Quicksort e Select Aleatorizados

CLRS Secs 7.3, 7.4 e 9.2

Relembremos o Particione

Rearranja $A[p\,..\,r)$ de modo que $p \le q < r$ e
$A[p\,..\,q) \le A[q] < A(q\,..\,r)$

PARTICIONE (A, p, r)
$x \leftarrow A[r-1]$ $\quad \triangleright$ x é o "pivô"
$i \leftarrow p$
para $j \leftarrow p$ **até** $r-2$ **faça**
  se $A[j] \le x$
    então $A[i] \leftrightarrow A[j]$
      $i \leftarrow i+1$
$A[i] \leftrightarrow A[r-1]$ $\quad \triangleright$ troca com o "pivô"
devolva i

Invariantes:

(i0) $A[p\,..\,i) \le x$ (i1) $x < A[i\,..\,j)$ (i2) $A[r-1] = x$

Relembremos o Particione

Rearranja $A[p\,..\,r)$ de modo que $p \le q < r$ e
$A[p\,..\,q) \le A[q] < A(q\,..\,r)$

```
PARTICIONE (A, p, r)
x ← A[r−1]                ▷ x é o "pivô"
i ← p
para j ← p até r − 2 faça
    se A[j] ≤ x
        então A[i] ↔ A[j]
              i ← i + 1
A[i] ↔ A[r − 1]           ▷ troca com o "pivô"
devolva i
```

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

Quicksort aleatorizado

PARTICIONE-ALEA(A, p, r)

$i \leftarrow$ RANDOM (p, r) $\quad\triangleright$ i uniforme em $[p\,.\,.\,r)$

$A[i] \leftrightarrow A[r-1]$

devolva PARTICIONE (A, p, r)

Quicksort aleatorizado

PARTICIONE-ALEA(A, p, r)

$i \leftarrow$ RANDOM (p, r) $\quad \triangleright$ i uniforme em $[p\,..\,r)$

$A[i] \leftrightarrow A[r-1]$

devolva PARTICIONE (A, p, r)

QUICKSORT-ALEA (A, p, r)

se $r - p > 1$

  então $q \leftarrow$ PARTICIONE-ALEA (A, p, r)

    QUICKSORT-ALEA (A, p, q)

    QUICKSORT-ALEA $(A, q+1, r)$

Quicksort aleatorizado

PARTICIONE-ALEA(A, p, r)

$i \leftarrow$ RANDOM (p, r) $\quad \triangleright$ i uniforme em $[p\,..\,r)$

$A[i] \leftrightarrow A[r-1]$

devolva PARTICIONE (A, p, r)

QUICKSORT-ALEA (A, p, r)

se $r - p > 1$

  então $q \leftarrow$ PARTICIONE-ALEA (A, p, r)

    QUICKSORT-ALEA (A, p, q)

    QUICKSORT-ALEA $(A, q+1, r)$

Consumo esperado de tempo para um vetor A arbitrário?

Quicksort aleatorizado

```
PARTICIONE-ALEA(A, p, r)
    i ← RANDOM (p, r)        ▷ i uniforme em [p . . r)
    A[i] ↔ A[r − 1]
    devolva PARTICIONE (A, p, r)
```

```
QUICKSORT-ALEA (A, p, r)
    se r − p > 1
        então q ← PARTICIONE-ALEA (A, p, r)
              QUICKSORT-ALEA (A, p, q)
              QUICKSORT-ALEA (A, q + 1, r)
```

Consumo esperado de tempo para um vetor A arbitrário?

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

Consumo de tempo esperado

Suponha $A[0\,.\,.\,n)$ é uma permutação de $\{1, \ldots, n\}$.

$$X_{ab} \quad = \quad \text{número de comparações entre } a \text{ e } b \text{ na linha 4 do PARTICIONE do QUICKSORT-ALEA;}$$

Queremos calcular

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

Consumo de tempo esperado

Supondo $a < b$,

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

Qual é a probabilidade de X_{ab} valer 1?

