CI2612: Algoritmos y Estructuras de Datos II

Blai Bonet

Universidad Simón Bolívar, Caracas, Venezuela

Cálculo de estadísticos de orden (selección)

© 2018 Blai Bonet

Objetivos

- Introducir el problema de selección del i-ésimo elemento en conjuntos de n elementos
- Presentar dos soluciones: un algoritmo randomizado que corre en tiempo lineal esperado y un algoritmo determinístico que corre en tiempo lineal
- Hacer análisis de correctitud y desempeño de ambos algoritmos

© 2018 Blai Bonet

Introducción

El i-ésimo estadístico de orden de un conjunto de n elementos es el **i-ésimo elemento más pequeño del conjunto**

Para un conjunto de n elementos:

- el primer estadístico de orden es el **menor elemento**
- el n-ésimo estadístico es el **mayor elemento**
- si n es par, el i-ésimo estadístico para $i = n/2$ es la **mediana**
- si $n = 2k + 1$ es impar, el i-ésimo estadístico para $i = \lfloor n/2 \rfloor = k$ es la **mediana inferior**
- si $n = 2k + 1$ es impar, el i-ésimo estadístico para $i = \lceil n/2 \rceil = k + 1$ es la **mediana superior**

© 2018 Blai Bonet

Problema de selección (cálculo de estadísticos)

Queremos diseñar un algoritmo que resuelva el siguiente problema:

Input: conjunto A con n **elementos distintos** e índice $i \in \{1, \ldots, n\}$

Output: $x \in A$ tal que x es $>$ a exactamente $i-1$ elementos en A

El problema de selección se resuelve en tiempo $O(n \log n)$ si primero ordenamos el conjunto

En esta clase mostramos dos algoritmos **óptimos**:

- un **algoritmo randomizado** que calcula el i-ésimo elemento en **tiempo esperado** $O(n)$
- un **algoritmo determinístico** que calcula el i-ésimo elemento en tiempo $O(n)$ en el **peor caso**

© 2018 Blai Bonet

Mínimos y máximos

Primero nos ocupamos de los casos especiales $i = 1$ (cálculo del mínimo) e $i = n$ (cálculo del máximo)

Ambos problemas pueden resolverse en tiempo lineal con un recorrido sobre todos los elementos del conjunto

Input: arreglo $A[p \ldots r]$ con $n = r - p + 1$ **elementos distintos**

Output: máximo elemento x en $A[p \ldots r]$

```
1  Maximum(array A, int p, int r)
2      m = A[p]
3      for i = p + 1 to r
4          if m < A[i]
5              m = A[i]
6      return m
```

© 2018 Blai Bonet

Selección en tiempo lineal esperado

El problema general parece más difícil que calcular el máximo/mínimo

Sin embargo veremos que podemos hacerlo en **tiempo lineal**

Primero mostramos un algoritmo de tipo **dividir y conquistar** que es parecido a `Quicksort` excepto que hace la recursión sobre **un sólo lado** de la partición

Esta característica hace que el tiempo sea $\Theta(n)$ a diferencia del tiempo $\Theta(n \log n)$ de `Quicksort`

El algoritmo `Randomized-Select` utiliza `Randomized-Partition` al igual que `Randomized-Quicksort`

© 2018 Blai Bonet

Selección en tiempo lineal esperado

Input: arreglo $A[p \ldots r]$ con $n = r - p + 1$ **elementos distintos** e índice $i \in \{1, \ldots, n\}$

Output: $x \in A$ tal que x es $>$ a exactamente $i-1$ elementos en A

```
1  Randomized-Select(array A, int p, int r, int i)
2      if p == r
3          return A[p]
4      q = Randomized-Partition(A, p, r)
5      k = q - p + 1                    % rango de A[q]: A[q] es k-ésimo
6      if i == k
7          return A[q]
8      else if i < k                        % i-ésimo está en A[p...q-1]
9          return Randomized-Select(A, p, q-1, i)
10     else                          % i-esimo es (i-k)-ésimo en A[q+1...r]
11         return Randomized-Select(A, q+1, r, i-k)
```

© 2018 Blai Bonet

Correctitud

Demostración por inducción en el tamaño n. Se asume que todos los elementos son **distintos**

