Teoria da Computação

Guilherme Henrique de Souza Nakahata

Universidade Estadual do Paraná - Unespar

12 de Junho de 2024

Algoritmos de ordenação

- Bubble Sort;
- Merge Sort;
- Quick Sort;
- Insertion Sort;
- Heap Sort;

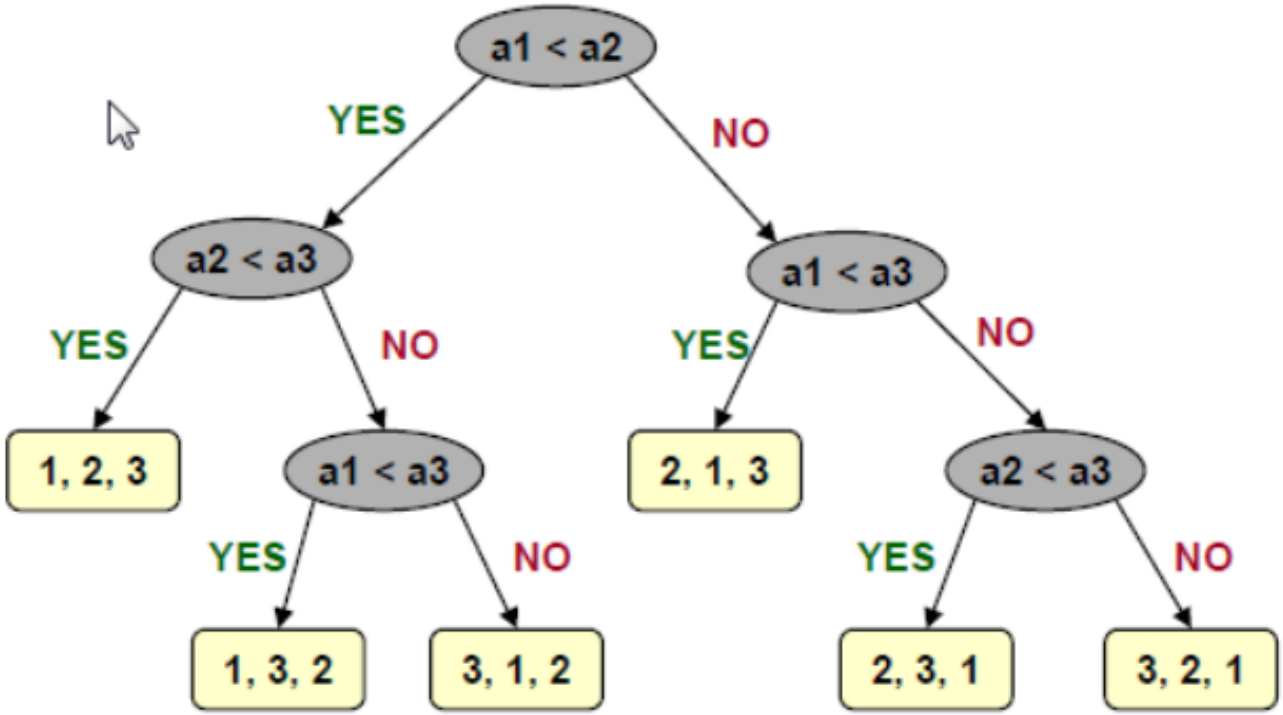
Limite inferior para ordenação

- Qual é o limite inferior para ordenação usando a operação de comparação entre os elementos?

Limite inferior para ordenação

- Qual é o limite inferior para ordenação usando a operação de comparação entre os elementos?
- É possível ordenar com tempo inferior a $\Omega(nlgn)$ no pior caso?

Ordenação por tempo linear



Limite inferior para ordenação

- Teorema: Qualquer árvore decisão que ordena n elementos deve ter altura pelo menos $\Omega(nlgn)$;
- Prova?

Limite inferior para ordenação

- Teorema: Qualquer árvore decisão que ordena n elementos deve ter altura pelo menos $\Omega(nlgn)$;
- Considere uma árvore de decisão de altura **h** que ordena **n** elementos;
- Qual o número de permutações?

