65	50

Clave	1							j		n
10	20	25	35	38	40	44	55	99	10	65	50

Ordenación por Inserción

Clave	1							j			n
99	20	25	35	38	40	44	55	99	10	65	50

Clave	1								j		n
10	20	25	35	38	40	44	55	99	10	65	50

Clave	1									j	n
65	10	20	25	35	38	40	44	55	99	65	50

Ordenación por Inserción

Clave	1							j			n
99	20	25	35	38	40	44	55	99	10	65	50

Clave	1								j		n
10	20	25	35	38	40	44	55	99	10	65	50

Clave	1									j	n
65	10	20	25	35	38	40	44	55	99	65	50

Clave	1										j
50	10	20	25	35	38	40	44	55	65	99	50

Ordenación por Inserción

Clave	1							j			n
99	20	25	35	38	40	44	55	99	10	65	50

Clave	1								j		n
10	20	25	35	38	40	44	55	99	10	65	50

Clave	1									j	n
65	10	20	25	35	38	40	44	55	99	65	50

Clave	1										j
50	10	20	25	35	38	40	44	55	65	99	50

FIN

10	20	25	35	38	40	44	50	55	65	99

Ordenación por Inserción

ORDENA-POR-INSERCION(A,n)

para $j \leftarrow 2$ hasta n **hacer**
  $clave \leftarrow A[j]$
  $i \leftarrow j - 1$
  mientras $i \geq 1$ AND $A[i] > clave$ **hacer**
    $A[i+1] \leftarrow A[i]$, $\triangleright$ Haciendo campo
    $i \leftarrow i - 1$
  $A[i+1] \leftarrow clave$, $\triangleright$ Insertando

Corrección del algoritmo

Para probar la correctitud de un algoritmo usamos invariantes.

Corrección del algoritmo

Para probar la correctitud de un algoritmo usamos invariantes.

Invariante i0

En la línea 1, se cumple que A[1..j-1] es creciente.

1						j				
20	25	35	40	44	55	38	99	10	65	50

Corrección del algoritmo

Para probar la correctitud de un algoritmo usamos invariantes.

Invariante i0

En la línea 1, se cumple que A[1..j-1] es creciente.

1						j				
20	25	35	40	44	55	38	99	10	65	50

Conclusión

Suponiendo que i0 siempre es válida, cuando j sea n+1, se entiende que A[1..n] es creciente.

Más invariantes

En la línea 4 ocurren las siguientes invariantes:

- i1 A[1..i] y A[i+2..j] son crecientes
- i2 A[1..i] $\leq$ A[i+2..j] son crecientes
- i3 A[i+2..j] $>$ llave.
- i4 A[1..i] + A[i+2..j] + llave no cambian.

Llave	1		i				j				n
38	20	25	35		40	44	55	99	10	65	50

Concluyendo

Las demostraciones en un algoritmo iterativo siempre seguirán los siguientes pasos:

1. Verificar que la relación vale al inicio de la primera iteración.
2. Demostrar que si la relación vale al inicio de la iteración, entonces ella valdrá al final.
3. Demostrar que si la relación vale al inicio de la última iteración, entonces la relación junto a la condición de parada demuestran la correctitud del algoritmo.

Asignaciones del algoritmo ($\leftarrow$)

Recordando el algoritmo de INSERCIÓN:

LINEAS 3-6(A,n)

3: $i \leftarrow j - 1$
4: **mientras** $i \geq 1$ AND $A[i] > clave$ **hacer**
5: $\quad A[i+1] \leftarrow A[i]$
6: $\quad i \leftarrow i - 1$
7: $A[i+1] \leftarrow clave$

Contando las asignaciones para el caso máximo:

Asignaciones del algoritmo ($\leftarrow$)

Recordando el algoritmo de INSERCIÓN:

LINEAS 3-6(A,n)

3: $i \leftarrow j - 1$
4: **mientras** $i \geq 1$ AND $A[i] > clave$ **hacer**
5: $\quad A[i+1] \leftarrow A[i]$
6: $\quad i \leftarrow i - 1$
7: $A[i+1] \leftarrow clave$

Contando las asignaciones para el caso máximo:

línea	**asignaciones**
3	$= 1$
4	
5	
6	
total	

Asignaciones del algoritmo ($\leftarrow$)

Recordando el algoritmo de INSERCIÓN:

LINEAS 3-6(A,n)

3: $i \leftarrow j - 1$
4: **mientras** $i \geq 1$ AND $A[i] > clave$ **hacer**
5: $\quad A[i+1] \leftarrow A[i]$
6: $\quad i \leftarrow i - 1$
7: $A[i+1] \leftarrow clave$

Contando las asignaciones para el caso máximo:

línea	**asignaciones**
3	$= 1$
4	$= 0$
5	
6	
total	

Asignaciones del algoritmo ($\leftarrow$)

Recordando el algoritmo de INSERCIÓN:

LINEAS 3-6(A,n)

3: $i \leftarrow j - 1$
4: **mientras** $i \geq 1$ AND $A[i] > clave$ **hacer**
5: $\quad A[i+1] \leftarrow A[i]$
6: $\quad i \leftarrow i - 1$
7: $A[i+1] \leftarrow clave$

Contando las asignaciones para el caso máximo:

línea	**asignaciones**
3	$= 1$
4	$= 0$
5	$\leq$ j-1
6	
total	

Asignaciones del algoritmo ($\leftarrow$)

Recordando el algoritmo de INSERCIÓN:

LINEAS 3-6(A,n)

3: $i \leftarrow j - 1$
4: **mientras** $i \geq 1$ AND $A[i] > clave$ **hacer**
5: $\quad A[i+1] \leftarrow A[i]$
6: $\quad i \leftarrow i - 1$
7: $A[i+1] \leftarrow clave$

Contando las asignaciones para el caso máximo:

línea	**asignaciones**
3	$= 1$
4	$= 0$
5	$\leq$ j-1
6	$\leq$ j-1
total	$\leq$ 2j-1 $\leq$ 2n - 1

Asignaciones del Algoritmo ORDENA-POR-INSERCION

línea		Asignaciones (número máximo)
1	$=$	$n-1+1$
2	$=$	$n-1$
3	$=$	$n-1$
4	$=$	0
5	$\leq$	$1+2+\ +(n-1)=n(n-1)/2$
6	$\leq$	$1+2+\ +(n-1)=n(n-1)/2$
7	$=$	$n-1$
total	$\leq$	n^2+3n-3

Consumo de tiempo del Algoritmo ORDENA-POR-INSERCION

- Suponiendo que cada línea de código consume 1 unidad de tiempo.

línea		Asignaciones (número máximo)
1	$=$	n
2	$=$	$n-1$
3	$=$	$n-1$
4	$\leq$	$2+3+\cdots+n=(n-1)(n+2)/2$
5	$\leq$	$1+2+\cdots+(n-1)=n(n-1)/2$
6	$\leq$	$1+2+\cdots+(n-1)=n(n-1)/2$
7	$=$	$n-1$
total	$\leq$	

Consumo de tiempo del Algoritmo ORDENA-POR-INSERCION

- Suponiendo que cada línea de código consume 1 unidad de tiempo.

línea		**Asignaciones (número máximo)**
1	$=$	n
2	$=$	$n-1$
3	$=$	$n-1$
4	$\leq$	$2+3+\cdots+n=(n-1)(n+2)/2$
5	$\leq$	$1+2+\cdots+(n-1)=n(n-1)/2$
6	$\leq$	$1+2+\cdots+(n-1)=n(n-1)/2$
7	$=$	$n-1$
total	$\leq$	$(3/2)n^2+(7/2)n-4$

Consumo de tiempo del Algoritmo ORDENA-POR-INSERCION

- Suponiendo que cada línea de código consume 1 unidad de tiempo.
- Suponiendo que cada línea consume un tiempo t_i.

línea		**Asignaciones (número máximo)**	
1	$=$	n	xt_1
2	$=$	$n-1$	xt_2
3	$=$	$n-1$	xt_3
4	$\leq$	$2+3+\cdots+n=(n-1)(n+2)/2$	xt_4
5	$\leq$	$1+2+\cdots+(n-1)=n(n-1)/2$	xt_5
6	$\leq$	$1+2+\cdots+(n-1)=n(n-1)/2$	xt_6
7	$=$	$n-1$	xt_7
total	$\leq$		

Consumo de tiempo del Algoritmo ORDENA-POR-INSERCION

- Suponiendo que cada línea de código consume 1 unidad de tiempo.
- Suponiendo que cada línea consume un tiempo t_i.

línea		**Asignaciones (número máximo)**	
1	$=$	n	xt_1
2	$=$	$n-1$	xt_2
3	$=$	$n-1$	xt_3
4	$\leq$	$2+3+\cdots+n=(n-1)(n+2)/2$	xt_4
5	$\leq$	$1+2+\cdots+(n-1)=n(n-1)/2$	xt_5
6	$\leq$	$1+2+\cdots+(n-1)=n(n-1)/2$	xt_6
7	$=$	$n-1$	xt_7
total	$\leq$	$((t_4+t_5+t_6)/2)\times n^2+$ $(t_1+t_2+t_3+t_4/2-t_5/2-t_6/2+t_7)\times n-$ $(t_2+t_3+t_4+t_7)$	

Consumo de tiempo del Algoritmo ORDENA-POR-INSERCION

$$\begin{aligned}\textbf{total} \quad = \quad & ((t_4 + t_5 + t_6)/2) \times n^2 + \\ & (t_1 + t_2 + t_3 + t_4/2 - t_5/2 - t_6/2 + t_7) \times n - \\ & (t_2 + t_3 + t_4 + t_7)\end{aligned}$$

Reemplazando t_i por una constante, tenemos:

Consumo de tiempo del Algoritmo ORDENA-POR-INSERCION

$$
\begin{aligned}
\textbf{total} \quad = \quad & ((t_4 + t_5 + t_6)/2) \times n^2 + \\
& (t_1 + t_2 + t_3 + t_4/2 - t_5/2 - t_6/2 + t_7) \times n - \\
& (t_2 + t_3 + t_4 + t_7)
\end{aligned}
$$

Reemplazando t_i por una constante, tenemos:

$$c_1 \times n^2 + c_2 \times n + c_3$$

Consumo de tiempo del Algoritmo ORDENA-POR-INSERCION

$$\begin{aligned}\textbf{total} \quad = \quad & ((t_4 + t_5 + t_6)/2) \times n^2 + \\ & (t_1 + t_2 + t_3 + t_4/2 - t_5/2 - t_6/2 + t_7) \times n - \\ & (t_2 + t_3 + t_4 + t_7)\end{aligned}$$

Reemplazando t_i por una constante, tenemos:

$$c_1 \times n^2 + c_2 \times n + c_3$$

- c_1, c_2, c_3 dependen del computador.
- n^2 se repetirá siempre, es algo propio del algoritmo.

Notación Asintótica

Intuitivamente...

$$O(f(n)) \approx \text{Funciones que no crecen más rápido que } f(n)$$
$$\approx \text{Funciones menores o iguales a un múltiplo de } f(n)$$

$n^2 \quad 100n^2 + 0.00000001 \quad n^2/9378 \quad \text{etc.}$

- $n^2 + 3n - 5$ tiene el mismo crecimiento asintótico que n^2
- $n^2 + 3n - 5$ no crece más rápido que n^2
- $n^2 + 3n - 5$ es $O(n^2)$
- $n^2 + 3n - 5 = O(n^2)$
- $n^3 + n^2 - 18n + 65$ No es $O(n^2)$

Ejercicios Propuestos

Ejercicio 1

En función de n, cuanto vale S al final del siguiente algoritmo?

$S \leftarrow 0$
para $i \leftarrow 2$ hasta $n - 2$ **hacer**
  para $j \leftarrow i$ hasta n **hacer**
    $S \leftarrow S + 1$

Ejercicios Propuestos

Ejercicio 1

En función de n, cuanto vale S al final del siguiente algoritmo?

$S \leftarrow 0$
para $i \leftarrow 2$ hasta $n - 2$ **hacer**
    para $j \leftarrow i$ hasta n **hacer**
        $S \leftarrow S + 1$

Solución

$S = (1/2)n^2 - (1/2)n - 3$

Ejercicios Propuestos

Ejercicio 2

En función de n, cuanto vale S al final del siguiente algoritmo?. Responda con una cota superior cercana.

$S \leftarrow 0$
$i \leftarrow n$
mientras $i > 0$ **hacer**
  para $j \leftarrow 1$ hasta i **hacer**
    $S \leftarrow S + 1$
  $i \leftarrow \lfloor i/2 \rfloor$

Ejercicios Propuestos

Ejercicio 3

Para el siguiente algoritmo que recibe un arreglo A, se pide el número de...

1. Asignaciones, y
2. Comparaciones.

$s \leftarrow 0$
para $i \leftarrow$ hasta n **hacer**
    $s \leftarrow s + A[i]$
$m \leftarrow s/n$
$k \leftarrow 1$
para $i \leftarrow 2$ hasta n **hacer**
    si $(A[i] - m)^2 < (A[k] - m)^2$ **entonces**
        $k \leftarrow i$

Repaso de la aula pasada

ORDENA-POR-INSERCION(A,n)

1: **para** $j \leftarrow 2$ hasta n **hacer**
2:     $clave \leftarrow A[j]$
3:     $i \leftarrow j - 1$
4:     **mientras** $i \geq 1$ AND $A[i] > clave$ **hacer**
5:         $A[i+1] \leftarrow A[i]$, $\triangleright$ Haciendo campo
6:         $i \leftarrow i - 1$
7:     $A[i+1] \leftarrow clave$, $\triangleright$ Insertando

Invariantes

- i0 En la línea 1 se cumple que $A[1 \ldots j-1]$ es creciente.
  - Suponiendo que i0 siempre es válida, cuando j sea n+1, entonces $A[1 \ldots n]$ estará ordenado en forma creciente.

Repaso de la aula pasada

- La correctitud en general consiste en:
  1. Verificar que la relación vale al inicio de la primera iteración.
  2. Demostrar que la relación vale también al final.
  3. Demostrar que si la relación vale al final de la última iteración, entonces la relación junto a la condición de parada demuestran la correctitud del algoritmo.
- Existen más invariantes que deben ser demostradas para probar cada paso del algoritmo.

Repaso de la aula pasada

Asignaciones $\leftarrow$ del algoritmo INSERCIÓN.

$\leq n^2 + 3n - 3$

Repaso de la aula pasada

Asignaciones $\leftarrow$ del algoritmo INSERCIÓN.

$\leq n^2 + 3n - 3$

Consumo de tiempo (cada línea consume 1 unidad de tiempo)

$\leq (3/2)n^2 + (7/3)n - 4$

Repaso de la aula pasada

Asignaciones $\leftarrow$ del algoritmo INSERCIÓN.

$\leq n^2 + 3n - 3$

Consumo de tiempo (cada línea consume 1 unidad de tiempo)

$\leq (3/2)n^2 + (7/3)n - 4$

Consumo de tiempo (cada línea consume una unidad t_i de tiempo)

$\leq ((t_4 + t_5 + t_6)/2) \times n^2 + (t_1 + t_2 + t_3 + t_4/2 - t_5/2 - t_6/2 + t_7) \times n$ - $(t_2 + t_3 + t_4 + t_7)$

Repaso de la aula pasada

Asignaciones $\leftarrow$ del algoritmo INSERCIÓN.

$\leq n^2 + 3n - 3$

Consumo de tiempo (cada línea consume 1 unidad de tiempo)

$\leq (3/2)n^2 + (7/3)n - 4$

Consumo de tiempo (cada línea consume una unidad t_i de tiempo)

$\leq ((t_4 + t_5 + t_6)/2) \times n^2 + (t_1 + t_2 + t_3 + t_4/2 - t_5/2 - t_6/2 + t_7) \times n$ - $(t_2 + t_3 + t_4 + t_7)$

Simplificando en constantes el consumo de tiempo t_i

$\leq c_1 \times n^2 + c_2 \times n + c_3$ Donde:

- c_1, c_2, c_3 dependen del computador.
- n^2 se repetirá siempre, es algo propio del algoritmo.

Repaso de la aula pasada

Entendiendo asintóticamente...

n	$(3/2)n^2 + (7/2)n - 4$	$(3/2)n^2$
64	6364	6144
128	25020	24576
256	99196	98304
512	395004	3932116
1024	1576444	1572864
2048	6298620	6291456
4096	25180156	25165824
8192	100691964	100663296
16384	402710524	402653184
32768	1610727420	1610612736

Repaso de la aula pasada

Entendiendo asintóticamente...

n	$(3/2)n^2 + (7/2)n - 4$	$(3/2)n^2$
64	6364	6144
128	25020	24576
256	99196	98304
512	395004	3932116
1024	1576444	1572864
2048	6298620	6291456
4096	25180156	25165824
8192	100691964	100663296
16384	402710524	402653184
32768	1610727420	1610612736

Conclusión

$(3/2)n^2$ domina a los otros términos.

Recordando las primeras aulas de análisis asintótico

Notación O

$O(f(n))$	significa	la familia de funciones que no crecen más que $f(n)$.
	incluye	el conjunto de funciones que son menores que $f(n)$.
	indica	una cota superior

Recordando las primeras aulas de análisis asintótico

Notación O

$O(f(n))$	significa	la familia de funciones que no crecen más que $f(n)$.
	incluye	el conjunto de funciones que son menores que $f(n)$.
	indica	una cota superior

Notación

$T(n) = O(f(n))$ se lee "$T(n)$ es O de $(f(n))$".

- $T(n) \leq cf(n)$, en que c es una constante positiva, y $n \geq n_0$.

Recordando las primeras aulas de análisis asintótico

Notación O

$O(f(n))$	significa	la familia de funciones que no crecen más que $f(n)$.
	incluye	el conjunto de funciones que son menores que $f(n)$.
	indica	una cota superior

Notación

$T(n) = O(f(n))$ se lee "$T(n)$ es O de $(f(n))$".

- $T(n) \leq cf(n)$, en que c es una constante positiva, y $n \geq n_0$.

Ejemplo

$3n^2$ es $O(n^2)$. Probamos:

- Consideramos $T(n) = 3n^2$ y $f(n) = n^2$.
- Según definición, $3n^2 \leq cn^2$, para una constante positiva c y un $n \geq n_0$
- Para la prueba, consideramos $c = 3$ y $n_0 = 1$.

Recordando las primeras aulas de análisis asintótico

Notación Ω

$\Omega(f(n))$	significa	la familia de funciones que no crecen menos que $f(n)$.
	incluye	el conjunto de funciones que son mayores que $f(n)$.
	indica	una cota inferior

Recordando las primeras aulas de análisis asintótico

Notación Ω

$\Omega(f(n))$	significa	la familia de funciones que no crecen menos que $f(n)$.
	incluye	el conjunto de funciones que son mayores que $f(n)$.
	indica	una cota inferior

Notación

$T(n) = \Omega(f(n))$ se lee "$T(n)$ es *omega* de $(f(n))$".

- $T(n) \geq cf(n)$, en que c es una constante positiva, y $n \geq n_0$.

Recordando las primeras aulas de análisis asintótico

Notación Ω

$\Omega(f(n))$	significa	la familia de funciones que no crecen menos que $f(n)$.
	incluye	el conjunto de funciones que son mayores que $f(n)$.
	indica	una cota inferior

Notación

$T(n) = \Omega(f(n))$ se lee "$T(n)$ es *omega* de $(f(n))$".

- $T(n) \geq cf(n)$, en que c es una constante positiva, y $n \geq n_0$.

Ejemplo

$0.01n^2$ es $\Omega(n^2)$. Probamos:

- Consideramos $T(n) = 0.01n^2$ y $f(n) = n^2$.
- Según definición, $0.01n^2 \geq cn^2$, para una constante positiva c y un $n \geq n_0$
- Para la prueba, consideramos $c = 0.001$ y $n_0 = 1$.

Recordando las primeras aulas de análisis asintótico

Notación Θ

$\Theta(f(n))$	significa	familia de funciones que crecen en un orden igual a $f($
	conforma	las familias pertenecientes a $O(f(n))$ y $\Omega(f(n))$

Recordando las primeras aulas de análisis asintótico

Notación Θ

$\Theta(f(n))$	significa	familia de funciones que crecen en un orden igual a $f($
	conforma	las familias pertenecientes a $O(f(n))$ y $\Omega(f(n))$

Notación

$T(n) = \Theta(f(n))$ se lee "$T(n)$ es *theta* de $(f(n))$".

- $c_1 f(n) \leq T(n) \leq c_2 f(n)$, en que c_1 y c_2 es una constante positiva, y $n \geq n_0$.

Recordando las primeras aulas de análisis asintótico

Notación Θ

$\Theta(f(n))$	significa	familia de funciones que crecen en un orden igual a $f($
	conforma	las familias pertenecientes a $O(f(n))$ y $\Omega(f(n))$

Notación

$T(n) = \Theta(f(n))$ se lee "$T(n)$ es *theta* de $(f(n))$".

- $c_1 f(n) \leq T(n) \leq c_2 f(n)$, en que c_1 y c_2 es una constante positiva, y $n \geq n_0$.

Ejemplo

$(3/2)n^2$ es $\Theta(n^2)$. Probamos:

- Consideramos $T(n) = (3/2)n^2$ y $f(n) = n^2$.
- Según definición, $c_1 n^2 \leq (3/2)n^2 \leq c_2 n^2$, para las constantes positivas c_1 y c_2, además $n \geq n_0$
- Para la prueba, consideramos $c_1 = (1/2)$, $c_2 = 2$ y $n_0 = 1$.

Análisis asintótico del algoritmo de INSERCIÓN

ORDENA-POR-INSERCION(A,p,r)

Análisis asintótico del algoritmo de INSERCIÓN

ORDENA-POR-INSERCION(A,p,r)

para $j \leftarrow p + 1$ hasta r **hacer**
  $clave \leftarrow A[j]$
  $i \leftarrow j - 1$
  mientras $i \geq p$ AND $A[i] > clave$ **hacer**
    $A[i+1] \leftarrow A[i]$, $\triangleright$ Haciendo campo
    $i \leftarrow i - 1$
  $A[i+1] \leftarrow clave$, $\triangleright$ Insertando

£Cuánto tiempo consume el algoritmo?

Supongamos que $n = r - p + 1$

Consumo de tiempo

línea	Consumo de tiempo
1	$O(n)$
2	$O(n)$
3	$O(n)$
4	$nO(n)$
5	$nO(n)$
6	$nO(n)$
7	$O(n)$
Total	$O(3n^2 + 4n) = \mathrm{O}(n^2)$

Observaciones

- Las líneas 4-6 son ejecutadas $\leq n$ veces. Cada ejecución consume $O(n)$. Todas juntas consumen $nO(n)$.
- Probar $nO(n) = O(n^2)$.
- Probar $O(n^2) + O(n^2) + O(n^2) = O(3n^2)$.
- Probar $O(3n^2 + 4n) = O(n^2)$.

Algoritmo de Intercalación

Problema

Para los segmentos $A[p \ldots q]$ y $A[q+1 \ldots r]$, ambos crecientes, deseamos ordenar $A[p \ldots r]$ de modo que quede en orden creciente.

Algoritmo de Intercalación

Problema

Para los segmentos A[p. . . q] y A[q+1 . . . r], ambos crecientes, deseamos ordenar A[p. . . r] de modo que quede en orden creciente.

Entra

p				q						r
10	35	38	60	25	40	45	50	65	77	99

Algoritmo de Intercalación

Problema

Para los segmentos A[p. . . q] y A[q+1 . . . r], ambos crecientes, deseamos ordenar A[p. . . r] de modo que quede en orden creciente.

Entra

p				q						r
10	35	38	60	25	40	45	50	65	77	99

Sale

1										n
10	25	35	38	40	45	50	60	65	77	99

Algoritmo de Intercalación

Vaciamos los valores en un arreglo temporal B. Asegurarse de dejar el segundo bloque con los valores invertidos.