Consumo de tempo esperado

Supondo $a < b$,

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

Qual é a probabilidade de X_{ab} valer 1?

$$\mathbb{P}(X_{ab}{=}1) \; = \; \frac{1}{b-a+1} + \frac{1}{b-a+1} \; = \; \mathbb{E}[X_{ab}]$$

Consumo de tempo esperado

Supondo $a < b$,

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

Qual é a probabilidade de X_{ab} valer 1?

$$\mathbb{P}(X_{ab}{=}1) \;=\; \frac{1}{b-a+1} + \frac{1}{b-a+1} \;=\; \mathbb{E}[X_{ab}]$$

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

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

Consumo de tempo esperado

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

CLRS (A.7), p.1060

Conclusões

O consumo de tempo esperado
do algoritmo QUICKSORT-ALEA é $\mathrm{O}(n \log n)$.

Do exercício 7.4-4 do CLRS temos que

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

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

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

Exemplo: as medianas são 34 e 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** $A[k]$

SELECT-ORD pode ser implementado para consumir tempo $\mathrm{O}(n \lg n)$.

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** $A[k]$

SELECT-ORD pode ser implementado para consumir tempo $\mathrm{O}(n \lg n)$.

Dá para fazer melhor?

Menor

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

MENOR (A, n)
```
menor ← A[1]
para k ← 2 até n faça
    se A[k] < menor
        então menor ← A[k]
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)

$\text{menor} \leftarrow \min\{A[1], A[2]\}$
$\text{segmenor} \leftarrow \max\{A[1], A[2]\}$
para $k \leftarrow 3$ **até** n **faça**
    se $A[k] < \text{menor}$
        então $\text{segmenor} \leftarrow \text{menor}$
            $\text{menor} \leftarrow A[k]$
    senão se $A[k] < \text{segmenor}$
        então $\text{segmenor} \leftarrow A[k]$
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)

$k \leftarrow$ RANDOM (p, r) $\quad \triangleright$ k uniforme em $[p \,..\, r)$

$A[k] \leftrightarrow A[r-1]$

devolva PARTICIONE (A, p, r)

SELECT-ALEA (A, p, r, k) $\quad \triangleright$ devolve k-ésimo menor de $A[p \,..\, r)$

se $r - p = 1$ **então devolva** $A[p]$

$q \leftarrow$ PARTICIONE-ALEA (A, p, r)

$\text{num}_{\leq}\text{pivô} \leftarrow q - p + 1$ $\quad \triangleright$ nº de elementos em $A[p \,..\, q]$

se $k = \text{num}_{\leq}\text{pivô}$

  então devolva $A[q]$

se $k < \text{num}_{\leq}\text{pivô}$

  então devolva SELECT-ALEA (A, p, q, k)

  senão devolva SELECT-ALEA $(A, q{+}1, r, k{-}\text{num}_{\leq}\text{pivô})$

Select aleatorizado

PARTICIONE-ALEA(A, p, r)
$k \leftarrow$ RANDOM(p, r) ▷ k uniforme em $[p \,..\, r)$
$A[k] \leftrightarrow A[r-1]$
devolva PARTICIONE(A, p, r)