Limite inferior para ordenação

- Teorema: Qualquer árvore decisão que ordena n elementos deve ter altura pelo menos $\Omega(nlgn)$;
- Considere uma árvore de decisão de altura **h** que ordena **n** elementos;
- Qual o número de permutações?
- **n!**;

Limite inferior para ordenação

- Teorema: Qualquer árvore decisão que ordena n elementos deve ter altura pelo menos $\Omega(nlgn)$;
- Considere uma árvore de decisão de altura **h** que ordena **n** elementos;
- Qual o número de permutações?
- **n!**;
- Primeiro elemento pode ter n combinações possíveis;
- O segundo n-1;
- O terceiro n-2;
- ...;

Limite inferior para ordenação

- Teorema: Qualquer árvore decisão que ordena n elementos deve ter altura pelo menos $\Omega(nlgn)$;
- Considere uma árvore de decisão de altura **h** que ordena **n** elementos;
- **n!** permutações;
- Toda árvore binária tem no máximo quantas folhas?

Limite inferior para ordenação

- Teorema: Qualquer árvore decisão que ordena n elementos deve ter altura pelo menos $\Omega(nlgn)$;
- Considere uma árvore de decisão de altura **h** que ordena **n** elementos;
- **n!** permutações;
- Toda árvore binária tem no máximo quantas folhas?
- 2^h;
- Logo:
- $n! \leq 2^h$

Limite inferior para ordenação

- Aproximação de Stirling:

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

- Logo:
- $n! > (\frac{n}{e})^n$

Limite inferior para ordenação

- Logo:
- $n! > (\frac{n}{e})^n$
- Assim, $lg(n!) > lg(\frac{n}{e})^n$

Limite inferior para ordenação

- Logo:
- $n! > (\frac{n}{e})^n$
- Assim, $lg(n!) > lg(\frac{n}{e})^n$
- $nlg\frac{n}{e}$
- $nlgn - nlge$;
- **?**

Limite inferior para ordenação

- $2^h \geq n!$;

Limite inferior para ordenação

- $2^h \geq n!$;
- Logaritmos em ambos os lados;
- $\log_2(2^h) \geq \log_2(n!)$
- $\log_b(a^c) = c \log_b(a)$
- $h \log_2(2) \geq \log_2(n!)$
- $h \cdot 1 \geq \log_2(n!)$
- $h \geq \log_2(n!)$

