Outline

- Correctitud de Quicksort
- Complejidad de Quicksort:
  - Peor caso
  - Mejor caso
  - Caso promedio

Outline

- Correctitud de Quicksort
- Complejidad de Quicksort:
  - Peor caso
  - Mejor caso
  - Caso promedio

Quicksort termina

¿Cómo lo demostramos?

Podemos partir de la base de que $\boldsymbol{partition}$ termina

```
quicksort(A, i, f):
    if f − i ≥ 0:
        p = partition(A, i, f)
        quicksort(A, i, p − 1)
        quicksort(A, p + 1, f)
```

Inicialmente Quicksort se llama con los índices

$$i = 0$$

$$f = n - 1$$

$$d = f - i = n - 1$$

Cuando $d < 0$ entonces QuickSort termina. En otro caso se llama recursivamente con dos nuevos intervalos:

$$d_1 = p - 1 - i < d$$

$$d_2 = f - p - 1 < d$$

Esto significa que para cada nivel de recursión el d siempre disminuye al menos en 1, lo que limita la profundidad de recursión a una profundidad máxima de $n - 1$. Por lo tanto, QuickSort termina.

Quicksort ordena

¿Cómo lo demostramos?

Podemos partir de la base de que $\boldsymbol{partition}$ es correcto

Que partition sea correcto significa que escoge un pivote p, y separa el intervalo en 2: los elementos menores y los mayores al pivote, y retorna el índice en que queda p en el arreglo final.

Luego de partition, el pivote queda en su posición ordenada en el arreglo. Esto se debe a que todos los elementos a la izquierda del pivote son menores a este, y todos los elementos a la derecha son mayores.

Sea N el conjunto de todos los elementos del arreglo, y $P = \emptyset$ el conjunto de todos los elementos que han sido pivote. Luego de cada llamada a partition, el pivote p se extrae de N y se agrega a P, mientras que el resto de los elementos de N pertenecen a algún intervalo de las llamadas recursivas de QuickSort. Quicksort solo termina cuando $N = \emptyset$, por lo tanto eventualmente todos los elementos están en P, por lo que han sido pivotes alguna vez. Esto significa que todos los elementos están en su posición ordenada.

Outline

- Correctitud de Quicksort
- Complejidad de Quicksort:
  - Peor caso
  - Mejor caso
  - Caso promedio

Peor caso

Vimos la clase pasada que el peor caso de QuickSort se da cuando la posición final del pivote p es i o f

Si esto pasa en cada llamada a partition, ¿Cuánto tiempo toma?

La situación antes descrita se expresa con la recurrencia

$$T(n) = \begin{cases} 1 & n = 1 \\ n + T(n-1) & n > 1 \end{cases}$$

Esto se traduce a la siguiente sumatoria:

$$\sum_{i=1}^{n} i = \frac{n \cdot (n-1)}{2} \in \Theta(n^2)$$

Outline

- Correctitud de Quicksort
- Complejidad de Quicksort:
  - Peor caso
  - Mejor caso
  - Caso promedio

Mejor caso

Discutimos que el mejor caso es cuando el pivote divide el intervalo de manera equitativa.

¿Qué tiempo toma el algoritmo en este caso?

Expresemos la recurrencia

La situación antes descrita se expresa con la recurrencia

$$T(n) = \begin{cases} 1 & n = 1 \\ n + T\left(\left\lceil \frac{n-1}{2} \right\rceil\right) + T\left(\left\lfloor \frac{n-1}{2} \right\rfloor\right) & n > 1 \end{cases}$$

Podemos simplificar la expresión acotándolo por arriba:

$$T\left(\left\lceil \frac{n-1}{2} \right\rceil\right) \geq T\left(\left\lfloor \frac{n-1}{2} \right\rfloor\right)$$

Por lo tanto queda simplificado a

$$T(n) \leq \begin{cases} 1 & n = 1 \\ n + 2 \cdot T\left(\left\lceil \frac{n-1}{2} \right\rceil\right) & n > 1 \end{cases}$$

Además, podemos seguir acotándolo:

$$T\left(\left\lceil \frac{n-1}{2} \right\rceil\right) < T\left(\frac{n}{2}\right)$$

Por lo tanto queda simplificado a

$$T(n) \leq \begin{cases} 1 & n = 1 \\ n + 2 \cdot T\left(\frac{n}{2}\right) & n > 1 \end{cases}$$