Caso $n = 1$: Entonces $i = 1$, $p = r$, y el i-ésimo es $A[p] = A[r]$

Caso $n > 1$: `Randomized-Partition` reordena A y retorna **pivote** q tal que $A[j] < A[q] < A[k]$ para todo j y k con $p \leq j < q < k \leq r$

Para $k = q - p + 1$, tenemos:

– el k-ésimo de $A[p \ldots r]$ es $A[q]$

– si $i < k$, el i-ésimo de $A[p \ldots r]$ es el i-ésimo de $A[p \ldots q-1]$

– si $i > k$, el i-ésimo de $A[p \ldots r]$ es el $(i-k)$-ésimo de $A[q+1 \ldots r]$

Entonces, por HI, `Randomized-Select` es correcto □

© 2018 Blai Bonet

Análisis de peor caso

Existen varios peores casos que toman tiempo $\Theta(n^2)$

Considere el caso $i = 1$ (queremos el mínimo de $A[p \ldots r]$) y suponga que **todas las llamadas** a `Randomized-Partition` retornan un pivote q tal que $A[q]$ es el **máximo** de $A[p \ldots r]$

Entonces, $q = r$ y la recursión es sobre $A[p \ldots r-1]$ de tamaño $n-1$

Se realizan $n-1$ recursiones hasta que $p = r$ cuando se retorna $A[p]$

Cada llamada a `Randomized-Partition` toma tiempo $\Theta(k)$ donde k es el tamaño de $A[p \ldots r]$

El tiempo total es:

$$T(n) = \sum_{k=1}^{n} \Theta(k) = \Theta\left(\sum_{k=1}^{n} k\right) = \Theta\left(\frac{n(n+1)}{2}\right) = \Theta(n^2)$$

© 2018 Blai Bonet

Análisis de tiempo esperado (1 de 8)

Veremos que el tiempo esperado de `Randomized-Select` es $\Theta(n)$ para **cualquier entrada** (e.g. no existe entrada para la cual `Randomized-Select` corre en tiempo esperado $\Theta(n^2)$)

Definimos la **variable aleatoria** $T(n)$ para el tiempo de corrida sobre una entrada cualquiera $A[p \ldots r]$ con **elementos distintos** de tamaño $n = r - p + 1$

El pivote q retornado por `Randomized-Partition` es cualquier elemento de $A[p \ldots r]$ con **igual probabilidad**

Por lo tanto, el subarreglo $A[p \ldots q]$ tiene k elementos con probabilidad $1/n$ para todo $1 \leq k \leq n$

© 2018 Blai Bonet

Análisis de tiempo esperado (2 de 8)

Definimos la **variable aleatoria indicadora** X_k dada por

$$X_k = \mathbb{I}\{\text{el subarreglo } A[p \ldots q] \text{ tiene } k \text{ elementos}\}$$

Como los elementos en $A[p \ldots r]$ son distintos:

$$\begin{aligned}\mathbb{E}[X_k] &= \mathbb{P}(\text{el subarreglo } A[p \ldots q] \text{ tiene } k \text{ elementos}) \\ &= \mathbb{P}(A[q] \text{ es el } k\text{-ésimo de } A[p \ldots r]) \\ &= 1/n\end{aligned}$$

Acotamos $T(n)$ considerando el subproblema más costoso:

$$T(n) \leq \Theta(n) + \sum_{k=1}^{n} X_k \cdot \max\{T(k-1), T(n-k)\}$$

© 2018 Blai Bonet

Análisis de tiempo esperado (3 de 8)

Asumimos que $T(n)$ es **monotona creciente**:

$$T(n) \;\leq\; \Theta(n) + \sum_{k=1}^{n} X_k \cdot T(\max\{k-1, n-k\})$$

Tomamos **valor esperado** en ambos lados

$$\begin{aligned}
\mathbb{E}[T(n)] \;&\leq\; \mathbb{E}[\Theta(n)] + \sum_{k=1}^{n} \mathbb{E}[X_k \cdot T(\max\{k-1, n-k\})] \\
&=\; \Theta(n) + \sum_{k=1}^{n} \mathbb{E}[X_k] \cdot \mathbb{E}[T(\max\{k-1, n-k\})] \\
&=\; \Theta(n) + \frac{1}{n} \sum_{k=1}^{n} \mathbb{E}[T(\max\{k-1, n-k\})]
\end{aligned}$$

