BCC202 - Estruturas de Dados I

Aula 14: Ordenação: MergeSort

Pedro Silva

Universidade Federal de Ouro Preto, UFOP
Departamento de Computação, DECOM
Email: **silvap@ufop.edu.br**

2021

Conteúdo

Conteúdo

O projeto de muitos algoritmos eficientes é baseado no método da divisão e conquista.

- Mergesort
- Quichsort
- ...

Conteúdo

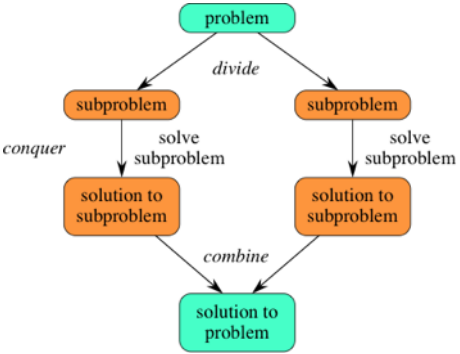
Conceitos Fundamentais

- É preciso revolver um problema com uma entrada grande.
- Para facilitar a resolução do problema, a entrada é quebrada em pedaços menores (**DIVISÃO**).
- Cada pedaço da entrada é então tratado separadamente (**CONQUISTA**).
- Ao final, os resultados parciais são **combinados** para gerar o resultado final procurado.

- A técnica de **divisão e conquista** consiste de **3 passos**:
  - **Divisão**: Dividir o problema original em subproblemas menores.
  - **Conquista**: Resolver cada subproblema recursivamente.
  - **Combinação**: Combinar as soluções encontradas, compondo uma solução para o problema original.

Ilustração

A seguir, uma ilustração da estratégia divisão e conquista.



Visão Geral

- Algoritmos baseados em divisão e conquista são, em geral, **recursivos**.
- A maioria dos algoritmos de divisão e conquista divide o problema de tamanho **n** em subproblemas da mesma natureza, de tamanho n/b, sendo **n** e **b** inteiros.
- Vantagens:
  - **Uso eficiente da memória cache**: ao final da fase de divisão, grande parte dos dados necessários para a fase de combinação já estão disponíveis na cache.
    - Com isso, requerem um número menor de acessos à memória.
  - **São altamente paralelizáveis**: se existirem vários processadores disponíveis, a estratégia propiciará eficiência.

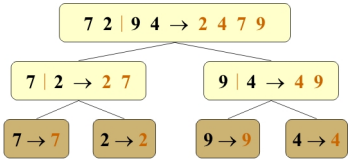
Utilização na ordenação

- Métodos de ordenação que utilizam-se da **Divisão e Conquista**:
  - **MergeSort**: aula de hoje.
  - **QuickSort**: próxima aula teórica.
- Principal diferença:
  - **MergeSort**: sempre divide o problema de forma balanceada (subproblemas de mesmo tamanho).
  - **QuickSort**: utiliza o conceito de **pivô** para dividir o problema em subproblemas (subproblemas potencialmente de tamanhos diferentes).

Conteúdo

Funcionamento

- A execução do **MergeSort** pode ser facilmente descrita por uma árvore binária:
  - Cada nó representa uma chamada recursiva do **MergeSort**.
  - O nó raiz é a chamada inicial.
  - Os nós folha são vetores de 1 número (caso base).



Ilustração

A seguir, mais detalhes sobre o passo a passo do *mergesort*.