SELECT-ALEA(A, p, r, k) ▷ devolve k-ésimo menor de $A[p \,..\, r)$
se $r - p = 1$ **então devolva** $A[p]$
$q \leftarrow$ PARTICIONE-ALEA(A, p, r)
$\mathrm{num}_{\leq}\mathrm{piv\hat{o}} \leftarrow q - p + 1$ ▷ nº de elementos em $A[p \,..\, q]$
se $k = \mathrm{num}_{\leq}\mathrm{piv\hat{o}}$
  então devolva $A[q]$
se $k < \mathrm{num}_{\leq}\mathrm{piv\hat{o}}$
  então devolva SELECT-ALEA(A, p, q, k)
  senão devolva SELECT-ALEA$(A, q{+}1, r, k{-}\mathrm{num}_{\leq}\mathrm{piv\hat{o}})$

Consumo esperado de tempo?

Consumo de tempo esperado

Suponha $A[0\,.\,.\,n)$ permutação de $\{1,\ldots,n\}$.

$$X_{ab} \quad = \quad \text{número de comparações entre } a \text{ e } b \text{ na linha 4 do PARTICIONE do SELECT-ALEA.}$$

Observe que X_{ab} não é a mesma de antes.

De novo, queremos calcular

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

Consumo de tempo esperado

Vamos supor que $k = n$.
Supondo $a < b$,

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

Qual é a probabilidade de X_{ab} valer 1?

Consumo de tempo esperado

Vamos supor que $k = n$.
Supondo $a < b$,

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

Qual é a probabilidade de X_{ab} valer 1?

$$\mathbb{P}(X_{ab}{=}1) = \frac{1}{n-a+1} + \frac{1}{n-a+1} = \mathbb{E}[X_{ab}]$$

Consumo de tempo esperado

Vamos supor que $k = n$.
Supondo $a < b$,

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

Qual é a probabilidade de X_{ab} valer 1?

$$\mathbb{P}(X_{ab}=1) = \frac{1}{n-a+1} + \frac{1}{n-a+1} = \mathbb{E}[X_{ab}]$$

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

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

Consumo de tempo esperado

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

Consumo de tempo esperado

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

Observação: É possível estender essa análise para um k arbitrário.

« Conference on Topological and Geometric Graph Theory The photon/XOR system »

Blum-style analysis of Quickselect

Oct 9, 2007

So you're all familiar with ~~Avrim Blum's~~ Motwani and Raghavan's (?) slick analysis of quicksort in CLRS, avoiding probabilistic recurrences and easily getting the correct constant in the leading term, right? (If not, I review it below.) What CLRS doesn't tell you is that the same analysis works almost as well for quickselect.

Quicksort

Randomized quicksort is a recursive algorithm that, in each recursive call chooses a "pivot" element uniformly at random from the values given as input to the call, partitions the data into subsets that are less than, equal to, and greater than the pivot, and recursively sorts the subsets that are less and greater. With some care the partition can be done with one comparison of each data item to the pivot:

```
def quicksort(L):
    if len(L) < 2: return L
    pivot_pos = random integer in range 0 .. len(L)-1
    x = L[pivot_pos]
    parts = [[],[x],[]]
    for y in L[:pivot_pos] + L[pivot_pos+1:]:
        parts[cmp(y,x)+1].append(y)
    return quicksort(parts[0]) + parts[1] + quicksort(parts[2])
```

An equivalent view of the algorithm (though not how one would usually program it) is that, before the recursion starts, we pick a random permutation σ; then, in each recursive call, we pick the pivot to be the value among the inputs given to that call that's earliest in σ. It's not hard to see that this gives the same uniform probability that any pivot is chosen. What we want to count is the number of calls to cmp over the course of the algorithm.

The expected number of comparisons is (by linearity of expectation) the same as the sum, over pairs of data values, of the probability that the algorithm compares that pair. If x_i denotes the value in position i of the sorted output array, and $i < j$, then x_i and x_j are compared exactly when one of these two values is the earliest in σ in the range of values $x_i, x_{i+1}, \ldots, x_{j-1}, x_j$. For, until a pivot within this range is chosen, x_i and x_j will stay in the same subproblem as each other, but after such a pivot is chosen, they will be separated. Each item in this range is equally likely to be the first one chosen as pivot, so the probability that the first in this range is one of the two endpoints is exactly $\frac{2}{j-i+1}$. Thus, using this expression as the probability that a pair is compared, the expected number of comparisons is (using the known logarithmic evaluation of harmonic numbers)

$$\sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} \frac{2}{j-i+1} < 2nH_n = 2n\ln n + O(n).$$

Quickselect

Quickselect is a variant of quicksort that finds the kth smallest of a set of items by using the same partition strategy as quicksort but then only recursing within one of the subsets of the partition, the one containing the desired value:

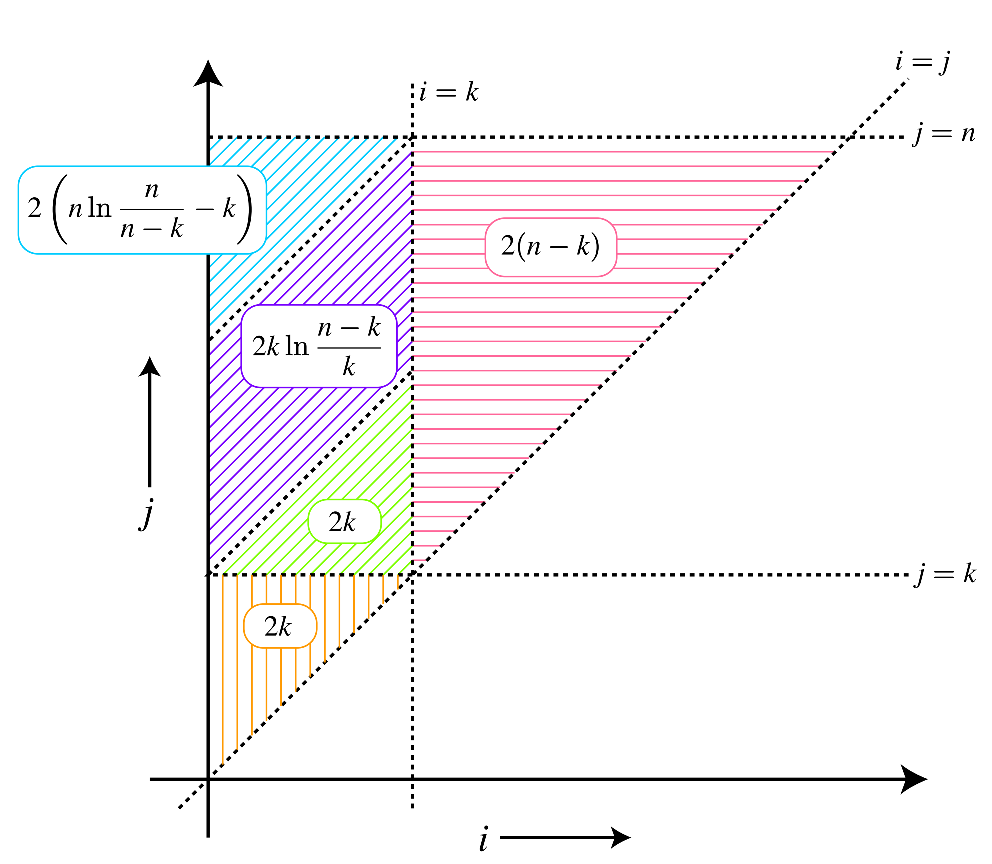
```
def quickselect(L,k):
    if len(L) < 2: return L[k]
    pivot_pos = random integer in range 0 .. len(L)-1
    x = L[pivot_pos]
    parts = [[],[x],[]]
    for y in L[:pivot_pos] + L[pivot_pos+1:]:
        parts[cmp(y,x)+1].append(y)
    if k < len(parts[0]):
        return quickselect(parts[0],k)
    if k >= len(parts[0]) + len(parts[1]):
        return quickselect(parts[2],k - len(parts[0]) - len(parts[1]))
    return x
```

The results of calling this should be the same as calling quicksort on L and then indexing the kth entry in the sorted array.