© 2018 Blai Bonet

Análisis de tiempo esperado (4 de 8)

$$\mathbb{E}[X_k \cdot T(\max\{k-1, n-k\})] \;\stackrel{?}{=}\; \mathbb{E}[X_k] \cdot \mathbb{E}[T(\max\{k-1, n-k\})]$$

El valor de la v.a. X_k depende del **pivote aleatorio** escogido por `Randomized-Partition`

El valor de la v.a. $T(\max\{k-1, n-k\})$ depende de los **próximos pivotes aleatorios** escogidos por `Randomized-Partition`

Dichas escogencias se realizan con la función `Random(p,r)`

Cada llamada a `Random(p,r)` retorna un **nuevo entero aleatorio** que es **independiente** de los otros enteros aleatorios (pasados y futuros)

(Resp. Ejercicio 9.2-2)

© 2018 Blai Bonet

Análisis de tiempo esperado (5 de 8)

Veamos $\sum_{k=1}^{n} \mathbb{E}[T(\max\{k-1, n-k\})]$. Considere

$$m_k \;\doteq\; \max\{k-1, n-k\} \;=\; \begin{cases} n-k & \text{si } k \leq \lceil n/2 \rceil \\ k-1 & \text{si } k > \lceil n/2 \rceil \end{cases}$$

Cuando $n = 2\ell$ es par:

$$\begin{aligned}
T(m_1) \;&=\; T(n-1) \\
T(m_2) \;&=\; T(n-2) \\
&\cdots \\
T(m_\ell) \;&=\; T(\ell) \;=\; T(\lfloor n/2 \rfloor) \\
T(m_{\ell+1}) \;&=\; T(\ell) \;=\; T(\lfloor n/2 \rfloor) \\
&\cdots \\
T(m_n) \;&=\; T(n-1)
\end{aligned}$$

© 2018 Blai Bonet

Análisis de tiempo esperado (5 de 8)

Veamos $\sum_{k=1}^{n} \mathbb{E}[T(\max\{k-1, n-k\})]$. Considere

$$m_k \;\doteq\; \max\{k-1, n-k\} \;=\; \begin{cases} n-k & \text{si } k \leq \lceil n/2 \rceil \\ k-1 & \text{si } k > \lceil n/2 \rceil \end{cases}$$

Cuando $n = 2\ell$ es par:

$$\sum_{k=1}^{n} \mathbb{E}[T(\max\{k-1, n-k\})] \;=\; 2 \sum_{k=\lfloor n/2 \rfloor}^{n-1} \mathbb{E}[T(k)]$$

© 2018 Blai Bonet

Análisis de tiempo esperado (5 de 8)

Veamos $\sum_{k=1}^{n} \mathbb{E}[T(\max\{k-1, n-k\})]$. Considere

$$m_k \doteq \max\{k-1, n-k\} = \begin{cases} n-k & \text{si } k \leq \lceil n/2 \rceil \\ k-1 & \text{si } k > \lceil n/2 \rceil \end{cases}$$

Cuando $n = 2\ell + 1$ es impar:

$$\begin{aligned} T(m_1) &= T(n-1) \\ T(m_2) &= T(n-2) \\ &\cdots \\ T(m_\ell) &= T(n-\ell) = T(\ell+1) = T(\lceil n/2 \rceil) \\ T(m_{\ell+1}) &= T(n-\ell-1) = T(\ell) = T(\lfloor n/2 \rfloor) \\ T(m_{\ell+2}) &= T(\ell+2-1) = T(\ell+1) = T(\lceil n/2 \rceil) \\ &\cdots \\ T(m_n) &= T(n-1) \end{aligned}$$

© 2018 Blai Bonet

Análisis de tiempo esperado (5 de 8)

Veamos $\sum_{k=1}^{n} \mathbb{E}[T(\max\{k-1, n-k\})]$. Considere

$$m_k \doteq \max\{k-1, n-k\} = \begin{cases} n-k & \text{si } k \leq \lceil n/2 \rceil \\ k-1 & \text{si } k > \lceil n/2 \rceil \end{cases}$$

Cuando $n = 2\ell + 1$ es impar:

$$\sum_{k=1}^{n} \mathbb{E}[T(\max\{k-1, n-k\})] = \mathbb{E}[T(\lfloor n/2 \rfloor)] + 2 \sum_{k=\lceil n/2 \rceil}^{n-1} \mathbb{E}[T(k)]$$

$$\leq 2 \sum_{k=\lfloor n/2 \rfloor}^{n-1} \mathbb{E}[T(k)]$$

© 2018 Blai Bonet

Análisis de tiempo esperado (5 de 8)