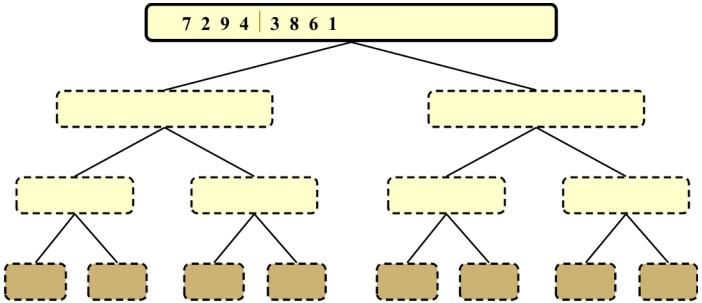

Animação


Vídeo

Visão Geral

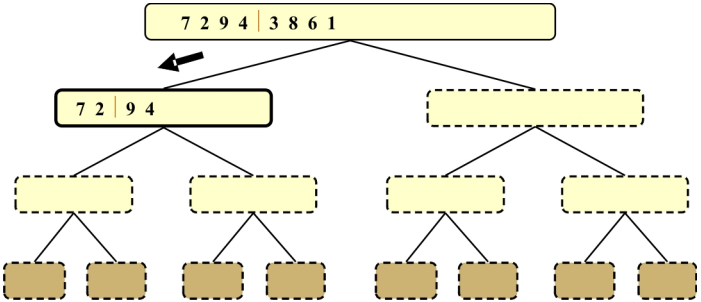
- Divida o vetor em duas partes.
- Ordene as duas partes usando chamadas recursivas.
- Intercale as duas partes ordenadas, obtendo um conjunto ordenado de todos os elementos.

Execução



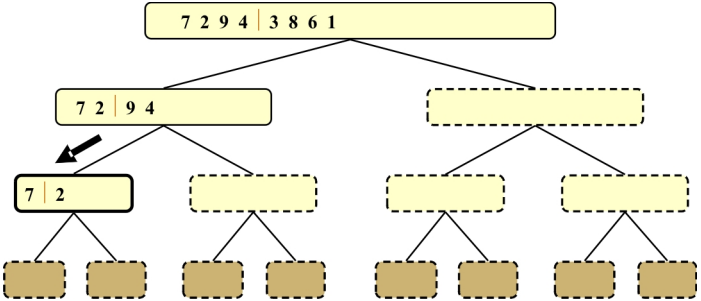
Partição do problema (sempre no meio do vetor).

Execução



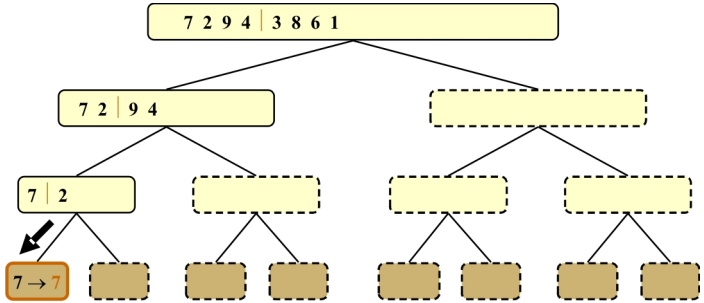
Chamada recursiva para primeira partição.

Execução



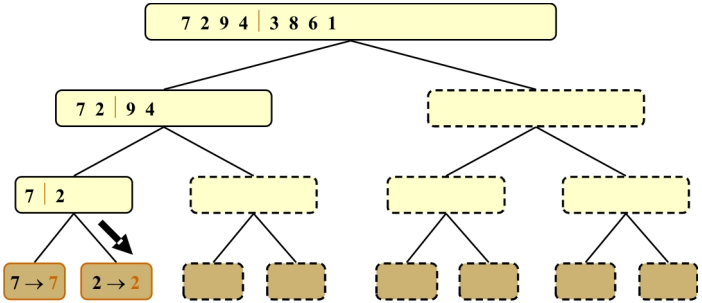
Chamada recursiva.

Execução



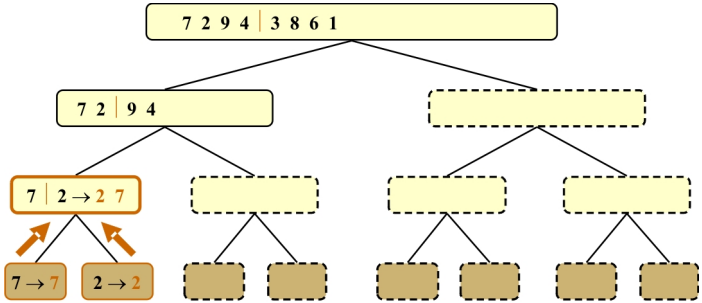
Chamada recursiva: caso base encontrado.

