

Quicksort e Select Aleatorizados

CLRS Secs 7.3, 7.4 e 9.2

Relembremos o Particione

Rearranja $A[p..r]$ de modo que $p \leq q \leq r$ e
 $A[p..q-1] \leq A[q] < A[q+1..r]$

PARTICIONE (A, p, r)

- 1 $x \leftarrow A[r]$ ▷ x é o “pivô”
- 2 $i \leftarrow p-1$
- 3 para $j \leftarrow p$ até $r-1$ faça
- 4 se $A[j] \leq x$
- 5 então $i \leftarrow i+1$
- 6 $A[i] \leftrightarrow A[j]$
- 7 $A[i+1] \leftrightarrow A[r]$
- 8 devolva $i+1$

Invariantes: no começo de cada iteração de 3–6,

(i0) $A[p..i] \leq x$ (i1) $A[i+1..j-1] > x$ (i2) $A[r] = x$

Relembramos o Particione

Rearranja $A[p..r]$ de modo que $p \leq q \leq r$ e
 $A[p..q-1] \leq A[q] < A[q+1..r]$

PARTICIONE (A, p, r)

- 1 $x \leftarrow A[r]$ $\triangleright x$ é o “pivô”
- 2 $i \leftarrow p-1$
- 3 para $j \leftarrow p$ até $r-1$ faça
- 4 se $A[j] \leq x$
- 5 então $i \leftarrow i+1$
- 6 $A[i] \leftrightarrow A[j]$
- 7 $A[i+1] \leftrightarrow A[r]$
- 8 devolva $i+1$

Consumo de tempo: $\Theta(n)$ onde $n := r - p$.

Quicksort aleatorizado

PARTICIONE-ALEA(A, p, r)

1 $i \leftarrow \text{RANDOM}(p, r)$

2 $A[i] \leftrightarrow A[r]$

3 devolva PARTICIONE(A, p, r)

Quicksort aleatorizado

PARTICIONE-ALEA(A, p, r)

- 1 $i \leftarrow \text{RANDOM}(p, r)$
- 2 $A[i] \leftrightarrow A[r]$
- 3 devolva PARTICIONE(A, p, r)

QuickSort-Ale(A, p, r)

- 1 se $p < r$
- 2 então $q \leftarrow \text{PARTICIONE-ALEA}(A, p, r)$
- 3 QuickSort – Ale($A, p, q - 1$)
- 4 QuickSort – Ale($A, q + 1, r$)

Como analisar?

Vamos estimar o tempo médio...

Como analisar?

Vamos estimar o tempo médio...do que, mesmo?

Como analisar?

Vamos estimar o tempo médio...do que, mesmo?

- ▶ Fixe uma permutação

Como analisar?

Vamos estimar o tempo médio...do que, mesmo?

- ▶ Fixe uma permutação
- ▶ Considere as várias instâncias do QuickSort-Ale em cima dela, com a distribuição de probabilidade dada pelos sorteios ao longo da execução.

Como analisar?

Vamos estimar o tempo médio...do que, mesmo?

- ▶ Fixe uma permutação
- ▶ Considere as várias instâncias do QuickSort-Ale em cima dela, com a distribuição de probabilidade dada pelos sorteios ao longo da execução.
- ▶ Determine o tempo médio **para essa permutação**.

Quicksort aleatorizado

PARTICIONE-ALEA(A, p, r)

1 $i \leftarrow \text{RANDOM}(p, r)$

2 $A[i] \leftrightarrow A[r]$

3 **devolva** PARTICIONE(A, p, r)

Quicksort aleatorizado

PARTICIONE-ALEA(A, p, r)

- 1 $i \leftarrow \text{RANDOM}(p, r)$
- 2 $A[i] \leftrightarrow A[r]$
- 3 devolva $\text{PARTICIONE}(A, p, r)$

QuickSort-Ale (A, p, r)

