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

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

Quicksort aleatorizado

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

Quicksort aleatorizado

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

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

Quicksort aleatorizado

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

QUICKSORT-ALEA (A, p, r)
1  se r − p > 1
2     então q ← PARTICIONE-ALEA (A, p, r)
3           QUICKSORT-ALEA (A, p, q)
4           QUICKSORT-ALEA (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)    ▷ i uniforme em [p . . r)
2  A[i] ↔ A[r − 1]
3  devolva PARTICIONE (A, p, r)

QUICKSORT-ALEA (A, p, r)
1  se r − p > 1
2     então q ← PARTICIONE-ALEA (A, p, r)
3           QUICKSORT-ALEA (A, p, q)
4           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 ← A[r−1]          ▷ x é o "pivô"
2  i ← p
3  para j ← p até r − 2 faça
4      se A[j] ≤ x
5          então A[i] ↔ A[j]
6                i ← i + 1
7  A[i] ↔ 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} = número de comparações entre a e b
na linha 4 do PARTICIONE do QUICKSORT-ALEA;

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

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?

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

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?

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

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

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

Consumo de tempo esperado

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

CLRS (A.7), p.1060

Conclusões

O consumo de tempo esperado
do algoritmo QUICKSORT-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:

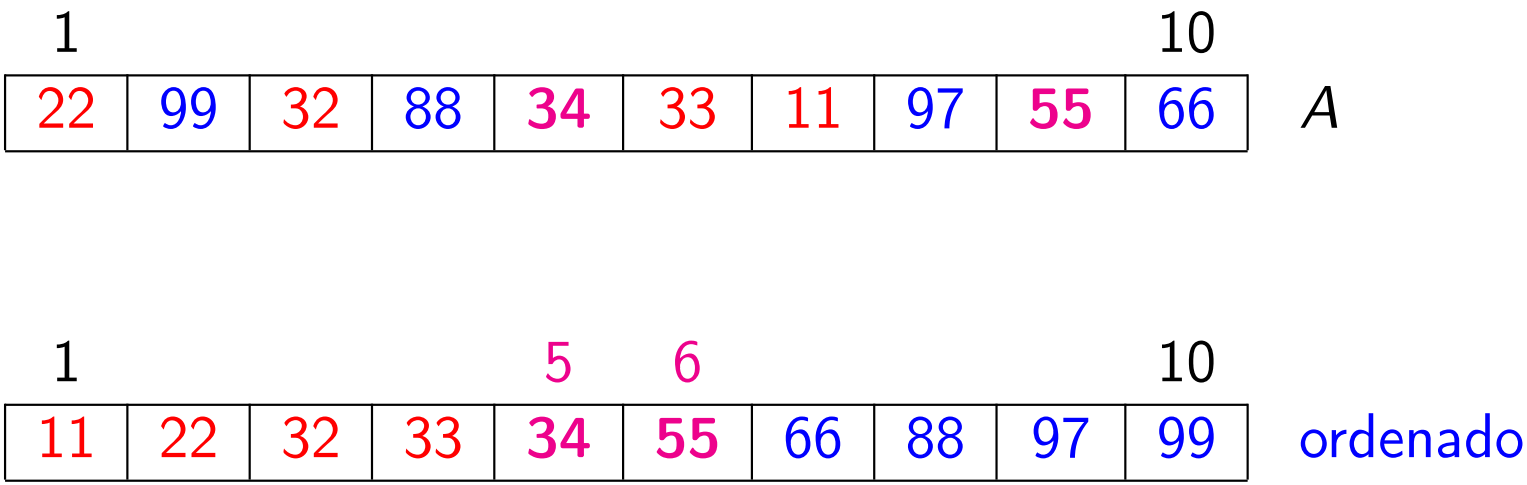
1									10	
22	99	32	88	34	**33**	11	97	55	66	A

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:



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 $O(n \lg n)$.

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 $O(n \lg n)$.

Dá para fazer melhor?

Menor

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

```
MENOR (A, n)
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  t ← RANDOM(p, r)        ▷ t uniforme em [p .. r)
2  A[t] ↔ A[r − 1]
3  devolva PARTICIONE(A, p, r)

SELECT-ALEA (A, p, r, k)    ▷ devolve k-ésimo menor de A[p .. r)
1  se r − p = 1 então devolva A[p]
2  q ← PARTICIONE-ALEA(A, p, r)
3  num≤pivô ← q − p + 1     ▷ nº de elementos em A[p .. q]
4  se k = num≤pivô
5       então devolva A[q]
6  se k < num≤pivô
7       então devolva SELECT-ALEA(A, p, q, k)
8       senão devolva SELECT-ALEA(A, q+1, r, k−num≤pivô)
```

