Quicksort e Select Aleatorizados

CLRS Secs 7.3, 7.4 e 9.2

Relembremos o Particione

Rearranja $A[p\,..\,r]$ de modo que $p \le q \le r$ e
$A[p\,..\,q{-}1] \le A[q] < A[q{+}1\,..\,r]$

```
PARTICIONE (A, p, r)
1  x ← A[r]          ▷ x é o "pivô"
2  i ← p−1
3  para j ← p até r − 1 faça
4      se A[j] ≤ x
5          então i ← i + 1
6                A[i] ↔ A[j]
7  A[i+1] ↔ A[r]
8  devolva i + 1
```

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

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

Relembremos o Particione

Rearranja $A[p\,..\,r]$ de modo que $p \le q \le r$ e
$A[p\,..\,q{-}1] \le A[q] < A[q{+}1\,..\,r]$

```
PARTICIONE (A, p, r)
1  x ← A[r]          ▷ x é o "pivô"
2  i ← p−1
3  para j ← p até r − 1 faça
4      se A[j] ≤ x
5          então i ← i + 1
6                A[i] ↔ A[j]
7  A[i+1] ↔ A[r]
8  devolva i + 1
```

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

Quicksort aleatorizado

```
PARTICIONE-ALEA(A, p, r)
1  i ← RANDOM (p, r)
2  A[i] ↔ A[r]
3  devolva PARTICIONE (A, p, r)
```

Quicksort aleatorizado

```
PARTICIONE-ALEA(A, p, r)
1  i ← RANDOM (p, r)
2  A[i] ↔ A[r]
3  devolva PARTICIONE (A, p, r)

QUICKSORT-ALE (A, p, r)
1  se p < r
2     então q ← PARTICIONE-ALEA (A, p, r)
3           QUICKSORT-ALE (A, p, q − 1)
4           QUICKSORT-ALE (A, q + 1, r)
```

Quicksort aleatorizado

```
PARTICIONE-ALEA(A, p, r)
1  i ← RANDOM (p, r)
2  A[i] ↔ A[r]
3  devolva PARTICIONE (A, p, r)

QUICKSORT-ALE (A, p, r)
1  se p < r
2     então q ← PARTICIONE-ALEA (A, p, r)
3           QUICKSORT-ALE (A, p, q − 1)
4           QUICKSORT-ALE (A, q + 1, r)
```

Consumo esperado de tempo para um vetor A arbitrário?

Quicksort aleatorizado

```
PARTICIONE-ALEA(A, p, r)
1  i ← RANDOM (p, r)
2  A[i] ↔ A[r]
3  devolva PARTICIONE (A, p, r)

QUICKSORT-ALE (A, p, r)
1  se p < r
2      então q ← PARTICIONE-ALEA (A, p, r)
3            QUICKSORT-ALE (A, p, q − 1)
4            QUICKSORT-ALE (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 ← A[r]          ▷ x é o "pivô"
2  i ← p−1
3  para j ← p até r − 1 faça
4      se A[j] ≤ x
5          então i ← i + 1
6                A[i] ↔ A[j]
7  A[i+1] ↔ A[r]
8  devolva i + 1
```

Consumo de tempo esperado

Suponha os elementos de A são distintos.

X_{ab} = número de comparações entre o a-ésimo e o b-ésimo menor na linha 4 do PARTICIONE do QUICKSORT-ALE;

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 $a < b$,

$$X_{ab} = \begin{cases} 1 & \text{se primeiro pivô que é o } i\text{-ésimo menor} \\ & \text{para } i \text{ em } \{a, \ldots, b\} \text{ é o } a\text{-ésimo ou o } b\text{-ésimo menor} \\ 0 & \text{caso contrário} \end{cases}$$

Qual a probabilidade de X_{ab} valer 1?

Consumo de tempo esperado

Supondo $a < b$,

$$X_{ab} = \begin{cases} 1 & \text{se primeiro pivô que é o } i\text{-ésimo menor} \\ & \text{para } i \text{ em } \{a, \ldots, b\} \text{ é o } a\text{-ésimo ou o } b\text{-ésimo menor} \\ 0 & \text{caso contrário} \end{cases}$$

Qual a probabilidade de X_{ab} valer 1?