- 1 se $p < r$
- 2 então $q \leftarrow \text{PARTICIONE-ALEA}(A, p, r)$
- 3 $\text{QuickSort-Ale}(A, p, q - 1)$
- 4 $\text{QuickSort-Ale}(A, q + 1, r)$

Quicksort aleatorizado

PARTICIONE-ALEA(A, p, r)

- 1 $i \leftarrow \text{RANDOM}(p, r)$
- 2 $A[i] \leftrightarrow A[r]$
- 3 devolva PARTICIONE(A, p, r)

QuickSort-Ale (A, p, r)

- 1 se $p < r$
- 2 então $q \leftarrow \text{PARTICIONE-ALEA}(A, p, r)$
- 3 QuickSort – Ale ($A, p, q - 1$)
- 4 QuickSort – Ale ($A, q + 1, r$)

Consumo esperado de tempo?

Quicksort aleatorizado

$\text{PARTICIONE-ALEA}(A, p, r)$

- 1 $i \leftarrow \text{RANDOM}(p, r)$
- 2 $A[i] \leftrightarrow A[r]$
- 3 devolva $\text{PARTICIONE}(A, p, r)$

$\text{QuickSort-Ale}(A, p, r)$

- 1 se $p < r$
- 2 então $q \leftarrow \text{PARTICIONE-ALEA}(A, p, r)$
- 3 $\text{QuickSort-Ale}(A, p, q - 1)$
- 4 $\text{QuickSort-Ale}(A, q + 1, r)$

Consumo esperado de tempo?

Basta contar o número esperado de comparações na linha 4 do **PARTICIONE**.

Consumo esperado de tempo

Basta contar o número esperado de comparações na linha 4 do **PARTICIONE**.

```
PARTICIONE ( $A, p, r$ )  
1  $x \leftarrow A[r]$   $\triangleright x$  é o “pivô”  
2  $i \leftarrow p-1$   
3 para  $j \leftarrow p$  até  $r-1$  faça  
4     se  $A[j] \leq x$   
5         então  $i \leftarrow i + 1$   
6              $A[i] \leftrightarrow A[j]$   
7  $A[i+1] \leftrightarrow A[r]$   
8 devolva  $i + 1$ 
```

Consumo de tempo esperado

Suponha $A[p..r]$ permutação de $1..n$.

X_{ab} = número de comparações entre a e b na linha 4 do PARTICIONE do QuickSort-Ale;

Queremos calcular

$$\begin{aligned} X &= \text{total de comparações "A[j] \le x"} \\ &= \sum_{a=1}^{n-1} \sum_{b=a+1}^n X_{ab} \end{aligned}$$

Consumo de tempo esperado

Supondo $a < b$,

$$X_{ab} = \begin{cases} 1 & \text{se primeiro pivô em } \{a, \dots, b\} \text{ é } a \text{ ou } b \\ 0 & \text{caso contrário} \end{cases}$$

Qual a probabilidade de X_{ab} valer 1?

Consumo de tempo esperado

Supondo $a < b$,

$$X_{ab} = \begin{cases} 1 & \text{se primeiro pivô em } \{a, \dots, b\} \text{ é } a \text{ ou } b \\ 0 & \text{caso contrário} \end{cases}$$

Qual a probabilidade de X_{ab} valer 1?

$$\Pr\{X_{ab}=1\} = \frac{1}{b-a+1} + \frac{1}{b-a+1} = E[X_{ab}]$$

Consumo de tempo esperado

Supondo $a < b$,

$$X_{ab} = \begin{cases} 1 & \text{se primeiro pivô em } \{a, \dots, b\} \text{ é } a \text{ ou } b \\ 0 & \text{caso contrário} \end{cases}$$

Qual a probabilidade de X_{ab} valer 1?

$$\Pr\{X_{ab}=1\} = \frac{1}{b-a+1} + \frac{1}{b-a+1} = E[X_{ab}]$$

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

$$E[X] = \text{????}$$

Consumo de tempo esperado

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

Consumo de tempo esperado

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

Consumo de tempo esperado

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

Consumo de tempo esperado

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

Consumo de tempo esperado

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

Consumo de tempo esperado

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

Conclusões

O consumo de tempo esperado do algoritmo
QuickSort-Ale é $O(n \log n)$.