Execução



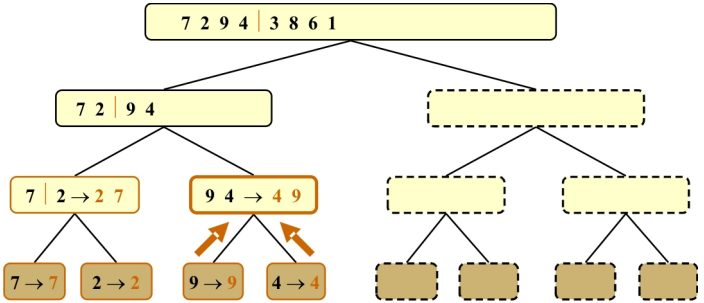
Chamada recursiva: caso base encontrado.

Execução



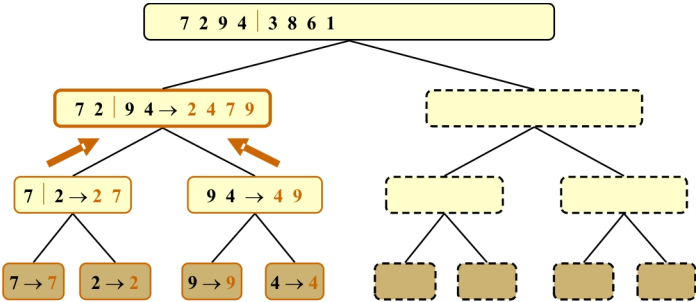
Operação de merge (intercalação).

Execução



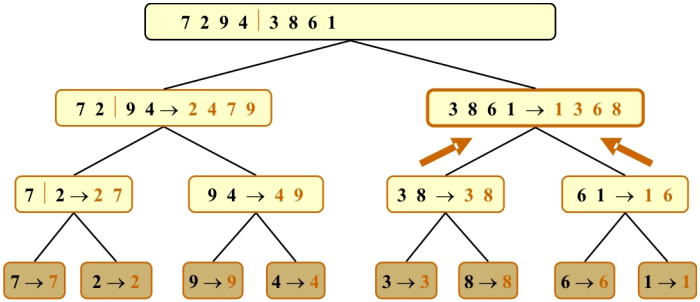
Chamadas recursivas, casos bases e merge (intercalação).

Execução



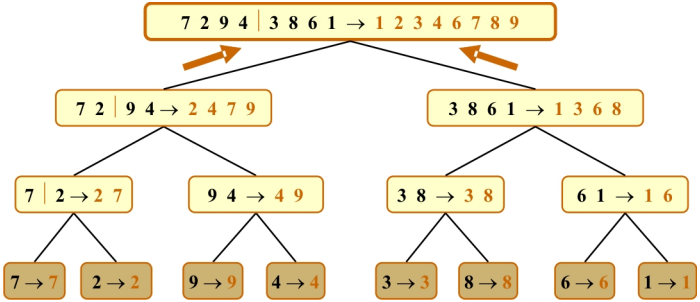
Operação de merge (intercalação).

Execução



Execução do MergeSort para a outra partição.

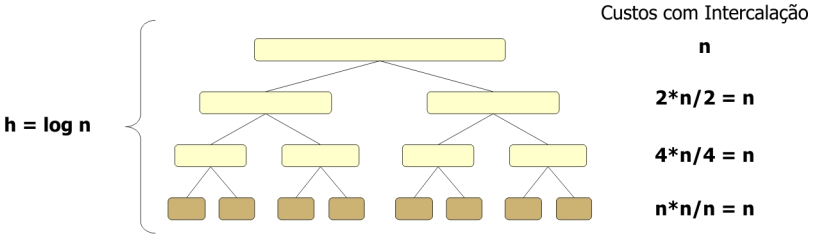
Execução



Finalmente o último merge (intercalação).

Ordem de Complexidade

- A altura h da árvore de execução é $O(log\ n)$.
- A quantidade de operações em cada nível da árvore é assintoticamente igual a $O(n)$.
- Logo: algoritmo é $O(n\ log\ n)$.



