



Algoritmos de ordenação eficientes - Merge Sort

emanoelim@utfpr.edu.br

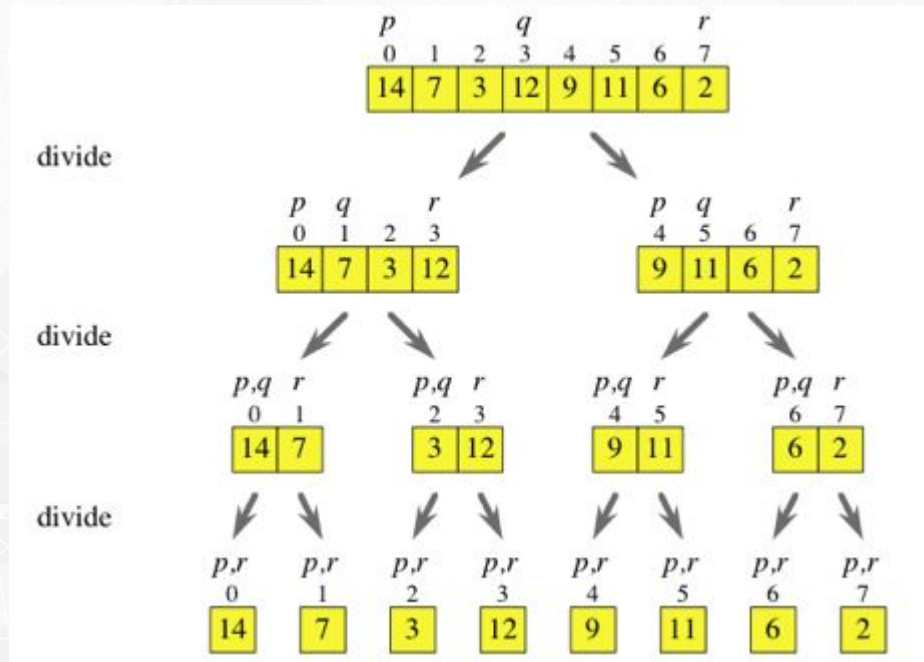
Merge Sort

Este algoritmo usa uma técnica conhecida como “**divisão e conquista**”: em vez de tratar o problema como um todo, essa técnica quebra o problema em problemas menores, encontrando as soluções destas partes menores e então combinando estas soluções para obter uma solução geral.

Merge Sort

- O algoritmo merge sort divide o vetor de entrada em duas metades e assim sucessivamente, até chegar em subvetores de tamanho 1 (caso base), visto que um vetor com apenas um item já está trivialmente ordenado.
- Considere por exemplo, o vetor formado pelos inteiros:
14, 7, 3, 12, 9, 11, 6, 2

Merge Sort



Merge Sort

- Uma vez divididos, os subvetores são agrupados em ordem, usando ideia de **intercalação / fusão** de dois vetores ordenados.
- Daí vem o nome merge sort: merge = fundir em inglês.
- Exemplo: intercalar os vetores ordenados v1 e v2 em v3:

Merge Sort

Intercalação:

1)

v1	0	2	5	10	
	↑				
v2	1	4	9	12	20
	↑				

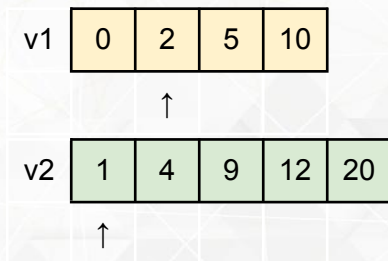
Entre 0 e 1, o menor é 0, então v3 guarda 0:

v3	0							
----	---	--	--	--	--	--	--	--

Merge Sort

Intercalação:

2)



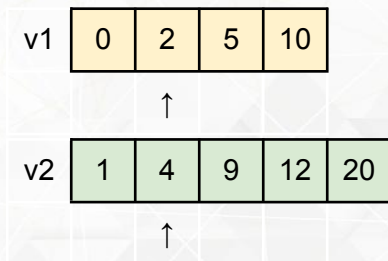
Entre 2 e 1, o menor é 1, então v3 guarda 1:



Merge Sort

Intercalação:

3)



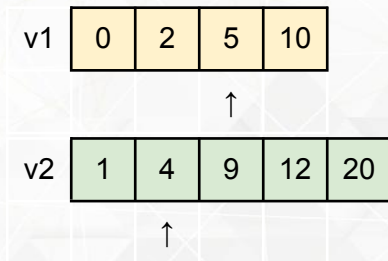
Entre 2 e 4, o menor é 2, então v3 guarda 2:



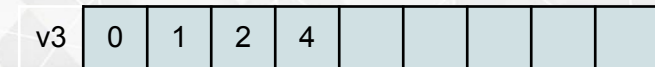
Merge Sort

Intercalação:

4)



Entre 5 e 4, o menor é 4, então v3 guarda 4:



Merge Sort

Intercalação:

5)

v1	0	2	5	10	
			↑		
v2	1	4	9	12	20
			↑		

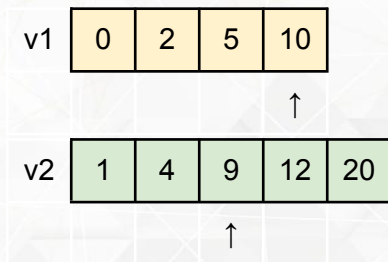
Entre 5 e 9, o menor é 5, então v3 guarda 5:

v3	0	1	2	4	5				
----	---	---	---	---	---	--	--	--	--

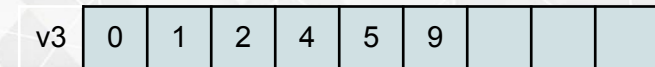
Merge Sort

Intercalação:

6)



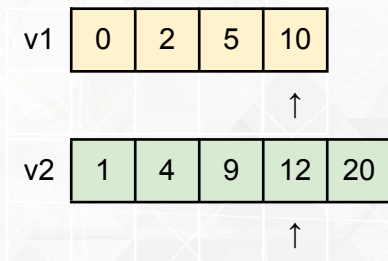
Entre 10 e 9, o menor é 9, então v3 guarda 9:



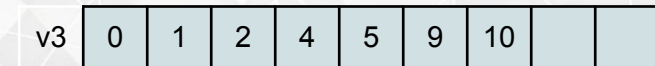
Merge Sort

Intercalação:

7)



Entre 10 e 12, o menor é 10, então v3 guarda 10:



Merge Sort

Intercalação:

8) Não há mais elementos em v1, então basta adicionar os elementos restantes de v2 em v3

v2	1	4	9	12	20
			↑		

v3	0	1	2	4	5	9	10	12	
----	---	---	---	---	---	---	----	----	--

Merge Sort

Intercalação:

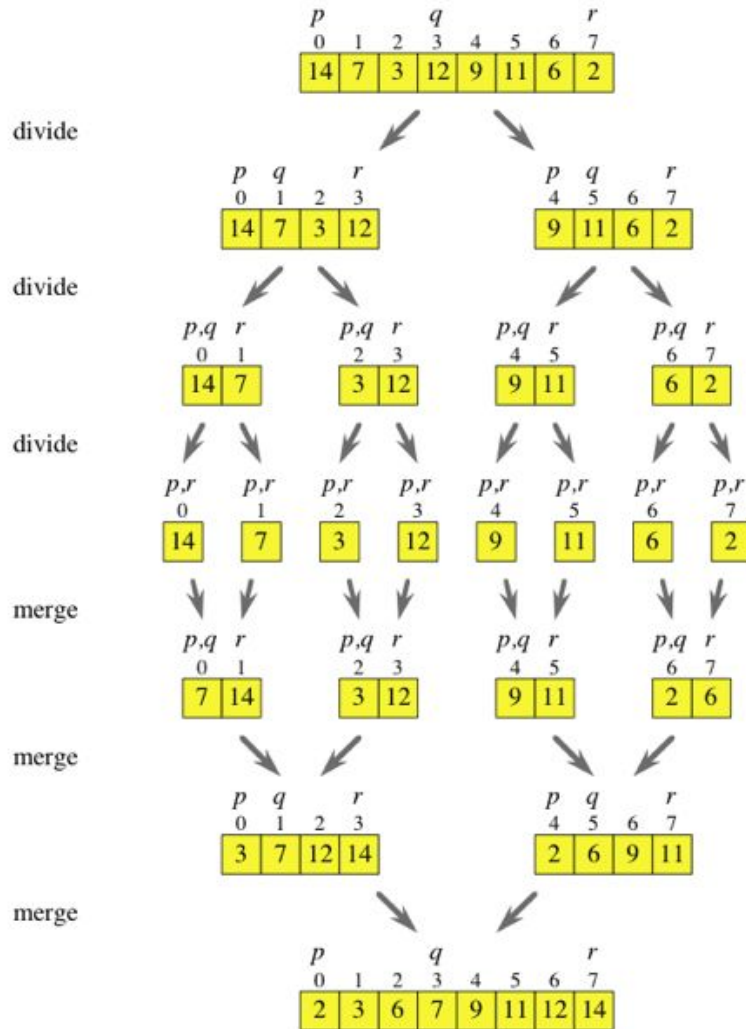
9) Não há mais elementos em v1, então basta adicionar os elementos restantes de v2 em v3