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

Consumo de tempo esperado

Supondo $a < b$,

$$X_{ab} = \begin{cases} 1 & \text{se primeiro pivô que é o } i\text{-ésimo menor} \\ & \text{para } i \text{ em } \{a, \ldots, b\} \text{ é o } a\text{-ésimo ou o } b\text{-ésimo menor} \\ 0 & \text{caso contrário} \end{cases}$$

Qual a probabilidade de X_{ab} valer 1?

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

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

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

Consumo de tempo esperado

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

CLRS (A.7), p.1060

Conclusões

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

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:

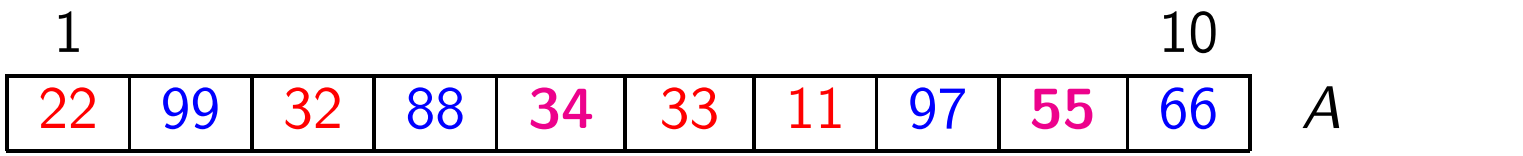
1									10	
22	99	32	88	34	**33**	11	97	55	66	A

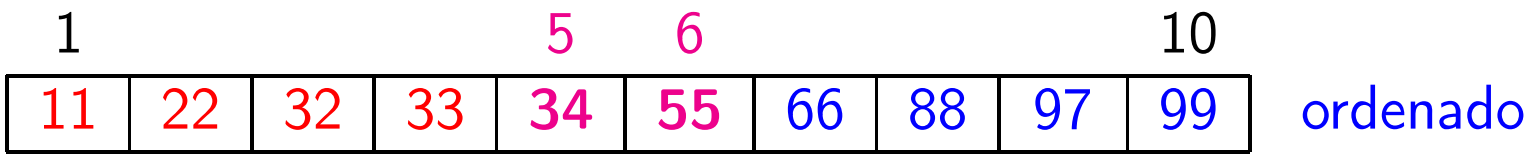
1			4						10	
11	22	32	**33**	34	55	66	88	97	99	ordenado

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:





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 ← A[1]
2   para k ← 2 até n faça
3       se A[k] < menor
4           então menor ← 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]}
2   segmenor ← max{A[1], A[2]}
3   para k ← 3 até n faça
4       se A[k] < menor
5           então segmenor ← menor
6                 menor ← A[k]
7       senão se A[k] < segmenor
8           então segmenor ← A[k]
9   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 ← RANDOM (p, r)
2  A[k] ↔ A[r]
3  devolva PARTICIONE (A, p, r)

SELECT-ALEA (A, p, r, k)
1  se p = r então devolva A[p]
2  q ← 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 ← RANDOM (p, r)
2  A[k] ↔ A[r]
3  devolva PARTICIONE (A, p, r)

SELECT-ALEA (A, p, r, k)
1  se p = r então devolva A[p]
2  q ← 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?

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 SELECT-ALEA.

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

De novo, queremos calcular

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

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 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 a probabilidade de X_{ab} valer 1?

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

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 a probabilidade de X_{ab} valer 1?

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

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

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

Consumo de tempo esperado

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

Consumo de tempo esperado

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

Exercício: Refaça os cálculos para um k arbitrário.

Conclusões

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

Conclusões

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

Será que existe algoritmo de ordenação
cujo consumo de tempo é melhor que $\Theta(n \lg n)$?

Conclusões

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

Será que existe algoritmo de ordenação
cujo consumo de tempo é melhor que $\Theta(n \lg n)$?

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?