Do exercício 7.4-4 do CLRS temos que

O consumo de tempo esperado do algoritmo
QuickSort-Ale é $\Theta(n \log n)$.

Prática

1. Eliminar recursão de cauda e começar pela “metade menor”, para economizar pilha.

Prática

1. Eliminar recursão de cauda e começar pela “metade menor”, para economizar pilha.
2. Parar a recursão quando n chega num certo ponto ($n \approx 10$). No final, fazer uma passada do **ORDENA-POR-INSERÇÃO**.

Prática

1. Eliminar recursão de cauda e começar pela “metade menor”, para economizar pilha.
2. Parar a recursão quando n chega num certo ponto ($n \approx 10$). No final, fazer uma passada do **ORDENA-POR-INSERÇÃO**.
3. **TIMSORT**. Algo parecido, mas baseado no **MERGESORT**. Usado no python.

Prática

1. Eliminar recursão de cauda e começar pela “metade menor”, para economizar pilha.
2. Parar a recursão quando n chega num certo ponto ($n \approx 10$). No final, fazer uma passada do **ORDENA-POR-INSERÇÃO**.
3. **TIMSORT**. Algo parecido, mas baseado no **MERGESORT**. Usado no python.

Prática

1. Eliminar recursão de cauda e começar pela “metade menor”, para economizar pilha.
2. Parar a recursão quando n chega num certo ponto ($n \approx 10$). No final, fazer uma passada do [ORDENA-POR-INSERÇÃO](#).
3. [TIMSORT](#). Algo parecido, mas baseado no [MERGESORT](#). Usado no python. Não está em livros ainda, mas está na Wikipedia e no código do python, super-explicado.

k -ésimo menor elemento

CLRS 9

k -ésimo menor

Problema: Encontrar o k -ésimo menor elemento de $A[1..n]$.

Suponha $A[1..n]$ sem elementos repetidos.

Exemplo: 33 é o 4º menor elemento de:

1									10
22	99	32	88	34	33	11	97	55	66

A

1			4						10
11	22	32	33	34	55	66	88	97	99

ordenado

Mediana

Mediana é o $\lfloor \frac{n+1}{2} \rfloor$ -ésimo menor ou o $\lceil \frac{n+1}{2} \rceil$ -ésimo menor elemento.

Exemplo: a mediana é 34 ou 55:

1									10
22	99	32	88	34	33	11	97	55	66

A

1				5	6				10
11	22	32	33	34	55	66	88	97	99

ordenado

k -ésimo menor

Recebe $A[1..n]$ e k tal que $1 \leq k \leq n$
e devolve valor do k -ésimo menor elemento de $A[1..n]$.

```
SELECT-ORD ( $A, n, k$ )  
1  ORDENE ( $A, n$ )  
2  devolva  $A[k]$ 
```

O consumo de tempo do **SELECT-ORD** é $\Theta(n \lg n)$.

k -ésimo menor

Recebe $A[1..n]$ e k tal que $1 \leq k \leq n$
e devolve valor do k -ésimo menor elemento de $A[1..n]$.

```
SELECT-ORD ( $A, n, k$ )  
1  ORDENE ( $A, n$ )  
2  devolva  $A[k]$ 
```

O consumo de tempo do **SELECT-ORD** é $\Theta(n \lg n)$.

Dá para fazer melhor?

Menor

Recebe um vetor $A[1..n]$ e devolve o valor do **menor** elemento.

MENOR (A, n)

- 1 **menor** $\leftarrow A[1]$
- 2 para $k \leftarrow 2$ até n faça
- 3 se $A[k] < \text{menor}$
- 4 então **menor** $\leftarrow A[k]$
- 5 devolva **menor**

O consumo de tempo do algoritmo **MENOR** é $\Theta(n)$.

