Aula 10
Limite inferior para ordenação e algoritmos lineares

Projeto e Análise de Algoritmos

Professor Eurinardo Rodrigues Costa
Universidade Federal do Ceará
Campus Russas

2021.1

Sumário

Aulas Passadas

Limite inferior para ordenação por comparação $O(n \log n)$

Ordenação por contagem

Aulas Passadas

Aulas Passadas

- Algoritmos de Ordenação **Por Comparação**

Aulas Passadas

- Algoritmos de Ordenação **Por Comparação**

  Insertion-Sort é $O(n^2)$

Aulas Passadas

- Algoritmos de Ordenação **Por Comparação**
  Insertion-Sort é $O(n^2)$
  Merge-Sort é $O(n \log n)$,

Aulas Passadas

- Algoritmos de Ordenação **Por Comparação**

  Insertion-Sort é $O(n^2)$
  Merge-Sort é $O(n \log n)$,
  Heap-Sort é $O(n \log n)$ e

Aulas Passadas

- Algoritmos de Ordenação **Por Comparação**

  Insertion-Sort é $O(n^2)$
  Merge-Sort é $O(n \log n)$,
  Heap-Sort é $O(n \log n)$ e

  $$\text{Quick-Sort} \begin{cases} O(n^2) \text{ no pior caso} \\ \end{cases}$$

Aulas Passadas

- Algoritmos de Ordenação **Por Comparação**

  Insertion-Sort é $O(n^2)$
  Merge-Sort é $O(n \log n)$,
  Heap-Sort é $O(n \log n)$ e

  $$\text{Quick-Sort} \begin{cases} O(n^2) \text{ no pior caso} \\ O(n \log n) \text{ no caso médio e melhor caso} \end{cases}$$

Limite inferior para ordenação por comparação

Algoritmo 1: Insertion-Sort

Entrada: Vetor $A[1 \cdots n]$ e inteiro n (tamanho de A)
Saída: A ordenado

```
para j ← 2 até n faça
    chave ← A[j]
    %inserir A[j] na sequencia ordenada A[1 ··· j − 1]
    i ← j − 1
    enquanto (A[i] > chave) e (i > 0) faça
        A[i + 1] ← A[i]
        i ← i − 1
    A[i + 1] ← chave
```

