Algoritmos

Pedro Hokama

Fontes

- [*clrs*] 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=PLXFMmlk03Dt7Q0xr1PIAriY5623cKiH7V`
  `https://www.youtube.com/playlist?list=PLXFMmlk03Dt5EMI2s2WQBsLsZl7A5HEK6`
- 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.

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)

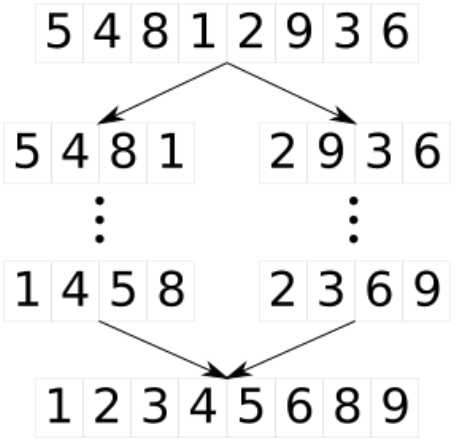
Por que veremos novamente o MergeSort?

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

Problema da Ordenação

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

MergeSort



Pseudo-Código para o MergeSort:

Algoritmo 1: MergeSort

Entrada: Um arranjo com n
Saída: Um arranjo com os mesmos números ordenados

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

$i = 1$;
$j = 1$;
para k *de 1 até* m **faça**
    se $A[i] < B[j]$ **então**
        $C[k] = A[i]$;
        $i++$;
    senão
        $C[k] = B[j]$;
        $j++$;
devolva C;
Exercício: verificar finalizações (se A ou B acabarem etc).

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 ⩽ 6m

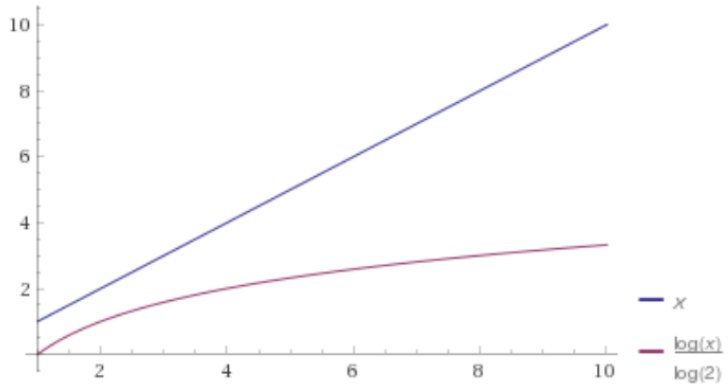
- Analisar o MergeSort é um pouco mais desafiador pois cada problema faz 2 chamadas recursivas, causando uma explosão de subproblemas. A boa notícia é que cada vez que dividimos um subproblema em dois, cada um deles tem metade do tamanho.
- De fato é esse equilíbrio entre a quantidade de subproblemas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

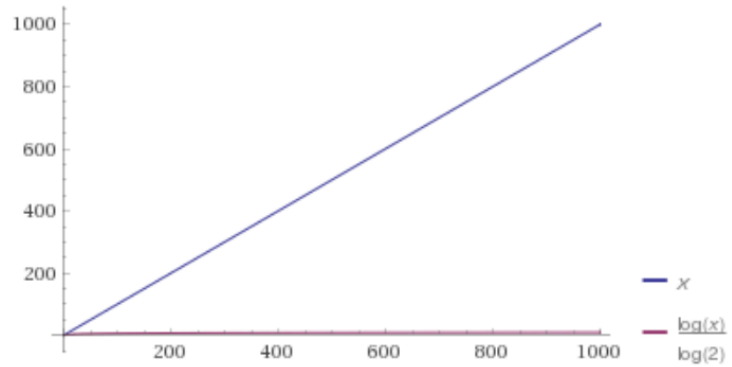
Complexidade do MergeSort

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.

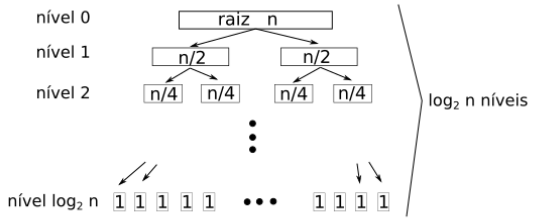




Complexidade do MergeSort

- 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

Resposta:

- $\log_2 n + 1$ (nível 0)

Qual o padrão? Em cada nível $j = 0, 1, \ldots, \log_2 n$, 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

Teorema

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

Demonstração.