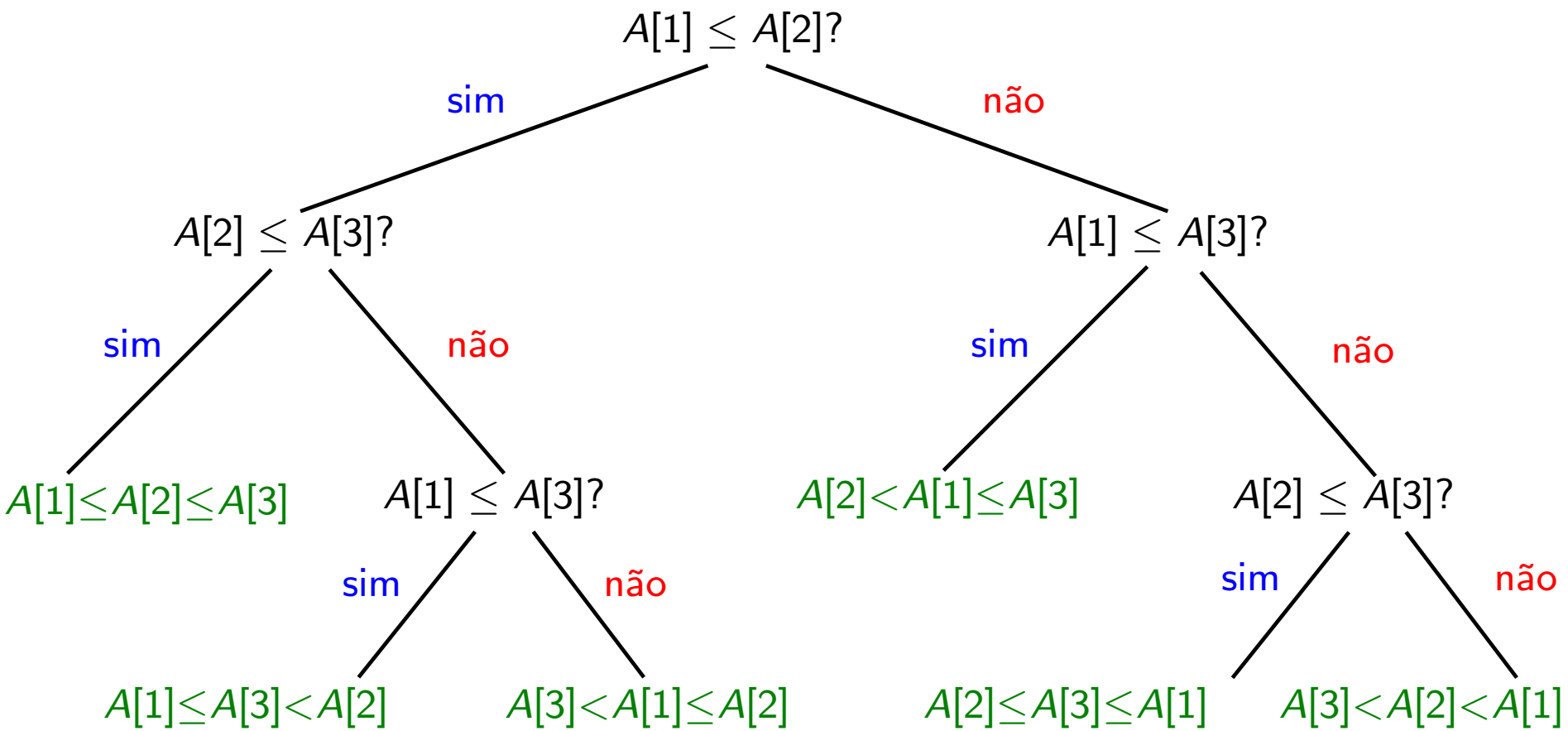
NÃO, se o algoritmo é baseado em comparações.

Prova?

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

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 $h = 0$.
Suponha que a afirmação vale
para toda árvore binária de altura menor que h, para $h \geq 1$.

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, 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, 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 $h \geq \lg(n!)$.

Limite inferior

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

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

Limite inferior

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

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

Portanto,

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

Limite inferior

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

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

Portanto,

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

Alternativamente, a fórmula de Stirling diz que

$$n! \;=\; \sqrt{2\pi\, n}\left(\frac{n}{e}\right)^n \left(1 + \Theta\left(\frac{1}{n}\right)\right).$$

Limite inferior

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

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

Portanto,

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

Alternativamente, a fórmula de Stirling diz que

$$n! \;=\; \sqrt{2\pi\, n}\left(\frac{n}{e}\right)^n \left(1 + \Theta\left(\frac{1}{n}\right)\right).$$

Disso, temos que $h \;\geq\; \lg(n!) \;\geq\; \lg\left(\frac{n}{e}\right)^n \;=\; n(\lg n - \lg e)$.

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.

Aula que vem:

Algoritmos de ordenação lineares! Como assim???