v2	1	4	9	12	20
					↑

v3	0	1	2	4	5	9	10	12	20
----	---	---	---	---	---	---	----	----	----

Merge Sort

Aplicando a ideia de
intercalação no vetor:
14, 7, 3, 12, 9, 11, 6, 2



Merge Sort

- O algoritmo merge sort possui uma estrutura recursiva: o problema vai sendo quebrado em problemas menores até chegar ao caso base, e então as soluções são agrupadas para gerar a solução geral.
- Isso indica uma estrutura recursiva.

Merge Sort

Procedimento: MERGE-SORT

Entradas:

- A : um vetor.
- p, r : índices iniciais e finais de um subvetor de A .

Resultado: Os elementos do subvetor $A[p\dots r]$ ordenados em ordem crescente.

1. Se $p < r$, faça o seguinte:
 - a. Iguale q a $(p + r) / 2$.
 - b. Chame recursivamente $\text{MERGE-SORT}(A, p, q)$.
 - c. Chame recursivamente $\text{MERGE-SORT}(A, q + 1, r)$.
 - d. Chame $\text{MERGE}(A, p, q, r)$.
2. Caso contrário, o subvetor $A[p\dots r]$ tem, no máximo, um elemento. Portanto, já está ordenado. Apenas retorne sem fazer nada.

Merge Sort

Onde MERGE é a função que faz a intercalação de dois subvetores (já ordenados) de A:

Merge Sort

Procedimento: MERGE

Entradas:

- A : um vetor.
- p, q, r : índices para A . Considera-se que cada um dos subvetores $A[p...q]$ e $A[q + 1...r]$ já está ordenado.

Resultado: O subvetor $A[p...q]$ contém os elementos originalmente em $A[p...q]$ e $A[q + 1...r]$, mas agora o subvetor $A[p...q]$ inteiro já está ordenado.

1. Iguale $n1$ a $q - p + 1$ e iguale $n2$ a $r - q$.
2. Sejam $B[0...n1]$ e $C[0...n2]$ novos vetores.
3. Copie $A[p...q]$ para $B[0...n1]$ e $A[q + 1...r]$ para $C[0...n2]$.
4. Enquanto i for menor que $n1$ e j for menor que $n2$:
 - a. Se $B[i] \leq C[j]$, então:
 - i. iguale $A[k]$ a $B[i]$ e incremente i .
 - b. Caso contrário:
 - i. iguale $A[k]$ a $C[j]$ e incremente j .
 - c. Incremente k .

Merge Sort

A figura traz um exemplo de ordenação usando o merge sort onde os números em *itálico* mostram a ordem na qual as chamadas são feitas.