Veamos $\sum_{k=1}^{n} \mathbb{E}[T(\max\{k-1, n-k\})]$. Considere

$$m_k \doteq \max\{k-1, n-k\} = \begin{cases} n-k & \text{si } k \leq \lceil n/2 \rceil \\ k-1 & \text{si } k > \lceil n/2 \rceil \end{cases}$$

En ambos casos:

$$\sum_{k=1}^{n} \mathbb{E}[T(\max\{k-1, n-k\})] \leq 2 \sum_{k=\lfloor n/2 \rfloor}^{n-1} \mathbb{E}[T(k)]$$

© 2018 Blai Bonet

Análisis de tiempo esperado (6 de 8)

$$\mathbb{E}[T(n)] \leq \Theta(n) + \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} \mathbb{E}[T(k)]$$

Mostraremos que $\mathbb{E}[T(n)] = O(n)$ por **sustitución** (i.e. inducción)

Sea $f(n)$ la función representada por el término $\Theta(n)$, y a una constante tal que $f(n) \leq an$ para todo $n > 0$

Tesis inductiva: $\mathbb{E}[T(n)] \leq cn$ para n grande ($n \geq n_0$) y $T(n) = O(1)$ para $n < n_0$

© 2018 Blai Bonet

Análisis de tiempo esperado (7 de 8)

Para $n \geq 2n_0$,

$$\begin{aligned}
\mathbb{E}[T(n)] &\leq \Theta(n) + \frac{2}{n}\sum_{k=\lfloor n/2 \rfloor}^{n-1} \mathbb{E}[T(k)] \\
&\leq an + \frac{2}{n}\sum_{k=\lfloor n/2 \rfloor}^{n-1} ck \\
&= an + \frac{2c}{n}\left(\sum_{k=1}^{n-1} k - \sum_{k=1}^{\lfloor n/2 \rfloor - 1} k\right) \\
&\leq an + \frac{2c}{n}\left(\frac{(n-1)n}{2} - \frac{(n/2-2)(n/2-1)}{2}\right) \\
&\leq an + \frac{3cn}{4} + \frac{c}{2} \\
&= cn - \left(\frac{cn}{4} - \frac{c}{2} - an\right)
\end{aligned}$$

© 2018 Blai Bonet

Análisis de tiempo esperado (8 de 8)

Terminamos mostrando $cn - (\frac{cn}{4} - \frac{c}{2} - an) \leq cn$ para $n \geq 2n_0$

Es decir,

$$\frac{cn}{4} - \frac{c}{2} - an \geq 0$$

$$n\left(\frac{c}{4} - a\right) \geq \frac{c}{2}$$

Si $c > 4a$, entonces $n(c/4 - a) \geq c/2$ cuando $n \geq 2c/(c - 4a)$

Por lo tanto, c y n_0 deben ser tales que $c > 4a$ y $n_0 > c/(c-4a)$ para que $\mathbb{E}[T(n)] = O(n)$ cuando **todos los elementos son distintos**

□

© 2018 Blai Bonet

Selección en tiempo lineal en el peor caso

Ahora damos un algoritmo de selección que corre en tiempo lineal en el peor caso

Algoritmo de **interés teórico** ya que el algoritmo randomizado es más simple y tiene mejor desempeño en la práctica

El algoritmo es parecido al anterior pero garantiza que los "splits" sobre el arreglo son siempre balanceados

El algoritmo utiliza una versión de `Partition` al cual se le pasa el elemento pivote

© 2018 Blai Bonet

Algoritmo de selección de tiempo lineal

El algoritmo realiza lo siguiente:

- Divide los n elementos de la entrada en $\lceil \frac{n}{5} \rceil$ grupos de hasta 5 elementos cada grupo (i.e. grupos pequeños)
- Calcula la mediana de cada grupo con `Insertion-Sort`
- Calcula la **mediana de las medianas** de forma recursiva
- Utiliza la mediana de las medianas como **pivote** para particionar los elementos de la entrada, garantizando una **partición balanceada**
- Hace recursión sobre el subproblema apropiado como en el algoritmo anterior