Segundo menor

Recebe um vetor $A[1..n]$ e devolve o valor do **segundo menor** elemento, supondo $n \geq 2$.

SEG-MENOR (A, n)

```
1  menor ← min{A[1], A[2]}  segmenor ← max{A[1], A[2]}
2  para k ← 3 até n faça
3      se A[k] < menor
4          então segmenor ← menor
5              menor ← A[k]
6      senão se A[k] < segmenor
7          então segmenor ← A[k]
8  devolva segmenor
```

O consumo de tempo do SEG-MENOR é $\Theta(n)$.

Algoritmo linear?

Será que conseguimos fazer um **algoritmo linear**
para a mediana?
para o k -ésimo menor?

Algoritmo linear?

Será que conseguimos fazer um **algoritmo linear**
para a mediana?
para o k -ésimo menor?

Sim!

Usaremos o **PARTICIONE** do **QUICKSORT!**

Select aleatorizado

PARTICIONE-ALEA(A, p, r)

- 1 $k \leftarrow \text{RANDOM}(p, r)$
- 2 $A[k] \leftrightarrow A[r]$
- 3 devolva PARTICIONE(A, p, r)

SELECT-ALEA(A, p, r, k)

- 1 se $p = r$ então devolva $A[p]$
- 2 $q \leftarrow \text{PARTICIONE-ALEA}(A, p, r)$
- 3 se $k = q - p + 1$
- 4 então devolva $A[q]$
- 5 se $k < q - p + 1$
- 6 então devolva SELECT-ALEA($A, p, q - 1, k$)
- 7 senão devolva SELECT-ALEA($A, q + 1, r, k - (q - p + 1)$)

Select aleatorizado

PARTICIONE-ALEA(A, p, r)

- 1 $k \leftarrow \text{RANDOM}(p, r)$
- 2 $A[k] \leftrightarrow A[r]$
- 3 devolva PARTICIONE(A, p, r)

SELECT-ALEA(A, p, r, k)

- 1 se $p = r$ então devolva $A[p]$
- 2 $q \leftarrow \text{PARTICIONE-ALEA}(A, p, r)$
- 3 se $k = q - p + 1$
- 4 então devolva $A[q]$
- 5 se $k < q - p + 1$
- 6 então devolva SELECT-ALEA($A, p, q - 1, k$)
- 7 senão devolva SELECT-ALEA($A, q + 1, r, k - (q - p + 1)$)

Consumo esperado de tempo?

Quebrar em fases

KT 13.5

Uma **fase** é uma sucessão de chamadas recursivas

Quebrar em fases

KT 13.5

Uma **fase** é uma sucessão de chamadas recursivas a fase muda quando o tamanho do vetor cai de pelo menos $3/4$

Quebrar em fases

KT 13.5

Uma **fase** é uma sucessão de chamadas recursivas a fase muda quando o tamanho do vetor cai de pelo menos $3/4$

Na fase k , o tamanho do vetor é $\leq \left(\frac{3}{4}\right)^k n$

Quebrar em fases

KT 13.5

Uma **fase** é uma sucessão de chamadas recursivas a fase muda quando o tamanho do vetor cai de pelo menos $3/4$

Na fase k , o tamanho do vetor é $\leq \left(\frac{3}{4}\right)^k n$

Se X_k é o número de chamadas da fase k , vale

Quebrar em fases

KT 13.5

Uma **fase** é uma sucessão de chamadas recursivas a fase muda quando o tamanho do vetor cai de pelo menos $3/4$

Na fase k , o tamanho do vetor é $\leq \left(\frac{3}{4}\right)^k n$

Se X_k é o número de chamadas da fase k , vale

$$X \leq \sum_{k \geq 0} X_k \left(\frac{3}{4}\right)^k n$$

Quebrar em fases

KT 13.5

Uma **fase** é uma sucessão de chamadas recursivas a fase muda quando o tamanho do vetor cai de pelo menos 3/4

