Fundamentos de análisis y diseño de algoritmos

Quicksort

Características del Quicksort

Procedimiento PARTITION

Quicksort

Análisis de complejidad

Quicksort con aleatoriedad

Quicksort

- En el peor caso tiene un tiempo de $\Theta(n^2)$
- Es una de las mejores opciones en la práctica, debido a que su tiempo promedio es de $\Theta(n lg n)$
- Los factores constantes en $\Theta(n lg n)$ son pequeños comparados con los de los otros algoritmos
- Es ordenación *in-place*
- Técnica: Dividir, Conquistar y Combinar

Quicksort

Dado un arreglo A[p..r]

Dividir: A[p..r] es particionado en dos subarreglos no vacíos A[p..q] y A[q+1..r] tal que cada elemento en A[p..q] sea menor o igual que los elementos en A[q+1..r]

Conquistar: Ordenar los dos subarreglos A[p..q] y A[q+1..r] recursivamente utilizando QuickSort

Combinar: Ya que los subarreglos son ordenados *in-place*, no es necesario llevar a cabo la tarea de combinar

Quicksort

QUIKSORT basa su funcionamiento en un procedimiento llamado PARTITION

PARTITION(A,p,r)

Precondición: p y r son índices válidos en A

Poscondicion: x=A[p], se intercambian de posicion los datos en A, de tal forma que los elementos con índice menor o igual a j, son menores que x, y aquellos con índice mayor a j, son mayores o iguales a x

Quicksort

PARTITION(A,p,r)

5	3	2	6	4	1	3	7

p r

$x = 5$

i → 5 p

j → 5

3 3 2 1 4 | 6 5 7

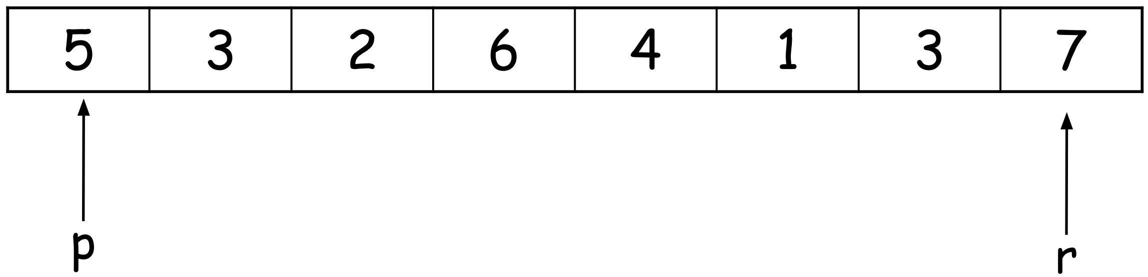
j i

$i < j$

p — j | j+1 r

Quicksort

PARTITION(*A*,p,r)



$x \leftarrow A[p]$

Quicksort

PARTITION(A,p,r)

5	3	2	6	4	1	3	7

p r

$x \leftarrow A[p]$

p r p r

3	3	2	1	4	6	5	7

$x \leftarrow 3$

1 2 | 3 3 4 j 5 | 6 7

j i j i

Quicksort

PARTITION(A,p,r)

5	3	2	6	4	1	3	7

p r

$x \leftarrow A[p]$

A[p..q] A[q+1..r]

3	3	2	1	4	6	5	7

j

Los elementos en A[p..q] son todos menores que x

Los elementos en A[q+1..r] son todos mayores o iguales a x

Quicksort

PARTITION(A,p,r)

5	3	2	6	4	1	3	7

p

r

$x \leftarrow A[p]$

PARTITION cambia de lugar los elementos en A (*in-place*)

A[p..q]

A[q+1..r]

3	3	2	1	4	6	5	7

j

Los elementos en A[p..q] son todos mejores que x

Los elementos en A[q+1..r] son todos mayores o iguales a x

Quicksort

```
PARTITION(A,p,r)
  x ← A[p]
  i ← p-1
  j ← r+1
  while TRUE
     do repeat j ← j-1
          until A[j] ≤ x
          repeat i ← i+1
          until A[i] ≥ x
          if i<j
             then exchange A[i] ↔ A[j]
             else return j
```