Select aleatorizado

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

SELECT-ALEA (A, p, r, k)    ▷ devolve k-ésimo menor de A[p .. r)
1  se r − p = 1 então devolva A[p]
2  q ← PARTICIONE-ALEA(A, p, r)
3  num_≤pivô ← q − p + 1    ▷ nº de elementos em A[p .. q]
4  se k = num_≤pivô
5      então devolva A[q]
6  se k < num_≤pivô
7      então devolva SELECT-ALEA(A, p, q, k)
8      senão devolva SELECT-ALEA(A, q+1, r, k−num_≤pivô)
```

Consumo esperado de tempo?

Consumo de tempo esperado

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

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 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?

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

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?

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

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

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

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.

Consumo de tempo esperado

Veja Blum-style analysis of Quickselect, por David Eppstein

Conclusões

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 $O(n \lg n)$.

Existe algoritmo assintoticamente melhor?

NÃO, se o algoritmo é baseado em comparações.

Prova?

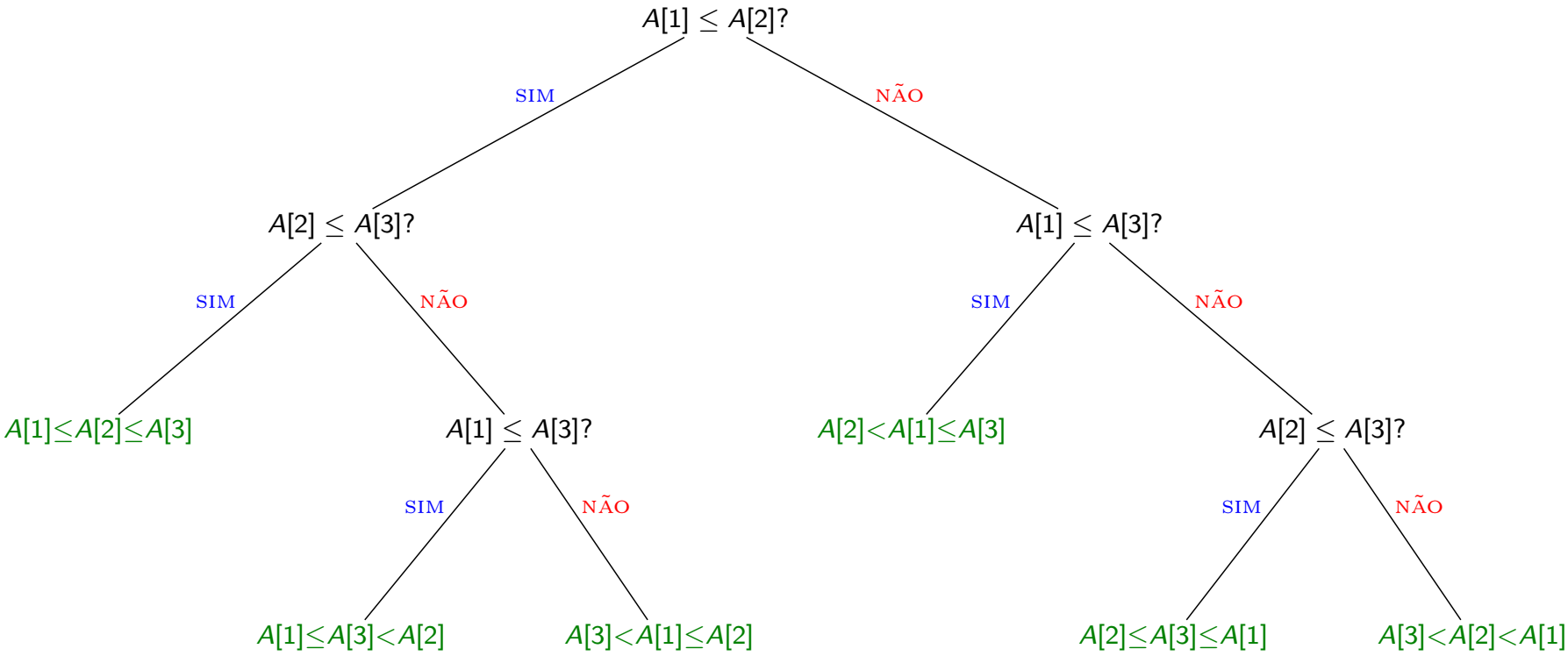
Qualquer algoritmo baseado em comparações é uma "árvore de decisão".

Ordenação por inserção

```
ORDENA-POR-INSERÇÃO (A, n)
1  para j ← 2 até n faça
2      chave ← A[j]
3      k ← j − 1
4      enquanto k ≥ 1 e A[k] > chave faça
5          A[k + 1] ← A[k]      ▷ desloca
6          k ← k − 1
7      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 $A[1\,..\,n]$.

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.

Aula que vem:

Algoritmos de ordenação lineares! Como assim???