Na fase k , o tamanho do vetor é $\leq \left(\frac{3}{4}\right)^k n$

Se X_k é o número de chamadas da fase k , vale

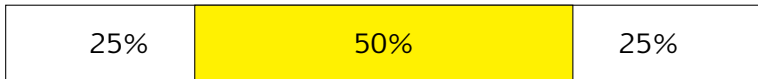
$$X \leq \sum_{k \geq 0} X_k \left(\frac{3}{4}\right)^k n$$

Logo,

$$E(X) \leq n \sum_{k \geq 0} E(X_k) \left(\frac{3}{4}\right)^k$$

Com pivô sortudo

Suponha que cada pivô caia na faixa boa



Com pivô sortudo

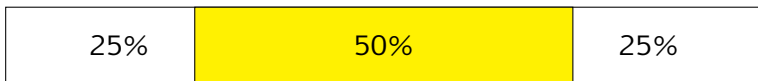
Suponha que cada pivô caia na faixa boa



a chamada recursiva faz mudar de fase.

Com pivô sortudo

Suponha que cada pivô caia na faixa boa



a chamada recursiva faz mudar de fase.

Difícil dar sorte sempre, mas quanto tempo precisa esperar?

Um pouco mais de probabilidade

Considere uma sequência de sorteios de **BOM** ou **MAU**, onde a probabilidade de **BOM** é $p > 0$ a cada sorteio.

quantos sorteios em média para aparecer o primeiro **BOM**?

Um pouco mais de probabilidade

Considere uma sequência de sorteios de **BOM** ou **MAU**, onde a probabilidade de **BOM** é $p > 0$ a cada sorteio.

quantos sorteios em média para aparecer o primeiro **BOM**?

X = instante do primeiro sorteio favorável

Um pouco mais de probabilidade

Considere uma sequência de sorteios de BOM ou MAU, onde a probabilidade de BOM é $p > 0$ a cada sorteio.

quantos sorteios em média para aparecer o primeiro BOM?

X = instante do primeiro sorteio favorável

$$\Pr[X = j] = (1 - p)^{j-1} p$$

Um pouco mais de probabilidade

Considere uma sequência de sorteios de BOM ou MAU, onde a probabilidade de BOM é $p > 0$ a cada sorteio.

quantos sorteios em média para aparecer o primeiro BOM?

X = instante do primeiro sorteio favorável

$$\Pr[X = j] = (1 - p)^{j-1} p$$

$$E(X) = \sum_{j \geq 1} j \Pr[X = j] = \sum_{j \geq 1} j (1 - p)^{j-1} p$$

Um pouco mais de probabilidade

Considere uma sequência de sorteios de **BOM** ou **MAU**, onde a probabilidade de **BOM** é $p > 0$ a cada sorteio.

quantos sorteios em média para aparecer o primeiro **BOM**?

X = instante do primeiro sorteio favorável

$$\Pr[X = j] = (1 - p)^{j-1} p$$

$$E(X) = \sum_{j \geq 1} j \Pr[X = j] = \sum_{j \geq 1} j (1 - p)^{j-1} p$$

$$\leq p \sum_{j=1}^{\infty} j (1 - p)^{j-1} = \frac{1}{p}$$

Terminando

Para um pivô cair na faixa boa, $p = 1/2$. Logo, o número esperado de sorteios para que o tamanho do vetor caia por $3/4$ é ≤ 2 .

Terminando

Para um pivô cair na faixa boa, $p = 1/2$. Logo, o número esperado de sorteios para que o tamanho do vetor caia por $3/4$ é ≤ 2 .

Da análise anterior,

$$\begin{aligned} E(X) &\leq n \sum_{k \geq 0} E(X_k) \left(\frac{3}{4}\right)^k \\ &\leq 2n \sum_{k \geq 0} \left(\frac{3}{4}\right)^k \\ &\leq 8n \end{aligned}$$

Conclusões

O consumo de tempo esperado do algoritmo **SELECT-ALEA**
é $O(n)$.

Próxima aula

CLRS Cap 6 Heapsort