Árvore de decisão do Insertion-Sort

Árvore de decisão do Insertion-Sort

Considere o vetor $[a_1, a_2, a_3]$

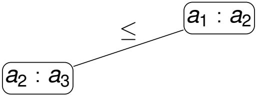
Árvore de decisão do Insertion-Sort

Considere o vetor $[a_1, a_2, a_3]$

$\boxed{a_1 : a_2}$

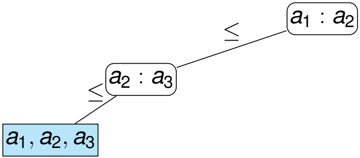
Árvore de decisão do Insertion-Sort

Considere o vetor $[a_1, a_2, a_3]$



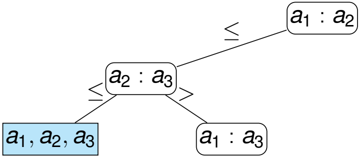
Árvore de decisão do Insertion-Sort

Considere o vetor $[a_1, a_2, a_3]$



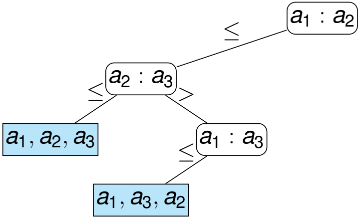
Árvore de decisão do Insertion-Sort

Considere o vetor $[a_1, a_2, a_3]$



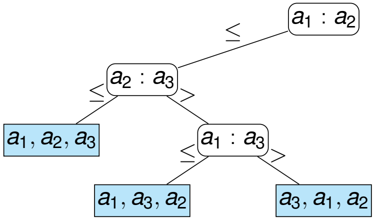
Árvore de decisão do Insertion-Sort

Considere o vetor $[a_1, a_2, a_3]$



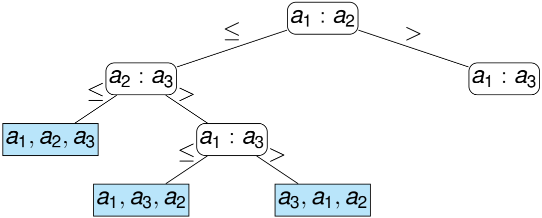
Árvore de decisão do Insertion-Sort

Considere o vetor $[a_1, a_2, a_3]$



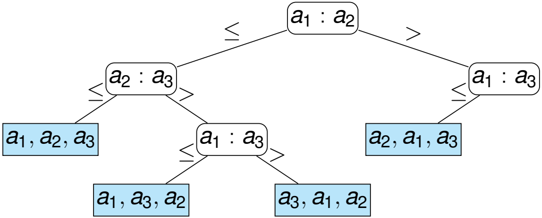
Árvore de decisão do Insertion-Sort

Considere o vetor $[a_1, a_2, a_3]$



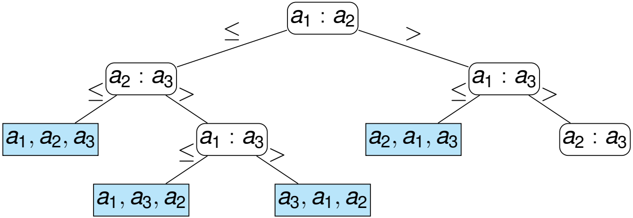
Árvore de decisão do Insertion-Sort

Considere o vetor $[a_1, a_2, a_3]$



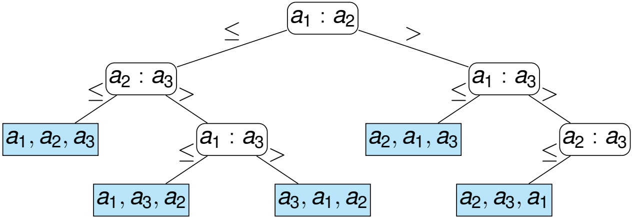
Árvore de decisão do Insertion-Sort

Considere o vetor $[a_1, a_2, a_3]$



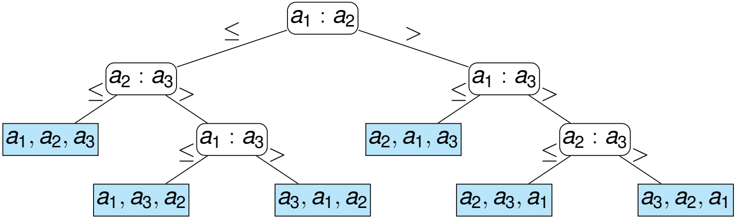
Árvore de decisão do Insertion-Sort

Considere o vetor $[a_1, a_2, a_3]$

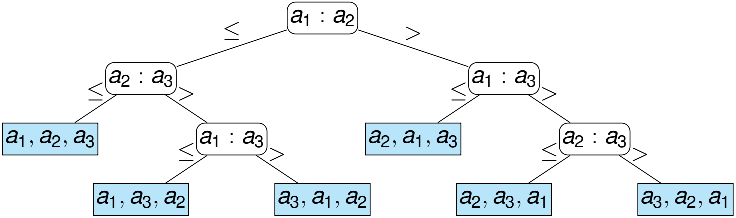


Árvore de decisão do Insertion-Sort

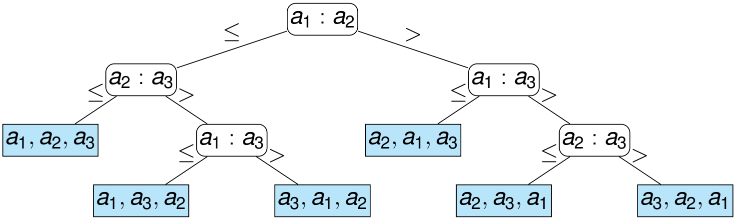
Considere o vetor $[a_1, a_2, a_3]$



Árvore de decisão do Insertion-Sort

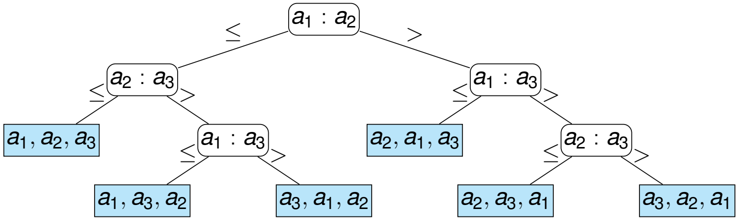


Árvore de decisão do Insertion-Sort



Observações

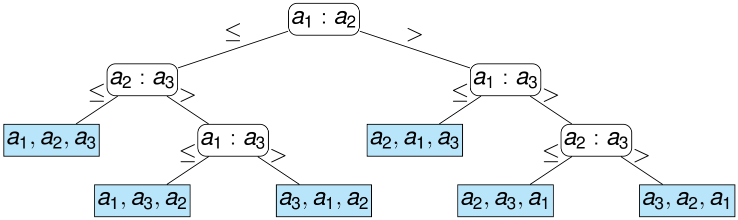
Árvore de decisão do Insertion-Sort



Observações

- Pior caso $\rightarrow$ altura da árvore de decisão

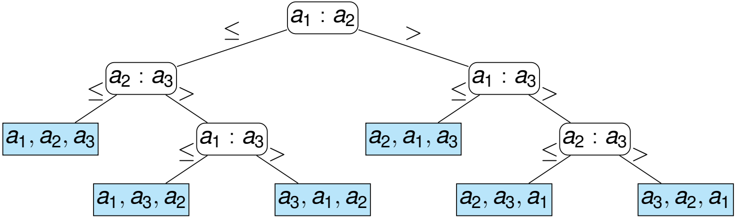
Árvore de decisão do Insertion-Sort



Observações

- Pior caso $\rightarrow$ altura da árvore de decisão
- Número de folhas da árvore de decisão é $n!$

Árvore de decisão do Insertion-Sort



Observações

- Pior caso $\rightarrow$ altura da árvore de decisão
- Número de folhas da árvore de decisão é $n!$ (total de permutações dos elementos do vetor).