- Em cada nível $j = 0, 1, \ldots, \log_2 n$ 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 · operações por nível

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

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

□

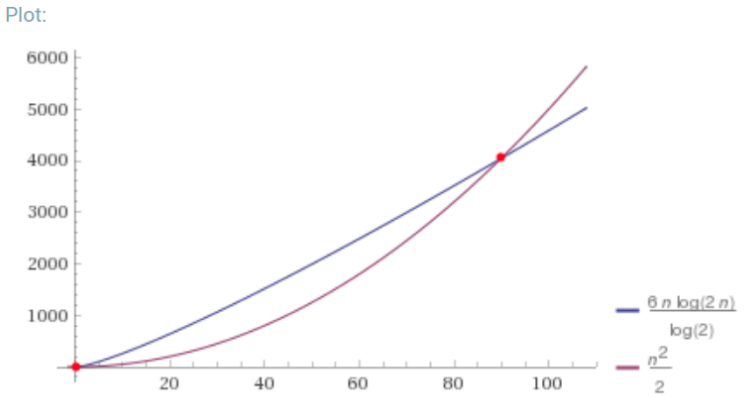
Princípios da Análise de Algoritmos

- O que fizemos no caso do MergeSort foi uma análise de Pior Caso, ou seja, qualquer que seja a entrada sabemos que o algoritmo executaram em tempo $\leq 6n \log_2 n + 6n$.
- Esse limite também é aplicado se um adversário tentasse atribuir números de forma a deixar o algoritmo lento.
- Essa análise é particularmente interessante por não precisar entender a aplicação do problema, padrões de entrada e ela é usualmente mais útil e mais fácil que as alternativas:
  - Análise de Caso Médio (Exige assumir alguma distribuição da entrada, ter conhecimento do domínio, mais difícil de ser feita)
  - Desempenho em *Benchmarks*
  - Análise de Melhor Caso (inútil na maior parte do tempo)

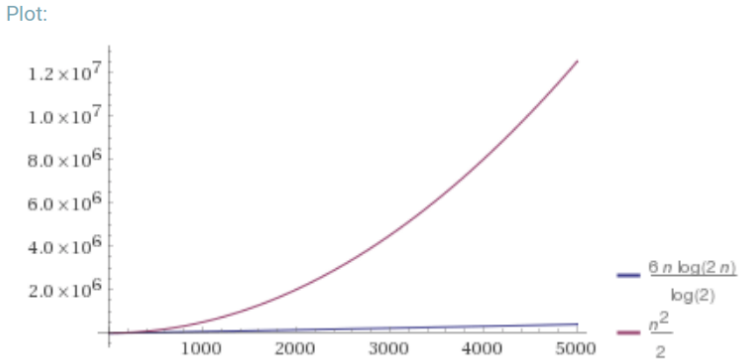
Princípios da Análise de Algoritmos

- Iremos fazer uma Análise **Assintótica** dos algoritmos, o que significa que estamos interessados no comportamento deles para instâncias **grandes**.
- Isso nos permite dizer que um algoritmo é mais rápido que outro assumindo que o tamanho n da instância é suficientemente grande.
- Por exemplo, podemos dizer com segurança que um algoritmo que executa em $6n \log_2 n + 6n$ é mais rápido que um que executa em $\frac{1}{2}n^2$
- Note que isso pode não ser verdade para n pequeno, mas a partir de algum $n = n_0$ sempre será verdade.
- De fato, para n pequeno, tanto faz o algoritmo que você use. Estamos interessados em resolver problemas grandes!

Princípios da Análise de Algoritmos



Princípios da Análise de Algoritmos



Princípios da Análise de Algoritmos

- Você poderia imaginar que com o avanço dos hardwares, bastaria eu usar um computador melhor.
- Na verdade quanto maior o poder computacional, MAIOR é a disparidade entre algoritmos eficientes.
- Podemos pensar no tamanho do problema que podemos resolver com computadores mais potentes usando diferentes algoritmos.
- Suponha que você tem dois algoritmos para um problema.

Algoritmo A	Algoritmo B
n	n^2

- Suponha que você fez um grande investimento e comprou um computador 4 vezes mais potente. Com o algoritmo A você pode resolver um problema 4 vezes maior, enquanto com o algoritmo B você só pode resolver um problema 2 vezes maior.

Análise Assintótica

- É a linguagem que os cientistas da computação (sérios) usam para discutir o desempenho em alto nível de algoritmos.
- É fundamental para o vocabulário do cientista da computação. Quando alguém diz "O MergeSort executa em $\Theta(n \log n)$, e o InsertionSort executa em $o(n^2)$" o que exatamente ele está dizendo?
- A análise assintótica é uma ferramenta adequada pois:
  - É simples o bastante para suprimir detalhes de arquitetura/linguagem/compilador.
  - Mas é complexa o bastante para permitir a comparação entre diferentes algoritmos, especialmente em instâncias grandes.

Análise Assintótica

Ideia geral

Suprimir fatores constantes e termos de ordens inferiores.

- Por exemplo em:

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

6n é um termo de ordem inferior, 6 é constante então resultaria em:

$$n \log n$$

Exercício: A base do log também não importa. Por que?

- Então quando dizemos que o tempo de execução do MergeSort é $O(n \log n)$, ou de maneira geral quando dizemos que um algoritmo é $O(f(n))$. Estamos dizendo que depois de eliminar os termos de ordem inferior e constantes acabamos apenas com $f(n)$.

Análise assintótica de funções quadráticas - termos de menor ordem

Considere a função quadrática $3n^2 + 10n + 50$:

n	$3n^2 + 10n + 50$	$3n^2$
64	12978	12288
128	50482	49152
512	791602	786432
1024	3156018	3145728
2048	12603442	12582912
4096	50372658	50331648
8192	201408562	201326592
16384	805470258	805306368
32768	3221553202	3221225472

- Como se vê, $3n^2$ é o termo dominante quando n é grande.
- De um modo geral, podemos nos concentrar nos termos dominantes e esquecer os demais.