© 2018 Blai Bonet

Mediana de las medianas

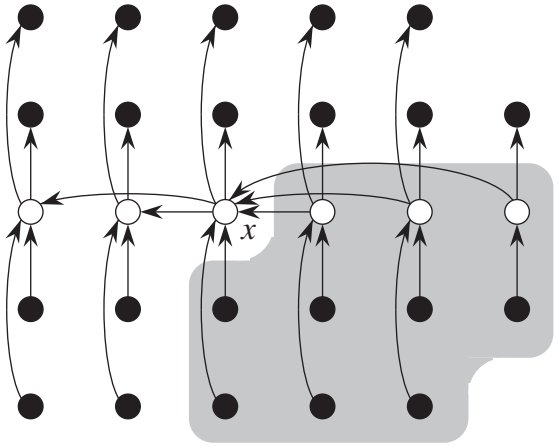

Imagen de Cormen et al. Intro. to Algorithms. MIT Press

© 2018 Blai Bonet

Algoritmo de selección de tiempo lineal

```
1  Select(array A, int p, int r, int i)
2      if r - p + 1 <= N                          % constante N a determinar
3          Insertion-Sort(A, p, r)
4          return A[i]
5
6      ℓ = ⌈(r - p + 1) / 5⌉                          % número de grupos
7      let M[1...ℓ] be new array                  % mediana de cada grupo
8      for j = 0 to ℓ - 1
9          u = min { p + 5 * j + 4, r }
10         Insertion-Sort(A, p + 5 * j, u)
11         M[j+1] = A[⌈(p + 5 * j + u)/2⌉]   % mediana (j+1)-ésimo grupo
12     x = Select(M, 1, ℓ, ⌈ℓ/2⌉)                % mediana de medianas
13
14     q = Partition-With-Pivot(A, p, r, x)
15     k = q - p + 1                    % rango de A[q]: A[q] es k-ésimo
16     if i == k
17         return A[q]                      % k-ésimo en A[p...r] es A[q]
18     else if i < k                         % i-ésimo está en A[p...q-1]
19         return Select(A, p, q-1, i)
20     else                          % i-ésimo es (i-k)-ésimo en A[q+1...r]
21         return Select(A, q+1, r, i-k)
```

© 2018 Blai Bonet

Correctitud

El mismo argumento de antes

Lo único que ha cambiado es como se escoge el pivote:

– Antes el pivote era escogido de forma aleatoria

– Ahora el pivote es la mediana de las mediana

Si bajo la escogencia aleatoria el algoritmo es correcto, entonces lo tiene que ser con la nueva escogencia

© 2018 Blai Bonet

Análisis de peor caso (1 de 4)

Acotamos el número de elementos mayores estrictos al pivote x:

Como los elementos son distintos, la mitad de las medianas son $\geq x$

Por lo tanto, la mitad del los grupos contribuyen 3 elementos **mayores estrictos** a x excepto (ver figura):

– el grupo al cual pertenece x

– el último grupo que puede tener menos de 5 elementos (cuando n no es divisible por 5)

El número de elementos mayores estrictos a x es al menos

$$3\left(\left\lceil\frac{1}{2}\left\lceil\frac{n}{5}\right\rceil\right\rceil-2\right) \;\geq\; \frac{3n}{10}-6$$

© 2018 Blai Bonet

Análisis de peor caso (2 de 4)

El número de elementos **menores estrictos** a x es n menos el número de elementos mayores a x menos 1. Dicho número es a lo sumo

$$n - \left(\frac{3n}{10} - 6\right) - 1 \;=\; \frac{7n}{10} + 6 - 1 \;=\; \frac{7n}{10} + 5$$

De forma similar se muestra que el número de elementos mayores estrictos a x es a lo sumo $7n/10 + 5$

Como consecuencia, la **segunda recursión** es sobre un arreglo de tamaño menor o igual a $\lceil 7n/10 \rceil + 5$

La **primera recursión** (para calcular la mediana de las medianas) es siempre sobre un arreglo de tamaño $\lceil n/5 \rceil$

© 2018 Blai Bonet

Análisis de peor caso (3 de 4)

Por razones que serán claras en breves, elegimos $N = 140$ (se pueden usar otros valores)

Asumiendo que $T(n)$ es una función monotona, obtenemos

$$T(n) \;\leq\; \begin{cases} O(1) & \text{si } n < 140 \\ T(\lceil n/5 \rceil) + T(\lceil 7n/10 + 5 \rceil) + O(n) & \text{si } n \geq 140 \end{cases}$$

donde el término $O(n)$ contabiliza todos los tiempos excepto las dos recursiones