1st 2nd 3th 4th 5th 6th

1 2 4 8 6 7

5	3	2	6	4	1	3	7

Siga el algoritmo PARTITION(A,1,8)

1 5 6 8

3 3 2 1 4 | 6 5 7

j i

P(A, 1, 5)

P(A, 6, 8)

1 2 3 4 5 6 7 8

1 2 | 3 3 4 | 5 | 6 7

j i j i

P(1, 2), P(3, 5) P(6, 6) P(7, 8)

Quicksort

```
PARTITION(A,p,r)
 x ← A[p]
 i ← p-1
 j ← r+1
 while TRUE
    do repeat j ← j-1
         until A[j]≤x
         repeat i ← i+1
         until A[i]≥x
         if i<j
            then exchange A[i] ↔ A[j]
            else return j
```

5	3	2	6	4	1	3	7

i

j

Quicksort

```
PARTITION(A,p,r)
 x ← A[p]
 i ← p-1
 j ← r+1
 while TRUE
    do repeat j ← j-1
         until A[j]≤x
         repeat i ← i+1
         until A[i]≥x
         if i<j
            then exchange A[i] ↔ A[j]
            else return j
```

5	3	2	6	4	1	3	7

i

j

Quicksort

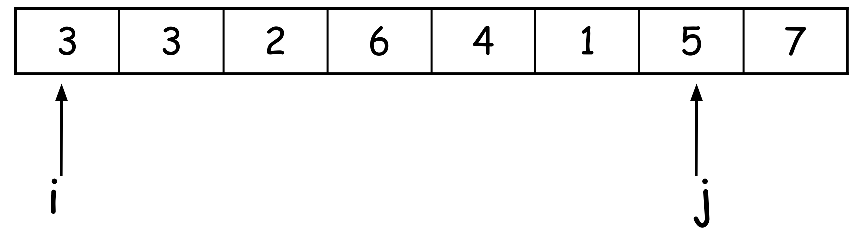
```
PARTITION(A,p,r)
 x ← A[p]
 i ← p-1
 j ← r+1
 while TRUE
    do repeat j ← j-1
         until A[j]≤x
         repeat i ← i+1
         until A[i]≥x
         if i<j
            then exchange A[i] ↔ A[j]
            else return j
```

5	3	2	6	4	1	3	7

i   j

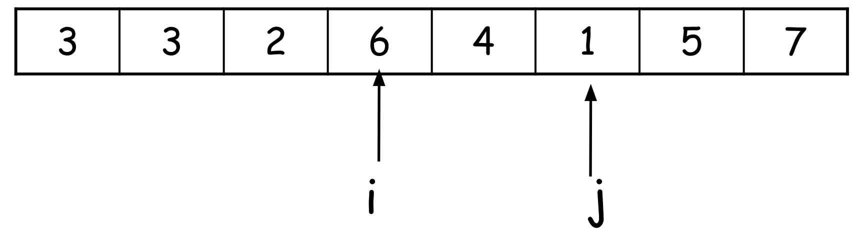
Quicksort

```
PARTITION(A,p,r)
 x ← A[p]
 i ← p-1
 j ← r+1
 while TRUE
    do repeat j ← j-1
          until A[j]≤x
          repeat i ← i+1
          until A[i]≥x
          if i<j
             then exchange A[i] ↔ A[j]
             else return j
```



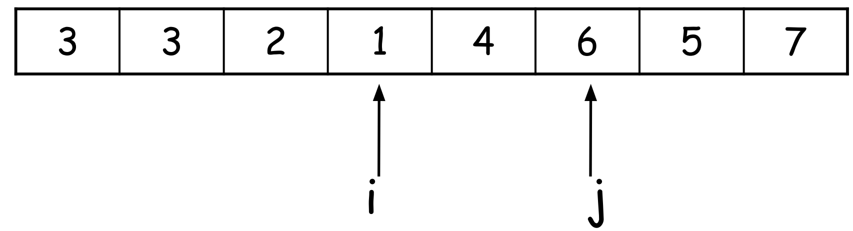
Quicksort

```
PARTITION(A,p,r)
 x ← A[p]
 i ← p-1
 j ← r+1
 while TRUE
    do repeat j ← j-1
         until A[j]≤x
         repeat i ← i+1
         until A[i]≥x
         if i<j
            then exchange A[i] ↔ A[j]
            else return j
```



