Estrutura de Dados (CCA410)

Aula 08 – Algoritmos de Ordenação Eficientes

(Merge, Quick, Heap Sort)

Prof. Luciano Rossi
Prof. Leonardo Anjoletto Ferreira
Prof. Flavio Tonidandel
Prof. Fabio Suim

Ciência da Computação
Centro Universitário FEI

2° Semestre de 2025

Algoritmos de ordenação

Algoritmos de ordenação

Definição Formal

Seja $A = [a_1, a_2, \ldots, a_n]$ uma sequência de n elementos de um conjunto S, onde existe uma relação de ordem total $\leqslant$ definida sobre S.

Um algoritmo de ordenação é uma função f tal que:

$$f(A) = A' = [a'_1, a'_1, \ldots, a'_n]$$

- A' é uma permutação de A
- $a'_1 \leqslant a'_2 \leqslant \ldots \leqslant a'_n$
- O algoritmo sempre termina em tempo finito
- Para qualquer entrada válida, produz a saída correta

Algoritmos de ordenação

Propriedades Fundamentais

- **Estabilidade:** Um algoritmo é estável se preserva a ordem relativa de elementos com chaves iguais
- **In-place:** Opera com espaço adicional $O(1)$, modificando a estrutura original
- **Adaptabilidade:** Performa melhor em sequências parcialmente ordenadas
- **Complexidade:** Caracterizada pelo número de comparações e movimentações necessárias

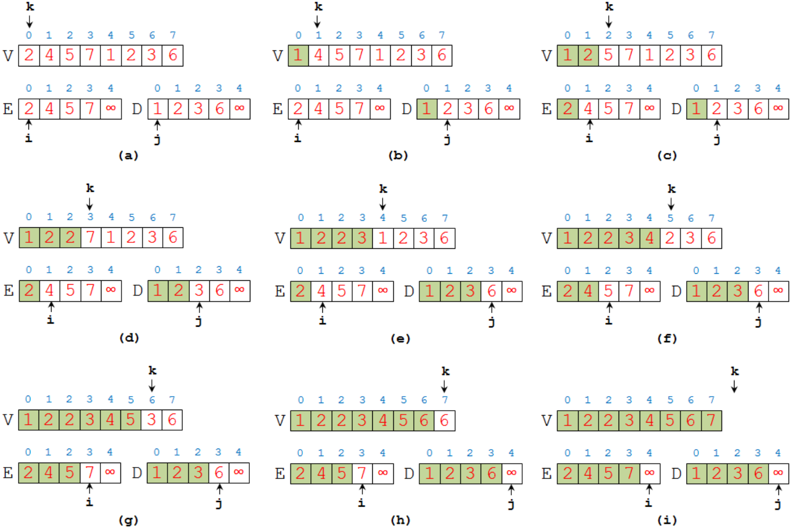
Merge Sort

Definição

O Merge Sort é um algoritmo de ordenação baseado na estratégia “dividir para conquistar” que funciona recursivamente dividindo o array em metades menores até chegar a elementos individuais, e depois combina (merge) essas partes de volta de forma ordenada.

Algoritmos de ordenação

Merge Sort – Exemplo



centro universitário FEI

Algoritmos de ordenação

Merge Sort - Algoritmo

$Merge(V, p, q, r)$

$n_1 \leftarrow q - p + 1$
$n_2 \leftarrow r - q$
sejam $E[0 \ldots n_1]$ e $D[0 \ldots n_2]$ novos vetores
para $i \leftarrow 0$ até $n_1 - 1$ faça
    $E[i] \leftarrow V[p + i]$
para $j \leftarrow 0$ até $n_2 - 1$ faça
    $D[j] \leftarrow V[q + j + 1]$
$E[n_1] \leftarrow \infty$
$D[n_2] \leftarrow \infty$
$i \leftarrow 0$
$j \leftarrow 0$
para $k \leftarrow p$ até r faça
    se $E[i] \leq D[j]$ então
        $V[k] \leftarrow E[i]$
        $i \leftarrow i + 1$
    senão
        $V[k] \leftarrow D[j]$
        $j \leftarrow j + 1$