Algoritmo de Intercalación

Vaciamos los valores en un arreglo temporal B. Asegurarse de dejar el segundo bloque con los valores invertidos.

	p					q					r
B	20	25	35	40	44	50	99	65	55	38	10
A											

Algoritmo de Intercalación

Vaciamos los valores en un arreglo temporal B. Asegurarse de dejar el segundo bloque con los valores invertidos.

	i										j
B	20	25	35	40	44	50	99	65	55	38	10
	k										
A											

Algoritmo de Intercalación

Vaciamos los valores en un arreglo temporal B. Asegurarse de dejar el segundo bloque con los valores invertidos.

	i									j	
B	20	25	35	40	44	50	99	65	55	38	10
		k									
A	10										

Algoritmo de Intercalación

Vaciamos los valores en un arreglo temporal B. Asegurarse de dejar el segundo bloque con los valores invertidos.

		i								j	
B	20	25	35	40	44	50	99	65	55	38	10
			k								
A	10	20									

Algoritmo de Intercalación

Vaciamos los valores en un arreglo temporal B. Asegurarse de dejar el segundo bloque con los valores invertidos.

			i							j	
B	20	25	35	40	44	50	99	65	55	38	10
				k							
A	10	20	25								

Algoritmo de Intercalación

Vaciamos los valores en un arreglo temporal B. Asegurarse de dejar el segundo bloque con los valores invertidos.

				i						j	
B	20	25	35	40	44	50	99	65	55	38	10
					k						
A	10	20	25	35							

Algoritmo de Intercalación

Vaciamos los valores en un arreglo temporal B. Asegurarse de dejar el segundo bloque con los valores invertidos.

				i					j		
B	20	25	35	40	44	50	99	65	55	38	10
						k					
A	10	20	25	35	38						

Algoritmo de Intercalación

Vaciamos los valores en un arreglo temporal B. Asegurarse de dejar el segundo bloque con los valores invertidos.

					i				j		
B	20	25	35	40	44	50	99	65	55	38	10
							k				
A	10	20	25	35	38	40					

Algoritmo de Intercalación

Vaciamos los valores en un arreglo temporal B. Asegurarse de dejar el segundo bloque con los valores invertidos.

						i			j		
B	20	25	35	40	44	50	99	65	55	38	10
								k			
A	10	20	25	35	38	40	44				

Algoritmo de Intercalación

Vaciamos los valores en un arreglo temporal B. Asegurarse de dejar el segundo bloque con los valores invertidos.

							i		j		
B	20	25	35	40	44	50	99	65	55	38	10
									k		
A	10	20	25	35	38	40	44	50			

Algoritmo de Intercalación

Vaciamos los valores en un arreglo temporal B. Asegurarse de dejar el segundo bloque con los valores invertidos.

							i	j			
B	20	25	35	40	44	50	99	65	55	38	10
										k	
A	10	20	25	35	38	40	44	50	55		

Algoritmo de Intercalación

Vaciamos los valores en un arreglo temporal B. Asegurarse de dejar el segundo bloque con los valores invertidos.

							i=j				
B	20	25	35	40	44	50	99	65	55	38	10
											k
A	10	20	25	35	38	40	44	50	55	65	

Algoritmo de Intercalación

Vaciamos los valores en un arreglo temporal B. Asegurarse de dejar el segundo bloque con los valores invertidos.

							i=j				
B	20	25	35	40	44	50	99	65	55	38	10
A	10	20	25	35	38	40	44	50	55	65	99

Algoritmo de Intercalación

INTERCALA(A,p,q,r)

$\triangleright$ $B[p..r]$ es un vector auxiliar
para $i \leftarrow p$ **hasta** q **hacer**
    $B[i] \leftarrow A[i]$
para $j \leftarrow q + 1$ **hasta** r **hacer**
    $B[r + q + 1 - j] \leftarrow A[j]$
$i \leftarrow p$
$j \leftarrow r$
para $k \leftarrow p$ **hasta** r **hacer**
    si $B[i] \leq B[j]$ **entonces**
        $A[k] \leftarrow B[i]$
        $i \leftarrow i + 1$
    si no
        $A[k] \leftarrow B[j]$
        $j \leftarrow j - 1$

Ejercicios

- Si cada línea de código consume 1 unidad de tiempo. Cuál sería el consumo total?. $n = r - p + 1$
- Cómo sería el análisis asintótico por cada línea?.
- Demuestre que el algoritmo Intercalación es $\Omega(n)$.

Soluciones

Si cada línea de código consume <u>1 unidad</u> de tiempo...

línea		**consumo**
1	=	1
2	=	$q - p + 2 = n - r + q + 1$
3	=	$q - p + 1 = n - r + q$
4	=	$r - (q+1) + 2 = n - q + p$
5	=	$r - (q+1) + 1 = n - q + p - 1$
6	=	1
7	=	1
8	=	$r - p+2 = n+1$
9	=	$r - p+1 = n$
10-14	=	$2(r - p+1) = 2n$
Total	=	$8n - 2\ (r-p+1)+5 = 6n+5$

Soluciones

Si cada línea de código consume 1 unidad de tiempo...

línea		consumo
1	=	1
2	=	$q - p + 2 = n - r + q + 1$
3	=	$q - p + 1 = n - r + q$
4	=	$r - (q+1) + 2 = n - q + p$
5	=	$r - (q+1) + 1 = n - q + p - 1$
6	=	1
7	=	1
8	=	$r - p+2 = n+1$
9	=	$r - p+1 = n$
10-14	=	$2(r - p+1) = 2n$
Total	=	$8n - 2\,(r-p+1)+5 = \underline{6n+5}$

Soluciones

Análisis asintótico...

línea	**consumo**
1-4	$O(n)$
5-6	$O(1)$
7	$nO(1) = O(n)$
8	$nO(1) = O(n)$
9-14	$nO(1) = O(n)$
Total	$O(4n + 1) = O(n)$

El algoritmo por lo tanto consume $O(n)$.

Soluciones

Demuestre que el algoritmo de intercalación es $\Omega(n)$

- Sabemos que el algoritmo consume $6n + 5$ unidades de tiempo, suponiendo que cada línea toma 1 unidad de tiempo.
- Probaremos por lo tanto que $6n + 5$ es $\Omega(n)$
- Según definición de Ω, para $c > 0$ y $n > n_0$

$$\begin{aligned} 6n + 5 &\geq cn \\ 6n + 5 &\geq 6n \\ c &= 6 \\ n_0 &= 1 \end{aligned} \qquad \ldots \textit{Probado}!$$

Podemos decir más aún..

- El algoritmo de intercalación es $O(n)$

Soluciones

Demuestre que el algoritmo de intercalación es $\Omega(n)$

- Sabemos que el algoritmo consume $6n + 5$ unidades de tiempo, suponiendo que cada línea toma 1 unidad de tiempo.
- Probaremos por lo tanto que $6n + 5$ es $\Omega(n)$
- Según definición de Ω, para $c > 0$ y $n > n_0$

$$\begin{aligned} 6n + 5 &\geq cn \\ 6n + 5 &\geq 6n \\ c &= 6 \\ n_0 &= 1 \end{aligned} \qquad \ldots \textit{Probado}!$$

Podemos decir más aún..

- El algoritmo de intercalación es $O(n)$
- El algoritmo de intercalación es $\Omega(n)$

Soluciones

Demuestre que el algoritmo de intercalación es $\Omega(n)$

- Sabemos que el algoritmo consume $6n + 5$ unidades de tiempo, suponiendo que cada línea toma 1 unidad de tiempo.
- Probaremos por lo tanto que $6n + 5$ es $\Omega(n)$
- Según definición de Ω, para $c > 0$ y $n > n_0$

$$\begin{aligned} 6n + 5 &\geq cn \\ 6n + 5 &\geq 6n \\ c &= 6 \\ n_0 &= 1 \end{aligned} \qquad \ldots \textit{Probado}!$$

Podemos decir más aún..

- El algoritmo de intercalación es $O(n)$
- El algoritmo de intercalación es $\Omega(n)$
- Por lo tanto el algoritmo de intercalación es $\Theta(n)$.

Divide y vencerás (Divide and Conquer)

- Estos algoritmos tienen 3 pasos:
  1. **Dividir**: Generación de subproblemas.
  2. **Conquistar**: Resolver cada subproblemas de forma recursiva.
  3. **Combinar**: Cada solución de los subproblemas es combinada.

ntender el concepto de subproblema es importante para probar los problemas de Programación Dinámica.

Divide y vencerás (Divide and Conquer)

- Estos algoritmos tienen 3 pasos:
  1. **Dividir**: Generación de subproblemas.
  2. **Conquistar**: Resolver cada subproblema de forma recursiva.
  3. **Combinar**: Cada solución de los subproblema es combinada.

Entender el concepto de subproblema es importante para probar los problemas de Programación Dinámica.

Merge Sort

Problema

Reordenar $A[p..r]$ de modo que esté en orden creciente.

Entra

	p								r
A	55	33	66	44	99	11	77	22	88

Sale

	p								r
A	11	22	33	44	55	66	77	88	99

Merge-Sort

	p				q				r
A	55	33	66	44	99	11	77	22	88

Merge-Sort

	p				q				r
A	55	33	66	44	99	11	77	22	88

	p		q		r				
A	55	33	66	44	99				

Merge-Sort

	p				q				r
A	55	33	66	44	99	11	77	22	88

	p		q		r				
A	55	33	66	44	99				

	p	q	r						
A	55	33	66						

Merge-Sort

	p				q				r
A	55	33	66	44	99	11	77	22	88

	p		q		r				
A	55	33	66	44	99				

	p	q	r						
A	55	33	66						

	p	r							
A	55	33							

Merge-Sort

	p				q				r
A	33	55	66	44	99	11	77	22	88

	p		q		r				
A	33	55	66	44	99				

	p	q	r						
A	33	55	66						

	p	r							
A	33	55							

Merge-Sort

	p				q				r
A	33	55	66	44	99	11	77	22	88

	p		q		r				
A	33	55	66	44	99				

	p	q	r						
A	33	55	66						

			$p = r$						
A			66						

Merge-Sort

	p				q				r
A	33	55	66	44	99	11	77	22	88

	p		q		r				
A	33	55	66	44	99				

	p	q	r						
A	33	55	66						

Merge-Sort

	p				q				r
A	33	55	66	44	99	11	77	22	88

	p		q		r				
A	33	55	66	44	99				

Merge-Sort

	p				q				r
A	33	55	66	44	99	11	77	22	88

	p		q		r				
A	33	55	66	44	99				

				p	r				
A				44	99				

Merge-Sort

	p				q				r
A	33	55	66	44	99	11	77	22	88

	p		q		r				
A	33	55	66	44	99				

				p	r				
A				44	99				

Merge-Sort

	p				q				r
A	33	55	66	44	99	11	77	22	88

	p		q		r				
A	33	55	66	44	99				

Merge-Sort

	p				q				r
A	33	44	55	66	99	11	77	22	88

	p		q		r				
A	33	44	55	66	99				

Merge-Sort

	p				q				r
A	33	44	55	66	99	11	77	22	88

Merge-Sort

	p				q				r
A	33	44	55	66	99	11	77	22	88

						p			r
A						11	77	22	88

Merge-Sort

	p				q				r
A	33	44	55	66	99	11	77	22	88

						p			r
A						11	77	22	88

						p	r		
A						11	77		

Merge-Sort

	p				q				r
A	33	44	55	66	99	11	77	22	88

						p			r
A						11	77	22	88

						p	r		
A						11	77		

Merge-Sort

	p				q				r
A	33	44	55	66	99	11	77	22	88

						p			r
A						11	77	22	88

Algoritmos – p.141/1063

Merge-Sort

	p				q				r
A	33	44	55	66	99	11	77	22	88

						p			r
A						11	77	22	88

								p	r
A								22	88

Merge-Sort

	p				q				r
A	33	44	55	66	99	11	77	22	88

						p			r
A						11	77	22	88

								p	r
A								22	88

Merge-Sort

	p				q				r
A	33	44	55	66	99	11	77	22	88

						p			r
A						11	77	22	88

Merge-Sort

	p				q				r
A	33	44	55	66	99	11	22	77	88

						p			r
A						11	22	77	88

Merge-Sort

	p				q				r
A	33	44	55	66	99	11	22	77	88

Merge-Sort

	p				q				r
A	11	22	33	44	55	66	77	88	99

Algoritmo de MergeSort

MERGE-SORT(A,p,r)

1: **si** $p < r$ **entonces**
2: $\quad q \leftarrow \lfloor (p+r)/2 \rfloor$
3: $\quad$ MERGE-SORT(A,p,q-1)
4: $\quad$ MERGE-SORT(A,q+1,r)
5: $\quad$ INTERCALA(A,p,q,r)

Invariantes

i1 Al final de la línea 3, los elementos entre p y $q-1$ están ordenados.

i2 Al final de la línea 4, los elementos entre $q+1$ y r están ordenados.

i3 Al final de la línea 5, los elementos entre p y r están ordenados.

Algoritmo de MergeSort

MERGE-SORT(A,p,r)

```
1: si p < r entonces
2:     q ← ⌊(p + r)/2⌋
3:     MERGE-SORT(A,p,q-1)
4:     MERGE-SORT(A,q+1,r)
5:     INTERCALA(A,p,q,r)
```

Corrección del algoritmo (Prueba inductiva)

- Para el primer caso, cuando el arreglo tiene tamaño $n = 1$, es decir, $p = r$, el algoritmo sólo trabaja hasta la línea 1 dejando el arreglo intacto. Este arreglo por tener 1 elemento ya está ordenado.
- Para el caso de $n > 1$, el problema será subdivido en subproblemas y al final de las llamadas recursivas, el algoritmo de intercala retornará un arreglo ordenado de tamaño n.
- Note que el algoritmo de intercalación no puede ser probado a no ser que las invariantes i1 e i2 sean falsas.

Consumo de Tiempo del Merge-Sort

línea		Consumo en la línea
1	$=$	$\theta(1)$
2	$=$	$\theta(1)$
3	$=$	$T(\lceil n/2 \rceil)$
4	$=$	$T(\lfloor n/2 \rfloor)$
5	$=$	$\theta(n)$
T(n)	$=$	$T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + \theta(n+2)$

Consumo de Tiempo del Merge-Sort

línea		Consumo en la línea
1	=	$\theta(1)$
2	=	$\theta(1)$
3	=	$T(\lceil n/2 \rceil)$
4	=	$T(\lfloor n/2 \rfloor)$
5	=	$\theta(n)$
T(n)	=	$T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + \theta(n+2)$

Conclusión

Por la segunda regla del Teorema Maestro, obtenemos que la complejidad del Merge-Sort queda en $\theta(nlg(n))$

Teorema Maestro

Suponga:

$$T(n) = aT(n/b) + f(n)$$

para algún $a \geq 1$ y $b > 1$ y donde n/b significa $\lceil n/b \rceil$ o $\lfloor n/b \rfloor$ entonces en general:

$$\begin{array}{lll} \text{Si } f(n) = O(n^{log_b a - e}) & \text{entonces} & T(n) = \theta(n^{log_b a}) \\ \text{Si } f(n) = \Theta(n^{log_b a}) & \text{entonces} & T(n) = \theta(n^{log_b a} lg\, n) \\ \text{Si } f(n) = \Omega(n^{log_b a + e}) & \text{entonces} & T(n) = \theta(f(n)) \end{array}$$

para un $e > 0$

Teorema Maestro Simplificado

Suponga

$$T(n) = aT(n/b) + cn^k$$

para algun $a \geq 1$ y $b > 1$ y donde n/b significa $\lceil n/b \rceil$ o $\lfloor n/b \rfloor$. Entonces

$$\text{en general:}\quad \begin{array}{lll} \text{Si } a > b^k & \text{entonces} & T(n) = \theta(n^{log_b a}) \\ \text{Si } a = b^k & \text{entonces} & T(n) = \theta(n^k lg\, n) \\ \text{Si } a < b^k & \text{entonces} & T(n) = \theta(n^k) \end{array}$$

Repaso de aulas pasadas

- La correctitud en general consiste en:
  1. Verificar que la relación vale al inicio de la primera iteración.
  2. Demostrar que la relación vale también al final.
  3. Demostrar que si la relación vale al final de la última iteración, entonces la relación junto a la condición de parada demuestran la correctitud del algoritmo.
- Existen más invariantes que deben ser demostradas para probar cada paso del algoritmo.

Repaso de la aula pasada

Asignaciones $\leftarrow$ del algoritmo INSERCIÓN.

$\leq n^2 + 3n - 3$

Repaso de la aula pasada

Asignaciones $\leftarrow$ del algoritmo INSERCIÓN.

$\leq n^2 + 3n - 3$

Consumo de tiempo (cada línea consume 1 unidad de tiempo)

$\leq (3/2)n^2 + (7/3)n - 4$

Repaso de la aula pasada

Asignaciones $\leftarrow$ del algoritmo INSERCIÓN.

$\leq n^2 + 3n - 3$

Consumo de tiempo (cada línea consume 1 unidad de tiempo)

$\leq (3/2)n^2 + (7/3)n - 4$

Consumo de tiempo (cada línea consume una unidad t_i de tiempo)

$\leq ((t_4 + t_5 + t_6)/2) \times n^2 + (t_1 + t_2 + t_3 + t_4/2 - t_5/2 - t_6/2 + t_7) \times n$ - $(t_2 + t_3 + t_4 + t_7)$

Repaso de la aula pasada

Asignaciones $\leftarrow$ del algoritmo INSERCIÓN.

$\leq n^2 + 3n - 3$

Consumo de tiempo (cada línea consume 1 unidad de tiempo)

$\leq (3/2)n^2 + (7/3)n - 4$

Consumo de tiempo (cada línea consume una unidad t_i de tiempo)

$\leq ((t_4 + t_5 + t_6)/2) \times n^2 + (t_1 + t_2 + t_3 + t_4/2 - t_5/2 - t_6/2 + t_7) \times n$ - $(t_2 + t_3 + t_4 + t_7)$

Simplificando en constantes el consumo de tiempo t_i

$\leq c_1 \times n^2 + c_2 \times n + c_3$ Donde:

- c_1, c_2, c_3 dependen del computador.
- n^2 se repetirá siempre, es algo propio del algoritmo.

Divide y vencerás (Divide and Conquer)

- Estos algoritmos tienen 3 pasos:
    1. **Dividir**: Generación de subproblemas.
    2. **Conquistar**: Resolver cada subproblemas de forma recursiva.
    3. **Combinar**: Cada solución de los subproblemas es combinada.

ntender el concepto de subproblema es importante para probar los problemas de Programación Dinámica.

Divide y vencerás (Divide and Conquer)

- Estos algoritmos tienen 3 pasos:
  1. **Dividir**: Generación de subproblemas.
  2. **Conquistar**: Resolver cada subproblema de forma recursiva.
  3. **Combinar**: Cada solución de los subproblema es combinada.

Entender el concepto de subproblema es importante para probar los problemas de Programación Dinámica.

Merge Sort

Problema

Reordenar $A[p..r]$ de modo que esté en orden creciente.

Entra

	p								r
A	55	33	66	44	99	11	77	22	88

Sale

	p								r
A	11	22	33	44	55	66	77	88	99

Algoritmo de MergeSort

MERGE-SORT(A,p,r)

1: **si** $p < r$ **entonces**
2: $\quad q \leftarrow \lfloor (p + r)/2 \rfloor$
3: $\quad$ MERGE-SORT(A,p,q-1)
4: $\quad$ MERGE-SORT(A,q+1,r)
5: $\quad$ INTERCALA(A,p,q,r)

Invariantes

- i1 Al final de la línea 3, los elementos entre p y $q - 1$ están ordenados.
- i2 Al final de la línea 4, los elementos entre $q + 1$ y r están ordenados.
- i3 Al final de la línea 5, los elementos entre p y r están ordenados.

Algoritmo de MergeSort

MERGE-SORT(A,p,r)

1: **si** $p < r$ **entonces**
2: $\quad q \leftarrow \lfloor (p + r)/2 \rfloor$
3: $\quad$ MERGE-SORT(A,p,q-1)
4: $\quad$ MERGE-SORT(A,q+1,r)
5: $\quad$ INTERCALA(A,p,q,r)

Corrección del algoritmo (Prueba inductiva)

- Para el primer caso, cuando el arreglo tiene tamaño $n = 1$, es decir, $p = r$, el algoritmo sólo trabaja hasta la línea 1 dejando el arreglo intacto. Este arreglo por tener 1 elemento ya está ordenado.
- Para el caso de $n > 1$, el problema será subdivido en subproblemas y al final de las llamadas recursivas, el algoritmo de intercala retornará un arreglo ordenado de tamaño n.
- Note que el algoritmo de intercalación no puede ser probado a no ser que las invariantes i1 e i2 sean falsas.

Consumo de Tiempo del Merge-Sort

línea		Consumo en la línea
1	=	$\Theta(1)$
2	=	$\Theta(1)$
3	=	$T(\lceil n/2 \rceil)$
4	=	$T(\lfloor n/2 \rfloor)$
5	=	$\Theta(n)$
T(n)	=	$T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + \Theta(n+2)$

Consumo de Tiempo del Merge-Sort

línea		Consumo en la línea
1	$=$	$\Theta(1)$
2	$=$	$\Theta(1)$
3	$=$	$T(\lceil n/2 \rceil)$
4	$=$	$T(\lfloor n/2 \rfloor)$
5	$=$	$\Theta(n)$
T(n)	$=$	$T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + \Theta(n+2)$

Conclusión

Por la segunda regla del Teorema Mestre, obtenemos que la complejidad del Merge-Sort queda en $\Theta(nlg(n))$

Teorema Maestro

Suponga:

$$T(n) = aT(n/b) + f(n)$$

para algún $a \geq 1$ y $b > 1$ y donde n/b significa $\lceil n/b \rceil$ o $\lfloor n/b \rfloor$ entonces en general:

Si $f(n) = O(n^{log_b a - e})$	entonces	$T(n) = \Theta(n^{log_b a})$
Si $f(n) = \Theta(n^{log_b a - e})$	entonces	$T(n) = \Theta(n^{log_b a} lgn)$
Si $f(n) = \omega(n^{log_b a - e})$	entonces	$T(n) = \Theta(f(n))$

Teorema Maestro Simplificado

Suponga

$$T(n) = aT(n/b) + cn^k$$

para algún $a \geq 1$ y $b > 1$ y donde n/b significa $\lceil n/b \rceil$ o $\lfloor n/b \rfloor$. Entonces

$$\text{en general:}\quad \begin{array}{lll} \text{Si } a > b^k & \text{entonces} & T(n) = \Theta(n^{\log_b a}) \\ \text{Si } a = b^k & \text{entonces} & T(n) = \Theta(n^k \lg n) \\ \text{Si } a < b^k & \text{entonces} & T(n) = \Theta(n^k) \end{array}$$

Problema del Segmento de Suma Máxima

Un segmento de un vector $A[1..n]$ es cualquier subvector de la forma $A[e..d]$. $1 \leq e \leq d \leq n$

Problema

Dado un vector $A[1..n]$ de números enteros, determinar un segmento $A[e..d]$ de suma máxima:

Entra

1									n
31	-41	59	26	-53	58	97	-93	-23	84

Problema del Segmento de Suma Máxima

Un segmento de un vector $A[1..n]$ es cualquier subvector de la forma $A[e..d]$. $1 \leq e \leq d \leq n$

Problema

Dado un vector $A[1..n]$ de números enteros, determinar un segmento $A[e..d]$ de suma máxima:

Entra

1									n
31	-41	59	26	-53	58	97	-93	-23	84

Sale

1		3				7			n
31	-41	59	26	-53	58	97	-93	-23	84

Problema del Segmento de Suma Máxima

Un segmento de un vector $A[1..n]$ es cualquier subvector de la forma $A[e..d]$. $1 \leq e \leq d \leq n$

Problema

Dado un vector $A[1..n]$ de números enteros, determinar un segmento $A[e..d]$ de suma máxima:

Entra

1									n
31	-41	59	26	-53	58	97	-93	-23	84

Sale

1		3				7			n
31	-41	59	26	-53	58	97	-93	-23	84

- $A[e...d] = A[3..7]$ es el Segmento de Suma Máxima
- $A[3..7]$ tiene suma de 187.

Algoritmo Arroz con Leche (básico)

SEG-MAX-3(A,n)

```
sumamax ← 0
e ← 0, d ← −1, ▷ A[e..d] es vacío
para i ← 1 hasta n hacer
    para f ← i hasta n hacer
        suma ← 0
        para k ← i hasta f hacer
            suma ← suma + A[k]
        si suma > sumamax entonces
            sumamax ← suma, e ← i, d ← f
devolver e,d,sumamax
```

Problema del Segmento de Suma Máxima - Correctitud

i0 En la línea 3 se cumple que: A[e..d] es un segmento de suma máxima.

Problema del Segmento de Suma Máxima - Correctitud

i0 En la línea 3 se cumple que: A[e..d] es un segmento de suma máxima.

i1 En la linea 3 se cumple que: $sumamax = A[e] + A[e+1] + ... + A[d]$

Problema del Segmento de Suma Máxima - Correctitud

i0 En la línea 3 se cumple que: A[e..d] es un segmento de suma máxima.

i1 En la linea 3 se cumple que: $sumamax = A[e] + A[e+1] + ... + A[d]$

i2 En la línea 4 se cumple que: $sumamax \geq 0$ además, sumamax tiene el mayor de sumatoria de los segmentos posibles desde i hasta n.