Quicksort

```
PARTITION(A,p,r)
 x ← A[p]
 i ← p-1
 j ← r+1
 while TRUE
    do repeat j ← j-1
         until A[j]≤x
         repeat i ← i+1
         until A[i]≥x
         if i<j
            then exchange A[i] ↔ A[j]
            else return j
```



Quicksort

```
PARTITION(A,p,r)
 x ← A[p]
 i ← p-1
 j ← r+1
 while TRUE
    do repeat j ← j-1
         until A[j]≤x
         repeat i ← i+1
         until A[i]≥x
         if i<j
            then exchange A[i] ↔ A[j]
            else return j
```

3	3	2	1	4	6	5	7

j i

Retorna j=5 y los elementos en A quedan reubicados como se muestra

Quicksort

```
PARTITION(A,p,r)
 x ← A[p]
 i ← p-1
 j ← r+1
 while TRUE
    do repeat j ← j-1
         until A[j]≤x
         repeat i ← i+1
         until A[i]≥x
         if i<j
            then exchange A[i] ↔ A[j]
            else return j
```

1	10	9	3	2	8	3	9

Muestre el contenido de A
y el valor de j retornado
para el llamado
PARTITION(A, 1,8)

[ ] 3 | 10 9 3 2 8 3 9

p
i
p
j

Quicksort

```
PARTITION(A,p,r)
 x ← A[p]
 i ← p-1
 j ← r+1
 while TRUE
    do repeat j ← j-1
          until A[j]≤x
          repeat i ← i+1
          until A[i]≥x
          if i<j
             then exchange A[i] ↔ A[j]
             else return j
```

5	5	5	5	5	5	5	5

j i

Muestre el contenido de A
y el valor de j retornado
para el llamado
PARTITION(A, 1,8)

Quicksort

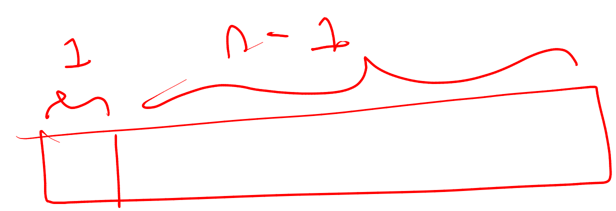
```
PARTITION(A,p,r)
 x ← A[p]
 i ← p-1
 j ← r+1
 while TRUE
    do repeat j ← j-1
         until A[j]≤x
         repeat i ← i+1
         until A[i]≥x
         if i<j
           then exchange A[i] ↔ A[j]
           else return j
```

10	1	9	3	2	8	3	7

Muestre el contenido de A
y el valor de j retornado
para el llamado
PARTITION(A, 1,8)

7 1 9 3 2 8 3 | 10

j i



pivote
min
0
max

Quicksort

```
PARTITION(A,p,r)
 x ← A[p]
 i ← p-1
 j ← r+1
 while TRUE
    do repeat j ← j-1
         until A[j]≤x
         repeat i ← i+1
         until A[i]≥x
         if i<j
           then exchange A[i] ↔ A[j]
           else return j
```

6	1	5	8	9	4	7	7

Muestre el contenido de A
y el valor de j retornado
para el llamado
PARTITION(A, 1,8)

Quicksort

QUICKSORT(*A*,p,r)

Idea: realizar particiones sucesivas de *A* hasta llegar a particiones triviales que en conjunto dejen ordenado al arreglo *A*

Quicksort

```
PARTITION(A,p,r)
 x ← A[p]
 i ← p-1
 j ← r+1
 while TRUE
    do repeat j ← j-1
         until A[j]≤x
         repeat i ← i+1
         until A[i]≥x
         if i<j
            then exchange A[i] ↔ A[j]
            else return j
```

5	1	8	4	9	3	7	6

3 1 4 | 8 9 5 7 6

Muestre el contenido de A
y el valor de j retornado
para el llamado
PARTITION(A, 1,8)