Vamos a resolver esta recurrencia para un 2^k tal que

$$n \leq 2^k < 2n$$

$$T(n) \leq T\left(2^k\right) \leq 2^k + 2 \cdot T\left(2^{k-1}\right)$$

$$= 2^k + 2 \cdot \left[2^{k-1} + 2 \cdot T\left(2^{k-2}\right)\right]$$

$$= 2^k + 2^k + 2^2 \cdot T\left(2^{k-2}\right)$$

$$= 2^k + 2^k + 2^2 \cdot \left[2^{k-2} + 2 \cdot T\left(2^{k-3}\right)\right]$$

$$= 2^k + 2^k + 2^k + 2^3 \cdot T\left(2^{k-3}\right)$$

$$\vdots$$

$$= i \cdot 2^k + 2^i \cdot T\left(2^{k-i}\right)$$

Cuando $i = k$, $T\left(2^{k-i}\right) = 1$ por el caso base, por lo tanto

$$T(n) \leq k \cdot 2^k + 2^k$$

Tenemos entonces que

$$T(n) \leq k \cdot 2^k + 2^k$$

Por construcción de k

$$2^k < 2n$$

Por lo tanto

$$T(n) \leq k \cdot 2^k + 2^k < \log(2n) \cdot 2n + 2n$$

Finalmente

$$T(n) \in O(n \cdot \log n)$$

Outline

- Correctitud de Quicksort
- Complejidad de Quicksort:
  - Peor caso
  - Mejor caso
  - Caso promedio

Caso promedio

Quicksort hace dos operaciones:

- Comparar elementos
- Intercambio de elementos

¿Cuál de estos nos interesa contar?

La cantidad de comparaciones, ya que es mayor a la cantidad de intercambios, por lo tanto tiene un mayor impacto en la complejidad del algoritmo.

Definición

Definimos los datos a ordenar como

$$a_1, a_2, \cdots, a_n$$

Tal que $a_i < a_j$ si $i < j$. Esto significa que el elemento a_i es el i-ésimo elemento en el arreglo ordenado.

Una instancia del problema es una permutación de estos elementos.

Contemos comparaciones

Escojamos dos valores a_i y a_j tal que $i < j$

¿Cuando se comparan estos dos elementos?

¿Es posible que se comparen más de una vez?

¿Es posible que no se comparen ninguna vez?

Se comparan cuando uno de ellos dos es pivote.

No se vuelven a comparar ya que el pivote se elimina de la llamada recursiva.

Si, es posible que no se comparen. Esto pasa cuando un valor a_p se escoge como pivote tal que $i < p < j$, ya que a_i y a_j se separan en dos subintervalos distintos.

Contemos comparaciones

Por lo tanto cada par de elementos se compara 0 o 1 vez

Definimos C_{ij} como el número de comparaciones de a_i y a_j

$$C_{ij} = 0 \text{ o } C_{ij} = 1$$

¿Hay elementos que tengan mayor probabilidad de compararse?

$j - i - 1$ es la cantidad de elementos que hay entre i y j

Si $i \ll j$ existe una gran probabilidad de que se escoja como pivote un elemento entre ellos dos antes de que i o j sean pivote. Por lo tanto no se compararían.

Por otro lado si ambos están muy cerca, hay pocos elementos que pueden escogerse como pivote, por lo que es más probable que se comparen.

Si $i = j - 1$ no hay elemento entre ellos asi que siempre se compararán.

Contemos comparaciones

¿Cuál es la probabilidad exacta de que $C_{ij} = 1$?

Cada elemento de un subarreglo tiene la misma probabilidad de ser escogido como pivote en una llamada a partition.

Tenemos 3 casos:

- $p < i$ o $p > j \Longrightarrow$ esto se decide en otra llamada a partition
- $p = i$ o $p = j \Longrightarrow a_i$ y a_j se comparan
- $i < p < j \Longrightarrow a_i$ y a_j nunca se comparan

Por lo tanto la probabilidad depende únicamente de i y j, y no del subarreglo en el que se llama partition.