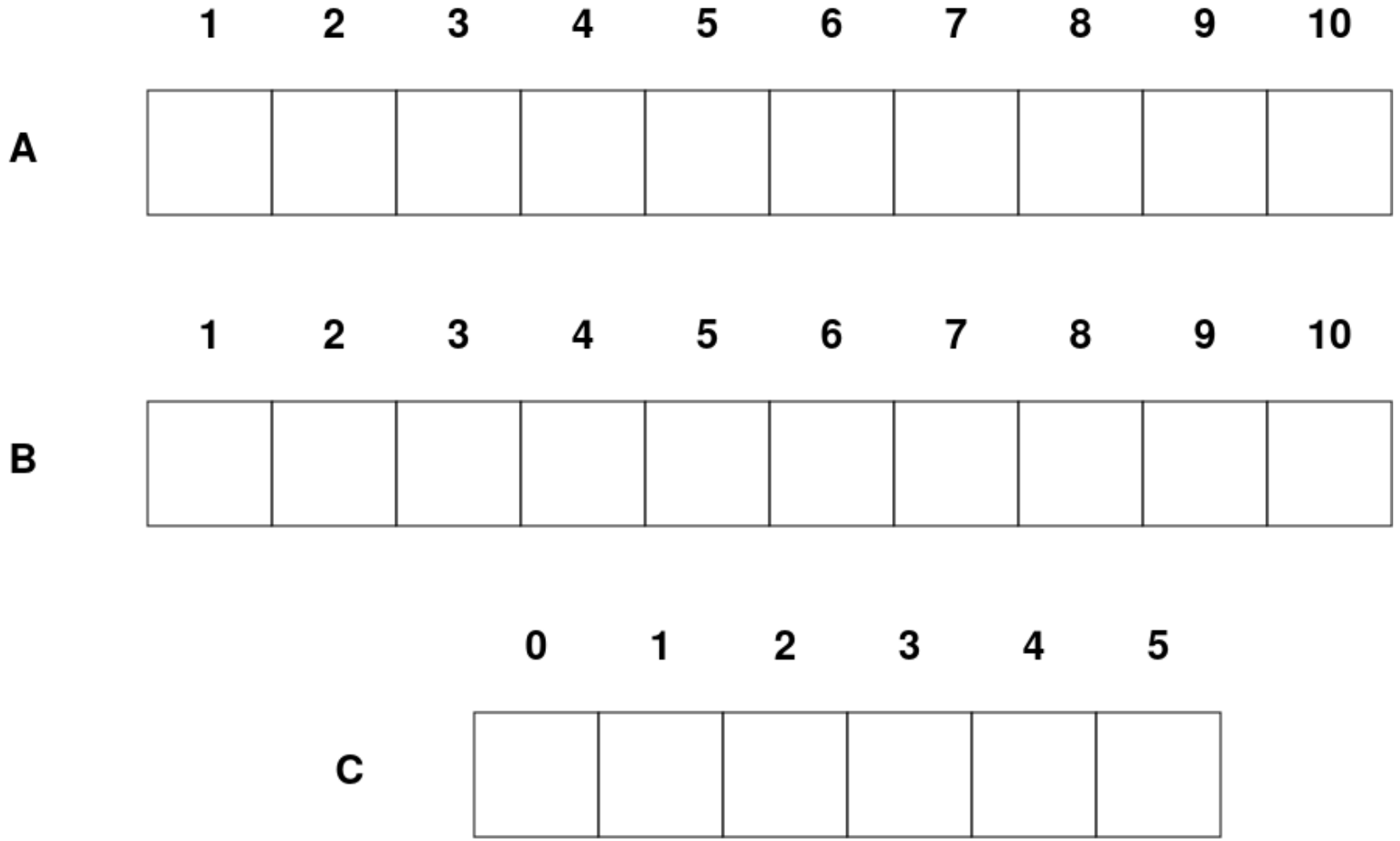
Limite inferior para ordenação

- $h \geq n \log_2(n) - n \log_2(e)$;
- $h = \Omega(nlgn)$.

Algoritmos de ordenação de complexidade linear

- Counting Sort;
- Radix Sort;
- Bucket Sort;