Merge Sort

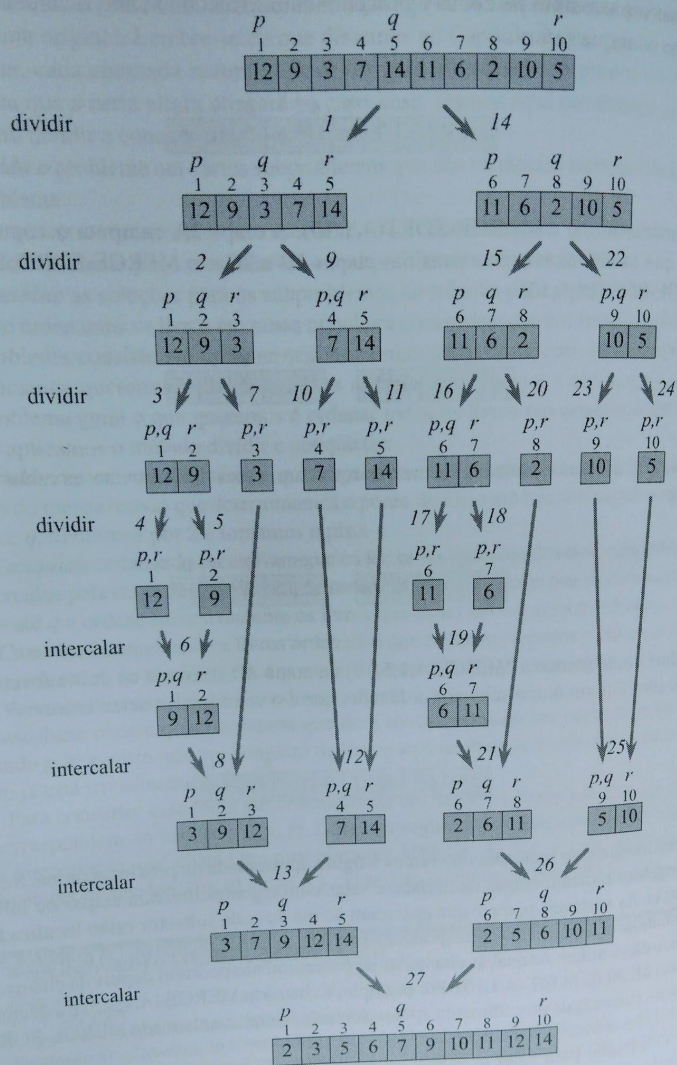
Procedimento: MERGE-SORT

Entradas:

- A : um vetor.
- p, r : índices iniciais e finais de um subvetor de A .

Resultado: Os elementos do subvetor $A[p...r]$ ordenados em ordem crescente.

- Se $p < r$, faça o seguinte:
 - Igual q a $(p + r) / 2$.
 - Chame recursivamente $\text{MERGE-SORT}(A, p, q)$.
 - Chame recursivamente $\text{MERGE-SORT}(A, q + 1, r)$.
 - Chame $\text{MERGE}(A, p, q, r)$.
- Caso contrário, o subvetor $A[p...r]$ tem, no máximo, um elemento. Portanto, já está ordenado. Apenas retorne sem fazer nada.



Merge Sort

Complexidade de tempo:

- Independente de o vetor já estar ordenado de forma crescente (melhor caso) ou de forma decrescente (pior caso), o algoritmo sempre irá dividir o problema por dois até chegar ao caso base e então juntar as soluções novamente. Assim, a complexidade será a mesma nos dois casos.

Merge Sort

Complexidade de tempo:

- A parte de intercalação, que chamamos de MERGE terá no máximo custo $O(n)$, quando precisar intercalar o vetor inteiro na última etapa. A partir disso é possível montar a relação de recorrência do merge sort:

$$T(n) = 2T(n / 2) + n$$

$$T(1) = 1$$

Merge Sort

Complexidade de tempo:

Resolvendo alguns termos:

$$T(n) = 2T(n / 2) + n$$

$$2T(n / 2) = 2(2T(n / 4) + n / 2) = 4T(n / 4) + n$$

$$4T(n / 4) = 4(2T(n / 8) + n / 4) = 8T(n / 8) + n$$

$$8T(n / 8) = 8(2T(n / 16) + n / 8) = 16T(n / 16) + n$$

...

Merge Sort

Complexidade de tempo:

Generalizando:

$$T(n) = 2T(n / 2) + n$$

$$2T(n / 2) = 4T(n / 4) + n$$

$$4T(n / 4) = 8T(n / 8) + n$$

$$8T(n / 8) = 16T(n / 16) + n$$

...

$$2^k T(n / 2^k) = 2^{k+1} T(n / 2^{k+1}) + n$$

Merge Sort

Complexidade de tempo:

Isso irá se repetir até que n seja igual a 1, ou seja até 2^k ser igual a n .

$$T(n) = 2T(n/2) + n$$

$$2T(n/2) = 4T(n/4) + n$$

$$4T(n/4) = 8T(n/8) + n$$

$$8T(n/8) = 16T(n/16) + n$$

...

$$nT(n/n) = n * 1$$

Se $2^k = n$, então $k = \log_2 n$, ou seja, o termo n deve ser somado $\log_2 n$ vezes, levando a uma complexidade de tempo $O(n \log_2 n)$.

Merge Sort

Complexidade de espaço:

- Até então analisamos a complexidade de espaço basicamente pelo tamanho da pilha de chamadas da função.
- Vimos por exemplo que os algoritmos de ordenação por seleção, inserção e bolha tinham complexidade de espaço $O(1)$.
- Esses algoritmos, além de não serem recursivos, têm um fato em comum: todos conseguem ordenar um vetor sem precisar de um vetor auxiliar, pois vão realizando trocas dentro do próprio vetor.