Pretty much the same analysis as before goes through, changed only in the last part. The expected number of comparisons is the same as the sum, over pairs of data values, of the probability that the algorithm compares that pair. Values x_i and x_j are compared exactly when one of these two values is the earliest in σ in a certain range of values, but now the range is bigger (making the probability of comparison smaller): it's the minimal consecutive range of values containing x_i, x_j, and x_k. For, if a pivot is chosen between x_i and x_j then they are placed in separate subproblems and not compared, while if a pivot is chosen between both of these values and x_k then both values are eliminated and not compared. Thus, we need only replace the probability $\frac{2}{j-i+1}$ in the quicksort analysis by the slightly more complicated formula

$$\frac{2}{\max(j-i+1, j-k+1, k-i+1)},$$

sum over all pairs, and we're done. The following diagram shows graphically how the sum decomposes the (i, j) plane into regions, for $k < n/2$ (the case for larger k is symmetric); along the colored lines within each region the probability given by the formula is constant.



- Within the tangerine-colored triangle on the lower left, i and j are both less than k. The qth column within the triangle, through the vertical line $i = k - q$, has approximately q entries within it, each of which is approximately $2/q$, and there are k columns, so the sum within this triangle is (ignoring lower order terms) $2k$.
- Within the salmon-colored triangle on the upper right, i and j are both greater than k. The qth row within the triangle, through the horizontal line $k = k + q$, has approximately q entries within it, each of which is approximately $2/q$, and there are $n - k$ columns, so the sum within this triangle is (ignoring lower order terms) $2(n - k)$.
- The remaining terms in the sum have i less than k and j greater than k, but it is convenient to break them down further into three smaller regions. Within the avocado-colored triangle above and to the left of the point (k, k), the distance between i and j is less than k. The qth diagonal within this triangle has approximately q entries within it, each of which is approximately $2/q$, and there are k diagonals, so the sum within this triangle is (ignoring lower order terms) $2k$.
- Within the lilac-colored parallelogram, each diagonal has k equal terms, each of the form $2/q$ for some q. If the same pattern of terms continued down to the main diagonal, the sum of these terms would be $2k$ times a harmonic number; instead, we get $2k$ times the difference of two harmonic numbers: $2k(\ln(n - k) - \ln k)$.