Quicksort

```
PARTITION(A,p,r)
 x ← A[p]
 i ← p-1
 j ← r+1
 while TRUE
    do repeat j ← j-1
         until A[j]≤x
         repeat i ← i+1
         until A[i]≥x
         if i<j
            then exchange A[i] ↔ A[j]
            else return j
```

3	1	4	8	9	5	7	6

j=3

Ahora se aplique PARTITION sobre cada partición resultante

3	1	4	8	9	5	7	6

PARTITION(A,1,3) PARTITION(A,4,8)

1 | 3 | 4 | 6 7 5 | 9 8

P(4,6) P(7,8)

5 | 7 6 8 | 9

6 | 7

Quicksort

```
PARTITION(A,p,r)
 x ← A[p]
 i ← p-1
 j ← r+1
 while TRUE
    do repeat j ← j-1
         until A[j]≤x
         repeat i ← i+1
         until A[i]≥x
         if i<j
            then exchange A[i] ↔ A[j]
            else return j
```

i≥ j≤

3	1	4	8	9	5	7	6

3 1 4 8 9 5 7 6

j=3

Ahora se aplique PARTITION sobre cada partición resultante

3	1	4	8	9	5	7	6

PARTITION(A,1,3) PARTITION(A,4,8)

1	3	4	6	7	5	9	8

PARTITION(A,1,1) PARTITION(A,4,6)

PARTITION(A,2,3) PARTITION(A,7,8)

Quicksort

```
PARTITION(A,p,r)
 x ← A[p]
 i ← p-1
 j ← r+1
 while TRUE
    do repeat j ← j-1
         until A[j]≤x
         repeat i ← i+1
         until A[i]≥x
         if i<j
            then exchange A[i] ↔ A[j]
            else return j
```

1	3	4	6	7	5	9	8

PARTITION(A,1,1) PARTITION(A,4,6)

PARTITION(A,2,3) PARTITION(A,7,8)

PARTITION(A,1,1) no se debe realizar, ya está ordenado el subarreglo

Quicksort

```
PARTITION(A,p,r)
 x ← A[p]
 i ← p-1
 j ← r+1
 while TRUE
    do repeat j ← j-1
         until A[j]≤x
         repeat i ← i+1
         until A[i]≥x
         if i<j
            then exchange A[i] ↔ A[j]
            else return j
```

1	3	4	6	7	5	9	8

PARTITION(A,1,1) PARTITION(A,4,6)

PARTITION(A,2,3) PARTITION(A,7,8)

1	3	4	5	7	6	8	9

PARTITION(A,1,1) PARTITION(A,4,4)

PARTITION(A,5,6)

PARTITION(A,2,2)

PARTITION(A,3,3) PARTITION(A,7,7)

PARTITION(A,8,8)

Quicksort

```
PARTITION(A,p,r)
 x ← A[p]
 i ← p-1
 j ← r+1
 while TRUE
    do repeat j ← j-1
         until A[j]≤x
         repeat i ← i+1
         until A[i]≥x
         if i<j
           then exchange A[i] ↔ A[j]
           else return j
```

1	3	4	6	7	5	9	8

PARTITION(A,1,1) PARTITION(A,4,6)

PARTITION(A,2,3) PARTITION(A,7,8)

1	3	4	5	7	6	8	9

PARTITION(A,1,1) PARTITION(A,4,4)

PARTITION(A,5,6)

PARTITION(A,2,2)

PARTITION(A,3,3) PARTITION(A,7,7)

PARTITION(A,8,8)

Quicksort

```
PARTITION(A,p,r)
 x ← A[p]
 i ← p-1
 j ← r+1
 while TRUE
    do repeat j ← j-1
         until A[j]≤x
         repeat i ← i+1
         until A[i]≥x
         if i<j
            then exchange A[i] ↔ A[j]
            else return j
```

1	3	4	6	7	5	9	8

PARTITION(A,1,1) PARTITION(A,4,6)

PARTITION(A,2,3) PARTITION(A,7,8)

1	3	4	5	7	6	8	9

PARTITION(A,1,1) PARTITION(A,4,4)

PARTITION(A,5,6)