Problema del Segmento de Suma Máxima - Correctitud

i0 En la línea 3 se cumple que: A[e..d] es un segmento de suma máxima.

i1 En la linea 3 se cumple que: $sumamax = A[e] + A[e+1] + ... + A[d]$

i2 En la línea 4 se cumple que: $sumamax \geq 0$ además, sumamax tiene el mayor de sumatoria de los segmentos posibles desde i hasta n.

i3 En la línea 6 se cumple que:
$soma = A[i] + A[i+1] + A[i+2] + ...A + [k-1]$

Consumo de Tiempo del Algoritmo Arroz con Leche

Suponiendo que cada línea consume 1 unidad de tiempo.

línea		Todas las Ejecuciones de la línea
1-2	=	
3	=	
4	=	
5	=	
6	=	
7	=	
8	=	
9	=	
10	=	
total	=	

Consumo de Tiempo del Algoritmo Arroz con Leche

Suponiendo que cada línea consume 1 unidad de tiempo.

línea		Todas las Ejecuciones de la línea
1-2	=	2
3	=	
4	=	
5	=	
6	=	
7	=	
8	=	
9	=	
10	=	
total	=	

Consumo de Tiempo del Algoritmo Arroz con Leche

Suponiendo que cada línea consume 1 unidad de tiempo.

línea		Todas las Ejecuciones de la línea
1-2	=	2
3	=	n+1
4	=	(n+1) + n + (n-1) + ... + 1
5	=	n + (n-1) + ... + 1
6	=	(2 + ... +(n+1)) + (2+...+n) + ... + 2
7	=	(1 + ... + n) + (1 + ... + (n-1))+...+1
8	=	n + (n-1) + (n-2) + ... + 1
9	$\leq$	n + (n-1) + (n-2) + ... + 1
10	=	1
total	=	

Consumo de Tiempo del Algoritmo Arroz con Leche

Suponiendo que cada línea consume 1 unidad de tiempo.

línea		Todas las Ejecuciones de la línea	
1-2	$=$	2	$= \Theta(1)$
3	$=$	n+1	$= \Theta(n)$
4	$=$	(n+1) + n + (n-1) + ... + 1	$= \Theta(n^2)$
5	$=$	n + (n-1) + ... + 1	$= \Theta(n^2)$
6	$=$	(2 + ... +(n+1)) + (2+...+n) + ... + 2	$= \Theta(n^3)$
7	$=$	(1 + ... + n) + (1 + ... + (n-1))+...+1	$= \Theta(n^3)$
8	$=$	n + (n-1) + (n-2) + ... + 1	$= \Theta(n^2)$
9	$\leq$	n + (n-1) + (n-2) + ... + 1	$= O(n^2)$
10	$=$	1	$= \Theta(1)$
total	$=$		

Consumo de Tiempo del Algoritmo Arroz con Leche

Suponiendo que cada línea consume 1 unidad de tiempo.

línea		Todas las Ejecuciones de la línea	
1-2	=	2	$= \Theta(1)$
3	=	n+1	$= \Theta(n)$
4	=	(n+1) + n + (n-1) + ... + 1	$= \Theta(n^2)$
5	=	n + (n-1) + ... + 1	$= \Theta(n^2)$
6	=	(2 + ... +(n+1)) + (2+...+n) + ... + 2	$= \Theta(n^3)$
7	=	(1 + ... + n) + (1 + ... + (n-1))+...+1	$= \Theta(n^3)$
8	=	n + (n-1) + (n-2) + ... + 1	$= \Theta(n^2)$
9	$\leq$	n + (n-1) + (n-2) + ... + 1	$= O(n^2)$
10	=	1	$= \Theta(1)$
total	=	$\Theta(2n^3 + 3n^2 + n + 2) + O(n^2)$	

Consumo de Tiempo del Algoritmo Arroz con Leche

Suponiendo que cada línea consume 1 unidad de tiempo.

línea		Todas las Ejecuciones de la línea	
1-2	=	2	$= \Theta(1)$
3	=	n+1	$= \Theta(n)$
4	=	(n+1) + n + (n-1) + ... + 1	$= \Theta(n^2)$
5	=	n + (n-1) + ... + 1	$= \Theta(n^2)$
6	=	(2 + ... +(n+1)) + (2+...+n) + ... + 2	$= \Theta(n^3)$
7	=	(1 + ... + n) + (1 + ... + (n-1))+...+1	$= \Theta(n^3)$
8	=	n + (n-1) + (n-2) + ... + 1	$= \Theta(n^2)$
9	$\leq$	n + (n-1) + (n-2) + ... + 1	$= O(n^2)$
10	=	1	$= \Theta(1)$
total	=	$\Theta(2n^3 + 3n^2 + n + 2) + O(n^2)$	$= \Theta(n^3)$

Algoritmo Arroz con Huevo

SEG-MAX-2(A,n)

$sumamax \leftarrow 0$
$e \leftarrow 0$, $d \leftarrow -1$, $\triangleright$ $A[e..d]$ es vacío
para $i \leftarrow 1$ hasta n **hacer**
    $suma \leftarrow 0$
    para $f \leftarrow i$ hasta n **hacer**
        $suma \leftarrow suma + A[f]$
        si $suma > sumamax$ **entonces**
            $sumamax \leftarrow suma$, $e \leftarrow i$, $d \leftarrow f$
devolver e,d,$sumamax$

Correctitud del Algoritmo Arroz con Huevo

i0 En la línea 3 se cumple que: A[e..d] es un segmento de suma máxima. Además, $e < i$.

i1 En la línea 3 se cumple que: $sumamax = A[e] + A[e+1] + ... + A[d]$.

i2 En la línea 5 se cumple que: $suma = A[i] + A[i+1] + ... + A[f-1]$.

Consumo de Tiempo del Algoritmo Arroz con Huevo

Suponiendo que cada línea consume 1 unidad de tiempo.

línea		todas las ejecuciones de la línea
1-2	$=$	
3	$=$	
4	$=$	
5	$=$	
6	$=$	
7	$=$	
8	$\leq$	
9	$=$	
Total	$=$	

Consumo de Tiempo del Algoritmo Arroz con Huevo

Suponiendo que cada línea consume 1 unidad de tiempo.

línea		todas las ejecuciones de la línea
1-2	$=$	2
3	$=$	
4	$=$	
5	$=$	
6	$=$	
7	$=$	
8	$\leq$	
9	$=$	
Total	$=$	

Consumo de Tiempo del Algoritmo Arroz con Huevo

Suponiendo que cada línea consume 1 unidad de tiempo.

línea		todas las ejecuciones de la línea
1-2	$=$	2
3	$=$	n+1
4	$=$	n
5	$=$	(n+1) + n + ... + 2
6	$=$	n + (n-1) + ... + 1
7	$=$	n + (n-1) + ... + 1
8	$\leq$	n + (n-1) + ... + 1
9	$=$	1
Total	$=$	

Consumo de Tiempo del Algoritmo Arroz con Huevo

Suponiendo que cada línea consume 1 unidad de tiempo.

línea		todas las ejecuciones de la línea	
1-2	$=$	2	$= \Theta(1)$
3	$=$	n+1	$= \Theta(n)$
4	$=$	n	$= \Theta(n)$
5	$=$	(n+1) + n + ... + 2	$= \Theta(n^2)$
6	$=$	n + (n-1) + ... + 1	$= \Theta(n^2)$
7	$=$	n + (n-1) + ... + 1	$= \Theta(n^2)$
8	$\leq$	n + (n-1) + ... + 1	$= O(n^2)$
9	$=$	1	$= \Theta(1)$
Total	$=$		

Consumo de Tiempo del Algoritmo Arroz con Huevo

Suponiendo que cada línea consume 1 unidad de tiempo.

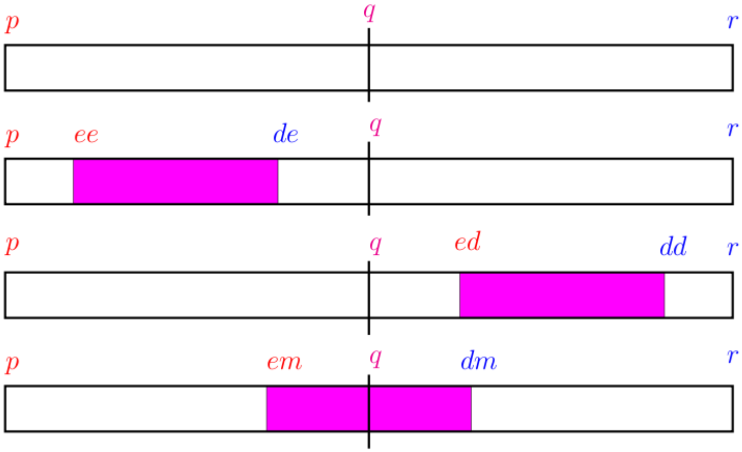
línea		todas las ejecuciones de la línea	
1-2	$=$	2	$= \Theta(1)$
3	$=$	n+1	$= \Theta(n)$
4	$=$	n	$= \Theta(n)$
5	$=$	(n+1) + n + ... + 2	$= \Theta(n^2)$
6	$=$	n + (n-1) + ... + 1	$= \Theta(n^2)$
7	$=$	n + (n-1) + ... + 1	$= \Theta(n^2)$
8	$\leq$	n + (n-1) + ... + 1	$= O(n^2)$
9	$=$	1	$= \Theta(1)$
Total	$=$	$\Theta(3n^2 + 2n + 2) + O(n^2)$	

Consumo de Tiempo del Algoritmo Arroz con Huevo

Suponiendo que cada línea consume 1 unidad de tiempo.

línea		todas las ejecuciones de la línea	
1-2	$=$	2	$= \Theta(1)$
3	$=$	n+1	$= \Theta(n)$
4	$=$	n	$= \Theta(n)$
5	$=$	(n+1) + n + ... + 2	$= \Theta(n^2)$
6	$=$	n + (n-1) + ... + 1	$= \Theta(n^2)$
7	$=$	n + (n-1) + ... + 1	$= \Theta(n^2)$
8	$\leq$	n + (n-1) + ... + 1	$= O(n^2)$
9	$=$	1	$= \Theta(1)$
Total	$=$	$\Theta(3n^2 + 2n + 2) + O(n^2)$	$= \Theta(n^2)$

Algoritmo División y Conquista



Algoritmo División y Vencerás

SEG-MAX-DV(A,p,r)

si $p = r$ **entonces**
    devolver $max(0, A[p])$
$q \leftarrow \lfloor (p + r)/2 \rfloor$
$maxizq \leftarrow$ SEG-MAX-DV(A,p,q)
$maxder \leftarrow$ SEG-MAX-DV(A,q+1,r)
$max2izq \leftarrow suma \leftarrow A[q]$
para $i \leftarrow q - 1$ desc p **hacer**
    $suma \leftarrow suma + A[i]$
    $max2izq \leftarrow max(max2izq, suma)$
$max2der \leftarrow suma \leftarrow A[q + 1]$
para $f \leftarrow q + 2$ hasta r **hacer**
    $suma \leftarrow suma + A[f]$
    $max2der \leftarrow max(max2der, suma)$
$maxcruz \leftarrow max2izq + max2der$
devolver $max(maxizq, maxcruz, maxder)$

Correctitud del ALgoritmo de Divide y Vencerás

- i0 maxizq es la suma máxima de un segmento de A[p..q].
- i1 maxder es la suma máxima de un segmento de A[q+1 .. r]
- i2 maxcruz es la suma máxima de un segmento de la forma A[i .. f] con i $\leq$ q y q+1 $\leq$ f.

Consumo de Tiempo del Algoritmo de División y Vencerás

Suponiendo que cada línea consume 1 unidad de tiempo.

línea		Todas las ejecuciones de la línea
1-2	=	
3	=	
4	=	
5	=	
6	=	
7-8	=	
9	=	
10	=	
11-12	=	
13-14	=	
Total	=	

Consumo de Tiempo del Algoritmo de División y Vencerás

Suponiendo que cada línea consume 1 unidad de tiempo.

línea		Todas las ejecuciones de la línea
1-2	=	2
3	=	$T(\lceil \frac{n}{2} \rceil)$
4	=	
5	=	
6	=	
7-8	=	
9	=	
10	=	
11-12	=	
13-14	=	
Total	=	

Consumo de Tiempo del Algoritmo de División y Vencerás

Suponiendo que cada línea consume 1 unidad de tiempo.

línea		Todas las ejecuciones de la línea
1-2	$=$	2
3	$=$	$T(\lceil \frac{n}{2} \rceil)$
4	$=$	$T(\lfloor \frac{n}{2} \rfloor)$
5	$=$	1
6	$=$	$\lceil \frac{n}{2} \rceil + 1$
7-8	$=$	$\lceil \frac{n}{2} \rceil$
9	$=$	1
10	$=$	$\lfloor \frac{n}{2} \rfloor + 1$
11-12	$=$	$\lfloor \frac{n}{2} \rfloor$
13-14	$=$	2
Total	$=$	

Consumo de Tiempo del Algoritmo de División y Vencerás

Suponiendo que cada línea consume 1 unidad de tiempo.

línea		Todas las ejecuciones de la línea	
1-2	$=$	2	$= \Theta(1)$
3	$=$	$T(\lceil \frac{n}{2} \rceil)$	$= T(\lceil \frac{n}{2} \rceil)$
4	$=$	$T(\lfloor \frac{n}{2} \rfloor)$	$= T(\lfloor \frac{n}{2} \rfloor)$
5	$=$	1	$= \Theta(1)$
6	$=$	$\lceil \frac{n}{2} \rceil + 1$	$= \Theta(n)$
7-8	$=$	$\lceil \frac{n}{2} \rceil$	$= \Theta(n)$
9	$=$	1	$= \Theta(1)$
10	$=$	$\lfloor \frac{n}{2} \rfloor + 1$	$= \Theta(n)$
11-12	$=$	$\lfloor \frac{n}{2} \rfloor$	$= \Theta(n)$
13-14	$=$	2	$= \Theta(1)$
Total	$=$		

Consumo de Tiempo del Algoritmo de División y Vencerás

Suponiendo que cada línea consume 1 unidad de tiempo.

línea		Todas las ejecuciones de la línea	
1-2	$=$	2	$= \Theta(1)$
3	$=$	$T(\lceil \frac{n}{2} \rceil)$	$= T(\lceil \frac{n}{2} \rceil)$
4	$=$	$T(\lfloor \frac{n}{2} \rfloor)$	$= T(\lfloor \frac{n}{2} \rfloor)$
5	$=$	1	$= \Theta(1)$
6	$=$	$\lceil \frac{n}{2} \rceil + 1$	$= \Theta(n)$
7-8	$=$	$\lceil \frac{n}{2} \rceil$	$= \Theta(n)$
9	$=$	1	$= \Theta(1)$
10	$=$	$\lfloor \frac{n}{2} \rfloor + 1$	$= \Theta(n)$
11-12	$=$	$\lfloor \frac{n}{2} \rfloor$	$= \Theta(n)$
13-14	$=$	2	$= \Theta(1)$
Total	$=$	$T(\lceil \frac{n}{2} \rceil) + T(\lfloor \frac{n}{2} \rfloor) + \Theta(4n+4)$	

Consumo de Tiempo del Algoritmo de División y Vencerás

Suponiendo que cada línea consume 1 unidad de tiempo.

línea		Todas las ejecuciones de la línea	
1-2	$=$	2	$= \Theta(1)$
3	$=$	$T(\lceil \frac{n}{2} \rceil)$	$= T(\lceil \frac{n}{2} \rceil)$
4	$=$	$T(\lfloor \frac{n}{2} \rfloor)$	$= T(\lfloor \frac{n}{2} \rfloor)$
5	$=$	1	$= \Theta(1)$
6	$=$	$\lceil \frac{n}{2} \rceil + 1$	$= \Theta(n)$
7-8	$=$	$\lceil \frac{n}{2} \rceil$	$= \Theta(n)$
9	$=$	1	$= \Theta(1)$
10	$=$	$\lfloor \frac{n}{2} \rfloor + 1$	$= \Theta(n)$
11-12	$=$	$\lfloor \frac{n}{2} \rfloor$	$= \Theta(n)$
13-14	$=$	2	$= \Theta(1)$
Total	$=$	$T(\lceil \frac{n}{2} \rceil) + T(\lfloor \frac{n}{2} \rfloor) + \Theta(4n + 4)$	

Consumo de Tiempo del Algoritmo División y Conquista

El consumo de tiempo $T(n)$ es:

$$T(n) = T(\lceil \tfrac{n}{2} \rceil) + T(\lfloor \tfrac{n}{2} \rfloor) + \Theta(4n + 4)$$

Podemos reescribirlo así

$$T(n) = 2T(\tfrac{n}{2}) + \Theta(n)$$

Aplicando el teorema maestro

Obtenemos que la fórmula cerrada del tiempo de ejecución del División y Vencerás para el problema del Segmento de Suma Máxima es:

$$T(n) = \Theta(n^{log_2 2} lg(n)) = \Theta(nlg(n))$$

Algoritmo Lineal para encontrar el Segmento de Suma Máximo

Propuesto por Jay Kadane.

SEG-MAX-LIN(A,n)

```
sumamax ← 0
e ← 0 d ← −1 ▷ A[e..d] es vacío
suma ← 0
i ← 1 f ← 0 ▷ A[i..f] es vacío
mientras f < n hacer
    f ← f + 1
    suma ← suma + A[f]
    si suma < 0 entonces
        suma ← 0 i ← f + 1
    si no, si suma > sumamax entonces
        sumamax ← suma e ← i d ← f
devolver  e,d,sumamax
```

Correctitud del Algoritmo Lineal

- En la línea 5 se cumple que:
  - i0 $A[e..d]$ es un segmento de suma máxima con $d \leq f$.
  - i1 $sumamax = A[e] + A[e+1] + A[e+2] + ... + A[d]$.
  - i2 $A[i..f]$ es un segmento de suma máxima con el término en f;
  - i3 $suma = A[i] + A[i+1] + A[i+2] + ... + A[f]$
  - i4 para $k = i, i+1, ..., f < 0$
- En la línea 9 se cumple que:
  - i5 $suma = A[i] + A[i+1] + A[i+2] + ... + A[f] < 0$.
  - i6 Las relaciones invariantes i4 e i5 implican que sea correcto:
    $para k = 1, 2, 3, ..., f$, vale que
    $A[k] + A[k+1] + A[k+2] + ... + A[f] < 0$

Consumo de Tiempo del Algoritmo Lineal

línea		Todas las ejecuciones de la línea	
1-2	$=$	2	$= \Theta(1)$
3-4	$=$	2	$= \Theta(1)$
5	$=$	n+1	$= \Theta(n)$
6-7	$=$	n	$= \Theta(n)$
8	$=$	n	$= \Theta(n)$
9-10	$=$	n	$= \Theta(n)$
11	$\leq$	n	$= \Theta(n)$
12	$=$	1	$= \Theta(1)$
Total	$=$	$\Theta(4n+3) + O(n)$	$= \Theta(n)$

Conclusiones

- El consumo de tiempo del algoritmo SEG-MAX-3 es $\Theta(n^3)$.

Conclusiones

- El consumo de tiempo del algoritmo SEG-MAX-3 es $\Theta(n^3)$.
- El consumo de tiempo del algoritmo SEG-MAX-2 es $\Theta(n^2)$.

Conclusiones

- El consumo de tiempo del algoritmo SEG-MAX-3 es $\Theta(n^3)$.
- El consumo de tiempo del algoritmo SEG-MAX-2 es $\Theta(n^2)$.
- El consumo de tiempo del algoritmo SEG-MAX-DV es $\Theta(nlg(n))$.

Conclusiones

- El consumo de tiempo del algoritmo SEG-MAX-3 es $\Theta(n^3)$.
- El consumo de tiempo del algoritmo SEG-MAX-2 es $\Theta(n^2)$.
- El consumo de tiempo del algoritmo SEG-MAX-DV es $\Theta(nlg(n))$.
- El consumo de tiempo del algoritmo SEG-MAX-LIN es $\Theta(n)$.

Divide y vencerás (Divide and Conquer)

- Estos algoritmos tienen 3 pasos:
  1. **Dividir**: Generación de subproblemas.
  2. **Conquistar**: Resolver cada subproblema de forma recursiva.
  3. **Combinar**: Cada solución de los subproblema es combinada.

ntender el concepto de subproblema es importante para probar los problemas de Programación Dinámica.

Divide y vencerás (Divide and Conquer)

- Estos algoritmos tienen 3 pasos:
  1. **Dividir**: Generación de subproblemas.
  2. **Conquistar**: Resolver cada subproblema de forma recursiva.
  3. **Combinar**: Cada solución de los subproblema es combinada.

Entender el concepto de subproblema es importante para probar los problemas de Programación Dinámica.

Algoritmo de MergeSort

MERGE-SORT(A,p,r)

```
si p < r entonces
    q ← ⌊(p + r)/2⌋
    MERGE-SORT(A,p,q-1)
    MERGE-SORT(A,q+1,r)
    INTERCALA(A,p,q,r)
```

Complejidad

$$T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + \theta(n+2) = \Theta(nlog(n))$$

Problema del Segmento de Suma Máxima

Un segmento de un vector $A[1..n]$ es cualquier subvector de la forma $A[e..d]$. $1 \leq e \leq d \leq n$

Problema

Dado un vector $A[1..n]$ de números enteros, determinar un segmento $A[e..d]$ de suma máxima:

Entra

1									n
31	-41	59	26	-53	58	97	-93	-23	84

Sale

1		3				7			n
31	-41	59	26	-53	58	97	-93	-23	84

- $A[e...d] = A[3..7]$ es el Segmento de Suma Máxima
- $A[3..7]$ tiene suma de 187.

Algoritmo Arroz con Leche (básico)

SEG-MAX-3(A,n) = $\Theta(n^3)$

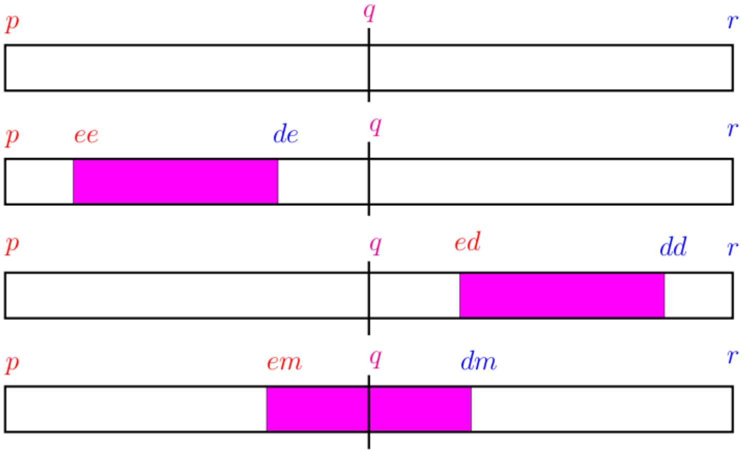
$sumamax \leftarrow 0$
$e \leftarrow 0$, $d \leftarrow -1$, $\triangleright$ $A[e..d]$ es vacío
para $i \leftarrow 1$ hasta n **hacer**
  para $f \leftarrow i$ hasta n **hacer**
    $suma \leftarrow 0$
    para $k \leftarrow i$ hasta f **hacer**
      $suma \leftarrow suma + A[k]$
    si $suma > sumamax$ **entonces**
      $sumamax \leftarrow suma$, $e \leftarrow i$, $d \leftarrow f$
devolver $e, d, sumamax$

Algoritmo Arroz con Huevo

SEG-MAX-2(A,n) $= \Theta(n^2)$

$sumamax \leftarrow 0$
$e \leftarrow 0,\ d \leftarrow -1,\ \triangleright\ A[e..d]$ es vacío
para $i \leftarrow 1$ hasta n **hacer**
    $suma \leftarrow 0$
    para $f \leftarrow i$ hasta n **hacer**
        $suma \leftarrow suma + A[f]$
        si $suma > sumamax$ **entonces**
            $sumamax \leftarrow suma,\ e \leftarrow i,\ d \leftarrow f$
devolver $e, d, sumamax$

Algoritmo División y Conquista



Algoritmo División y Vencerás

SEG-MAX-DV(A,p,r) $= \Theta(nlg(n))$

si $p = r$ **entonces**
    devolver $max(0, A[p])$
$q \leftarrow \lfloor (p + r)/2 \rfloor$
$maxizq \leftarrow$ SEG-MAX-DV(A,p,q)
$maxder \leftarrow$ SEG-MAX-DV(A,q+1,r)
$max2izq \leftarrow suma \leftarrow A[q]$
para $i \leftarrow q - 1$ desc p **hacer**
    $suma \leftarrow suma + A[i]$
    $max2izq \leftarrow max(max2izq, suma)$
$max2der \leftarrow suma \leftarrow A[q + 1]$
para $f \leftarrow q + 2$ hasta r **hacer**
    $suma \leftarrow suma + A[f]$
    $max2der \leftarrow max(max2der, suma)$
$maxcruz \leftarrow max2izq + max2der$
devolver $max(maxizq, maxcruz, maxder)$

Algoritmo Lineal para encontrar el Segmento de Suma Máximo

Propuesto por Jay Kadane.

SEG-MAX-LIN(A,n) $= \Theta(n)$

$sumamax \leftarrow 0$
$e \leftarrow 0$ $d \leftarrow -1$ $\triangleright$ $A[e..d]$ es vacío
$suma \leftarrow 0$
$i \leftarrow 1$ $f \leftarrow 0$ $\triangleright$ $A[i..f]$ es vacío
mientras $f < n$ **hacer**
    $f \leftarrow f + 1$
    $suma \leftarrow suma + A[f]$
    si $suma < 0$ **entonces**
        $suma \leftarrow 0$ $i \leftarrow f + 1$
    si no, si $suma > sumamax$ **entonces**
        $sumamax \leftarrow suma$ $e \leftarrow i$ $d \leftarrow f$
devolver $e, d, sumamax$

Conclusiones

- El consumo de tiempo del algoritmo SEG-MAX-3 es $\theta(n^3)$.