Simular com $[8, 2, 1, 6, 7, 0, 3, 5, 4, 9]$

Algoritmos de ordenação

Merge Sort - Algoritmo

$MergeSort(V, p, r)$

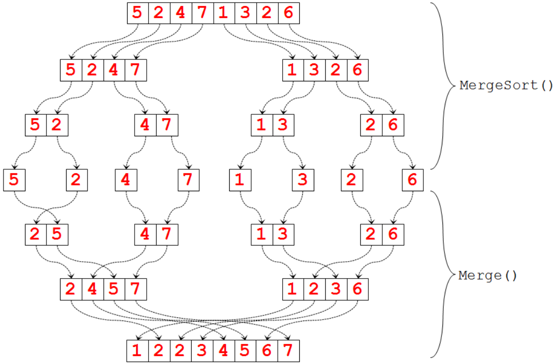
se $p < r$ então

$\quad q \leftarrow \lfloor (p+r)/2 \rfloor$

$\quad MergeSort(V, p, q)$

$\quad MergeSort(V, q+1, r)$

$\quad Merge(V, p, q, r)$



Simular com $[8, 2, 1, 6, 7, 0, 3, 5, 4, 9]$

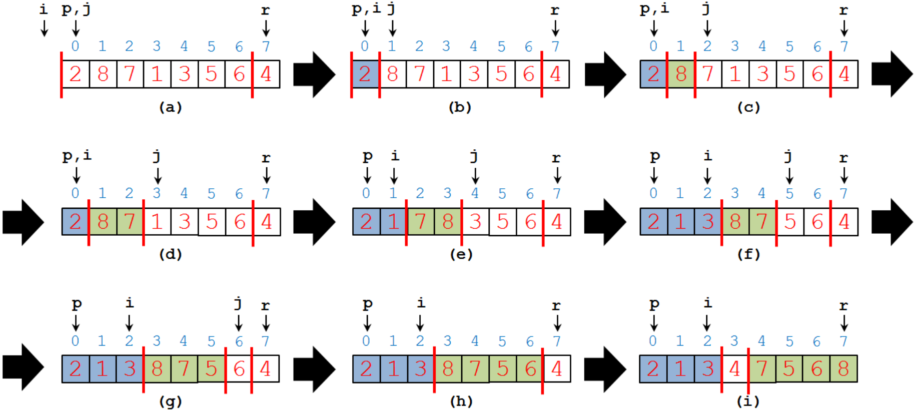
Quick Sort

Definição

O Quick Sort é um algoritmo de ordenação baseado na estratégia “dividir para conquistar” que funciona selecionando um elemento como pivô e particionando o array de forma que todos os elementos menores que o pivô fiquem à sua esquerda e todos os maiores fiquem à sua direita.

Algoritmos de ordenação

Quick Sort – Exemplo



Algoritmos de ordenação

Quick Sort - Algoritmo

$QuickSort(V, p, r)$

se $p < r$ então
    $q \leftarrow Partition(V, p, r)$
    $QuickSort(V, p, q-1)$
    $QuickSort(V, q+1, r)$

$Partition(V, p, r)$

$x \leftarrow V[r]$
$i \leftarrow p - 1$
para $j \leftarrow p$ até $r - 1$ faça
    se $V[j] \leq x$ então
        $i \leftarrow i + 1$
        trocar $V[i]$ e $V[j]$
trocar $V[i+1]$ e $V[r]$
retornar $i + 1$

Simular com $[8, 2, 1, 6, 7, 0, 3, 5, 4, 9]$

Heap Sort

Definição

O Heap Sort é um algoritmo de ordenação baseado na estrutura de dados heap (árvore binária completa) que funciona construindo um max-heap a partir do array e repetidamente extraindo o elemento máximo para formar o array ordenado.

Algoritmos de ordenação

Heap Sort – Exemplo

(a)

4	1	3	2	16	9	10	14	8	7
1	2	3	4	5	6	7	8	9	10

Array inicial

(b)

4	1	3	14	16	9	10	2	8	7
1	2	3	4	5	6	7	8	9	10

MaxHeapify(4)

(c)

4	1	10	14	16	9	3	2	8	7
1	2	3	4	5	6	7	8	9	10

MaxHeapify(3)

(d)

4	16	10	14	7	9	3	2	8	1
1	2	3	4	5	6	7	8	9	10

MaxHeapify(2)

(e)

16	14	10	8	7	9	3	2	4	1
1	2	3	4	5	6	7	8	9	10

MaxHeapify(1) - Max-heap!

(f)

14	8	10	4	7	9	3	2	1	16
1	2	3	4	5	6	7	8	9	10

Extrai 16

(g)

10	8	9	4	7	1	3	2	14	16
1	2	3	4	5	6	7	8	9	10

Extrai 14

(h)

9	8	3	4	7	1	2	10	14	16
1	2	3	4	5	6	7	8	9	10

Extrai 10

...

...	...	...	...	...	...	...	...	...	...
1	2	3	4	5	6	7	8	9	10

Continua...

(i)

1	2	3	4	7	8	9	10	14	16
1	2	3	4	5	6	7	8	9	10

Array ordenado!

centro universitário FEI

Algoritmos de ordenação

Heap Sort – Algoritmo

$HeapSort(V)$

- $BuildMaxHeap(V)$
- para $i \leftarrow V.tamanho$ até 2 faça
    - trocar $V[1]$ com $V[i]$
    - $V.heapSize \leftarrow V.heapSize - 1$
    - $MaxHeapify(V, 1)$

$BuildMaxHeap(V)$

- $V.heapSize \leftarrow V.tamanho$
- para $i \leftarrow \lfloor V.tamanho/2 \rfloor$ até 1 faça
    - $MaxHeapify(V, i)$

$MaxHeapify(V, i)$

- $esq \leftarrow 2 \cdot i$
- $dir \leftarrow 2 \cdot i + 1$
- se $esq \leq V.heapSize$ e $V[esq] > V[i]$ então
    - $maior \leftarrow esq$
- senão
    - $maior \leftarrow i$
- se $dir \leq V.heapSize$ e $V[dir] > V[maior]$ então
    - $maior \leftarrow dir$
- se $maior \neq i$ então
    - trocar $V[i]$ com $V[maior]$
    - $MaxHeapify(V, maior)$

Simular com $[8, 2, 1, 6, 7, 0, 3, 5, 4, 9]$

Considerações

Considerações

Tabela: Complexidade dos Algoritmos de Ordenação

Algoritmo	Melhor Caso	Caso Médio	Pior Caso	Memória	Estável
Bubble Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(1)$	Sim
Insertion Sort	$O(n)$	$O(n^2)$	$O(n^2)$	$O(1)$	Sim
Selection Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(1)$	Não
Merge Sort	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$	$O(n)$	Sim
Quick Sort	$O(n \log n)$	$O(n \log n)$	$O(n^2)$	$O(\log n)$	Não
Heap Sort	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$	$O(1)$	Não

centro universitário FEI

Estrutura de Dados (CCA410)

Aula 08 – Algoritmos de Ordenação Eficientes

(Merge, Quick, Heap Sort)

Prof. Luciano Rossi
Prof. Leonardo Anjoletto Ferreira
Prof. Flavio Tonidandel
Prof. Fabio Suim

Ciência da Computação
Centro Universitário FEI

2° Semestre de 2025