PARTITION(A,2,2)

PARTITION(A,3,3) PARTITION(A,7,7)

PARTITION(A,8,8)

1	3	4	5	6	7	8	9

Quicksort

```
QUICKSORT(A,p,r)
 if p<r
   then q ← PARTITION(A,p,r)
        QUICKSORT(A,p,q)
        QUICKSORT(A,q+1,r)
```

Aplique el algoritmo QUICKSORT(A), para A={3,4,1,7,8,2}

Quicksort

```
QUICKSORT(A,p,r)
 if p<r
   then q ← PARTITION(A,p,r)
          QUICKSORT(A,p,q)
          QUICKSORT(A,q+1,r)
```

Aplique el algoritmo QUICKSORT(A), para A={3,2,1,6,5,7}

Quicksort

```
QUICKSORT(A,p,r)
 if p<r
   then q←PARTITION(A,p,r)
          QUICKSORT(A,p,q)
          QUICKSORT(A,q+1,r)
```

Aplique el algoritmo QUICKSORT(A), para A={9,4,3,1,5,6,3}

Quicksort

```
QUICKSORT(A,p,r)
 if p<r
   then q ← PARTITION(A,p,r)
        QUICKSORT(A,p,q)
        QUICKSORT(A,q+1,r)
```

Aplique el algoritmo QUICKSORT(A), para A={1,2,3,4,5,6,7}

Quicksort

```
QUICKSORT(A,p,r)
 if p<r
  then q←PARTITION(A,p,r)
        QUICKSORT(A,p,q)
        QUICKSORT(A,q+1,r)
```

Aplique el algoritmo QUICKSORT(A), para A={1,7,6,5,4,3,2}

5,1,8,4,9,3,7,6 x=5

i 0 1 2 j

3,1,4 | 8,9,5,7,6 q=2 x=8

6,7, 5 | 9, 8

5 | 7, 6

6 7

Quicksort

Análisis de complejidad

- El tiempo de ejecución depende de que tan balanceado queden las particiones

- Si el particionamiento es balanceado, el algoritmo corre tan rápido como el Mergesort. De no serlo, corre tan lento como el Insertionsort

Quicksort

Análisis de complejidad

• ¿Cuál es el peor caso que puede ocurrir al particionar n elementos?

*subproblemas

$$T(n) = a\,T\left(\frac{n}{b}\right) + C(n) + D(n)$$

tamaño del subproblema

costo combinar

costo conquistar

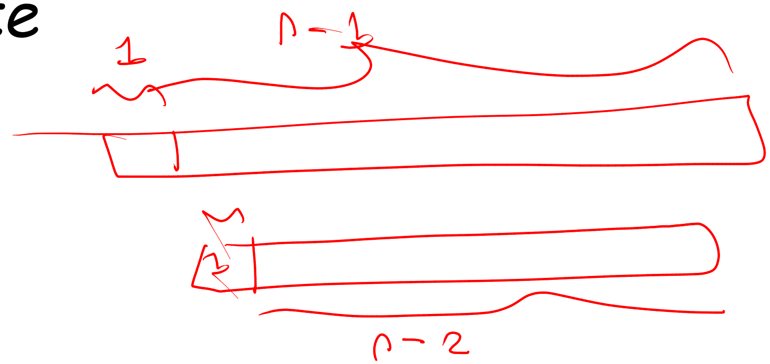
Quicksort

Análisis de complejidad

• ¿Cuál es el peor caso que puede ocurrir al particionar n elementos?

Que resulte una partición de (n-1) elementos y otra de 1.

Además, que cuando se particione sobre los (n-1) elementos, quede una partición de (n-2) y otra de 1. Así sucesivamente



Quicksort

Análisis de complejidad

- ¿Cuál es el peor caso que puede ocurrir al particionar n elementos?

Que resulte una partición de (n-1) elementos y otra de 1.