Conteúdo

Pseudocódigo

```
1 Algorithm: MERGESORT
  Input: int* v, int l, int r
2 begin
3     if l<r then
4         m ← (l + r)/2
5         MERGESORT(v, l, m)
6         MERGESORT(v, m + 1, r)
7         MERGE(v, l, m, r)
8     end
9 end
```

```
1  Algorithm: MERGE(int* v, int l, int m, int r)
2  begin
3      size_l ← (m − l + 1) size_r ← (r − m)
4      alocar vet_l[0...size_l − 1] e vet_r[0...size_r − 1]
5      for i ← 0 to i < size_l do
6          vet_l[i] ← v[i + l]
7      end
8      for j ← 0 to j < size_r do
9          vet_r[j] ← v[m + j + 1]
10     end
11     i ← 0 j ← 0
12     for k ← l to k ≤ r do
13         if i == size_l then
14             v[k] ← vet_r[j++]
15         end
16         else if  j == size_r then
17             v[k] ← vet_l[i++]
18         end
19         else if vet_l[i] ≤ vet_r[j] then
20             v[k] ← vet_l[i++]
21         end
22         else
23             v[k] ← vet_r[j++]
24         end
25     end
26     desalocar vet_l[0...size_l − 1] e vet_r[0...size_r − 1]
27 end
```

Tempo, Comparação e Espaço

Mergesort é um algoritmo estável.

- Complexidade de tempo no pior caso
  - $O(nlogn)$ comparações
  - $O(nlogn)$ trocas
- Complexidade de tempo no melhor caso
  - $O(nlogn)$ comparações
  - $O(nlogn)$ trocas
- Complexidade de espaço/consumo de espaço
  - O Mergesort usa vetores auxiliares (tamanho n) para fazer intercalação
  - O(n) extra.

MergeSort ITERATIVO

```
1  Algorithm: MERGE-SORT-ITER
   Input: int* v, int l, int r
2  begin
3      b ← 1
4      while b < n do
5          esq ← 0
6          while esq + b < n do
7              dir ← esq + 2 * b
8              if dir > n then
9                  dir ← n
10             end
11             MERGE(v, esq, esq + b − 1, dir − 1)
12             esq ← esq + 2 * b
13         end
14         b ← b * 2
15     end
16 end
```

Conteúdo

Conclusão

- Nesta aula, tivemos contato com o *algoritmo de ordenação* chamado **MergeSort**.
- Foram vistas duas versões: **Recursiva** e **Iterativa**.
- Apesar de mais simples de entender e implementar, a versão recursiva requer memória adicional.

Conclusão

- Quadro comparativo dos métodos de ordenação:

Algoritmo	Comparações			Movimentações			Espaço	Estável	In situ
	Melhor	Médio	Pior	Melhor	Médio	Pior			
Bubble	$O(n^2)$			$O(n^2)$			$O(1)$	**Sim**	**Sim**
Selection	$O(n^2)$			$O(n)$			$O(1)$	**Não***	**Sim**
Insertion	$O(n)$	$O(n^2)$		$O(1)$	$O(n^2)$		$O(1)$	**Sim**	**Sim**
Merge	$O(n \log n)$			$O(n \log n)$			$O(n)$	**Sim**	**Não**

* Existem versões estáveis.

Quicksort.

Conteúdo

Bibliografia

Os conteúdos deste material, incluindo figuras, textos e códigos, foram extraídos ou adaptados de:

- *Slides* **MO417 - Complexidade de Algoritmos I**, elaborados por Cid Carvalho de Souza, Cândida Nues da Silva e Orlando Lee e revisado por Zanoni Dias em agosto de 2011, `https://www.ic.unicamp.br/~zanoni/teaching/mo417/2011-2s/aulas/handout/04-ordenacao.pdf`. Acessado em 2021.

Cormen, Thomas H. and Leiserson, Charles E. and Rivest, Ronald L. and Stein, Clifford.
Introduction to Algorithms.
The MIT Press, 2011.

Exercício

- Dada a sequência de números: `3 4 9 2 5 1 8`.
- Ordene em ordem crescente utilizando o algoritmo aprendido em sala (**MergeSort**), apresentando a sequência dos números a cada passo (Teste de Mesa).