

Algoritmos

Pedro Hokama

- [cirs] Algoritmos: Teoria e Prática (Terceira Edição) Thomas H. Cormen, Charles Eric Leiserson, Ronald Rivest e Clifford Stein.
- [timr] Algorithms Illuminated Series, Tim Roughgarden
- Desmistificando Algoritmos, Thomas H. Cormen.
- Algoritmos, Sanjoy Dasgupta, Christos Papadimitriou e Umesh Vazirani
- Stanford Algorithms
<https://www.youtube.com/playlist?list=PLXFMlk03Dt7Q0xr1PIAriY5623cKiH7V>
<https://www.youtube.com/playlist?list=PLXFMlk03Dt5EMI2s2WQBsLsZ17A5HEK6>
- Conjunto de Slides dos Professores Cid C. de Souza, Cândida N. da Silva, Orlando Lee, Pedro J. de Rezende
- Conjunto de Slides do Professores Cid C. de Souza para a disciplina MO420
Qualquer erro é de minha responsabilidade.

1 / 20

2 / 20

John von Neumann

- John von Neumann (1903 - 1957) nascido na Hungria e de origem judaica. Naturalizado americano em 1937.
- Foi membro do Instituto de Estudos Avançados de Princeton, Nova Jérsei, do qual fazia parte Albert Einstein, Kurt Gödel e vários outros grandes cientistas.
- Dentre diversas contribuições para a matemática, ciência da computação, física, etc. Neumann descreveu em 1945 o algoritmo MergeSort.



MergeSort (Ordenação por Intercalação)

Por que veremos um algoritmo de 1945?

- É o algoritmo de escolha ainda hoje por ser realmente eficiente
- Muito melhor do que a complexidade quadrática (InsertionSort, SelectionSort, etc)

Por que veremos novamente o MergeSort?

- É claro sobre o método de divisão e conquista
- Vai preparar vocês para análises de complexidade mais complicadas.

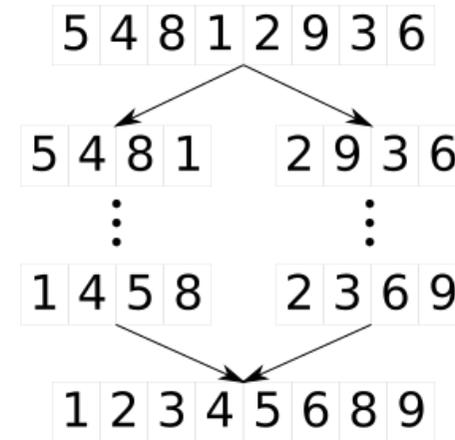
3 / 20

4 / 20

MergeSort

Problema da Ordenação

Dado um arranjo de n inteiros distintos, encontrar o arranjo $\pi = (\pi_1, \pi_2, \dots, \pi_n)$ que contenha os mesmos elementos mas ordenados de maneira não decrescente, ou seja, $\pi_i \leq \pi_j$ para qualquer $i < j$ e $i, j \in \{1, \dots, n\}$.



5 / 20

6 / 20

Pseudo-Código para o MergeSort:

Algoritmo 1: MergeSort

Entrada: Um arranjo com n

Saída: Um arranjo com os mesmos números ordenados

- 1 se $n \geq 2$ então
 - 2 A = Recursivamente ordenar a primeira metade do arranjo de entrada;
 - 3 B = Recursivamente ordenar a segunda metade do arranjo de entrada;
 - 4 C = Intercalar (Merge) as duas partes ordenadas A e B em uma;
 - 5 devolva C ;
-

Pseudo-Código para o Merge:

Algoritmo 2: Merge

Entrada: A e B arranjos ordenados com $m/2$

Saída: Arranjo C de tamanho m com os mesmos elementos de A e B mas ordenados

- 1 $i = 1$;
 - 2 $j = 1$;
 - 3 para k de 1 até m faça
 - 4 se $A[i] < B[j]$ então
 - 5 $C[k] = A[i]$;
 - 6 $i++$;
 - 7 senão
 - 8 $C[k] = B[j]$;
 - 9 $j++$;
 - 10 devolva C ;
 - 11 Exercício: verificar finalizações (se A ou B acabarem etc).
-

7 / 20

8 / 20

MergeSort

- Qual o tempo de execução do MergeSort? Quantas operações básicas faz o MergeSort? Qual o número de linhas de código executadas pelo MergeSort?
- Primeiro nos perguntaremos qual o tempo de execução do Merge?