Definimos $\mathbb{P}_{ij}$ = probabilidad de que $C_{ij} = 1$

$$\mathbb{P}_{ij} = \frac{\text{casos favorables}}{\text{todos los posibles casos}}$$

Casos favorables $= 2$. Casos desfavorables $= j - i - 1$

Contando comparaciones

Por lo tanto,

$$\mathbb{P}_{ij} = \frac{2}{j - i + 1}$$

Las comparaciones hechas por Quicksort en una llamada son:

$$\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} C_{ij}$$

El número promedio de comparaciones hechas por QuickSort se define como:

$$\mathbb{E}\left(\sum_{i=1}^{n-1}\sum_{j=i+1}^{n} C_{ij}\right) = \sum_{i=1}^{n-1}\sum_{j=i+1}^{n} \mathbb{E}(C_{ij}) = \sum_{i=1}^{n-1}\sum_{j=i+1}^{n} \mathbb{P}_{ij}$$

Ojo: esperanza no es parte de la materia del curso, está solo por formalidad.

$$\sum_{i=1}^{n-1}\sum_{j=i+1}^{n}\mathbb{P}_{ij}=\sum_{i=1}^{n-1}\sum_{j=i+1}^{n}\frac{2}{j-i+1}$$

$$=2\sum_{i=1}^{n-1}\left[\frac{1}{2}+\frac{1}{3}+\cdots+\frac{1}{n-i+1}\right]$$

$i = 1$	$\frac{1}{2}$	$\frac{1}{3}$	$\cdots$	$\frac{1}{n-4}$	$\frac{1}{n-3}$	$\frac{1}{n-2}$
$i = 2$	$\frac{1}{2}$	$\frac{1}{3}$	$\cdots$	$\frac{1}{n-4}$	$\frac{1}{n-3}$	
$i = 3$	$\frac{1}{2}$	$\frac{1}{3}$	$\cdots$	$\frac{1}{n-4}$		
$\vdots$	$\vdots$	$\vdots$				
$i = n-2$	$\frac{1}{2}$	$\frac{1}{3}$				
$i = n-1$	$\frac{1}{2}$					

Podemos reescribir esta sumatoria entonces como:

$$\frac{1}{2}(n-1)+\frac{1}{3}(n-2)+\cdots+\frac{1}{n-4}3+\frac{1}{n-3}2+\frac{1}{n-2}1$$

Esto es estrictamente menor que

$$\frac{1}{2}n+\frac{1}{3}n+\cdots+\frac{1}{n-4}n+\frac{1}{n-3}n+\frac{1}{n-2}n$$

Lo cual es estrictamente menor que

$$n\sum_{i=1}^{n}\frac{1}{i}$$

Calcular esta sumatoria no es fácil, pero podemos acotarla

$$\sum_{i=1}^{n} \frac{1}{i}$$

Veamos que relación tiene con

$$\int_{1}^{n} \frac{1}{x} dx = \ln n$$

El cual sería un valor conveniente para nosotros

Acotando por arriba:

$$\sum_{i=2}^{n} \frac{1}{i} < \int_{1}^{n} \frac{1}{x} dx$$

$$\sum_{i=1}^{n} \frac{1}{i} < \int_{1}^{n} \frac{1}{x} dx + 1$$

$$\sum_{i=1}^{n} \frac{1}{i} < \ln n + 1$$

Caso Promedio

Por lo tanto, el numero de comparaciones promedio es menor a

$$2n \cdot (\ln n + 1)$$

Por lo tanto la complejidad promedio de QuickSort es

$$O(n \cdot \log n)$$

¿Qué hemos visto?

Ordenación como **problema computacional**

Cotas inferiores para algoritmos que lo resuelven:

- Comparando e intercambiando elementos adyacentes: $\Omega(n^2)$
- Comparando e intercambiando elementos: $\Omega(n \cdot \log n)$

Resumen de algoritmos de ordenación

Algoritmo	Mejor caso	Caso promedio	Peor caso	Memoria adicional
SelectionSort	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(1)$
InsertionSort	$O(n)$	$O(n^2)$	$O(n^2)$	$O(1)$
QuickSort	$O(n \cdot \log n)$	$O(n \cdot \log n)$	$O(n^2)$	$O(1)$
MergeSort	$O(n \cdot \log n)$	$O(n \cdot \log n)$	$O(n \cdot \log n)$	$O(n)$
?????	$O(n \cdot \log n)$	$O(n \cdot \log n)$	$O(n \cdot \log n)$	$O(1)$

MergeSort, nuestro viejo amigo de matemáticas discretas

Y un último algoritmo que veremos este lunes