Counting Sort

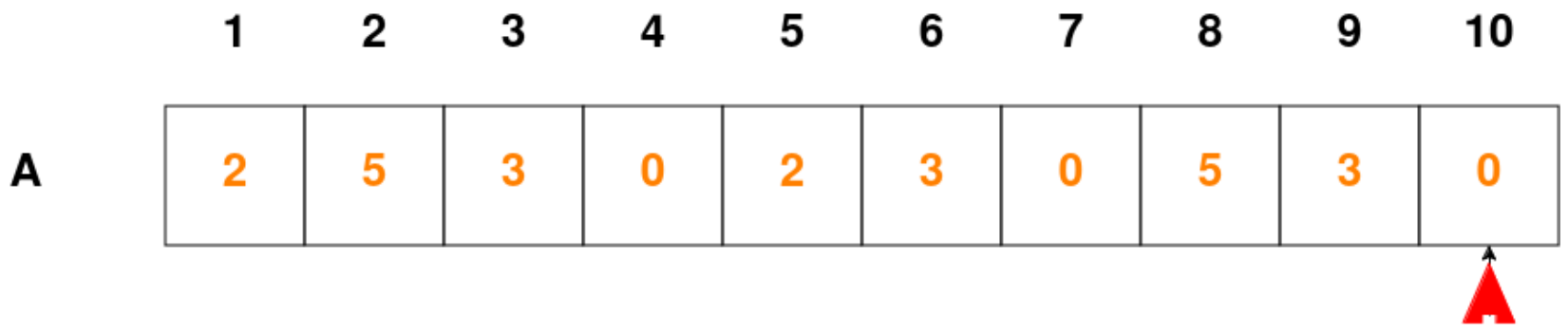


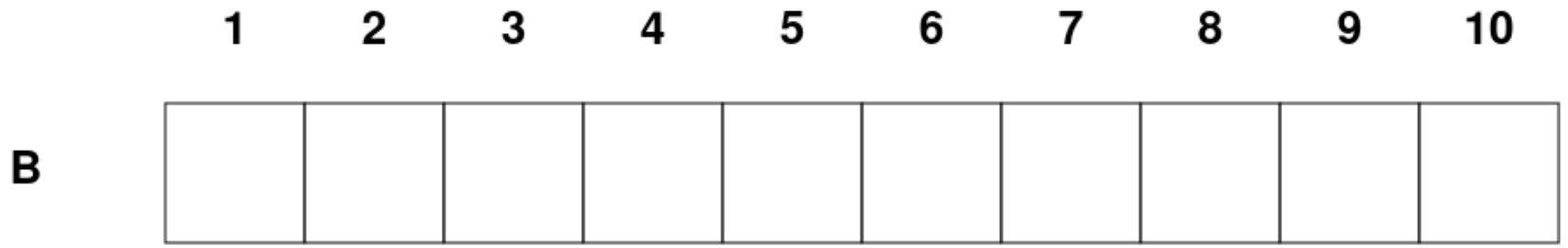
Counting Sort

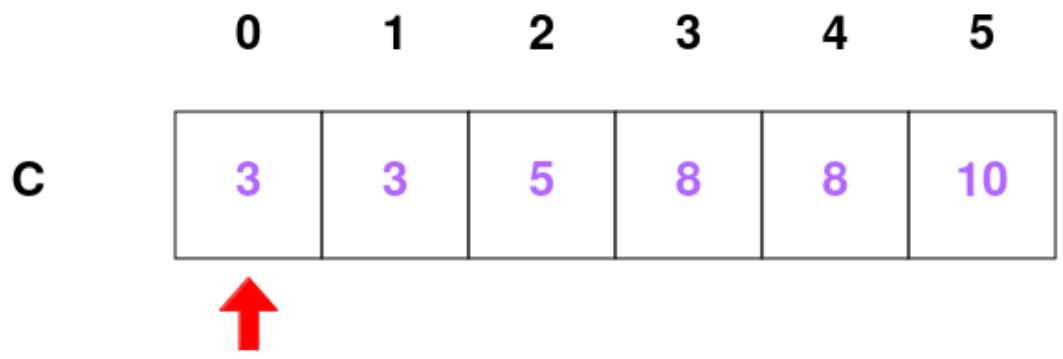
Counting Sort

Counting Sort

Counting Sort







Counting Sort

Counting Sort

```
Algoritmo Counting-Sort(A, B, k)
    para i de 1 até k faça
        C[i] <- 0
    para j de 1 até comprimento[A] faça
        C[A[j]] <- C[A[j]] + 1
    para i de 2 até k faça
        C[i] <- C[i] + C[i-1]
    para j de comprimento[A] até 1 faça
        B[C[A[j]]] <- A[j]
        C[A[j]] <- C[A[j]] - 1
```

Counting Sort

- $\Theta(k)$ { para $i \leftarrow 1$ to k do $C[i] \leftarrow 0$ }
- $\Theta(n)$ { para $j \leftarrow 1$ to comprimento[A] do $C[A[j]] \leftarrow C[A[j]] + 1$ }
- $\Theta(k)$ { para $i \leftarrow 2$ to k do $C[i] \leftarrow C[i] + C[i-1]$ }
- $\Theta(n)$ { para $j \leftarrow$ *comprimento*$[A]$ downto 1 do $B[C[A[j]]] \leftarrow A[j]$; $C[A[j]] \leftarrow C[A[j]] - 1$ }
- $\Theta(n+k)$

Counting Sort

- Análise de complexidade:
  - Counting Sort roda em tempo $\Theta(n + k)$. Se tivermos $k = O(n)$, então o algoritmo executa em tempo $\Theta(n)$.