- Within the turqoise-colored triangle at the upper right, the diagonal $j - i = q$ has approximately $n - q$ terms, each approximately $2/q$. The sum of $2(n - q)/q$, for q ranging from $n - k$ to n, can be broken into two parts, the sum of $2n/q$ and the sum of $-2q/q$; the latter sum is simply $-2k$. The first sum is again (factoring out the constant numerator) the difference of two harmonic numbers, so the total is $2(n\ln n - n\ln(n - k) - k$.

Adding all of these parts of the sum together, we find that the total expected number of comparisons made by quickselect is

$$\left(2n\left(1 + \ln\frac{n}{n-k}\right) + 2k\ln\frac{n-k}{k}\right)(1 + o(1)).$$

The worst case happens when $k = n/2$, for which the number of comparisons can be simplified to

$$2n(1 + \ln 2 + o(1)) \le 3.3863n + o(n).$$

A simplified analysis, perhaps more suitable for an undergraduate class, would be to expand the upper left rectangle into a larger triangle, with n diagonals each contributing less than 2 to the total; this sloppier analysis shows more easily that the number of comparisons is at most $4n$.

Conclusões

$\boxed{\text{O consumo de tempo esperado do algoritmo SELECT-ALEA é } O(n).}$

Conclusões

O consumo de tempo esperado
do algoritmo SELECT-ALEA é $\mathrm{O}(n)$.

Será que existe algoritmo de ordenação
cujo consumo de tempo é melhor que $\mathrm{O}(n \lg n)$?

Conclusões

O consumo de tempo esperado
do algoritmo SELECT-ALEA é $\mathrm{O}(n)$.

Será que existe algoritmo de ordenação
cujo consumo de tempo é melhor que $\mathrm{O}(n \lg n)$?

Por exemplo,
será que existe algoritmo de ordenação linear?

Ordenação: limite inferior

Problema: Rearranjar um vetor $A[1\,..\,n]$ de modo que ele fique em ordem crescente.

Existem algoritmos que consomem tempo $\mathrm{O}(n \lg n)$.

Ordenação: limite inferior

Problema: Rearranjar um vetor $A[1\,..\,n]$ de modo que ele fique em ordem crescente.

Existem algoritmos que consomem tempo $\mathrm{O}(n \lg n)$.

Existe algoritmo assintoticamente melhor?

Ordenação: limite inferior

Problema: Rearranjar um vetor $A[1\,..\,n]$ de modo que ele fique em ordem crescente.

Existem algoritmos que consomem tempo $\mathrm{O}(n \lg n)$.

Existe algoritmo assintoticamente melhor?

NÃO, se o algoritmo é baseado em comparações.

Prova?

Ordenação: limite inferior

Problema: Rearranjar um vetor $A[1\,..\,n]$ de modo que ele fique em ordem crescente.

Existem algoritmos que consomem tempo $\mathrm{O}(n \lg n)$.

Existe algoritmo assintoticamente melhor?

NÃO, se o algoritmo é baseado em comparações.

Prova?

Qualquer algoritmo baseado em comparações
é uma "árvore de decisão".

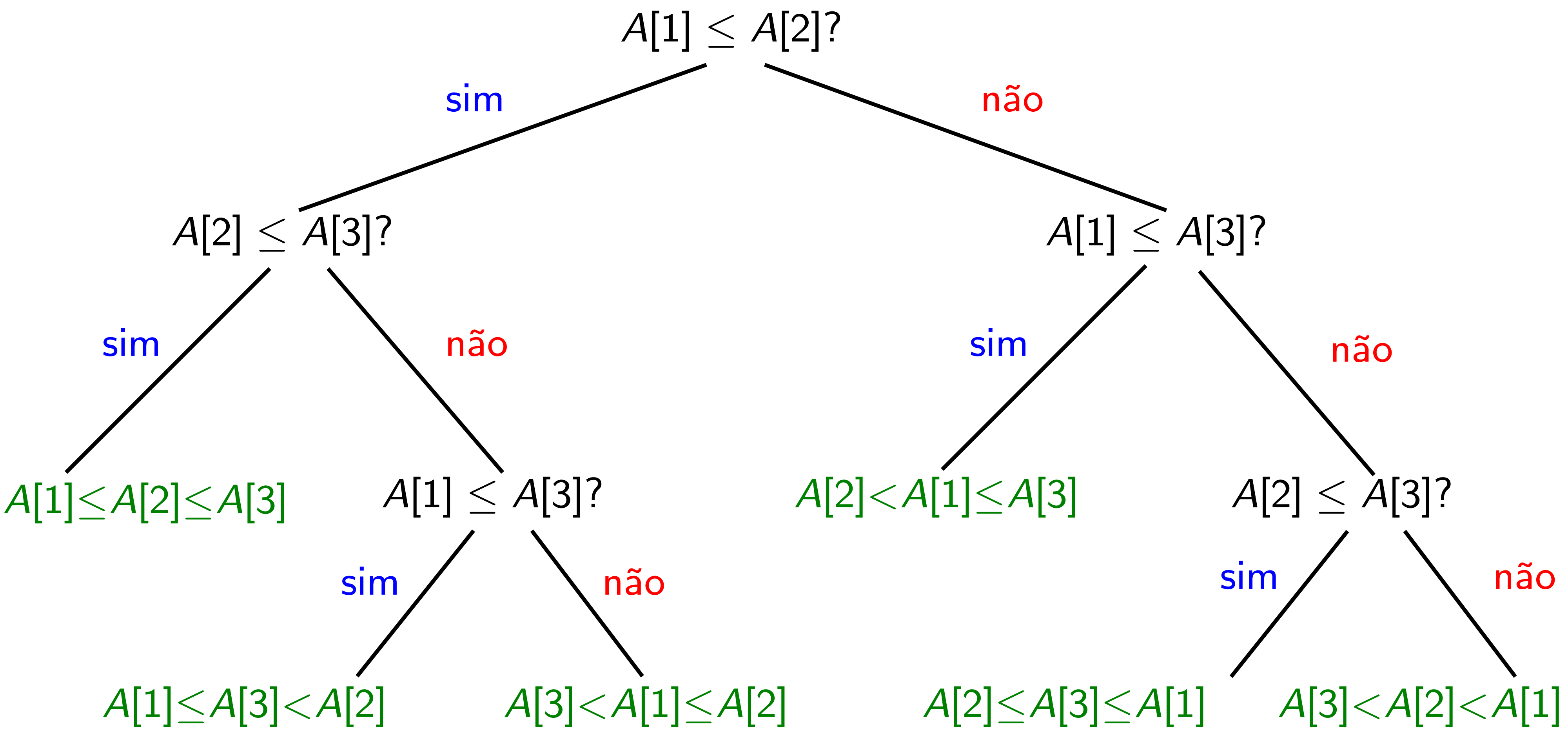
Ordenação por inserção

ORDENA-POR-INSERÇÃO (A, n)