Conclusiones

- El consumo de tiempo del algoritmo SEG-MAX-3 es $\theta(n^3)$.
- El consumo de tiempo del algoritmo SEG-MAX-2 es $\theta(n^2)$.

Conclusiones

- El consumo de tiempo del algoritmo SEG-MAX-3 es $\theta(n^3)$.
- El consumo de tiempo del algoritmo SEG-MAX-2 es $\theta(n^2)$.
- El consumo de tiempo del algoritmo SEG-MAX-DV es $\theta(nlg(n))$.

Conclusiones

- El consumo de tiempo del algoritmo SEG-MAX-3 es $\theta(n^3)$.
- El consumo de tiempo del algoritmo SEG-MAX-2 es $\theta(n^2)$.
- El consumo de tiempo del algoritmo SEG-MAX-DV es $\theta(nlg(n))$.
- El consumo de tiempo del algoritmo SEG-MAX-LIN es $\theta(n)$.

Problema de la Multiplicación

n = número de cifras

Problema

Dado dos números enteros $X[1..n]$ e $Y[1..n]$, calcular el producto $X \times Y$

Entra

	12											1
X	9	2	3	4	5	5	4	5	6	2	9	8
Y	0	6	3	2	8	4	9	9	3	8	4	4

Sale

23 X . Y 1

5	8	4	4	0	8	7	2	8	6	7	0	2	7	1	4	1	0	2	9	5	1	2

Recordando época de colegio

	9	2	3	4	5	5	4	5	6	2	9	8
x		6	3	2	8	4	9	9	3	8	4	4

Recordando época de colegio

									9	2	3	4	5	5	4	5	6	2	9	8		
								x		6	3	2	8	4	9	9	3	8	4	4		
										3	6	9	3	8	2	1	8	2	5	1	9	2
									3	6	9	3	8	2	1	8	2	5	1	9	2	
								7	3	8	7	6	4	3	6	5	0	3	8	4		
							2	7	7	0	3	6	6	3	6	8	8	9	4			
						8	3	1	1	0	9	9	1	0	6	6	8	2				
					8	3	1	1	0	9	9	1	0	6	6	8	2					
				3	6	9	3	8	2	1	8	2	5	1	9	2						
			7	3	8	7	6	4	3	6	5	0	3	8	4							
		1	8	4	6	9	1	0	9	1	2	5	9	6								
	2	7	7	0	3	6	6	3	6	8	8	9	4									
5	5	4	0	7	3	2	7	3	7	7	8	8										

Recordando época de colegio

											9	2	3	4	5	5	4	5	6	2	9	8
								x				6	3	2	8	4	9	9	3	8	4	4
										3	6	9	3	8	2	1	8	2	5	1	9	2
									3	6	9	3	8	2	1	8	2	5	1	9	2	
								7	3	8	7	6	4	3	6	5	0	3	8	4		
							2	7	7	0	3	6	6	3	6	8	8	9	4			
						8	3	1	1	0	9	9	1	0	6	6	8	2				
					8	3	1	1	0	9	9	1	0	6	6	8	2					
				3	6	9	3	8	2	1	8	2	5	1	9	2						
			7	3	8	7	6	4	3	6	5	0	3	8	4							
		1	8	4	6	9	1	0	9	1	2	5	9	6								
	2	7	7	0	3	6	6	3	6	8	8	9	4									
5	5	4	0	7	3	2	7	3	7	7	8	8										
5	8	4	4	0	8	7	2	8	6	7	0	2	7	1	4	1	0	2	9	5	1	2

Multiplicación al estilo colegio

MULTIPLICA-COLE(X,Y)

Requiere: $numero_1$ ▷ Primer numero
Requiere: $numero_2$ ▷ Segundo numero

si $cifras(numero_1) < cifras(numero_2)$ **entonces**
  swap($numero_1$, $numero_2$)
B ← Matriz de 0's de tamaño $cifras(numero_2) \times cifras(numero_1)$
para $i \leftarrow cifras(numero_2)$ hasta 1 dec **hacer**
  para $j \leftarrow cifras(numero_1)$ hasta 1 dec **hacer**
    ...
$S \leftarrow$ Suma de B
devolver S

Complexidad

$\Theta(n^2)$

Multiplicación de números

- Cómo podremos ganar a $\Theta(n^2)$?

Multiplicación de números

- Cómo podremos ganar a $\Theta(n^2)$?
  - que tal **Divide y Vencerás**?

Multiplicación de números

- Cómo podremos ganar a $\Theta(n^2)$?
  - que tal **Divide y Vencerás**?
- Dividimos un número en dos, así: $X = A * (10^{2^r}) + B$, $Y = C * (10^p) + D$.

Multiplicación de números

- Cómo podremos ganar a $\Theta(n^2)$?
  - que tal **Divide y Vencerás**?
- Dividimos un número en dos, así: $X = A * (10^{2r}) + B$, $Y = C * (10^{p}) + D$.
  - $r = \lceil n/2 \rceil$, donde n es la cantidad de cifras del primer número.
  - $p = \lceil m/2 \rceil$, donde m es la cantidad de cifras del segundo número.

Multiplicación de números

- Cómo podremos ganar a $\Theta(n^2)$?
  - que tal **Divide y Vencerás**?
- Dividimos un número en dos, así: $X = A * (10^{2r}) + B$, $Y = C * (10^{p}) + D$.
  - $r = \lceil n/2 \rceil$, donde n es la cantidad de cifras del primer número.
  - $p = \lceil m/2 \rceil$, donde m es la cantidad de cifras del segundo número.
  - Veamos un ejemplo:

Multiplicación de números

$$\begin{array}{cccccccccccc} & \color{red}{4} & & & \color{blue}{1} & & & \color{red}{4} & & & \color{blue}{1} \\ X & \boxed{3} & \boxed{1} & \boxed{4} & \boxed{1} & & Y & \boxed{5} & \boxed{9} & \boxed{3} & \boxed{6} \end{array}$$

$$A\ \boxed{3}\boxed{1} \quad B\ \boxed{4}\boxed{1} \quad C\ \boxed{5}\boxed{9} \quad D\ \boxed{3}\boxed{6}$$

$$X \cdot Y = A \cdot C \times 10^4 + (A \cdot D + B \cdot C) \times 10^2 + B \cdot D$$

$A \cdot C = 1829$ $\qquad (A \cdot D + B \cdot C) = 1116 + 2419 = 3535$

$B \cdot D = 1476$

$$\begin{array}{lcccccccc} A \cdot C & 1 & 8 & 2 & 9 & 0 & 0 & 0 & 0 \\ (A \cdot D + B \cdot C) & & & 3 & 5 & 3 & 5 & 0 & 0 \\ B \cdot D & & & & & 1 & 4 & 7 & 6 \\ \hline X \cdot Y = & 1 & 8 & 6 & 4 & 4 & 9 & 7 & 6 \end{array}$$

Algoritmo de Multi-DC

Algoritmo recibe enteros X[1..n] y Y[1..n], y devuelve X $\times$ Y.

MULT(X,Y,n)

si $n = 1$ **entonces**
  devolver X $\times$ Y
$q \leftarrow \lceil n/2 \rceil$
$A \leftarrow X[q+1..n]$
$B \leftarrow X[1..q]$
$C \leftarrow X[q+1..n]$
$D \leftarrow X[1..q]$
$E \leftarrow \text{MULT}(A, C, \lfloor (n/2) \rfloor)$
$F \leftarrow \text{MULT}(B, D, \lceil (n/2) \rceil)$
$G \leftarrow \text{MULT}(A, D, \lceil (n/2) \rceil)$
$H \leftarrow \text{MULT}(B, C, \lceil (n/2) \rceil)$
$R \leftarrow E x 10^{2\lceil n/2 \rceil} + (G + H) x 10^{\lceil n/2 \rceil} + F$
devolver R

Consumo de tiempo del Algoritmo Multi-DC

línea		Todas las Execuciones
1-2	=	$2x\Theta(1)$
3	=	$\Theta(1)$
4-7	=	$4\Theta(n)$
8	=	$T(\lfloor n/2 \rfloor)$
9-11	=	$T(\lceil n/2 \rceil)$
12	=	$\Theta(1)$
13	=	$\Theta(1)$
total	=	$T(\lfloor n/2 \rfloor) + 3T(\lceil n/2 \rceil) + \Theta(4n+2)$

- Suponiendo que está en la misma clase Θ de:
  - $T'(1) = 1$
  - $T'(n) = 4T'(n/2)) + n$, para $n = 2, 2^2, 2^3, \ldots$
- La complejidad sería...

Consumo de tiempo del Algoritmo Multi-DC

línea		Todas las Execuciones
1-2	=	$2x\Theta(1)$
3	=	$\Theta(1)$
4-7	=	$4\Theta(n)$
8	=	$T(\lfloor n/2 \rfloor)$
9-11	=	$T(\lceil n/2 \rceil)$
12	=	$\Theta(1)$
13	=	$\Theta(1)$
total	=	$T(\lfloor n/2 \rfloor) + 3T(\lceil n/2 \rceil) + \Theta(4n+2)$

- Suponiendo que está en la misma clase Θ de:
  - $T'(1) = 1$
  - $T'(n) = 4T'(n/2)) + n$, para $n = 2, 2^2, 2^3, \ldots$
- La complejidad sería... $\Theta(n^2)$
- Mejoró?

Analizando el Algoritmo Multi-DC

Analizando el Algoritmo Multi-DC

$X * Y = AB * BC$

- Suponga que tiene dos números X, Y, cada uno de dos cifras.
- En un cálculo matemático, suponiendo que cada multiplicación cuesta $S/1.00$ y cada suma cuesta $S/0.01$.
- Cuánto costaría multiplicar $X * Y$?

Analizando el Algoritmo Multi-DC

$X * Y = AB * BC$

- Suponga que tiene dos números X, Y, cada uno de dos cifras.
- En un cálculo matemático, suponiendo que cada multiplicación cuesta $S/1.00$ y cada suma cuesta $S/0.01$.
- Cuánto costaría multiplicar $X * Y$?

Entonces..

- Algoritmo Multi-DC costaría 4.03.

Analizando el Algoritmo Multi-DC

$X * Y = AB * BC$

- Suponga que tiene dos números X, Y, cada uno de dos cifras.
- En un cálculo matemático, suponiendo que cada multiplicación cuesta $S/1.00$ y cada suma cuesta $S/0.01$.
- Cuánto costaría multiplicar $X * Y$?

Entonces..

- Algoritmo Multi-DC costaría 4.03.
- Método Gauss lo hace por $S/3.06$.

$X * Y$ por solo $S/3.06$

X		a	b
Y		c	d
		ad	bd
	ac	bc	
$X \cdot Y$	ac	$ad + bc$	bd

$(a + b)(c + d) = ac + ad + bc + bd$

$\Rightarrow$

$ad + bc = (a + b)(c + d) - ac - bd$

$$\begin{aligned} g &= (a + b)(c + d) \\ e &= ac \\ f &= bd \\ h &= g - e - f \\ X * Y &= e * 10^2 + h * 10^1 + f \end{aligned}$$

Algoritmo Karatsuba

KARATSUBA(X,Y,n)

si $n \leq 3$ **entonces**
    devolver $X * Y$
$q \leftarrow \lceil n/2 \rceil$
$A \leftarrow X[q + 1..n]$
$B \leftarrow X[1..q]$
$C \leftarrow X[q + 1..n]$
$D \leftarrow X[1..q]$
$E \leftarrow \text{KARATSUBA}(A, C, \lfloor (n/2) \rfloor)$
$F \leftarrow \text{KARATSUBA}(B, D, \lceil (n/2) \rceil)$
$G \leftarrow \text{KARATSUBA}(A + B, C + D, \lceil (n/2) \rceil + 1)$
$H \leftarrow G - F - E$
$R \leftarrow E x 10^{2\lceil n/2 \rceil} + H x 10^{\lceil n/2 \rceil} + F$
devolver R

Análisis Algoritmo Karatsuba

Consumo de tiempo

Análisis Algoritmo Karatsuba

Consumo de tiempo

total $= T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + T(\lceil n/2 \rceil + 1) + \Theta(5n/2)$

Análisis Algoritmo Karatsuba

Consumo de tiempo

total $= T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + T(\lceil n/2 \rceil + 1) + \Theta(5n/2)$

Pertenece a la misma clase Θ

$$\begin{array}{rcl} T'(1) & = & 1 \\ T'(n) & = & 3T'(n/2) + n \quad n = 2, 2^2, 2^3, \ldots \end{array}$$

Análisis Algoritmo Karatsuba

Consumo de tiempo

total $= T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + T(\lceil n/2 \rceil + 1) + \Theta(5n/2)$

Pertenece a la misma clase Θ

$$\begin{array}{rcl} T'(1) & = & 1 \\ T'(n) & = & 3T'(n/2) + n \quad n = 2, 2^2, 2^3, \ldots \end{array}$$

Resolviendo la recurrencia

Asintóticamente $T'(n) = \Theta(n^{lg(3)}) \approx O(n^{1.585})$

Conclusiones

Colegio $\Theta(n^2)$

Conclusiones

$$
\begin{aligned}
\text{Colegio} \quad & \Theta(n^2) \\
\text{Karatsuba} \quad & \Theta(n^{lg(3)}) = O(n^{1.585})
\end{aligned}
$$

Conclusiones

Colegio $\Theta(n^2)$

Karatsuba $\Theta(n^{lg(3)}) = O(n^{1.585})$

Schönhage y Strassen $O(n \times lg(n) \times lg(lg(n)))$

Multiplicación de Matrices

Problema

Dadas 2 matrices $X[1..n, 1..n]$ y $Y[1..n, 1..n]$, calcular el producto $X \cdot Y$

Los algoritmos tradiciones de multiplicación de matrices, consumen tiempo $\Theta(n^3)$

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} \times \begin{pmatrix} e & f \\ g & h \end{pmatrix} = \begin{pmatrix} r & s \\ t & u \end{pmatrix}$$

$$r = ae + bg$$

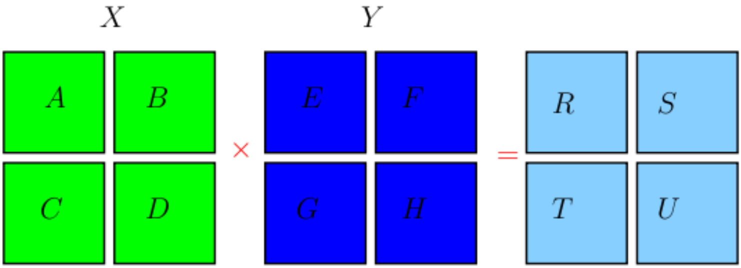
$$s = af + bh$$

$$t = ce + dg$$

$$u = cf + dh$$

La solución cuesta S/.8.04

Multiplicación de Matrices: División y Conquista



$$R = AE + BG$$
$$S = AF + BH$$
$$T = CE + DG$$
$$U = CF + DH$$

Algoritmo de Multi-Mat

El Algoritmo recibe enteros $X[1..n]$ ye $Y[1..n]$ y devuelve $X \cdot Y$

MULTI-M(X,Y,n)

```
si n=1 entonces
    devolver X · Y
(A,B,C,D) ← PARTICIONE(X,n)
(E,F,G,H) ← PARTICIONE(Y,n)
R ← MULTI-M(A,E,n/2) + MULTI-M(B,G,n/2)
S ← MULTI-M(A,F,n/2) + MULTI-M(B,H,n/2)
T ← MULTI-M(C,E,n/2) + MULTI-M(D,G,n/2)
U ← MULTI-M(C,F,n/2) + MULTI-M(D,H,n/2)
P ← CONSTRUYE-MAT(R,S,T,U)
devolver P
```

Consumo de tiempo de MULTI-MAT

$T(n)$ = Consumo de tiempo del algoritmo para multiplicar dos matrices de n líneas y n columnas.

línea		**Todas las Execuciones**
1-2	=	$\Theta(1)$
3	=	$\Theta(n^2)$
4	=	$\Theta(n^2)$
5	=	$T(n/2) + T(n/2)$
6	=	$T(n/2) + T(n/2)$
7	=	$T(n/2) + T(n/2)$
8	=	$T(n/2) + T(n/2)$
9	=	$\Theta(n^2)$
10	=	$\Theta(1)$
total	=	$8T(n/2) + \Theta(3n^2 + 1)$

Strassen $X \cdot Y$ por apenas S/ 7.18

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} \times \begin{pmatrix} e & f \\ g & h \end{pmatrix} = \begin{pmatrix} r & s \\ t & u \end{pmatrix}$$

$$\begin{aligned}
p_1 &= a(f-h) = af - ah \\
p_2 &= (a+b)h = ah + bh \\
p_3 &= (c+d)e = ce + de \\
p_4 &= d(g-e) = dg - de \\
p_5 &= (a+d)(e+h) = ae + ah + de + dh \\
p_6 &= (b-d)(g+h) = bg + bh - dg - dh \\
p_7 &= (a-c)(e+f) = ae + af - ce - cf
\end{aligned}$$

Strassen $X \cdot Y$ por apenas S/ 7.18

$$
\begin{aligned}
p_1 &= a(f-h) = af - ah \\
p_2 &= (a+b)h = ah + bh \\
p_3 &= (c+d)e = ce + de \\
p_4 &= d(g-e) = dg - de \\
p_5 &= (a+d)(e+h) = ae + ah + de + dh \\
p_6 &= (b-d)(g+h) = bg + bh - dg - dh \\
p_7 &= (a-c)(e+f) = ae + af - ce - cf \\
r &= p_5 + p_4 - p_2 + p_6 = ae + bg \\
s &= p_1 + p_2 = af + bh \\
t &= p_3 + p_4 = ce + dg \\
u &= p_5 + p_1 - p_3 - p_7 = cf + dh
\end{aligned}
$$

Algoritmo de Strassen

STRASSEN(X,Y,n)

si n=1 **entonces**
    devolver $X \cdot Y$
(A,B,C,D) $\leftarrow$ PARTICIONE(X,n)
(E,F,G,H) $\leftarrow$ PARTICIONE(Y,n)
$P_1 \leftarrow$ STRASSEN(A,F-H,n/2)
$P_2 \leftarrow$ STRASSEN(A+B,H,n/2)
$P_3 \leftarrow$ STRASSEN(C+D,E,n/2)
$P_4 \leftarrow$ STRASSEN(D,G-E,n/2)
$P_5 \leftarrow$ STRASSEN(A+D,E+H,n/2)
$P_6 \leftarrow$ STRASSEN(B-D,G+H,n/2)
$P_7 \leftarrow$ STRASSEN(A-C,E+F,n/2)
R $\leftarrow P_5 + P_4 - P_2 + P_6$
S $\leftarrow P_1 + P_2$
T $\leftarrow P_3 + P_4$
U $\leftarrow P_5 + P_1 - P_3 - P_7$
devolver P $\leftarrow$ CONSTRUYE-MAT(R,S,T,U)

Consumo del tiempo del Algoritmo Strassen

línea		Todas las Ejecuciones
1-2	=	$\Theta(1)$
3-4	=	$\Theta(n^2)$
5-11	=	$7T(n/2)$
12-15	=	$\Theta(4n^2)$
16	=	$\Theta(1)$
total	=	$7T(n/2) + \Theta(5n^2 + 2)$

Resolución de la Complejidad del Algoritmo Strassen

Aplicando la primera regla del teorema mestre, obtenemos que la fórmula cerrada del tiempo de ejecución del Algoritmo Strassen es:

$$T(n) = \Theta(n^{log_2 7})$$

$$T(n) = \Theta(n^{2.81})$$

Complejidades vistas en la aula anterior

Algoritmos de multiplicación de números

Colegio $\Theta(n^2)$

Karatsuba $\Theta(n^{lg(3)}) = O(n^{1.585})$

Schönhage y Strassen $O(n \times lg(n) \times lg(lg(n)))$

Algoritmos de multiplicación de matrices

Colegio $\Theta(n^3)$

Strassen $\Theta(n^{log_2 7}) = \Theta(n^{2.81})$

Coppersmith e Winograd $O(n^{2.38})$

Algoritmo de Partición

Problema

Ordenar un vector dado $A[p..r]$ y devolver un índice $q, p \leq q \leq r$ tal que:

$$A[p..q-1] \leq A[q] < A[q+1..r]$$

Entra

	p									r
A	99	33	55	77	11	22	88	66	33	44

Sale

	p				q					r
A	33	11	22	33	44	55	99	66	77	88

Particione

	p									r
A	99	33	55	77	11	22	88	66	33	44

Particione

	i	j								x
A	99	33	55	77	11	22	88	66	33	44

Particione

	i	j								x
A	99	33	55	77	11	22	88	66	33	44

Particione

	i	j								x
A	99	33	55	77	11	22	88	66	33	44

	i	j								x
A	33	99	55	77	11	22	88	66	33	44

Particione

	i	j								x
A	99	33	55	77	11	22	88	66	33	44

	i		j							x
A	33	99	55	77	11	22	88	66	33	44

	i			j						x
A	33	99	55	77	11	22	88	66	33	44

Particione

i		j								x
A	99	33	55	77	11	22	88	66	33	44

	i		j							x
A	33	99	55	77	11	22	88	66	33	44

	i				j					x
A	33	99	55	77	11	22	88	66	33	44

Particione

i	j									x
A	99	33	55	77	11	22	88	66	33	44

	i	j								x
A	33	99	55	77	11	22	88	66	33	44

		i				j				x
A	33	11	55	77	99	22	88	66	33	44

Particione

i		j								x
A	99	33	55	77	11	22	88	66	33	44

	i		j							x
A	33	99	55	77	11	22	88	66	33	44

		i				j				x
A	33	11	55	77	99	22	88	66	33	44

			i				j			x
A	33	11	22	77	99	55	88	66	33	44

Particione

	i		j								x
A		99	33	55	77	11	22	88	66	33	44

	i		j							x
A	33	99	55	77	11	22	88	66	33	44

		i				j				x
A	33	11	55	77	99	22	88	66	33	44

			i				j			x
A	33	11	22	77	99	55	88	66	33	44

			i					j		x
A	33	11	22	77	99	55	88	66	33	44

Particione

i		j								x
A	99	33	55	77	11	22	88	66	33	44
	i		j							x
A	33	99	55	77	11	22	88	66	33	44
		i				j				x
A	33	11	55	77	99	22	88	66	33	44
			i				j			x
A	33	11	22	77	99	55	88	66	33	44
				i						j
A	33	11	22	33	99	55	88	66	77	44

Particione

	i	j								x
A	99	33	55	77	11	22	88	66	33	44

	i		j							x
A	33	99	55	77	11	22	88	66	33	44

		i				j				x
A	33	11	55	77	99	22	88	66	33	44

			i				j			x
A	33	11	22	77	99	55	88	66	33	44

				i						j
A	33	11	22	33	99	55	88	66	77	44

	p				q					r
A	33	11	22	33	44	55	88	66	77	99

Algoritmo Partición

PARTICION(A,p,r)

1: $x \leftarrow A[r]$ $\triangleright$ x es el pivote
2: $i \leftarrow p - 1$
3: **para** $j \leftarrow p$ HASTA $r - 1$ **hacer**
4:     **si** $A[j] \leq x$ **entonces**
5:         $i \leftarrow i + 1$
6:         $A[i] \leftrightarrow A[j]$ $\triangleright$ swap
7: $A[i + 1] \leftrightarrow A[r]$ $\triangleright$ swap último
8: **devolver** $i + 1$

Invariantes

Al comienzo de cada ciclo 3-6

i0 $A[p..i] \leq x$

i1 $A[i+1..j-1] > x$

i2 $A[r] = x$

Consumo de Tiempo

Cuento tiempo consumo en función de $n = r - p$

línea	Consumo de tiempo en la línea
1-2	$=$
3	$=$
4	$=$
5-6	$=$
7-8	$=$
Total	$=$

Consumo de Tiempo

Cuento tiempo consumo en función de $n = r - p$

línea		Consumo de tiempo en la línea
1-2	$=$	$2 \times \Theta(1)$
3	$=$	$\Theta(n)$
4	$=$	$\Theta(n)$
5-6	$=$	$\Theta(n)$
7-8	$=$	$2O(n)$
Total	$=$	$\Theta(2n+4) + O(2n)$

Por tanto el algoritmo PARTICION consume tiempo $\theta(n)$

Quicksort

Reordena $A[p..r]$ en orden creciente.