Counting Sort

- Esse é algoritmo de ordenação **estável**?

Counting Sort

- Esse é algoritmo de ordenação **estável**?
- O algoritmo counting sort é **estável**;
- Propriedade fundamental para que o algoritmo seja utilizado com o **Radix Sort**;

Counting Sort

- Aspectos positivos:
  - Ordena vetores em tempo linear para o tamanho do vetor inicial;
  - Não realiza comparações;
  - É um algoritmo de ordenação estável;
- Aspectos Negativos:
  - ?

Counting Sort

- Aspectos positivos:
  - Ordena vetores em tempo linear para o tamanho do vetor inicial;
  - Não realiza comparações;
  - É um algoritmo de ordenação estável;
- Aspectos Negativos:
  - Necessita de dois vetores adicionais para sua execução, utilizando, assim, mais espaço na memória.

Radix Sort

- Algoritmo de ordenação não comparativo;
- Ordena inteiros ou strings de comprimento fixo;
- Digit by digit (ou character by character);
- Implementado de duas formas:
  - Least Significant Digit (LSD) - do dígito menos significativo ao mais significativo.
  - Most Significant Digit (MSD) - do dígito mais significativo ao menos significativo.

Radix Sort

- Divide cada número em seus dígitos componentes;
- Ordena os números começando pelo dígito menos significativo (LSD);
- Usa um algoritmo de ordenação estável (como Counting Sort) para ordenar os dígitos;
- Repete o processo para cada dígito, movendo para o próximo mais significativo;

Radix Sort

329		720		720		329
457		355		329		355
657		436		436		436
839	=>	457	=>	839	=>	457
436		657		355		657
720		329		457		720
355		839		657		839

Complexidade de Tempo

- A complexidade de tempo do Radix Sort é $O(d \cdot (n + k))$, onde:
  - d é o número de dígitos;
  - n é o número de elementos a serem ordenados;
  - k é o intervalo dos dígitos (por exemplo, 0-9 para números decimais);
- Radix Sort é eficiente quando d é relativamente pequeno (constante);
- Rodando em tempo de O(n).

Vantagens e Desvantagens

- Aspectos positivos:
  - Ordenação linear em muitos casos práticos.
  - Bom para números inteiros ou strings de comprimento fixo.
  - Não comparativo, evitando overhead de comparação.
- Aspectos Negativos:
  - Requer espaço adicional para armazenamento temporário.
  - Não é adequado para todos os tipos de dados (por exemplo, números de ponto flutuante).
  - Menos eficiente se d ou k forem muito grandes.

Bucket Sort

- Algoritmo:
  - Inicialize um arranjo de "baldes", inicialmente vazios;
  - Vá para o arranjo original, incluindo cada elemento em um balde;
  - Ordene todos os baldes não vazios;
  - Coloque os elementos dos baldes que não estão vazios no arranjo original.
- https://www.youtube.com/watch?v=ibtN8rY7V5k;

Atividade

- Implemente o algoritmo de ordenação counting sort com radix sort.

Bibliografia Básica

- LEWIS, H. R.; PAPADIMITRIOU, C. H. **Elementos de Teoria da Computação**. 2 ed. Porto Alegre: Bookman, 2000.
- VIEIRA, N. J. **Introdução aos Fundamentos da Computação**. Editora Pioneira Thomson Learning, 2006.
- DIVERIO, T. A.; MENEZES, P. B. **Teoria da Computação: Máquinas Universais e Computabilidade**. Série Livros Didáticos Número 5, Instituto de Informática da UFRGS, Editora Sagra Luzzato, 1 ed. 1999.

Obrigado!
Dúvidas?

Guilherme Henrique de Souza Nakahata

guilhermenakahata@gmail.com

https://github.com/GuilhermeNakahata/UNESPAR-2024