Además, que cuando se particione sobre los (n-1) elementos, quede una partición de (n-2) y otra de 1. Así sucesivamente

$\Theta(n)$

$T(n)= T(n-1) + \Theta(n)$, donde $\Theta(n)$ es el costo de hacer la particion sobre n elementos

$T(n) = T^b(n) + T^p(n)$

R, R no homogenea

$r = 1$

$r = 1$

$T(n) = A 1^n$

$T^p = Bn + \cancel{C} = Bn^2 + Cn$

$Bn^2 + Cn = B(n-1)^2 + C(n-1) + n \longrightarrow O(n^2)$

Quicksort

Análisis de complejidad

• ¿Cuál es el peor caso que puede ocurrir al particionar n elementos?

Que resulte una partición de (n-1) elementos y otra de 1.

Además, que cuando se particione sobre los (n-1) elementos, quede una partición de (n-2) y otra de 1. Así sucesivamente

$$T(n) = T(n-1) + \Theta(n)$$

$$= T(n-2) + \Theta(n-1) + \Theta(n)$$

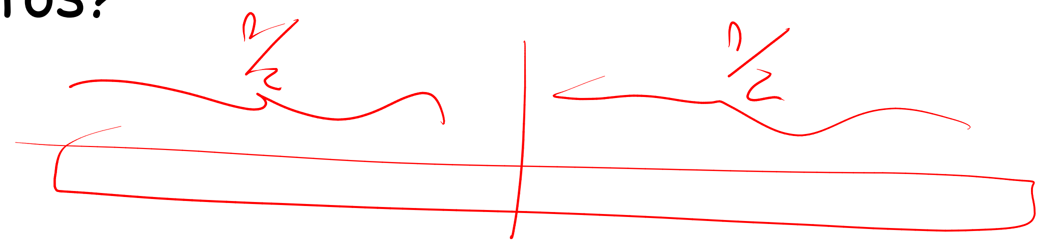
$$\ldots$$

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

Quicksort

Análisis de complejidad

• ¿Cuál es el mejor caso que puede ocurrir al particionar n elementos?



$$T(n) = 2T\left(\frac{n}{2}\right) + n \qquad n^{\log_b a} \to n^1$$

$f(n) = n$

2) n es $\Theta(n)$ ✓

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

Quicksort

Análisis de complejidad

• ¿Cuál es el mejor caso que puede ocurrir al particionar n elementos?

Que resulten 2 particiones, cada una de n/2 elementos

$$T(n)= 2T(n/2) + \Theta(n)$$

Por teorema maestro, se tiene que $T(n)=\Theta(n\lg n)$

Quicksort

Análisis de complejidad

- ¿Cuál es el caso promedio?

Quicksort

Análisis de complejidad

- ¿Cuál es el caso promedio?