Ordenação por contagem

Algoritmo 2: Counting-Sort(A, B, k)

Ordenação por contagem

Algoritmo 3: Counting-Sort(A, B, k)

Entrada: A: vetor com elementos em $\{0, \cdots, k\}$

Ordenação por contagem

Algoritmo 4: Counting-Sort(A, B, k)

Entrada: A: vetor com elementos em $\{0, \cdots, k\}$
Saída: B: vetor ordenado com os elementos de A

Ordenação por contagem

Algoritmo 5: Counting-Sort(A, B, k)

Entrada: A: vetor com elementos em $\{0, \cdots, k\}$

Saída: B: vetor ordenado com os elementos de A

$n \leftarrow$ tamanho de A

Ordenação por contagem

Algoritmo 6: Counting-Sort(A, B, k)

Entrada: A: vetor com elementos em $\{0, \cdots, k\}$

Saída: B: vetor ordenado com os elementos de A

$n \leftarrow$ tamanho de A

para $i \leftarrow 0$ **até** k **faça**

    $c[i] \leftarrow 0$;

Ordenação por contagem

Algoritmo 7: Counting-Sort(A, B, k)

Entrada: A: vetor com elementos em $\{0, \cdots, k\}$
Saída: B: vetor ordenado com os elementos de A

$n \leftarrow$ tamanho de A
para $i \leftarrow 0$ **até** k **faça**
    $c[i] \leftarrow 0$;
para $j \leftarrow 1$ **até** n **faça**
    $c[A[j]] + +$; % $c[i] \leftarrow$ qtd de elementos i

Ordenação por contagem

Algoritmo 8: Counting-Sort(A, B, k)

Entrada: A: vetor com elementos em $\{0, \cdots, k\}$

Saída: B: vetor ordenado com os elementos de A

```
n ← tamanho de A
para i ← 0 até k faça
    c[i] ← 0;
para j ← 1 até n faça
    c[A[j]] + +; % c[i] ← qtd de elementos i
para i ← 0 até k faça
    c[i] ← c[i] + c[i − 1];% c[i] ← qtd de elem. ≤ i
```

Ordenação por contagem

Algoritmo 9: Counting-Sort(A, B, k)

Entrada: A: vetor com elementos em $\{0, \cdots, k\}$
Saída: B: vetor ordenado com os elementos de A

```
n ← tamanho de A
para i ← 0 até k faça
    c[i] ← 0;
para j ← 1 até n faça
    c[A[j]] + +; % c[i] ← qtd de elementos i
para i ← 0 até k faça
    c[i] ← c[i] + c[i − 1];% c[i] ← qtd de elem. ≤ i
para j ← n até 1 faça
    B[c[A[j]]] ← A[j];
```

Ordenação por contagem

Algoritmo 10: Counting-Sort(A, B, k)

Entrada: A: vetor com elementos em $\{0, \cdots, k\}$

Saída: B: vetor ordenado com os elementos de A

$n \leftarrow$ tamanho de A
para $i \leftarrow 0$ **até** k **faça**
    $c[i] \leftarrow 0$;
para $j \leftarrow 1$ **até** n **faça**
    $c[A[j]]++$; % $c[i] \leftarrow$ qtd de elementos i
para $i \leftarrow 0$ **até** k **faça**
    $c[i] \leftarrow c[i] + c[i-1]$;% $c[i] \leftarrow$ qtd de elem. $\leq i$
para $j \leftarrow n$ **até** 1 **faça**
    $B[c[A[j]]] \leftarrow A[j]$;
    $c[A[j]]--$;

Ordenação por contagem

Algoritmo 11: Radix-Sort(A, d)

Ordenação por contagem

Algoritmo 12: Radix-Sort(A, d)

Entrada: A: vetor com elementos com d dígitos

Ordenação por contagem

Algoritmo 13: Radix-Sort(A, d)

Entrada: A: vetor com elementos com d dígitos

Saída: A: vetor ordenado

Ordenação por contagem

Algoritmo 14: Radix-Sort(A, d)

Entrada: A: vetor com elementos com d dígitos

Saída: A: vetor ordenado

```
para i ← 1 até d faça
    ordenar A com algoritmo estável considerando o dígito i;
```

Bibliografia

LEISERSON, C.E., STEIN, C., RIVEST, R.L., CORMEN T.H.
Algoritmos: teoria e prática, 3ed.
Editora Campus, ano 2012.

Obrigado!