QUICKSORT(A,p,r)

1: **si** $p < r$ **entonces**
2: q $\leftarrow$ PARTICIONE(A,p,r)
3: QUICKSORT(A,p,q-1)
4: QUICKSORT(A,q+1,r)

Invariantes

i0 Al comienzo de la línea 3
A[p..q-1] $\leq$ A[q] $<$ A[q+1..r]

Consumo de Tiempo del Algoritmo Quicksort

línea		Consumo de tiempo de la línea
1	=	
2	=	
3	=	
4	=	
Total	=	

Consumo de Tiempo del Algoritmo Quicksort

línea		Consumo de tiempo de la línea
1	=	$\Theta(1)$
2	=	$\Theta(n)$
3	=	$T(k)$
4	=	$T(n-k-1)$
Total	=	$T(k) - T(n-k-1) + \theta(n+1)$

Consumo de Tiempo de Quicksort mas detallado

$$
\begin{aligned}
T(0) &= \Theta(1) \\
T(1) &= \Theta(1) \\
T(n) &= T(k) + T(n-k-1) + \Theta(n) \text{ para } n > 1
\end{aligned}
$$

Consumo de Tiempo de Quicksort mas detallado

$$\begin{aligned} T(0) &= \Theta(1) \\ T(1) &= \Theta(1) \\ T(n) &= T(k) + T(n-k-1) + \Theta(n) \text{ para } n > 1 \end{aligned}$$

$$\begin{aligned} T(n) &= T(0) + T(n-1) + \Theta(n) \\ T(n) &= T(1) + T(n-2) + \Theta(n) \\ T(n) &= T(2) + T(n-3) + \Theta(n) \\ T(n) &= \ldots \\ T(n) &= T(n-1) + T(0) + \Theta(n) \end{aligned}$$

Consumo de Tiempo de Quicksort mas detallado

$$\begin{aligned} T(0) &= \Theta(1) \\ T(1) &= \Theta(1) \\ T(n) &= T(k) + T(n-k-1) + \Theta(n) \text{ para } n > 1 \end{aligned}$$

$$\begin{aligned} T(n) &= T(0) + T(n-1) + \Theta(n) \\ T(n) &= T(1) + T(n-2) + \Theta(n) \\ T(n) &= T(2) + T(n-3) + \Theta(n) \\ T(n) &= \ldots \\ T(n) &= T(n-1) + T(0) + \Theta(n) \end{aligned}$$

$$T(n) = \max_{0 \leq k \leq n-1} \{T(k) + T(n-k-1)\} + \Theta(n)$$

Consumo de Tiempo de Quicksort mas detallado

S(n) = T(n) ▷ Tiempo para el peor caso.

$$
\begin{aligned}
S(0) &= 1 \\
S(1) &= 1 \\
S(n) &= \max_{0 \leq k \leq n-1} \{S(k) + S(n-k-1)\} + n
\end{aligned}
$$

n	0	1	2	3	4	5

Consumo de Tiempo de Quicksort mas detallado

S(n) = T(n) ▷ Tiempo para el peor caso.

$$
\begin{aligned}
S(0) &= 1 \\
S(1) &= 1 \\
S(n) &= \max_{0 \leq k \leq n-1} \{S(k) + S(n-k-1)\} + n
\end{aligned}
$$

n	0	1	2	3	4	5
S(n)	1	1	2+2	5+3	9+4	14+5

Probando la complejidad

- Vamos suponer que $T(n) = \Theta(n^2)$

Probando la complejidad

- Vamos suponer que $T(n) = \Theta(n^2)$
- Para el caso base $n = 1$ probamos.

Probando la complejidad

- Vamos suponer que $T(n) = \Theta(n^2)$
- Para el caso base $n = 1$ probamos.
- Para $n > 1$

$$\begin{aligned} S(n) &= \max_{0 \leq k \leq n-1} \{S(n) + S(n-k-1)\} + n \\ &\leq \max_{0 \leq k \leq n-1} \{k^2 + (n-k-1)^2\} + n \\ &= (n-1)^2 + n \\ &= n^2 - n + 1 \end{aligned}$$

Probando la complejidad

- Vamos suponer que $T(n) = \Theta(n^2)$
- Para el caso base $n = 1$ probamos.
- Para $n > 1$

$$
\begin{aligned}
S(n) &= \max_{0 \le k \le n-1} \{S(n) + S(n-k-1)\} + n \\
&\le \max_{0 \le k \le n-1} \{k^2 + (n-k-1)^2\} + n \\
&= (n-1)^2 + n \\
&= n^2 - n + 1
\end{aligned}
$$

- $S(n) = \Theta(n^2)$ peor caso

Quicksort - mejor caso

M(n) = consumo de tiempo mínimo

$$\begin{array}{rcl} M(0) & = & 1 \\ M(1) & = & 1 \\ M(n) & = & \min\limits_{0 \leq k \leq n-1} \{M(k) + M(n-k-1)\} + n n > 1 \end{array}$$

Quicksort - mejor caso

M(n) = consumo de tiempo mínimo

$$\begin{array}{rcl} M(0) & = & 1 \\ M(1) & = & 1 \\ M(n) & = & \min_{0 \leq k \leq n-1} \{M(k) + M(n-k-1)\} + n n > 1 \end{array}$$

En el mejor de los casos $k = (n-1)/2$

$$R(n) = R(\lfloor \frac{n-1}{2} \rfloor) + R(\lceil \frac{n-1}{2} \rceil) + \Theta(n)$$

Quicksort - mejor caso

M(n) = consumo de tiempo mínimo

$$
\begin{aligned}
M(0) &= 1 \\
M(1) &= 1 \\
M(n) &= \min_{0 \leq k \leq n-1} \{M(k) + M(n-k-1)\} + nn > 1
\end{aligned}
$$

En el mejor de los casos $k = (n-1)/2$

$$R(n) = R(\lfloor \frac{n-1}{2} \rfloor) + R(\lceil \frac{n-1}{2} \rceil) + \Theta(n)$$

Solución de R(n) es $\Theta(n * lg(n))$

Consumo de Tiempo del Algoritmo Quicksort

Conclusiones.

- Peor de los Tiempos: $O(n^2)$
- Mejor de los Tiempos: $\Omega(n * lg(n))$

Consumo de Tiempo del Algoritmo Quicksort

Conclusiones.

- Peor de los Tiempos: $O(n^2)$
- Mejor de los Tiempos: $\Omega(n * lg(n))$
- Tiempo promedio es $\Theta(n * lg(n))$

Consumo de Tiempo del Algoritmo Quicksort

Conclusiones.

- Peor de los Tiempos: $O(n^2)$
- Mejor de los Tiempos: $\Omega(n * lg(n))$
- Tiempo promedio es $\Theta(n * lg(n))$ ¿Cómo?

Algoritmo de Partición

Problema

Ordenar un vector dado $A[p..r]$ y devolver un índice q, $p \leq q \leq r$ tal que:

$$A[p..q-1] \leq A[q] < A[q+1..r]$$

Entra

	p									r
A	99	33	55	77	11	22	88	66	33	44

Sale

	p				q					r
A	33	11	22	33	44	55	99	66	77	88

Particione

i		j								x
A	99	33	55	77	11	22	88	66	33	44
	i		j							x
A	33	99	55	77	11	22	88	66	33	44
		i				j				x
A	33	11	55	77	99	22	88	66	33	44
			i				j			x
A	33	11	22	77	99	55	88	66	33	44
				i						j
A	33	11	22	33	99	55	88	66	77	44
	p				q					r
A	33	11	22	33	44	55	88	66	77	99

Algoritmo Partición

PARTICION(A,p,r)

$x \leftarrow A[r]$ ▷ x es el pivote
$i \leftarrow p - 1$
para $j \leftarrow p$ HASTA $r - 1$ **hacer**
  si $A[j] \leq x$ **entonces**
    $i \leftarrow i + 1$
    $A[i] \leftrightarrow A[j]$ ▷ swap
$A[i+1] \leftrightarrow A[r]$ ▷ swap último
devolver $i + 1$

- El consumo de tiempo es $\Theta(n)$.

Algoritmo Partición

PARTICION(A,p,r)

1: $x \leftarrow A[r]$ ▷ x es el pivote
2: $i \leftarrow p - 1$
3: **para** $j \leftarrow p$ HASTA $r - 1$ **hacer**
4:     **si** $\underline{A[j] \leq x}$ **entonces**
5:         $i \leftarrow i + 1$
6:         $A[i] \leftrightarrow A[j]$ ▷ swap
7: $A[i+1] \leftrightarrow A[r]$ ▷ swap último
8: **devolver** $i + 1$

- El consumo de tiempo es $\Theta(n)$.
- El algoritmo depende si la <u>línea 4</u> tiene valor verdadero.

Quicksort

Reordena $A[p..r]$ en orden creciente.

QUICKSORT(A,p,r)

1: **si** $p < r$ **entonces**
2: q ← PARTICIONE(A,p,r)
3: QUICKSORT(A,p,q-1)
4: QUICKSORT(A,q+1,r)

Conclusiones

- Al comienzo de la línea 3 se cumple i_0: A[p..q-1] $\leq$ A[q] $<$ A[q+1..r]
- Peor de los Tiempos: $O(n^2)$
- Mejor de los Tiempos: $\Omega(n * lg(n))$
- Tiempo promedio es $\Theta(n * lg(n))$

Quicksort

Reordena $A[p..r]$ en orden creciente.

QUICKSORT(A,p,r)

1: **si** $p < r$ **entonces**
2: q ← PARTICIONE(A,p,r)
3: QUICKSORT(A,p,q-1)
4: QUICKSORT(A,q+1,r)

Conclusiones

- Al comienzo de la línea 3 se cumple i_0: A[p..q-1] $\leq$ A[q] $<$ A[q+1..r]
- Peor de los Tiempos: $O(n^2)$
- Mejor de los Tiempos: $\Omega(n * lg(n))$
- Tiempo promedio es $\Theta(n * lg(n))$ ¿Cómo?

Quicksort

Reordena $A[p..r]$ en orden creciente.

QUICKSORT(A,p,r)

```
1: si p < r entonces
2:     q ← PARTICIONE(A,p,r)
3:     QUICKSORT(A,p,q-1)
4:     QUICKSORT(A,q+1,r)
```

Conclusiones

- Al comienzo de la línea 3 se cumple i_0: A[p..q-1] $\leq$ A[q] $<$ A[q+1..r]
- Peor de los Tiempos: $O(n^2)$
- Mejor de los Tiempos: $\Omega(n * lg(n))$
- Tiempo promedio es $\Theta(n * lg(n))$ ¿Cómo?
  - El consumo de tiempo es proporcional al número de ejecuciones de la línea 4 de PARTICION.

Máximo

Problema

Encontrar el elemento máximo de un vector $A[1..n]$ de números enteros positivos diferentes entre sí.

MAX (A,n)

1: $max \leftarrow A[0]$
2: **para** $i \leftarrow 2$ hasta n **hacer**
3:     **si** $A[i] > max$ **entonces**
4:         $max \leftarrow A[i]$

Problema a analizar

Cuántas veces la línea 4 es ejecutada?

- En el peor caso?
- En el mejor caso?
- En el caso promedio?

Máximo

Problema

Encontrar el elemento máximo de un vector $A[1..n]$ de números enteros positivos diferentes entre sí.

MAX (A,n)

1: $max \leftarrow A[0]$
2: **para** $i \leftarrow 2$ hasta n **hacer**
3:     **si** $A[i] > max$ **entonces**
4:         $max \leftarrow A[i]$

Problema a analizar

Cuántas veces la línea 4 es ejecutada?

- En el peor caso? n-1
- En el mejor caso?
- En el caso promedio?

Máximo

Problema

Encontrar el elemento máximo de un vector $A[1..n]$ de números enteros positivos diferentes entre sí.

MAX (A,n)

1: $max \leftarrow A[0]$
2: **para** $i \leftarrow 2$ hasta n **hacer**
3:     **si** $A[i] > max$ **entonces**
4:         $max \leftarrow A[i]$

Problema a analizar

Cuántas veces la línea 4 es ejecutada?

- En el peor caso? n-1
- En el mejor caso? 0
- En el caso promedio?

Máximo

Problema

Encontrar el elemento máximo de un vector $A[1..n]$ de números enteros positivos diferentes entre sí.

MAX (A,n)

1: $max \leftarrow A[0]$
2: **para** $i \leftarrow 2$ hasta n **hacer**
3:     **si** $A[i] > max$ **entonces**
4:         $max \leftarrow A[i]$

Problema a analizar

Cuántas veces la línea 4 es ejecutada?

- En el peor caso? n-1
- En el mejor caso? 0
- En el caso promedio? (n-1+0)/2?

Operaciones

- Suponga que $A[1..n]$ es una permutación aleatoria uniforme de $1 \ldots n$.
- Cuál es la probabilidad $Pr\{\cdot\}$ de cada permutación? [...]
- Cuál es la probabilidad que $A[i]$ sea máximo? [...]
- Cuál es la probabilidad de E, dado F:
  - E Permutaciones $A[1..4]$ en que $A[1] = 1$ y $A[2] = 2$.
  - F Permutaciones $A[1..4]$ en que $A[1] = 1$.

  - $Pr\{E \cap F\} = Pr\{E\} =$ [...]
  - $Pr(F) =$ [...]
  - $Pr\{E|F\} = \frac{Pr\{E \cap F\}}{Pr\{F\}} =$ [...]

Aplicando conceptos

- Sea X = número total de ejecuciones de la línea 4.
- X_i = 1 o 0, dependiendo si $max \leftarrow A[i]$ es ejecutado o no.

Aplicando conceptos

- Sea X = número total de ejecuciones de la línea 4.
- $X_i = 1$ o 0, dependiendo si $max \leftarrow A[i]$ es ejecutado o no.
- $X = X_1 + X_2 + ... + X_n$
- Sea S el conjunto de todos los eventos posibles.
- $E[X] = \sum_{s \in S} X(s) \cdot Pr\{s\}$

Aplicando conceptos

- Sea X = número total de ejecuciones de la línea 4.
- $X_i = 1$ o 0, dependiendo si $max \leftarrow A[i]$ es ejecutado o no.
- $X = X_1 + X_2 + ... + X_n$
- Sea S el conjunto de todos los eventos posibles.
- $E[X] = \sum_{s \in S} X(s) \cdot Pr\{s\}$
- $E[X] = 0xPr\{X_i = 0\} + 1xPr\{X_i = 1\}$
- $Pr\{X_i = 1\}$ = probabilidad que $A[i]$ sea el máximo en $A[1..i]$.
- $E[x] = [...]$

Aplicando conceptos

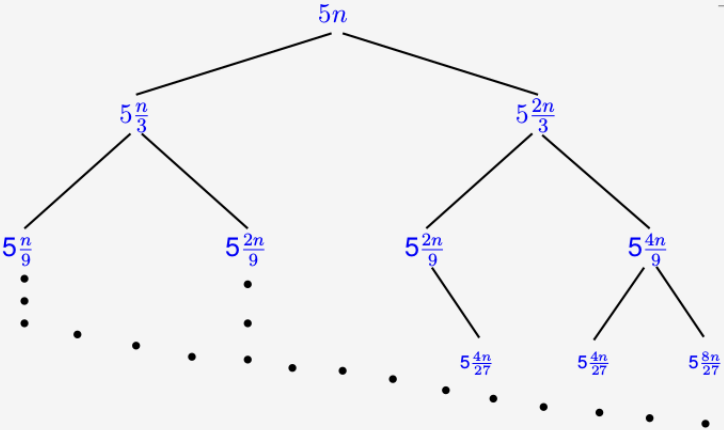
- Sea X = número total de ejecuciones de la línea 4.
- X_i = 1 o 0, dependiendo si $max \leftarrow A[i]$ es ejecutado o no.
- $X = X_1 + X_2 + ... + X_n$
- Sea S el conjunto de todos los eventos posibles.
- $E[X] = \sum_{s \in S} X(s) \cdot Pr\{s\}$
- $E[X] = 0xPr\{X_i = 0\} + 1xPr\{X_i = 1\}$
- $Pr\{X_i = 1\}$ = probabilidad que $A[i]$ sea el máximo en $A[1..i]$.
- $E[x]$ = 1+ln(x) Serie armónica

Consumo de tiempo de Quicksort (caso promedio)

Supongamos que nuestra recurrencia será partida en $\frac{1}{3}$ y $\frac{2}{3}$ entonces podemos colocar:

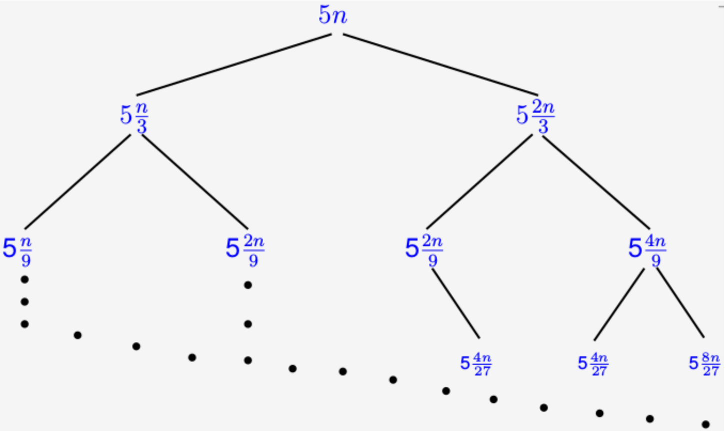
$$\begin{aligned} T(1) &= 1 \\ T(n) &= T(\left\lceil \frac{n}{3} \right\rceil) + T(\left\lfloor \frac{2n}{3} \right\rfloor) + 5n \end{aligned}$$

Consumo de tiempo de Quicksort (caso promedio)



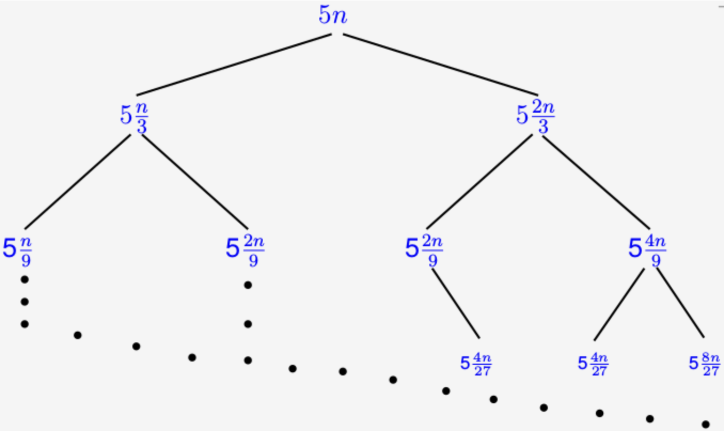
- ¿Cuántos niveles como máximo tendrá?
- ¿Cuánto suma cada nivel?

Consumo de tiempo de Quicksort (caso promedio)



- ¿Cuántos niveles como máximo tendrá? $\leq log_{3/2} n$
- ¿Cuánto suma cada nivel?

Consumo de tiempo de Quicksort (caso promedio)



- ¿Cuántos niveles como máximo tendrá? $\leq log_{3/2} n$
- ¿Cuánto suma cada nivel? $\leq 5n$

Consumo de tiempo de Quicksort (caso promedio)

$$
\begin{aligned}
T(n) &= \sum_{i=0}^{h} b_i \\
h &= Altura \\
b_i &= RecurrenciasNivel(i)
\end{aligned}
$$

Consumo de tiempo de Quicksort (caso promedio)

$$
\begin{aligned}
T(n) &= \sum_{i=0}^{h} b_i \\
h &= Altura \\
b_i &= RecurrenciasNivel(i) \\
b_i &\leq 5n
\end{aligned}
$$

Consumo de tiempo de Quicksort (caso promedio)

$$\begin{aligned} T(n) &= \sum_{i=0}^{h} b_i \\ h &= Altura \\ b_i &= RecurrenciasNivel(i) \\ b_i &\leq 5n \\ T(n) &= \underbrace{b_0 + b_1 + b_2 + ...}_{log_{3/2} n} \end{aligned}$$

Consumo de tiempo de Quicksort (caso promedio)

$$
\begin{aligned}
T(n) &= \sum_{i=0}^{h} b_i \\
h &= Altura \\
b_i &= RecurrenciasNivel(i) \\
b_i &\leq 5n \\
T(n) &= \underbrace{b_0 + b_1 + b_2 + \ldots}_{log_{3/2} n} \\
T(n) &\leq (5n) \times (log_{3/2} n)
\end{aligned}
$$

Consumo de tiempo de Quicksort (caso promedio)

$$
\begin{aligned}
T(n) &= \sum_{i=0}^{h} b_i \\
h &= Altura \\
b_i &= RecurrenciasNivel(i) \\
b_i &\leq 5n \\
T(n) &= \underbrace{b_0 + b_1 + b_2 + \ldots}_{log_{3/2} n} \\
T(n) &\leq (5n) \times (log_{3/2} n) \\
T(n) &= O(n * lg(n))
\end{aligned}
$$

Consumo de tiempo de Quicksort (caso promedio)

Regresemos a nuestra recurrencia

$$
\begin{aligned}
T(1) &= 1 \\
T(n) &= T(\lceil \frac{n}{3} \rceil) + T(\lfloor \frac{2n}{3} \rfloor) + 5n
\end{aligned}
$$

Resolviendo valores

n	T(n)
1	1

Suponga que $n > 3$. Entonces:

Consumo de tiempo de Quicksort (caso promedio)

Regresemos a nuestra recurrencia

$$\begin{aligned} T(1) &= 1 \\ T(n) &= T\left(\left\lceil \frac{n}{3} \right\rceil\right) + T\left(\left\lfloor \frac{2n}{3} \right\rfloor\right) + 5n \end{aligned}$$

Resolviendo valores

n	T(n)
1	1
2	1+1+5*2 = 12
3	1+12+5*3 = 28
4	12+12+5*4 = 44

Suponga que $n > 3$. Entonces:

Consumo de tiempo de Quicksort (caso promedio)

Regresemos a nuestra recurrencia

$$\begin{aligned} T(1) &= 1 \\ T(n) &= T\left(\left\lceil \frac{n}{3} \right\rceil\right) + T\left(\left\lfloor \frac{2n}{3} \right\rfloor\right) + 5n \end{aligned}$$

Resolviendo valores

n	T(n)
1	1
2	1+1+5*2 = 12
3	1+12+5*3 = 28
4	12+12+5*4 = 44

Vamos demostrar que $T(n) \leq 100nlg(n)$ para $n > 1$:

- Para $n = 2$ tenemos T(2) = 12 < 100*2*lg(2)
- Para $n = 3$ tenemos T(3) = 28 < 100*3*lg(3)

Suponga que $n > 3$. Entonces:

Continuación de la prueba

$$T(n) = T(\lceil n/3 \rceil) + T(\lfloor 2n/3 \rfloor) + 5n$$

Continuación de la prueba

$$
\begin{aligned}
T(n) &= T(\lceil n/3 \rceil) + T(\lfloor 2n/3 \rfloor) + 5n \\
&\overset{hi}{\leq} 100 \left\lceil \frac{n}{3} \right\rceil lg(\left\lceil \frac{n}{3} \right\rceil) + 100 \left\lfloor \frac{2n}{3} \right\rfloor lg(\left\lfloor \frac{2n}{3} \right\rfloor) + 5n
\end{aligned}
$$

Continuación de la prueba

$$
\begin{aligned}
T(n) &= T(\lceil n/3 \rceil) + T(\lfloor 2n/3 \rfloor) + 5n \\
&\overset{hi}{\leq} 100 \left\lceil \frac{n}{3} \right\rceil lg(\left\lceil \frac{n}{3} \right\rceil) + 100 \left\lfloor \frac{2n}{3} \right\rfloor lg(\left\lfloor \frac{2n}{3} \right\rfloor) + 5n \\
&< 100 \frac{n+2}{3} (lg(\frac{n}{3}) + 1) + 100 \frac{2n}{3} lg(\frac{2n}{3}) + 5n
\end{aligned}
$$

Continuación de la prueba

$$
\begin{aligned}
T(n) &= T(\lceil n/3\rceil) + T(\lfloor 2n/3\rfloor) + 5n \\
&\overset{hi}{\leq} 100\left\lceil \frac{n}{3}\right\rceil lg(\left\lceil \frac{n}{3}\right\rceil) + 100\left\lfloor \frac{2n}{3}\right\rfloor lg(\left\lfloor \frac{2n}{3}\right\rfloor) + 5n \\
&< 100\frac{n+2}{3}(lg(\frac{n}{3}) + 1) + 100\frac{2n}{3}lg(\frac{2n}{3}) + 5n \\
&< 100\frac{n+2}{3}lg(\frac{2n}{3}) + 100\frac{2n}{3}lg(\frac{2n}{3}) + 5n
\end{aligned}
$$

Continuación de la prueba

$$
\begin{aligned}
T(n) &= T(\lceil n/3 \rceil) + T(\lfloor 2n/3 \rfloor) + 5n \\
&\overset{hi}{\leq} 100\left\lceil \frac{n}{3} \right\rceil lg(\left\lceil \frac{n}{3} \right\rceil) + 100\left\lfloor \frac{2n}{3} \right\rfloor lg(\left\lfloor \frac{2n}{3} \right\rfloor) + 5n \\
&< 100\frac{n+2}{3}(lg(\frac{n}{3}) + 1) + 100\frac{2n}{3}lg(\frac{2n}{3}) + 5n \\
&< 100\frac{n+2}{3}lg(\frac{2n}{3}) + 100\frac{2n}{3}lg(\frac{2n}{3}) + 5n \\
&= 100\frac{n}{3}lg(\frac{2n}{3}) + 100\frac{2}{3}lg(\frac{2n}{3}) + 100\frac{2n}{3}lg(\frac{2n}{3}) + 5n
\end{aligned}
$$