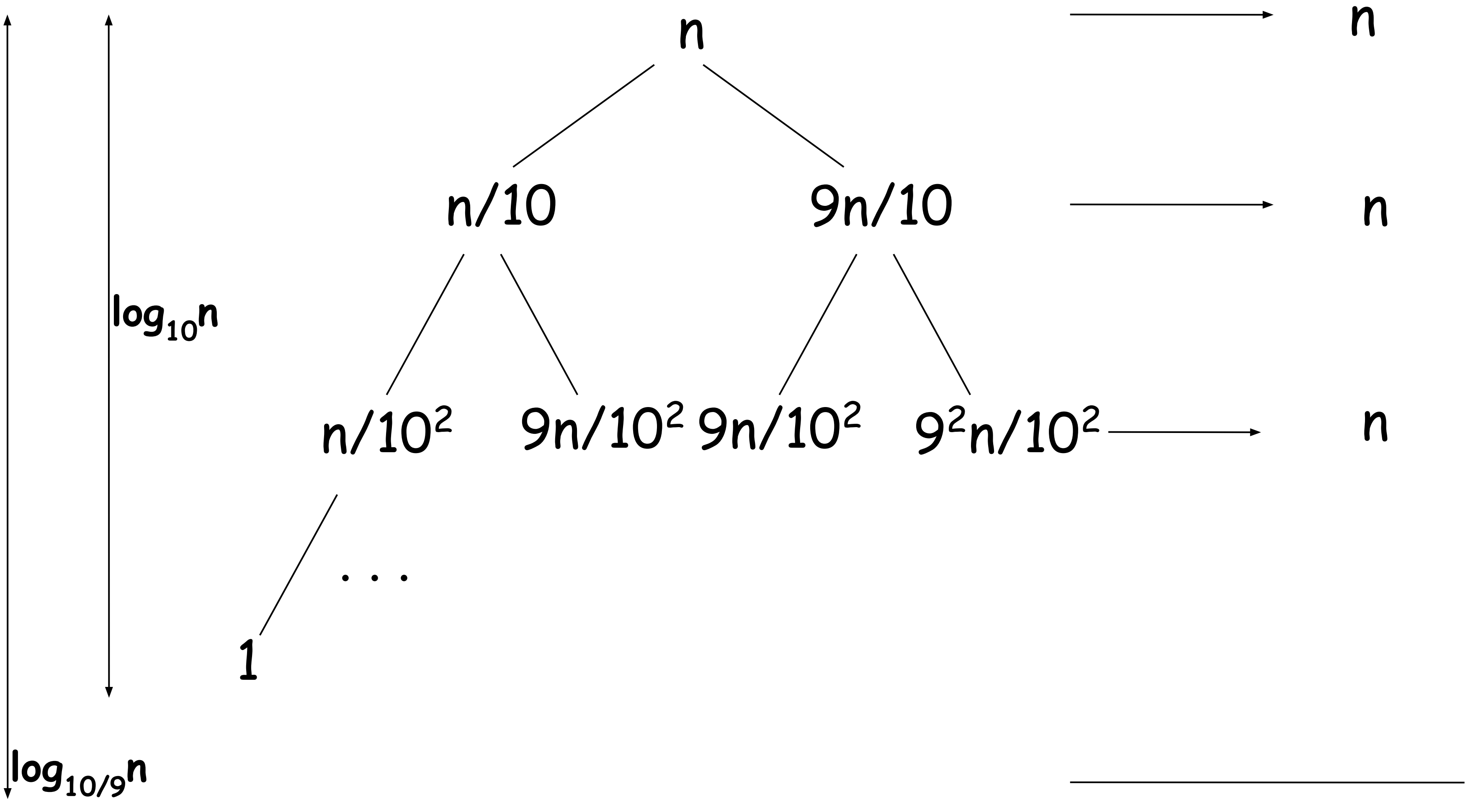
Se considera una proporción 9 a 1, esto es,

$$T(n)= T(9n/10) + T(n/10) + n$$

Quicksort

$$T(n) = T(9n/10) + T(n/10) + n$$

Quicksort



Total=Θ(nlgn)

Quicksort

Versiones aleatorias del QuickSort

Un algoritmo es aleatorio si su comportamiento no está determinado únicamente por su entrada, sino también por los valores producidos por un generador de números aleatorios

Random(a,b): devuelve un entero entre a y b, siendo cada entero igualmente probable

Quicksort

Versiones aleatorias del QuickSort

Un algoritmo es aleatorio si su comportamiento no está determinado únicamente por su entrada, sino también por los valores producidos por un generador de números aleatorios

Random(a,b): devuelve un entero entre a y b, siendo cada entero igualmente probable

Quicksort

```
PARTITION(A,p,r)
 x ← A[p]
 i ← p-1
 j ← r+1
 while TRUE
    do repeat j ← j-1
         until A[j]≤x
         repeat i ← i+1
         until A[i]≥x
         if i<j
            then exchange A[i] ↔ A[j]
            else return j
```

10	1	8	4	9	3	7	6

El problema de Quicksort es que depende de qué tan bueno resulte x, esto es, A[p]

Quicksort

Versiones aleatorias del QuickSort

Versión 1: permutar de forma aleatoria la entrada y luego llamar a Quicksort, esperando que el azar permita que el valor de x genere particiones balanceadas

Versión 2: Utilizar RANDOMIZED-PARTITION(A,p,r)

Quicksort

Versiones aleatorias del QuickSort

RANDOMIZED-PARTITION(A,p,r)

i ← RANDOM(p,r)

exchange A[p] ↔ A[i]

return PARTITION(A,p,r)

Referencias

Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. 2009. Introduction to Algorithms, Third Edition (3rd ed.). The MIT Press. Chapter 7

Gracias

Próximo tema:

Ordenamiento en tiempo lineal