Merge Sort

Complexidade de espaço:

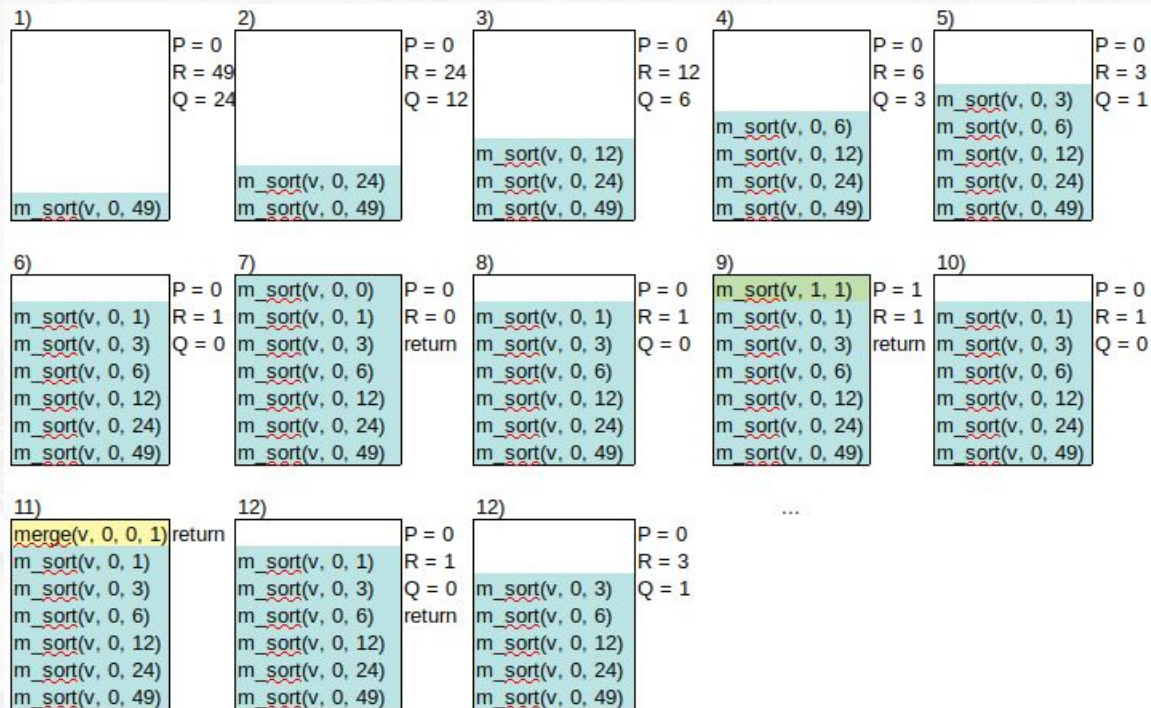
- No merge sort utilizamos vetores auxiliares na função de intercalação.
- São criados dois vetores: um com a metade esquerda do vetor recebido por parâmetro e um com a metade direita do vetor recebido por parâmetro.
- Nesta situação, teremos que analisar a complexidade de espaço considerando essa memória extra que o algoritmo irá gastar com os vetores auxiliares.

Merge Sort

Complexidade de espaço:

- Se fossem consideradas apenas as chamadas de função, ao montar a pilha de chamadas, veríamos que para uma entrada de tamanho n , até $\log_2 n$ chamadas seriam feitas até gerar o resultado final.
- Considere $n = 50$:

Merge Sort



Merge Sort

Complexidade de espaço:

Entretanto temos que considerar que na fase de intercalação:

- A 1ª chamada vai precisar de 2 vetores de tamanho $n/2$: $2 \times n / 2 = n$
- A 2ª chamada vai precisar de 2 vetores de tamanho $n/4$: $2 \times n / 4 = n / 2$
- A 3ª chamada vai precisar de 2 vetores de tamanho $n/8$: $2 \times n / 8 = n / 4$
- ...

Ao somar a memória extra para cada chamada temos:

$$n + n/2 + n/4 + n/8 + \dots$$

$$n + (1/2 + 1/4 + 1/8 + \dots) \rightarrow O(n)$$

Merge Sort

Observações:

Embora o merge sort tenha complexidade de tempo $O(n \log n)$, que é menor do que a complexidade de tempo da ordenação por inserção, por ex., que é $O(n^2)$, os fatores constantes da ordenação por inserção podem torná-la mais rápida para um n pequeno.

Merge Sort

Atividades:

1. Implementar o algoritmo MERGE (intercalação - esboço no no github).
2. Desenvolver uma versão não recursiva para o merge sort.

Merge Sort

Referências

- Cormen, T. H., “Desmistificando algoritmos”. 1ª ed. Rio de Janeiro, Elsevier, 2014.
- Explicação da pilha de chamadas do merge sort:
<https://www.youtube.com/watch?v=RZK6KVpaT3I>
- Overview Mergesort:
<https://www.khanacademy.org/computing/computer-science/algorithms/merge-sort/a/overview-of-merge-sort>