Continuación de la prueba

$$
\begin{aligned}
T(n) &= T(\lceil n/3 \rceil) + T(\lfloor 2n/3 \rfloor) + 5n \\
&\overset{hi}{\leq} 100\left\lceil \frac{n}{3} \right\rceil lg(\left\lceil \frac{n}{3} \right\rceil) + 100\left\lfloor \frac{2n}{3} \right\rfloor lg(\left\lfloor \frac{2n}{3} \right\rfloor) + 5n \\
&< 100\frac{n+2}{3}(lg(\frac{n}{3}) + 1) + 100\frac{2n}{3}lg(\frac{2n}{3}) + 5n \\
&< 100\frac{n+2}{3}lg(\frac{2n}{3}) + 100\frac{2n}{3}lg(\frac{2n}{3}) + 5n \\
&= 100\frac{n}{3}lg(\frac{2n}{3}) + 100\frac{2}{3}lg(\frac{2n}{3}) + 100\frac{2n}{3}lg(\frac{2n}{3}) + 5n \\
&= 100nlg(\frac{2n}{3}) + 67lg(\frac{2n}{3}) + 5n
\end{aligned}
$$

Continuación de la prueba

$$
\begin{array}{rcl}
T(n) & = & T(\lceil n/3 \rceil) + T(\lfloor 2n/3 \rfloor) + 5n \\
& \overset{hi}{\leq} & 100 \left\lceil \frac{n}{3} \right\rceil lg(\left\lceil \frac{n}{3} \right\rceil) + 100 \left\lfloor \frac{2n}{3} \right\rfloor lg(\left\lfloor \frac{2n}{3} \right\rfloor) + 5n \\
& < & 100 \frac{n+2}{3} (lg(\frac{n}{3}) + 1) + 100 \frac{2n}{3} lg(\frac{2n}{3}) + 5n \\
& < & 100 \frac{n+2}{3} lg(\frac{2n}{3}) + 100 \frac{2n}{3} lg(\frac{2n}{3}) + 5n \\
& = & 100 \frac{n}{3} lg(\frac{2n}{3}) + 100 \frac{2}{3} lg(\frac{2n}{3}) + 100 \frac{2n}{3} lg(\frac{2n}{3}) + 5n \\
& = & 100 n lg(\frac{2n}{3}) + 67 lg(\frac{2n}{3}) + 5n \\
& = & (100n + 67)((lg(\frac{2}{3}) = -0.58) + lg(n)) + 5n
\end{array}
$$

Continuación de la prueba

$$
\begin{aligned}
T(n) &= T(\lceil n/3 \rceil) + T(\lfloor 2n/3 \rfloor) + 5n \\
&\overset{hi}{\leq} 100 \left\lceil \frac{n}{3} \right\rceil lg(\left\lceil \frac{n}{3} \right\rceil) + 100 \left\lfloor \frac{2n}{3} \right\rfloor lg(\left\lfloor \frac{2n}{3} \right\rfloor) + 5n \\
&< 100 \frac{n+2}{3}(lg(\frac{n}{3}) + 1) + 100 \frac{2n}{3} lg(\frac{2n}{3}) + 5n \\
&< 100 \frac{n+2}{3} lg(\frac{2n}{3}) + 100 \frac{2n}{3} lg(\frac{2n}{3}) + 5n \\
&= 100 \frac{n}{3} lg(\frac{2n}{3}) + 100 \frac{2}{3} lg(\frac{2n}{3}) + 100 \frac{2n}{3} lg(\frac{2n}{3}) + 5n \\
&= 100 n lg(\frac{2n}{3}) + 67 lg(\frac{2n}{3}) + 5n \\
&= (100n + 67)((lg(\frac{2}{3}) = -0.58) + lg(n)) + 5n \\
&= 100 n lg(n) + 9 lg(n) + 5n - 39
\end{aligned}
$$

Continuación de la prueba

$$
\begin{aligned}
T(n) &= T(\lceil n/3 \rceil) + T(\lfloor 2n/3 \rfloor) + 5n \\
&\overset{hi}{\leq} 100\left\lceil \frac{n}{3} \right\rceil lg(\left\lceil \frac{n}{3} \right\rceil) + 100\left\lfloor \frac{2n}{3} \right\rfloor lg(\left\lfloor \frac{2n}{3} \right\rfloor) + 5n \\
&< 100\frac{n+2}{3}(lg(\frac{n}{3}) + 1) + 100\frac{2n}{3}lg(\frac{2n}{3}) + 5n \\
&< 100\frac{n+2}{3}lg(\frac{2n}{3}) + 100\frac{2n}{3}lg(\frac{2n}{3}) + 5n \\
&= 100\frac{n}{3}lg(\frac{2n}{3}) + 100\frac{2}{3}lg(\frac{2n}{3}) + 100\frac{2n}{3}lg(\frac{2n}{3}) + 5n \\
&= 100nlg(\frac{2n}{3}) + 67lg(\frac{2n}{3}) + 5n \\
&= (100n + 67)((lg(\frac{2}{3}) = -0.58) + lg(n)) + 5n \\
&= 100nlg(n) + 9lg(n) + 5n - 39 \\
&= O(100nlg(n))
\end{aligned}
$$

Continuación de la prueba

$$
\begin{aligned}
T(n) &= T(\lceil n/3\rceil) + T(\lfloor 2n/3\rfloor) + 5n \\
&\overset{hi}{\leq} 100\left\lceil\frac{n}{3}\right\rceil lg(\left\lceil\frac{n}{3}\right\rceil) + 100\left\lfloor\frac{2n}{3}\right\rfloor lg(\left\lfloor\frac{2n}{3}\right\rfloor) + 5n \\
&< 100\frac{n+2}{3}(lg(\frac{n}{3}) + 1) + 100\frac{2n}{3}lg(\frac{2n}{3}) + 5n \\
&< 100\frac{n+2}{3}lg(\frac{2n}{3}) + 100\frac{2n}{3}lg(\frac{2n}{3}) + 5n \\
&= 100\frac{n}{3}lg(\frac{2n}{3}) + 100\frac{2}{3}lg(\frac{2n}{3}) + 100\frac{2n}{3}lg(\frac{2n}{3}) + 5n \\
&= 100nlg(\frac{2n}{3}) + 67lg(\frac{2n}{3}) + 5n \\
&= (100n + 67)((lg(\frac{2}{3}) = -0.58) + lg(n)) + 5n \\
&= 100nlg(n) + 9lg(n) + 5n - 39 \\
&= O(100nlg(n)) \ldots Comprobado!
\end{aligned}
$$

Quicksort caso promedio

Sea $C(n)$ el número de comparaciones en la línea 4 de PARTICION, podemos decir que:

$$
\begin{aligned}
C(0) &= 0 \\
C(n) &= \frac{1}{n}\left(\sum_{k=0}^{n-1} C(k) + C(n-k-1)\right) + n - 1 \\
&\quad para \quad n > 0
\end{aligned}
$$

Simplificando:

$$
\begin{aligned}
C(0) &= 0 \\
C(n) &= \frac{2}{n}\sum_{k=0}^{n-1} C(k) + n - 1 \\
&\quad para \quad n > 0
\end{aligned}
$$

Quicksort caso promedio

Multiplicando ambos miembros por n

$$nC(n) = 2\sum_{k=0}^{n-1} C(k) + n^2 - n \quad (1)$$

Si $n = n - 1$

$$n(n-1)C(n-1) = 2\sum_{k=0}^{n-2} C(k) + (n-1)^2 - (n-1) \quad (2)$$

De (2) y (1) obtenemos:

$$\begin{aligned} nC(n) - (n-1)C(n-1) &= 2C(n-1) + 2n - 2 \quad (3) \\ nC(n) &= (n+1)C(n-1) + 2n - 2 \quad (4) \end{aligned}$$

Continuando con la prueba

Multiplicando ambos miembros por $\frac{1}{n(n+1)}$ obtenemos:

$$\frac{1}{n+1}C(n) = \frac{1}{n}C(n-1) + 2\frac{2}{n+1} - 2\frac{2}{n(n+1)} \tag{5}$$

Sustituyendo $S(n) = \frac{1}{n+1}C(n)$ llegamos a:

$$\begin{aligned} S(0) &= 0 \\ S(n) &= S(n-1) + \frac{2}{n+1} - \frac{2}{n(n+1)} \\ \textit{para} \quad & n > 1 \end{aligned}$$

Tiene como solución: [...]

Continuando con la prueba

Multiplicando ambos miembros por $\frac{1}{n(n+1)}$ obtenemos:

$$\frac{1}{n+1}C(n) = \frac{1}{n}C(n-1) + 2\frac{2}{n+1} - 2\frac{2}{n(n+1)} \tag{5}$$

Sustituyendo $S(n) = \frac{1}{n+1}C(n)$ llegamos a:

$$\begin{array}{rcl} S(0) & = & 0 \\ S(n) & = & S(n-1) + \dfrac{2}{n+1} - \dfrac{2}{n(n+1)} \\ & \textit{para} & n > 1 \end{array}$$

Tiene como solución: $S(n) = 2(H_{n+1} - 2 - \frac{1}{n+1})$

Continuando con la prueba

Concluimos que:

$$C(n) = 2(n+1)H_{n+1} - 4(n+1) + 2$$

Recordamos:

$$ln(n+1) < H_{n+1} < 1 + ln(n+1)$$

Entonces

$$C(n) = O(nln(n))$$

Quicksort Aleatorizado

PARTICIONE-ALE(A,p,r)

```
i ← RANDOM(p, r)
A[i] ↔ A[r]
devolver PARTICIONE(A,p,r)
```

QUICKSORT-ALE(A,p,r)

```
si p < r entonces
    q ← PARTICIONE-ALE(A,p,r)
    QUICKSORT-ALE(A,p,q-1)
    QUICKSORT-ALE(A,q+1,r)
```

Para encontrar el caso promedio, también analizaremos la línea 4 de PARTICIONE.

Consumo de tiempo esperado

Suponga que A[p..r] es una permutación de 1..n

X_{ab} = Número de comparaciones entre a y b en la línea 4 de PARTICIONE.

Queremos calcular:

$$\begin{aligned} X &= \text{Número de comparaciones } \underline{A[j] \leq x} \\ &= \sum_{a=1}^{n-1} \sum_{b=a+1}^{n} X_{ab} \end{aligned}$$

Recordemos:

- Variable aleatoria: X=k significa $s \in S : X(s) = k$
- Esperanza E:

$$\begin{aligned} \mathrm{E}[X] &= \sum_{k \in X(S)} k * Pr\{X = k\} \\ &= \sum_{s \in S} X(s) * Pr\{s\} \end{aligned}$$

Consumo de tiempo esperado

Suponiendo que a $<$ n,

- X_{an} es 1 si el primer pivote se encuentra en $\{a...b\}$ es a o b
- 0 caso contrario.
- Cuál es la probabilidad de X_{an} valer 1?

$$
\begin{aligned}
\mathtt{E}[X_{an}] &= Pr\{X_{an}\} \\
&= \frac{1}{b-a+1} +
\end{aligned}
$$

$$X = \sum_{a=1}^{n-1} \sum_{b=a+1}^{n} X_{ab}$$

$\mathtt{E} = ?$

Consumo de tiempo esperado

Suponiendo que a < n,

- X_{an} es 1 si el primer pivote se encuentra en {a...b} es a o b
- 0 caso contrario.
- Cuál es la probabilidad de X_{an} valer 1?

$$\begin{aligned} \mathtt{E}[X_{an}] &= Pr\{X_{an}\} \\ &= \frac{1}{b-a+1} + \frac{1}{b-a+1} \end{aligned}$$

$$X = \sum_{a=1}^{n-1} \sum_{b=a+1}^{n} X_{ab}$$

$\mathtt{E} = ?$

Consumo de tiempo esperado (Cormen, 7.4)

$$\mathtt{E}[X] = \sum_{a=1}^{n-1} \sum_{b=a+1}^{n} E[X_{ab}]$$

Consumo de tiempo esperado (Cormen, 7.4)

$$
\begin{aligned}
\mathtt{E}[X] &= \sum_{a=1}^{n-1} \sum_{b=a+1}^{n} E[X_{ab}] \\
&= \sum_{a=1}^{n-1} \sum_{b=a+1}^{n} Pr\{X_{ab} = 1\}
\end{aligned}
$$

Consumo de tiempo esperado (Cormen, 7.4)

$$
\begin{aligned}
\mathtt{E}[X] &= \sum_{a=1}^{n-1} \sum_{b=a+1}^{n} E[X_{ab}] \\
&= \sum_{a=1}^{n-1} \sum_{b=a+1}^{n} Pr\{X_{ab} = 1\} \\
&= \sum_{a=1}^{n-1} \sum_{b=a+1}^{n} \frac{2}{b - a + 1}
\end{aligned}
$$

Consumo de tiempo esperado (Cormen, 7.4)

$$
\begin{aligned}
\mathrm{E}[X] &= \sum_{a=1}^{n-1} \sum_{b=a+1}^{n} E[X_{ab}] \\
&= \sum_{a=1}^{n-1} \sum_{b=a+1}^{n} Pr\{X_{ab} = 1\} \\
&= \sum_{a=1}^{n-1} \sum_{b=a+1}^{n} \frac{2}{b-a+1} \\
&= \sum_{a=1}^{n-1} \sum_{k=1}^{n-a} \frac{2}{k+1}
\end{aligned}
$$

Consumo de tiempo esperado (Cormen, 7.4)

$$
\begin{aligned}
\mathtt{E}[X] &= \sum_{a=1}^{n-1} \sum_{b=a+1}^{n} E[X_{ab}] \\
&= \sum_{a=1}^{n-1} \sum_{b=a+1}^{n} Pr\{X_{ab} = 1\} \\
&= \sum_{a=1}^{n-1} \sum_{b=a+1}^{n} \frac{2}{b - a + 1} \\
&= \sum_{a=1}^{n-1} \sum_{k=1}^{n-a} \frac{2}{k + 1} \\
&< \sum_{a=1}^{n-1} 2(\frac{1}{1} + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{n})
\end{aligned}
$$

Consumo de tiempo esperado (Cormen, 7.4)

$$
\begin{aligned}
\mathtt{E}[X] &= \sum_{a=1}^{n-1} \sum_{b=a+1}^{n} E[X_{ab}] \\
&= \sum_{a=1}^{n-1} \sum_{b=a+1}^{n} Pr\{X_{ab} = 1\} \\
&= \sum_{a=1}^{n-1} \sum_{b=a+1}^{n} \frac{2}{b - a + 1} \\
&= \sum_{a=1}^{n-1} \sum_{k=1}^{n-a} \frac{2}{k + 1} \\
&< \sum_{a=1}^{n-1} 2(\frac{1}{1} + \frac{1}{2} + \frac{1}{3} + ... + \frac{1}{n}) \\
&< 2n(1 + ln(n))
\end{aligned}
$$

Consumo de tiempo esperado (Cormen, 7.4)

$$
\begin{aligned}
\mathtt{E}[X] &= \sum_{a=1}^{n-1} \sum_{b=a+1}^{n} E[X_{ab}] \\
&= \sum_{a=1}^{n-1} \sum_{b=a+1}^{n} Pr\{X_{ab} = 1\} \\
&= \sum_{a=1}^{n-1} \sum_{b=a+1}^{n} \frac{2}{b-a+1} \\
&= \sum_{a=1}^{n-1} \sum_{k=1}^{n-a} \frac{2}{k+1} \\
&< \sum_{a=1}^{n-1} 2(\frac{1}{1} + \frac{1}{2} + \frac{1}{3} + ... + \frac{1}{n}) \\
&< 2n(1 + ln(n))...Comprobado!
\end{aligned}
$$

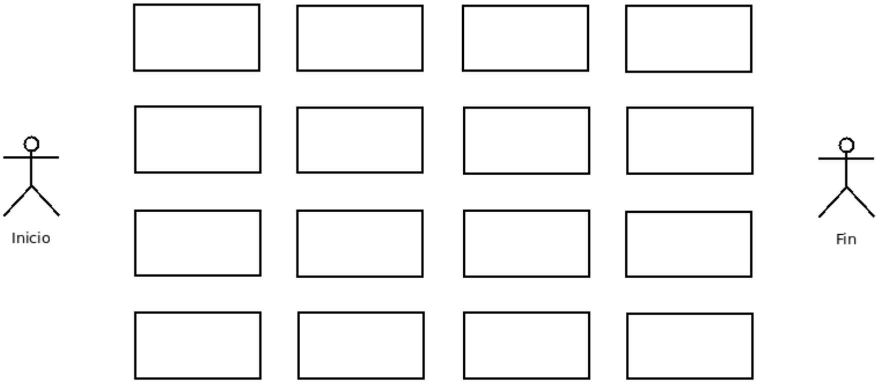
Conclusión

- Hemos visto diferentes formas de llegar a la complejidad promedio del algoritmo:
  - Análisis del árbol de recursión.
  - Análisis de recurrencia sobre el número de comparaciones.
  - Probabiliadades, encontrar el valor de la Esperanza.
- Quicksort aleatorio es $O(n \log(n))$.

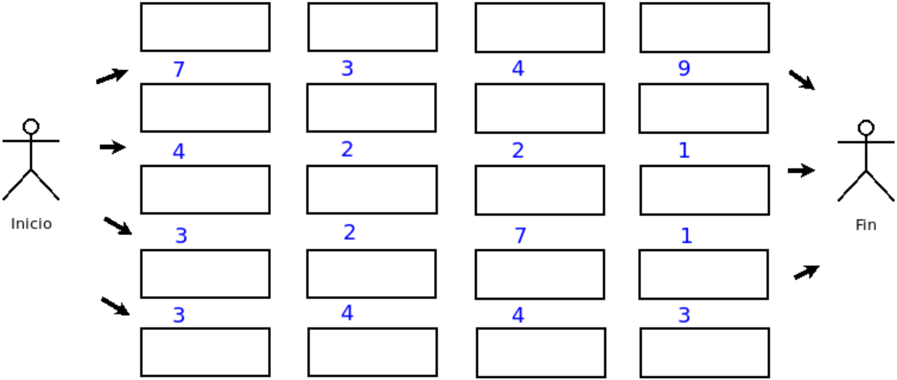
CASO PROMEDIO

Como vimos en la resolución de fórmulas, donde calculamos los valores para una cota superior ($<$), podemos realizar los cálculos para una cota inferior ($>$), de esa manera comprobaremos que el caso promedio es $\Omega(nlg(n))$, por lo tanto, un caso PROMEDIO, quicksort sería $\Theta(nlg(n)$

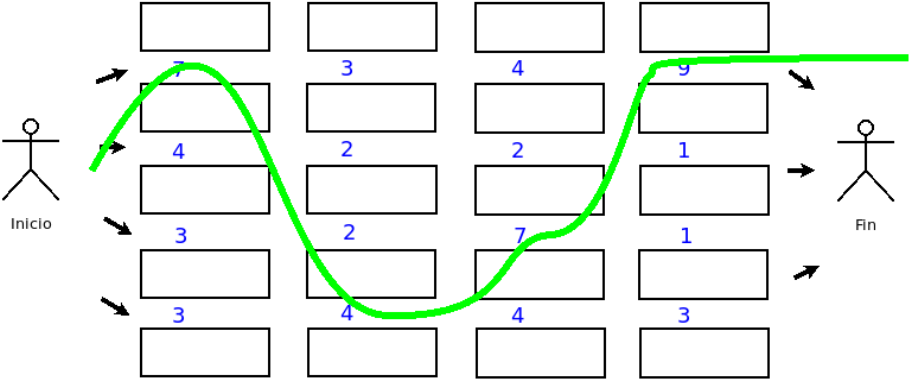
Motivación



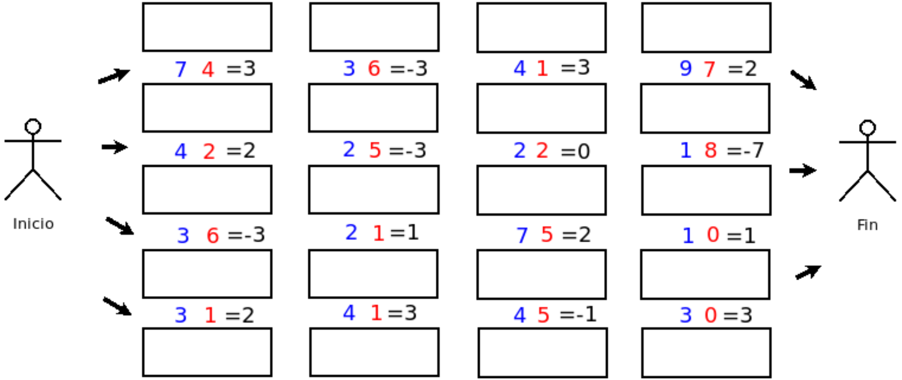
Motivación



Motivación

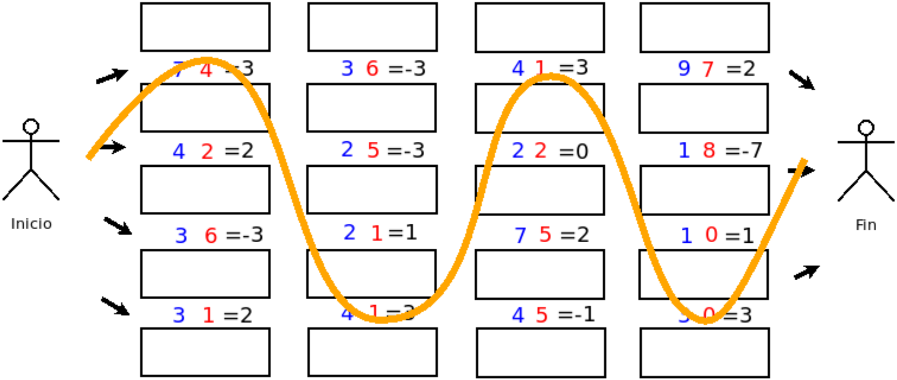


Motivación

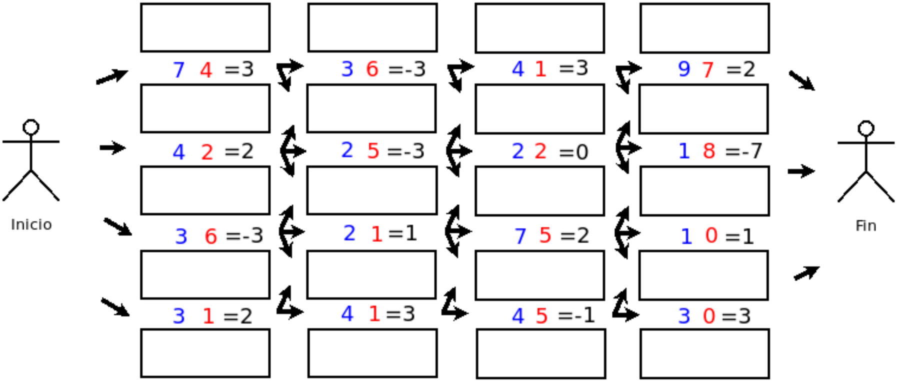


Inicio
7 4 =3
4 2 =2
3 6 =-3
3 1 =2
3 6 =-3
2 5 =-3
2 1 =1
4 1 =3
4 1 =3
2 2 =0
7 5 =2
4 5 =-1
9 7 =2
1 8 =-7
1 0 =1
3 0 =3
Fin

Motivación

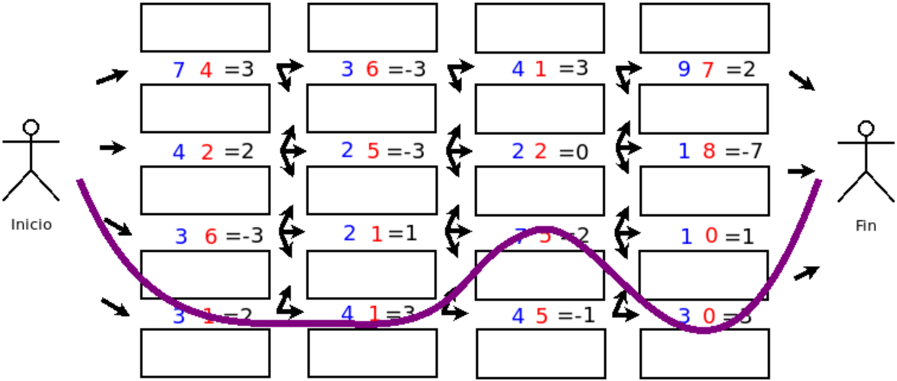


Motivación



Restricción de movimientos.

Motivación



Restricción de movimientos.