```
para j ← 2 até n faça
    chave ← A[j]
    k ← j − 1
    enquanto k ≥ 1 e A[k] > chave faça
        A[k + 1] ← A[k]      ▷ desloca
        k ← k − 1
    A[k + 1] ← chave      ▷ insere
```

Exemplo

ORDENA-POR-INSERÇÃO $(A[1\,..\,3])$:



Limite inferior

Considere uma árvore de decisão para $A[1\,..\,n]$.

Limite inferior

Considere uma árvore de decisão para $A[1\,..\,n]$.

Número de comparações, no pior caso?

Limite inferior

Considere uma árvore de decisão para $A[1\,..\,n]$.

Número de comparações, no pior caso?
Resposta: altura, h, da árvore de decisão.

Limite inferior

Considere uma árvore de decisão para $A[1\,..\,n]$.

Número de comparações, no pior caso?
Resposta: altura, h, da árvore de decisão.

Todas as $n!$ permutações de $1, \ldots, n$ devem ser folhas.

Limite inferior

Considere uma árvore de decisão para $A[1\,..\,n]$.

Número de comparações, no pior caso?
Resposta: altura, h, da árvore de decisão.

Todas as $n!$ permutações de $1, \ldots, n$ devem ser folhas.

Toda árvore binária de altura h tem no máximo 2^h folhas.

Limite inferior

Considere uma árvore de decisão para $A[1\,.\,.\,n]$.

Número de comparações, no pior caso?
Resposta: altura, h, da árvore de decisão.

Todas as $n!$ permutações de $1, \ldots, n$ devem ser folhas.

Toda árvore binária de altura h tem no máximo 2^h folhas.

Prova: Por indução em h. A afirmação vale para $h = 0$.

Limite inferior

Considere uma árvore de decisão para $A[1\,..\,n]$.

Número de comparações, no pior caso?
Resposta: altura, h, da árvore de decisão.

Todas as $n!$ permutações de $1, \ldots, n$ devem ser folhas.

Toda árvore binária de altura h tem no máximo 2^h folhas.

Prova: Por indução em h. A afirmação vale para $h = 0$.
Seja $h \geq 1$ e suponha que a afirmação vale
para toda árvore binária de altura menor que h.

Limite inferior

Considere uma árvore de decisão para $A[1\mathinner{\ldotp\ldotp} n]$.

Número de comparações, no pior caso?
Resposta: altura, h, da árvore de decisão.

Todas as $n!$ permutações de $1, \ldots, n$ devem ser folhas.

Toda árvore binária de altura h tem no máximo 2^h folhas.

Prova: Por indução em h. A afirmação vale para $h = 0$.
Seja $h \geq 1$ e suponha que a afirmação vale
para toda árvore binária de altura menor que h.
Número de folhas de árvore de altura h é a soma do
número de folhas das subárvores (da raiz), que têm altura $\leq h - 1$.

Limite inferior

Considere uma árvore de decisão para $A[1\,..\,n]$.

Número de comparações, no pior caso?
Resposta: altura, h, da árvore de decisão.

Todas as $n!$ permutações de $1, \ldots, n$ devem ser folhas.

Toda árvore binária de altura h tem no máximo 2^h folhas.

Prova: Por indução em h. A afirmação vale para $h = 0$.
Suponha que a afirmação vale
para toda árvore binária de altura menor que h, para $h \geq 1$.
Número de folhas de árvore de altura h é a soma do
número de folhas das subárvores (da raiz), que têm altura $\leq h - 1$.
Logo, o número de folhas de uma árvore de altura h é

$$\leq \; 2 \times 2^{h-1} \; = \; 2^h.$$

Limite inferior

Assim, devemos ter $2^h \geq n!$, donde $h \geq \lg(n!)$.

$$(n!)^2 \; = \left(\prod_{i=0}^{n-1} (n-i) \right) \left(\prod_{i=1}^{n} i \right)$$

Limite inferior

Assim, devemos ter $2^h \geq n!$, donde $h \geq \lg(n!)$.

$$(n!)^2 = \left(\prod_{i=0}^{n-1}(n-i)\right)\left(\prod_{i=1}^{n} i\right) = \left(\prod_{i=0}^{n-1}(n-i)\right)\left(\prod_{i=0}^{n-1}(i+1)\right)$$

Limite inferior

Assim, devemos ter $2^h \geq n!$, donde $h \geq \lg(n!)$.

$$(n!)^2 = \left(\prod_{i=0}^{n-1}(n-i)\right)\left(\prod_{i=0}^{n-1}(i+1)\right) = \prod_{i=0}^{n-1}(n-i)(i+1)$$

Limite inferior

Assim, devemos ter $2^h \geq n!$, donde $h \geq \lg(n!)$.

$$(n!)^2 \;=\; \prod_{i=0}^{n-1} (n-i)(i+1) \;=\; \prod_{i=0}^{n-1} \Big(n + i(n-1-i)\Big) \;\geq\; \prod_{i=0}^{n-1} n \;=\; n^n$$

Limite inferior

Assim, devemos ter $2^h \geq n!$, donde $h \geq \lg(n!)$.

$$(n!)^2 = \prod_{i=0}^{n-1} (n-i)(i+1) = \prod_{i=0}^{n-1} \Big(n + i(n-1-i)\Big) \geq \prod_{i=0}^{n-1} n = n^n$$

Portanto,

$$h \geq \lg(n!) \geq \frac{1}{2}\, n \lg n.$$

Conclusão

Todo algoritmo de ordenação
baseado em comparações faz

$$\Omega(n \lg n)$$

comparações no pior caso.

Conclusão

Todo algoritmo de ordenação
baseado em comparações faz

$$\Omega(n \lg n)$$

comparações no pior caso.

Aulas que vem:

Algoritmos de ordenação lineares! Como assim???