Pseudo-Código para o Merge:

Algoritmo 3: Merge

Entrada: A e B arranjos ordenados com $m/2$

Saída: Arranjo C de tamanho m com os mesmos elementos de A e B mas ordenados

```
1  $i = 1;$ 
2  $j = 1;$ 
3 para  $k$  de 1 até  $m$  faça
4   se  $A[i] < B[j]$  então
5      $C[k] = A[i];$ 
6      $i++;$ 
7   senão
8      $C[k] = B[j];$ 
9      $j++;$ 
10  fim
11 fim
12 devolva  $C;$ 
13 Exercício: verificar finalizações (se  $A$  ou  $B$  acabarem etc).
```

Um incremento de k e uma comparação

Essas 2 ou essas 2

m vezes

Um total de $4m + 2 \leq 6m$

9 / 20

10 / 20

Complexidade do MergeSort

Teorema

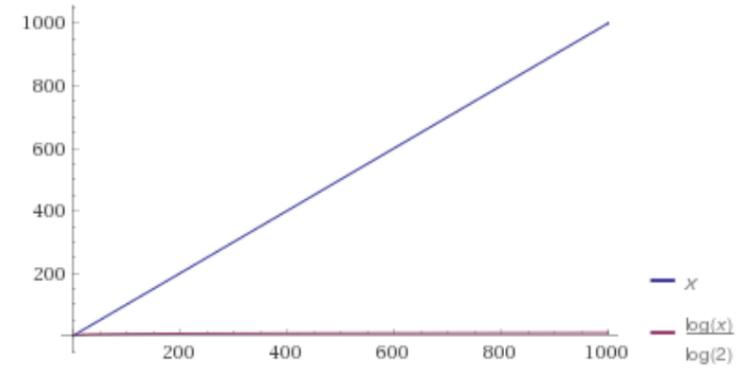
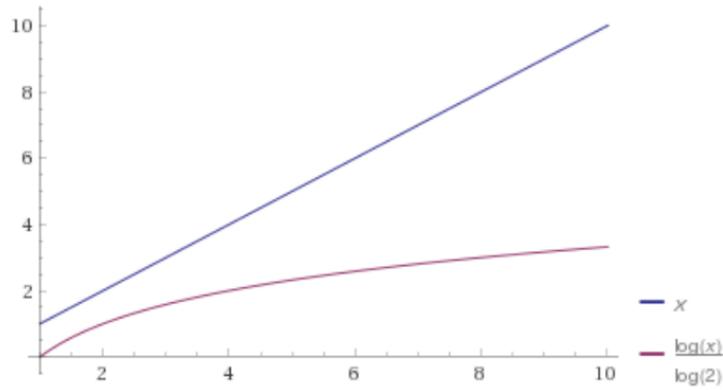
MergeSort exige menos de $6n \log_2 n + 6n$ operações para ordenar n números.

- Antes de provar esse teorema, nos perguntamos, será que esse limitante é bom?
- Relembre que os algoritmos mais triviais exigiam uma contante vezes n^2 , enquanto o MergeSort é uma constante vezes $n \log_2 n$.
- Outra breve lembrança é do que é um \log_2 , de maneira informal podemos dizer que \log_2 de um número, é a quantidade de vezes que você precisa dividir por 2 até chegar em 1.
- Então o $\log_2 4$ é 2, $\log_2 8 = 3$, $\log_2 16384 = 14$. Ou seja $\log_2 n$ é uma função que cresce devagar.

- Analisar o MergeSort é um pouco mais desafiador pois cada problema faz 2 chamadas recursivas, causando uma explosão de subproblemas. A boa notícia é que cada vez que dividimos um subproblema em dois, cada um deles tem metade do tamanho.
- De fato é esse equilíbrio entre a quantidade de subproblemas e o tamanho de cada subproblema que vai ditar a complexidade de um algoritmo recursivo.

11 / 20

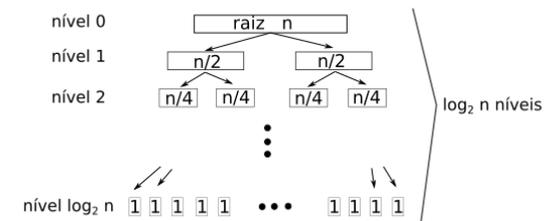
12 / 20



Complexidade do MergeSort

- Para demonstrar a complexidade do MergeSort iremos usar um recurso conhecido como árvore de recursão
- Ressalva: Alguns autores não consideram uma árvore de recursão como uma prova completa para uma afirmação.

Árvore de Recursão



Aproximadamente quantos níveis tem essa árvore de recursão?
Sendo n o número de elementos no vetor.

- a Um número constante (independente de n)
- b $\log_2 n$
- c \sqrt{n}
- d n

17 / 20

Qual o padrão? Em cada nível $j = 0, 1, \dots, \log_2 n$, existem ___ subproblemas, cada um com tamanho ___.

- a 2^j e 2^j
- b $n/2^j$ e $n/2^j$
- c 2^j e $n/2^j$
- d $n/2^j$ e 2^j

19 / 20

Resposta:

- $\log_2 n + 1$ (nível 0)

18 / 20

Teorema

MergeSort exige menos de $6n \log_2 n + 6n$ operações para ordenar n números.

Demonstração.

- Em cada nível $j = 0, 1, \dots, \log_2 n$ existem 2^j subproblemas, cada um de tamanho $n/2^j$.
- Total de operações no nível j :

$$\leq 2^j \cdot 6 \left(\frac{n}{2^j} \right)$$

$$= 6(n)$$

- Total de operações: número de níveis \cdot operações por nível

$$(\log_2 n + 1)6n$$

$$6n \log_2 n + 6n$$

□ 20 / 20