Algoritmo ávaro - teoría

I. Parberry, Problemns on Algorithms, Prentice Hall, 1995

A greedy algorithm starts with a solution to a very small subproblem and augments it successively to a solution for a big problem. The augmentation is done in a "greedy" fashion, that is, paying atention to short-term or local gain, withouth regard to wheter it will lead to a good long-term or globlal solution. As in real life, greedy algorithms sometimes lead to the best solution, sometimes lead to pretty good solutions, and sometimes lead to lousy solutions. The trick is to determine when to be greedy."

Algoritmo ávaro - teoría

I. Parberry, Problemns on Algorithms, Prentice Hall, 1995

A greedy algorithm starts with a solution to a very small subproblem and augments it successively to a solution for a big problem. The augmentation is done in a "greedy" fashion, that is, paying atention to short-term or local gain, withouth regard to wheter it will lead to a good long-term or globlal solution. As in real life, greedy algorithms sometimes lead to the best solution, sometimes lead to pretty good solutions, and sometimes lead to lousy solutions. The trick is to determine when to be greedy."

Algoritmo ávaro - teoría

I. Parberry, Problemns on Algorithms, Prentice Hall, 1995

A greedy algorithm starts with a solution to a very small subproblem and augments it successively to a solution for a big problem. The augmentation is done in a "greedy" fashion, that is, paying atention to short-term or local gain, withouth regard to wheter it will lead to a good long-term or globlal solution. As in real life, greedy algorithms sometimes lead to the best solution, sometimes lead to pretty good solutions, and sometimes lead to lousy solutions. The trick is to determine when to be greedy."

Algoritmo ávaro - teoría

I. Parberry, Problemns on Algorithms, Prentice Hall, 1995

A greedy algorithm starts with a solution to a very small subproblem and augments it successively to a solution for a big problem. The augmentation is done in a "greedy" fashion, that is, paying atention to short-term or local gain, withouth regard to wheter it will lead to a good long-term or globlal solution. As in real life, greedy algorithms sometimes lead to the best solution, sometimes lead to pretty good solutions, and sometimes lead to lousy solutions. The trick is to determine when to be greedy."

Algoritmo ávaro - teoría

I. Parberry, Problemns on Algorithms, Prentice Hall, 1995

A greedy algorithm starts with a solution to a very small subproblem and augments it successively to a solution for a big problem. The augmentation is done in a "greedy" fashion, that is, paying atention to short-term or local gain, withouth regard to wheter it will lead to a good long-term or globlal solution. As in real life, greedy algorithms sometimes lead to the best solution, sometimes lead to pretty good solutions, and sometimes lead to lousy solutions. The trick is to determine when to be greedy."

Algoritmo ávaro - teoría

I. Parberry, Problemns on Algorithms, Prentice Hall, 1995

A greedy algorithm starts with a solution to a very small subproblem and augments it successively to a solution for a big problem. The augmentation is done in a "greedy" fashion, that is, paying atention to short-term or local gain, withouth regard to wheter it will lead to a good long-term or globlal solution. As in real life, greedy algorithms sometimes lead to the best solution, sometimes lead to pretty good solutions, and sometimes lead to lousy solutions. The trick is to determine when to be greedy."

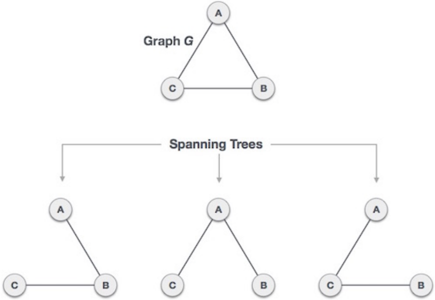
Algoritmo ávaro - teoría

I. Parberry, Problemns on Algorithms, Prentice Hall, 1995

A greedy algorithm starts with a solution to a very small subproblem and augments it successively to a solution for a big problem. The augmentation is done in a "greedy" fashion, that is, paying atention to short-term or local gain, withouth regard to wheter it will lead to a good long-term or globlal solution. As in real life, greedy algorithms sometimes lead to the best solution, sometimes lead to pretty good solutions, and sometimes lead to lousy solutions. The trick is to determine when to be greedy."

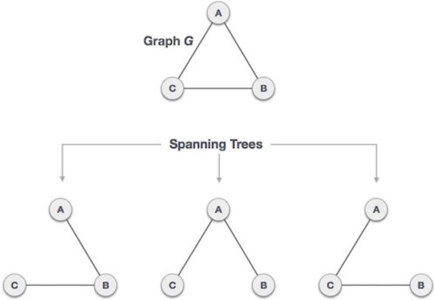
Minimum Spanning Tree - MST

- Sea $G = (V, E)$ un grafo conexo no dirigido de V vértices y E arcos, donde cada arco (u, v) tiene un peso $w(u, v)$ Encontrar el subgrafo conectado no dirigido $G' = (V', E')$ tal que $V' \equiv V$ y $E' \subseteq E$. Además $w(E') = \sum_{(u',v') \in E'} w(u', v')$ es el mínimo posible.



Minimum Spanning Tree - MST

- Sea $G = (V, E)$ un grafo conexo no dirigido de V vértices y E arcos, donde cada arco (u, v) tiene un peso $w(u, v)$ Encontrar el subgrafo conectado no dirigido $G' = (V', E')$ tal que $V' \equiv V$ y $E' \subseteq E$. Además $w(E') = \sum_{(u',v') \in E'} w(u', v')$ es el mínimo posible.

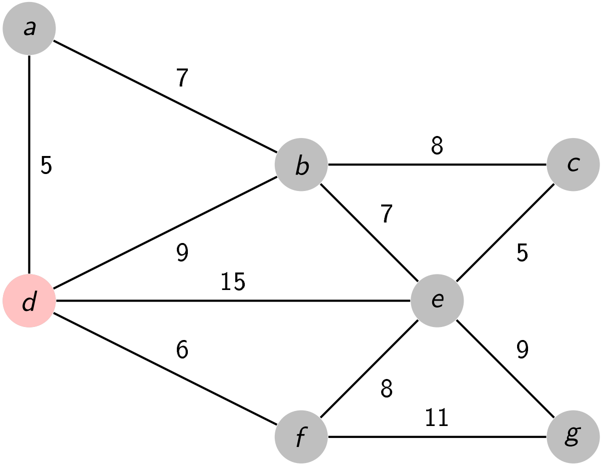


- Existen diferentes formas de encontrar el MST, sin embargo consideraremos 2 importantes: Algoritmo Prim y Kruskal.

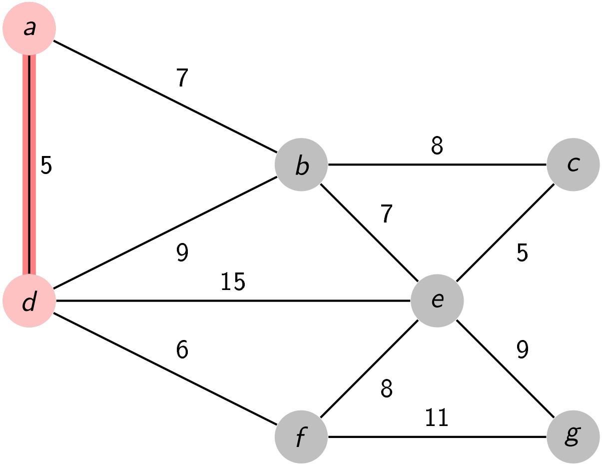
Algoritmo Prim

1. Seleccionar un vértice v de G. (Spanning Tree inicial).
2. Incrementar el árbol repetidamente con un vértice que no se encuentre en el MST.
3. Escoger aquel vértice que adicione un peso mínimo al nuevo MST.

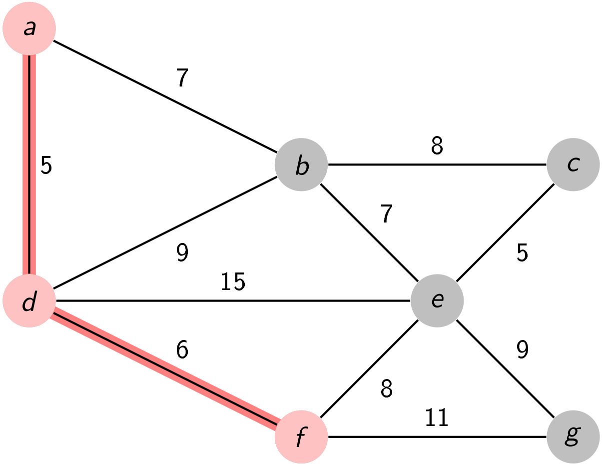
Algoritmo Prim



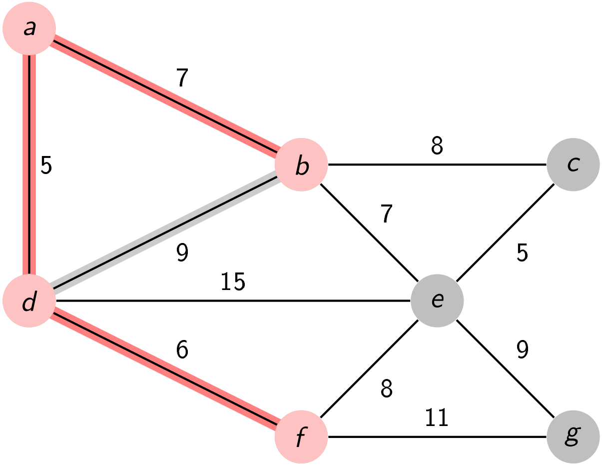
Algoritmo Prim



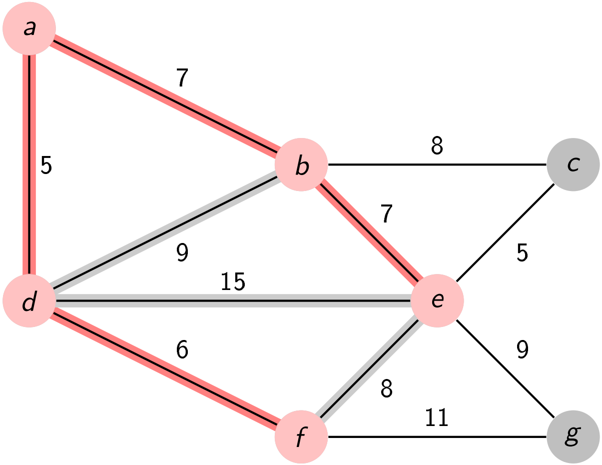
Algoritmo Prim



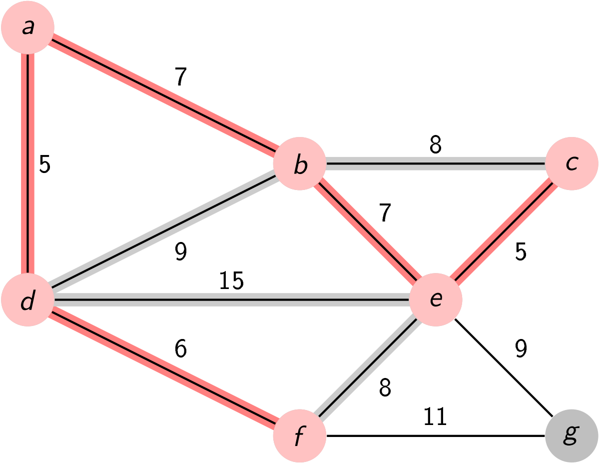
Algoritmo Prim



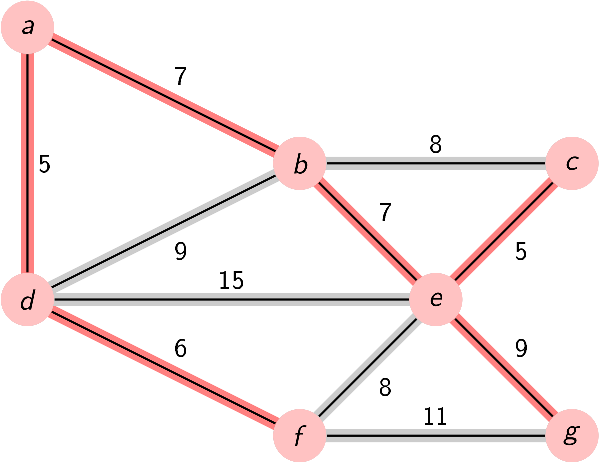
Algoritmo Prim



Algoritmo Prim



Algoritmo Prim



Algoritmo Prim

PRIM(G, V)

PQ $\leftarrow$ Cola de prioridad, ordenado de menor a mayor prioridad
para todo $v \in V$ **hacer**
    $ancestor(v) \leftarrow \lambda$
    PQ $\leftarrow (v, priority(v) = \infty)$
PQ $\leftarrow (s, 0)$ $\triangleright$ Sólo el vértice inicial tendrá prioridad 0.
mientras PQ no está vacío **hacer**
    $u \leftarrow$ **PQ**.pop()
    para todo $v \in V$ adyacente a u **hacer**
        si v está en **PQ** y $w(u, v) < priority(v)$ **entonces**
            $ancestor(v) \leftarrow u$
            $priority(v) = w(u, v)$ $\triangleright$ Actualizar la estructura después de esto
devolver $ancestor$

Consumo de tiempo

Líneas	**Tiempo**
2-4	$\|V\|$
6-11	Un poco complicado

Consumo de tiempo

Líneas	**Tiempo**
2-4	$\lvert V\rvert$
6-11	Un poco complicado
6	está en función de la línea 7 $\rightarrow O(\lvert V\rvert)$

Consumo de tiempo

Líneas	Tiempo
2-4	$\|V\|$
6-11	Un poco complicado
6	está en función de la línea 7 $\rightarrow O(\|V\|)$
8-11	aparentemente $O(\|V \cdot V\|)$

Consumo de tiempo

Líneas	Tiempo
2-4	$\lvert V \rvert$
6-11	Un poco complicado
6	está en función de la línea 7 $\rightarrow O(\lvert V \rvert)$
8-11	aparentemente $O(\lvert V \cdot V \rvert)$
	En realidad depende de la línea 9 con 7

Consumo de tiempo

Líneas	**Tiempo**
2-4	$\lvert V\rvert$
6-11	Un poco complicado
6	está en función de la línea 7 $\rightarrow O(\lvert V\rvert)$
8-11	aparentemente $O(\lvert V \cdot V\rvert)$
	En realidad depende de la línea 9 con 7
8	O($\lvert E\rvert$) cantidad de ocasiones

Consumo de tiempo

Líneas	Tiempo
2-4	$\lvert V \rvert$
6-11	Un poco complicado
6	está en función de la línea 7 $\rightarrow O(\lvert V \rvert)$
8-11	aparentemente $O(\lvert V \cdot V \rvert)$
	En realidad depende de la línea 9 con 7
8	O($\lvert E \rvert$) cantidad de ocasiones
11	La actualización en una lista de prioridad es $O(log(\lvert * \rvert)$

Consumo de tiempo

Líneas	Tiempo
2-4	$\|V\|$
6-11	Un poco complicado
6	está en función de la línea 7 $\rightarrow O(\|V\|)$
8-11	aparentemente $O(\|V \cdot V\|)$
	En realidad depende de la línea 9 con 7
8	$O(\|E\|)$ cantidad de ocasiones
11	La actualización en una lista de prioridad es $O(log(\|*\|)$

Consumo de tiempo

$O((|V| + |E|)log(|V|)) = O(|E|log(|V|))$

Correctitud

i0 Al final del ciclo de la línea 8, consideremos un árbol T que es un subgrafo de algún mininum spanning tree M.

Correctitud

i0 Al final del ciclo de la línea 8, consideremos un árbol T que es un subgrafo de algún mininum spanning tree M.

i1 En cada iteración del ciclo de las líneas 6-11, tenemos exactamente 1 operación POP en la línea 7.

Correctitud

i0 Al final del ciclo de la línea 8, consideremos un árbol T que es un subgrafo de algún mininum spanning tree M.

i1 En cada iteración del ciclo de las líneas 6-11, tenemos exactamente 1 operación POP en la línea 7.

i2 En la línea 10, el $ancestor(v)$ es u si el peso de u, v es mínimo.

Correctitud

- i0 Al final del ciclo de la línea 8, consideremos un árbol T que es un subgrafo de algún mininum spanning tree M.
- i1 En cada iteración del ciclo de las líneas 6-11, tenemos exactamente 1 operación POP en la línea 7.
- i2 En la línea 10, el $ancestor(v)$ es u si el peso de u, v es mínimo.
- Al inicio, solo tenemos un nodo en T y al final tendremos todos los nodos de V en T por lo que se probaría la correctitud.

Correctitud

i0 Al final del ciclo de la línea 8, consideremos un árbol T que es un subgrafo de algún mininum spanning tree M.

i1 En cada iteración del ciclo de las líneas 6-11, tenemos exactamente 1 operación POP en la línea 7.

i2 En la línea 10, el $ancestor(v)$ es u si el peso de u, v es mínimo.

- Al inicio, solo tenemos un nodo en T y al final tendremos todos los nodos de V en T por lo que se probaría la correctitud.
- Es correcto?. Vemos que cada vez T crece con un nodo e. Vamos probar por el absurdo. Supongamos que $e \notin M$.

Correctitud

i0 Al final del ciclo de la línea 8, consideremos un árbol T que es un subgrafo de algún mininum spanning tree M.

i1 En cada iteración del ciclo de las líneas 6-11, tenemos exactamente 1 operación POP en la línea 7.

i2 En la línea 10, el $ancestor(v)$ es u si el peso de u, v es mínimo.

- Al inicio, solo tenemos un nodo en T y al final tendremos todos los nodos de V en T por lo que se probaría la correctitud.
- Es correcto?. Vemos que cada vez T crece con un nodo e. Vamos probar por el absurdo. Supongamos que $e \notin M$.
  - Inválido porque el algoritmo adicionaría siempre tal nodo e (ciclo).

Correctitud

i0 Al final del ciclo de la línea 8, consideremos un árbol T que es un subgrafo de algún mininum spanning tree M.

i1 En cada iteración del ciclo de las líneas 6-11, tenemos exactamente 1 operación POP en la línea 7.

i2 En la línea 10, el $ancestor(v)$ es u si el peso de u, v es mínimo.

- Al inicio, solo tenemos un nodo en T y al final tendremos todos los nodos de V en T por lo que se probaría la correctitud.
- Es correcto?. Vemos que cada vez T crece con un nodo e. Vamos probar por el absurdo. Supongamos que $e \notin M$.
  - Inválido porque el algoritmo adicionaría siempre tal nodo e (ciclo).
  - Mismo posible, no cambiaría el resultado de M ya que i2 solo actualiza M por un peso minimal.

Correctitud

- i0 Al final del ciclo de la línea 8, consideremos un árbol T que es un subgrafo de algún mininum spanning tree M.
- i1 En cada iteración del ciclo de las líneas 6-11, tenemos exactamente 1 operación POP en la línea 7.
- i2 En la línea 10, el $ancestor(v)$ es u si el peso de u, v es mínimo.

- Al inicio, solo tenemos un nodo en T y al final tendremos todos los nodos de V en T por lo que se probaría la correctitud.
- Es correcto?. Vemos que cada vez T crece con un nodo e. Vamos probar por el absurdo. Supongamos que $e \notin M$.
  - Inválido porque el algoritmo adicionaría siempre tal nodo e (ciclo).
  - Mismo posible, no cambiaría el resultado de M ya que i2 solo actualiza M por un peso minimal.

Conclusión

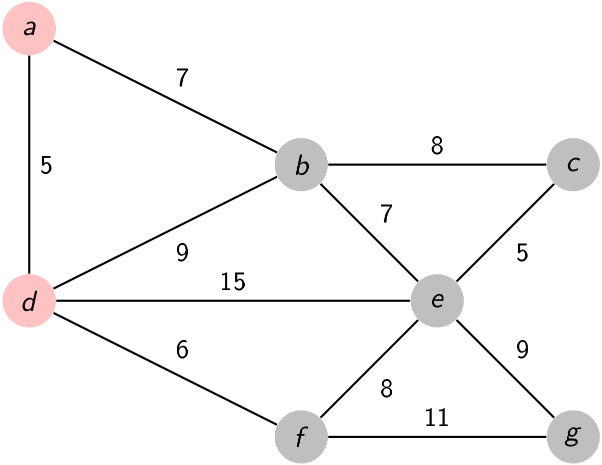
El algoritmo Prim analiza nodo por nodo y construye el MST a partir de todos los arcos que salen de cada nodo analizado. El algoritmo usa una cola de prioridad y realiza un análisis por vez.

Algoritmo Kruskal

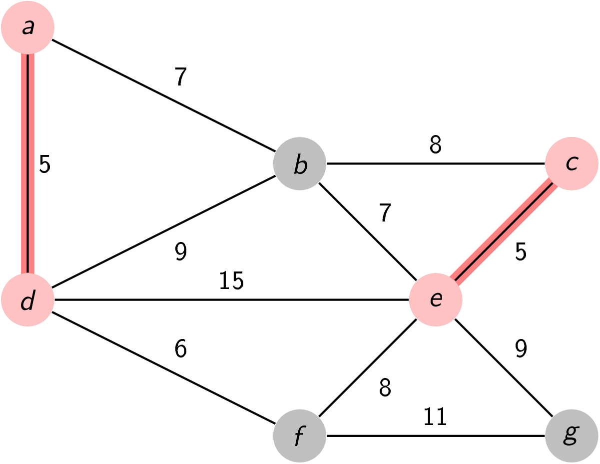
1. Son creados $|\mathbf{V}|$ árboles. Llamaremos a este conjunto B
2. Creamos un conjunto C con todas las aristas del grafo.
3. Mientras C no sea vacío:
   - Eliminar un arco de peso mínimo de C.
   - Si tal arco conecta dos árboles diferentes, entonces adicionarlos a B.
   - Combinar tales árboles en uno solo.

Al acabar el algoritmo, el conjunto de árboles B se habrá convertido en un MST.