© 2018 Blai Bonet

Análisis de peor caso (4 de 4)

Para terminar mostramos $T(n) \leq cn$ para una constante c

Sea c una constante suficientemente grande tal que $T(n) \leq cn$ para todo $n < 140$, y a la constante para el término $O(n)$ para todo $n > 0$

$$\begin{aligned} T(n) \;&\leq\; T(\lceil n/5 \rceil) + T(\lceil 7n/10 \rceil + 5) + O(n) \\ &\leq\; c\lceil n/5 \rceil + c(\lceil 7n/10 \rceil + 5) + an \\ &\leq\; c(n/5 + 1) + c(7n/10 + 6) + an \\ &=\; c(9n/10 + 7) + an \\ &=\; cn - (cn/10 - 7c - an) \end{aligned}$$

Para $n \geq 140$ y $c \geq 20a$, se verifica $cn/10 - 7c - an \geq 0$ y entonces $T(n) \leq cn$ para todo n □

© 2018 Blai Bonet

Observaciones finales

`Randomized-Select` es un algoritmo simple pero su análisis es complejo

`Select` es un algoritmo complejo pero su análisis es más simple

© 2018 Blai Bonet

Resumen

- Sin asumir nada particular sobre los elementos, podemos hacer selección en tiempo lineal sin necesidad de ordenar los elementos
- `Randomized-Select` es un algoritmo randomizado que corre en tiempo esperado lineal para cualquier entrada (de elementos distintos)
- La contraparte determinista es el algoritmo `Select` que corre en tiempo lineal (para elementos distintos)
- En la práctica, `Randomized-Select` tienen mejor desempeño que `Select`

© 2018 Blai Bonet

Ejercicios (1 de 3)

1. Escriba el pseudocódigo de `Minimum(A, p, r)`
2. (9.2-1) Muestre que `Randomized-Select` no hace recursión sobre arreglos de tamaño 0
3. (9.2-3) Escriba una versión iterativa de `Randomized-Select`
4. `Randomized-Select` asume que todos los elementos son distintos para ofrecer una garantía de tiempo lineal esperado. Modifique el algoritmo para que corra en tiempo lineal esperado aún cuando existan elementos repetidos. (**Ayuda:** defina una rutina de partición que devuelve dos indices i y j tal que todos los elementos entre i y j son iguales)
5. Escriba el pseudocódigo de `Partition-With-Pivot(A, p, r, x)`

© 2018 Blai Bonet

Ejercicios (2 de 3)

6. (9.3-1) ¿Se mantendrá el tiempo lineal si cambiamos `Select` para que divida la entrada en grupos de 7 elementos en lugar de 5 elementos? ¿Que sucede si dividimos la entrada en grupos de 3 elementos?
7. (9.3-2) Muestre que cuando $n \geq 140$, al menos $\lceil n/4 \rceil$ elementos son menores a la mediana de medianas x y al menos $\lceil n/4 \rceil$ son mayores a x
8. (9.3-3) De una versión determinística de `Quicksort` que corra en tiempo $O(n \log n)$ en el peor caso, asumiendo elementos distintos
9. Modifique `Select` para que trabaje en tiempo lineal aún cuando existan elementos repetidos

© 2018 Blai Bonet

Ejercicios (3 de 3)

10. (9.3-5) Suponga que tiene a su disposición una rutina `Median(A,p,r)` que calcula la mediana de $A[p \ldots r]$ en tiempo lineal. De un **algoritmo simple** de selección para calcular el i-ésimo elemento en tiempo lineal
11. (9.3-6) Los **k-ésimos cuantiles** de un conjunto de n elementos son $k-1$ elementos que dividen al conjunto en k partes de igual tamaño (o con diferencias de tamaño de a lo sumo 1 elemento). Describa un algoritmo que corra en tiempo $O(n \log k)$ en el peor caso para calcular los k-ésimos cuantiles de un conjunto de n elementos
12. (9.3-8) Sean $X[1 \ldots n]$ y $Y[1 \ldots n]$ dos arreglos ordenados de n elementos cada uno. Asumiendo que los $2n$ elementos son distintos, describa un algoritmo que corra en tiempo $O(\log n)$ para encontrar la mediana de los $2n$ elementos

© 2018 Blai Bonet