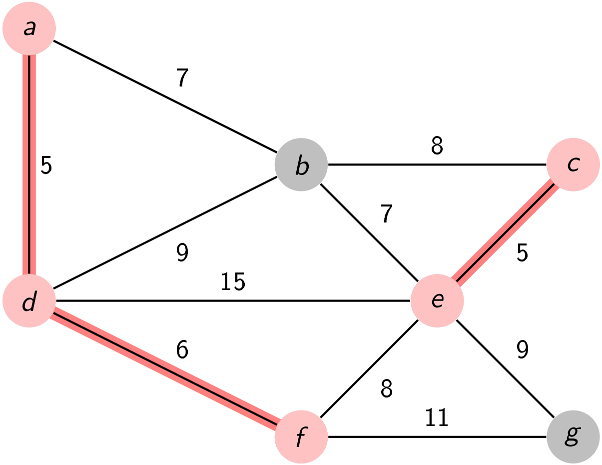
Algoritmo de Kruskal



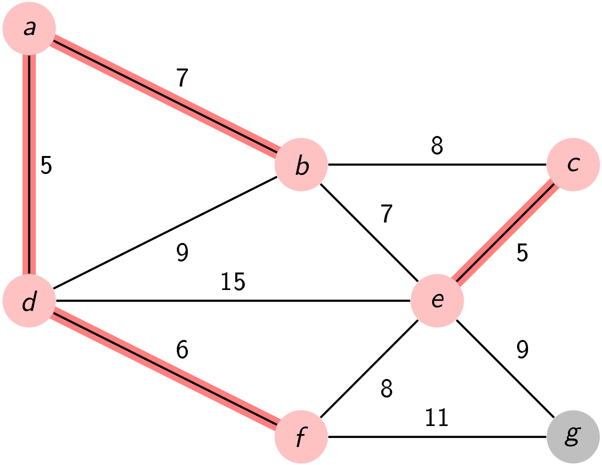
Algoritmo de Kruskal



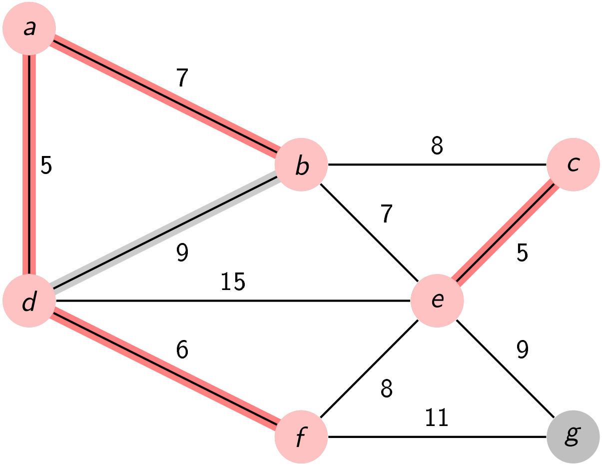
Algoritmo de Kruskal



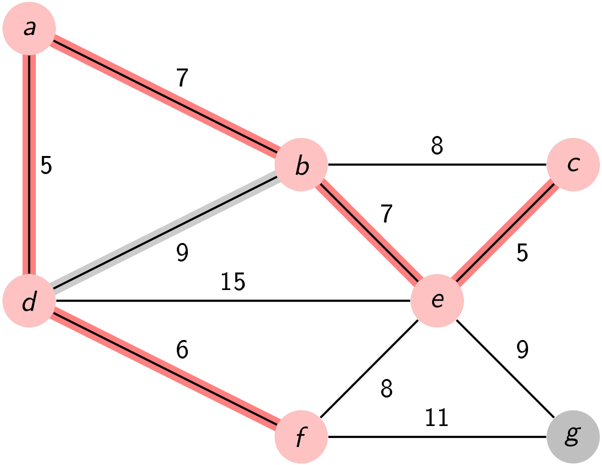
Algoritmo de Kruskal



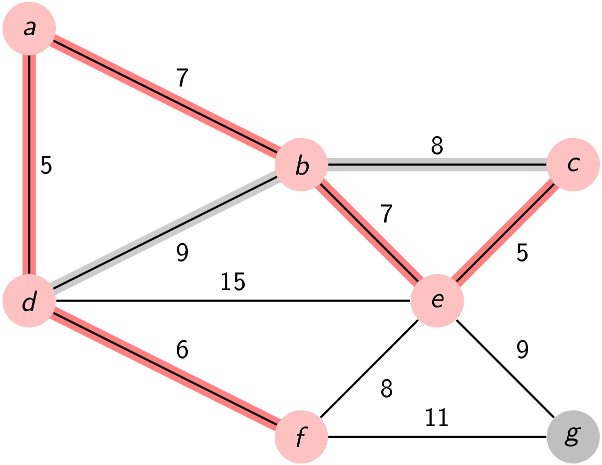
Algoritmo de Kruskal



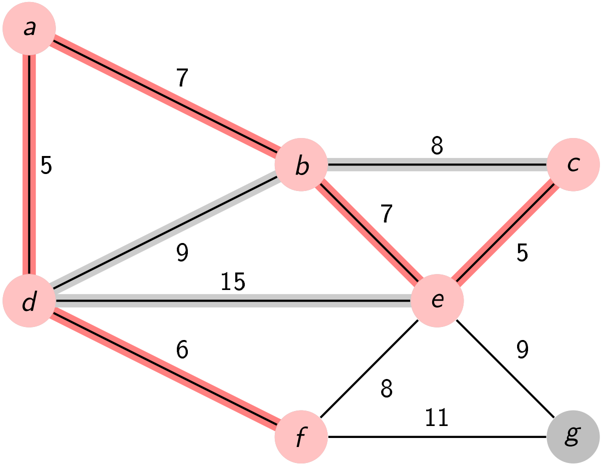
Algoritmo de Kruskal



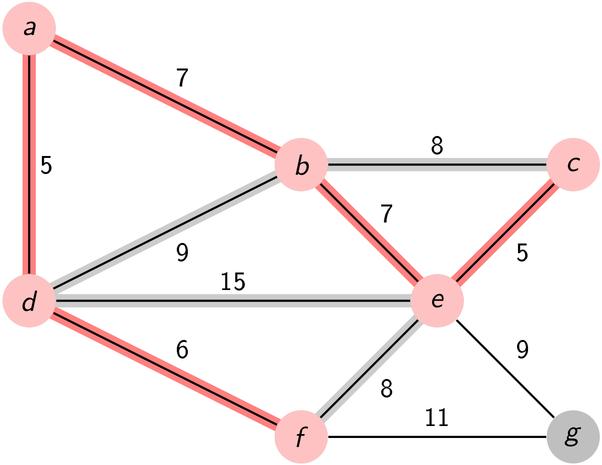
Algoritmo de Kruskal



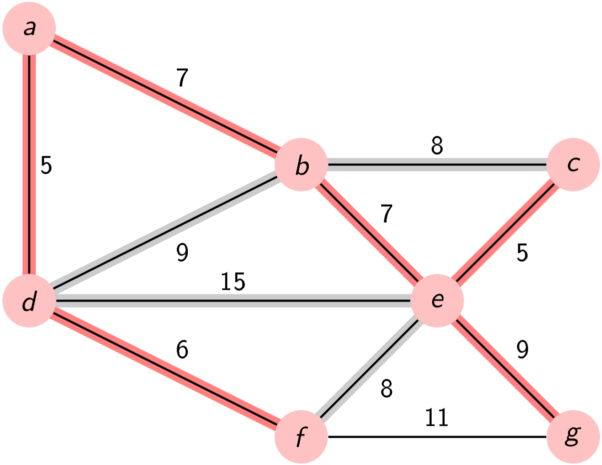
Algoritmo de Kruskal



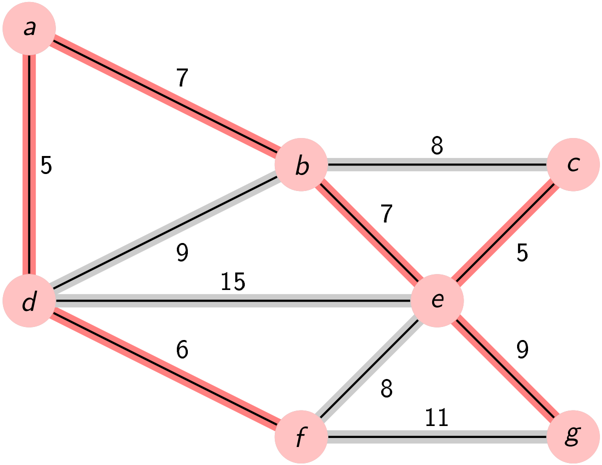
Algoritmo de Kruskal



Algoritmo de Kruskal



Algoritmo de Kruskal



Algoritmo Kruskal

Kruskal(G,V, E)

para todo $v \in V$ **hacer**
  $C \leftarrow C + \{v\}$ ▷ estructura Heap
$PQ \leftarrow V$
$T \leftarrow \emptyset$ ▷ árbol
mientras $ARCOS(T) < |E|$ y PQ no está vacío **hacer**
  $(u, v) \leftarrow$ PQ.pop()
  si $C(v) \not\equiv C(u)$ ▷ FINDSET **entonces**
    $T \leftarrow T + (v, u)$
    UNION($C(v), C(u)$)
devolver T

Complejidad

?

Algoritmo Kruskal

Kruskal(G,V, E)

para todo $v \in V$ **hacer**

  $C \leftarrow C + \{v\}$ $\triangleright$ estructura Heap

$PQ \leftarrow V$

$T \leftarrow \emptyset$ $\triangleright$ árbol

mientras $ARCOS(T) < |E|$ y PQ no está vacío **hacer**

  $(u, v) \leftarrow$ PQ.pop()

  si $C(v) \neq C(u)$ $\triangleright$ FINDSET **entonces**

    $T \leftarrow T + (v, u)$

    UNION$(C(v), C(u))$

devolver T

Complejidad

$O(|E|log(|E|)) = O(|E|log(|V|))$

Correctitud

¡0 ?

Prueba por el absurdo?

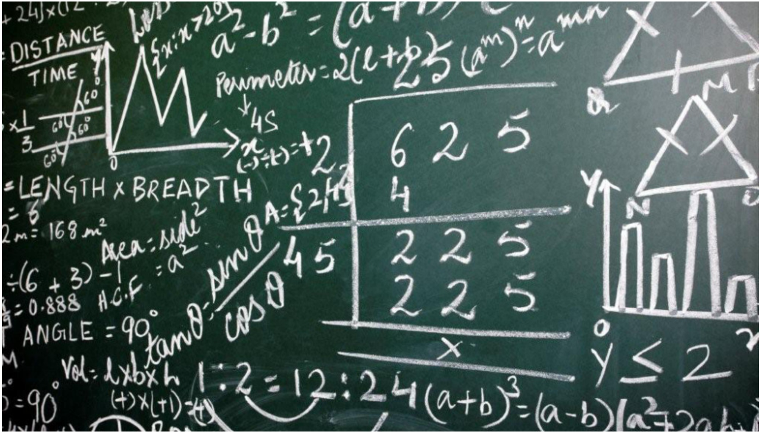
Conclusión

?

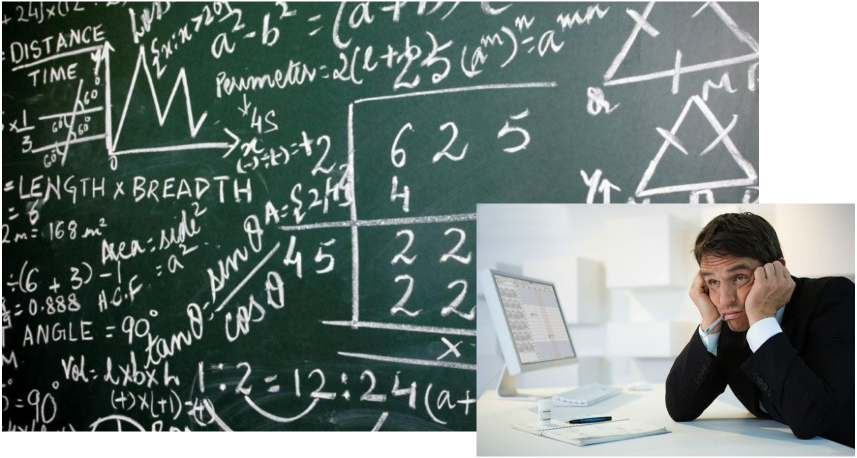
Lecturas recomendadas

- HEAP.
  - HEAPSORT
    - MAX-HEAPIFY
    - BUILD-MAX-HEAP
  - Cola de prioridad.
    - HEAP-MAX
    - HEAP-EXTRACT-MAX
    - HEAP-INCREASE-KEY
    - MAX-HEAP-INSERT
- UNION - DISJOINT
  - MAKESET
  - UNION
  - FINDSET

Por qué estudiar Complejidad P y NP?



Por qué estudiar Complejidad P y NP?



Análisis de algoritmo y su importancia

Ejemplo

- Necesito encontrar una palabra en un texto.

Análisis de algoritmo y su importancia

Ejemplo

- Necesito encontrar una palabra en un texto.
- Una forma sería buscar palabra por palabra en todo el texto.

Análisis de algoritmo y su importancia

Ejemplo

- Necesito encontrar una palabra en un texto.
- Una forma sería buscar palabra por palabra en todo el texto.
- Qué pasaría si el texto pesa 20 megas?

Análisis de algoritmo y su importancia

Ejemplo

- Necesito encontrar una palabra en un texto.
- Una forma sería buscar palabra por palabra en todo el texto.
- Qué pasaría si el texto pesa 20 megas?
- Qué pasaría si necesito hacer esto a menudo?

Análisis de algoritmo y su importancia

Ejemplo

- Necesito encontrar una palabra en un texto.
- Una forma sería buscar palabra por palabra en todo el texto.
- Qué pasaría si el texto pesa 20 megas?
- Qué pasaría si necesito hacer esto a menudo?
- Qué pasaría si no solo yo lo debo hacer, sino toda una comunidad de usuarios?

Análisis de algoritmo y su importancia

Ejemplo

- Necesito encontrar una palabra en un texto.
- Una forma sería buscar palabra por palabra en todo el texto.
- Qué pasaría si el texto pesa 20 megas?
- Qué pasaría si necesito hacer esto a menudo?
- Qué pasaría si no solo yo lo debo hacer, sino toda una comunidad de usuarios?
- Soluciones?

La importancia de estudiar P y NP

Supongamos que tenemos la siguiente ecuación:

$$y = x ln(x)$$

Cómo encontrar el valor de x para que y sea menor? existe el menor?

- Aplicando derivada:

$$\frac{dy}{dx} = x\frac{1}{x} + ln(x)$$

La importancia de estudiar P y NP

Supongamos que tenemos la siguiente ecuación:

$$y = x ln(x)$$

Cómo encontrar el valor de x para que y sea menor? existe el menor?

- Aplicando derivada:

$$\frac{dy}{dx} = x\frac{1}{x} + ln(x)$$

- Igualando a cero

$$0 = 1 + ln(x) \qquad \rightarrow \qquad x = \frac{1}{e}$$

La importancia de estudiar P y NP

Supongamos que tenemos la siguiente ecuación:

$$y = x ln(x)$$

Cómo encontrar el valor de x para que y sea menor? existe el menor?

- Aplicando derivada:

$$\frac{dy}{dx} = x\frac{1}{x} + ln(x)$$

- Igualando a cero

$$0 = 1 + ln(x) \qquad \rightarrow \qquad x = \frac{1}{e}$$

- Aplicando la segunda derivada:

$$\frac{d^2y}{d^2x} = \frac{1}{x}$$

La importancia de estudiar P y NP

Supongamos que tenemos la siguiente ecuación:

$$y = x ln(x)$$

Cómo encontrar el valor de x para que y sea menor? existe el menor?

- Aplicando derivada:

$$\frac{dy}{dx} = x\frac{1}{x} + ln(x)$$

- Igualando a cero

$$0 = 1 + ln(x) \qquad \rightarrow \qquad x = \frac{1}{e}$$

- Aplicando la segunda derivada:

$$\frac{d^2y}{d^2x} = \frac{1}{x}$$

- Reemplanzando x:

$$\frac{1}{\frac{1}{e}} = e$$

La importancia de estudiar P y NP

Supongamos que tenemos la siguiente ecuación:

$$y = x ln(x)$$

Cómo encontrar el valor de x para que y sea menor? existe el menor?

- Aplicando derivada:

$$\frac{dy}{dx} = x\frac{1}{x} + ln(x)$$

- Igualando a cero

$$0 = 1 + ln(x) \qquad \rightarrow \qquad x = \frac{1}{e}$$

- Aplicando la segunda derivada:

$$\frac{d^2y}{d^2x} = \frac{1}{x}$$

- Reemplanzando x:

$$\frac{1}{\frac{1}{e}} = e$$

- Es un valor positivo, por lo tanto SI EXISTE EL MENOR.

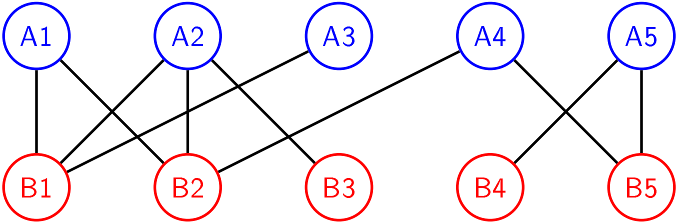
La importancia de estudiar P y NP

Conclusión

- Podíamos probar todos los valores de x y encontrar cuál era el que y era menor
- Cuando no podemos probar todos los valores, podemos reducir su complejidad.
- Debemos diferenciar entre "saber encontrar la respuesta" y "verificar la respuesta".
    - Si alguien nos hubiera dado el valor de x, no sabríamos si es el valor menor o no.

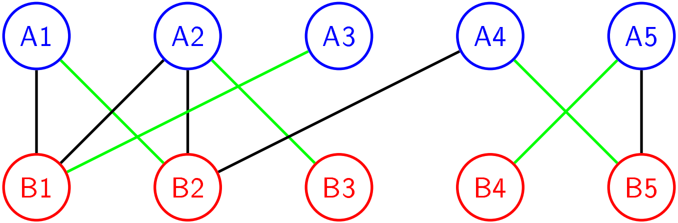
Problema del emparejamiento

- El siguiente grafo demuestra el interés de los chicos (grupo A) por las chicas (grupo B).
- Cómo seria un emparejamiento ideal, donde ninguno de los chicos choque en su interés por alguna chica?.



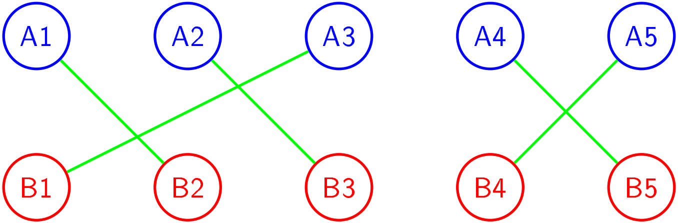
Problema del emparejamiento

- El siguiente grafo demuestra el interés de los chicos (grupo A) por las chicas (grupo B).
- Cómo seria un emparejamiento ideal, donde ninguno de los chicos choque en su interés por alguna chica?.



Problema del emparejamiento

- El siguiente grafo demuestra el interés de los chicos (grupo A) por las chicas (grupo B).
- Cómo seria un emparejamiento ideal, donde ninguno de los chicos choque en su interés por alguna chica?.



Problema de Emparejamiento

Dado un grafo bipartido, encontrar un conjunto de aristas que no tengan un vértice en común.

Clase P

Son aquellos problemas cuya solución puede ser encontrada en un tiempo razonable.

Qué es un tiempo razonable?

Que puede el tiempo puede ser calculado en forma polinómica.

Lineales $\theta(n)$

Cuadráticos $\theta(n^2)$

Cúbicos $\theta(n^3)$

Clase P

Son aquellos problemas cuya solución puede ser encontrada en un tiempo razonable.

Qué es un tiempo razonable?

Que puede el tiempo puede ser calculado en forma polinómica.

Lineales	$\theta(n)$
Cuadráticos	$\theta(n^2)$
Cúbicos	$\theta(n^3)$
..	

Con complejidades $lg(n)$ y $nlg(n)$

Clase P

Son aquellos problemas cuya solución puede ser encontrada en un tiempo razonable.

Qué es un tiempo razonable?

Que puede el tiempo puede ser calculado en forma polinómica.

$$
\begin{aligned}
\text{Lineales} \quad & \theta(n) \\
\text{Cuadráticos} \quad & \theta(n^2) \\
\text{Cúbicos} \quad & \theta(n^3) \\
& \vdots
\end{aligned}
$$

Con complejidades $lg(n)$ y $nlg(n)$

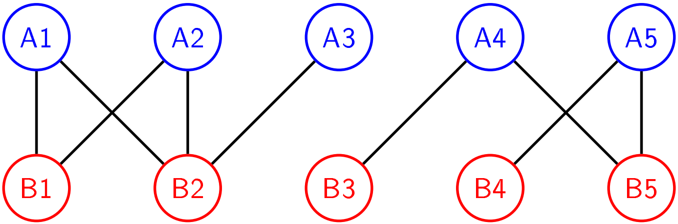
n a la tal $n^4, n^5, \ldots n^8 \ldots n^{10} \ldots$

Ejemplos de problemas con tipo P

- Ordenamiento.
- Máximo Comun Divisor (Euclides).
- Problemas de Programación Dinámica.

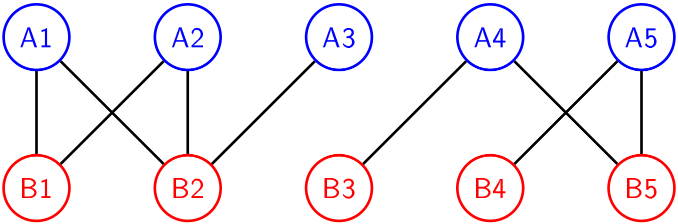
Problema del emparejamiento

Para el siguiente grafo bipartido, habrá un emparejamiento perfecto?



Problema del emparejamiento

Para el siguiente grafo bipartido, habrá un emparejamiento perfecto?

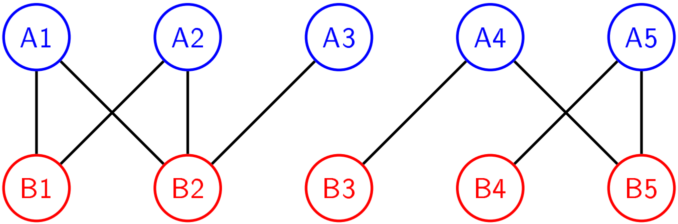


Respuesta

NO

Problema del emparejamiento

Para el siguiente grafo bipartido, habrá un emparejamiento perfecto?



Respuesta

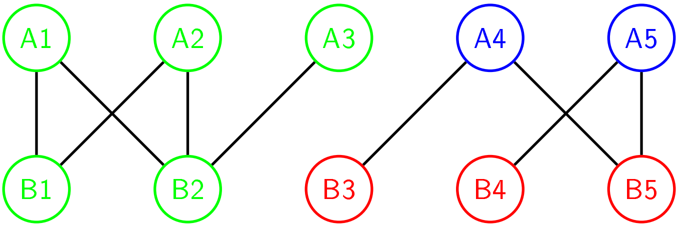
NO pero el problema es saber cómo es que lo sabemos?

Certificado

Definición

Toda aquella información necesaria para verificar si un problema de decisión es positivo/afirmativo.

- Por ejemplo para el caso anterior, tenemos $S \subseteq A$ tal que $|S| > |vecinos(S)|$.
- *Teorema de Hall*: Existirá un emparejamiento perfecto si $|S| \leq |vecinos(S)|$ para todo $S \subseteq A$.



Problemas NP

Definición

Son aquellos problemas cuya solución puede ser verificada en un tiempo razonable.

Problemas NP

Definición

Son aquellos problemas cuya solución puede ser verificada en un tiempo razonable.

Recordando P

Son aquellos problemas cuya solución puede ser encontrada en un tiempo razonable.

Problemas NP

Definición

Son aquellos problemas cuya solución puede ser verificada en un tiempo razonable.

Recordando P

Son aquellos problemas cuya solución puede ser encontrada en un tiempo razonable.

Problema del millenium

P = NP ???

NP Completos

Definición

Es el subconjunto de problemas de decisión en NP tal que todo problema en NP puede ser **reducido** en cada uno de los problemas de NP-completos

Reducción?

- Ejemplo: Problema del viajero.
- Cuando no se considera viajes de regreso, no hay ciclos, no necesariamente deben visitarse todos los lugares. El problema puede ser resuelto usando Programación Dinámica.

NP Completos

Definición

Es el subconjunto de problemas de decisión en NP tal que todo problema en NP puede ser **reducido** en cada uno de los problemas de NP-completos

Reducción?

- Ejemplo: Problema del viajero.
- Cuando no se considera viajes de regreso, no hay ciclos, no necesariamente deben visitarse todos los lugares. El problema puede ser resuelto usando Programación Dinámica.

Y qué pasa si la reducción me lleva a exponencial?

Satisfabilidad: Encontrar una solución que **satisfaza** en un conjunto de soluciones.

NP Completos

Definición

Es el subconjunto de problemas de decisión en NP tal que todo problema en NP puede ser **reducido** en cada uno de los problemas de NP-completos

Reducción?

- Ejemplo: Problema del viajero.
- Cuando no se considera viajes de regreso, no hay ciclos, no necesariamente deben visitarse todos los lugares. El problema puede ser resuelto usando Programación Dinámica.

Y qué pasa si la reducción me lleva a exponencial?

Satisfabilidad: Encontrar una solución que **satisfaza** en un conjunto de soluciones.

Y qué pasa si no consigo hacer una reducción polinómica?

Estarías encontrando un **NP-hard**.

NP completos

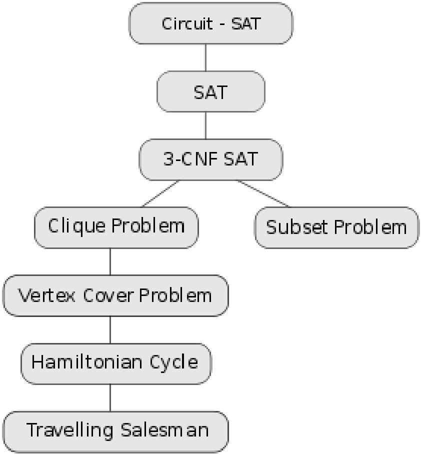
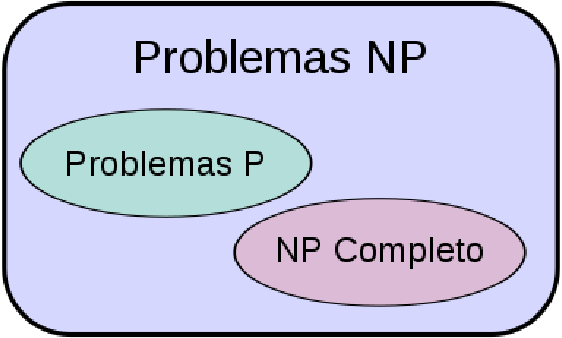


Figure: Reducción de problemas a NP-Completos, From:wikipedia

NP Completos



Qué aprendimos hoy?

- P significa que puedo encontrar una solución razonable. Además su solución es fácil de verificar.

Qué aprendimos hoy?

- P significa que puedo encontrar una solución razonable. Además su solución es fácil de verificar.
- NP significa que puedo verificar la solución, pero saber si esta existe o no... uhmmm es difícil.

Qué aprendimos hoy?

- P significa que puedo encontrar una solución razonable. Además su solución es fácil de verificar.
- NP significa que puedo verificar la solución, pero saber si esta existe o no... uhmmm es difícil.
- Reducir quiere decir que de ser algo "exponencial", puedo llevarlo a ser "polinómico".

Qué aprendimos hoy?

- P significa que puedo encontrar una solución razonable. Además su solución es fácil de verificar.
- NP significa que puedo verificar la solución, pero saber si esta existe o no... uhmmm es difícil.
- Reducir quiere decir que de ser algo "exponencial", puedo llevarlo a ser "polinómico".
- Si no puedo reducir, es un problema NP-dificil.

Gracias