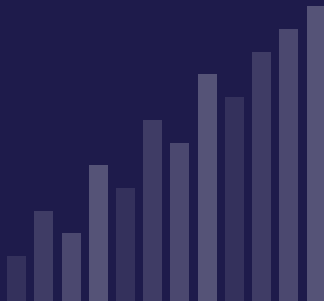

COMPLEMENTO 3 · BLOCO COMPLEMENTAR: ORDENAÇÃO AVANÇADA (OPCIONAL)

O limite inferior da ordenação por comparação

Seção 8.1

Estruturas de dados e seus algoritmos · José Dihego · IFBA



Roteiro da aula

Ao final, você será capaz de

- ▶ Modelar algoritmos de ordenação por comparação como árvores de decisão
- ▶ Provar que todo algoritmo desse tipo faz $\Omega(n \lg n)$ comparações no pior caso
- ▶ Usar a aproximação de Stirling para $\lg(n!)$
- ▶ Medir quão perto do limite o merge sort chega



Teoria

Árvore de decisão, altura e folhas, o teorema 8.1



Prática em C

Pior caso exato do merge sort em todas as $n!$ permutações



Experimento

Merge sort contra $\lceil \lg n! \rceil$ e contra $\lg n!$ para n grande



Conexões

Teoria da informação, compressão, criptografia e computação quântica

Árvores de decisão

- ▶ Um algoritmo de **comparação** só aprende sobre a ordem comparando pares $a_i \leq a_j$.
- ▶ Cada nó interno é uma comparação; cada folha é uma permutação de saída.
- ▶ Uma execução é um caminho da raiz a uma folha: o número de comparações é o comprimento do caminho.
- ▶ O **pior caso** é a **altura** da árvore.

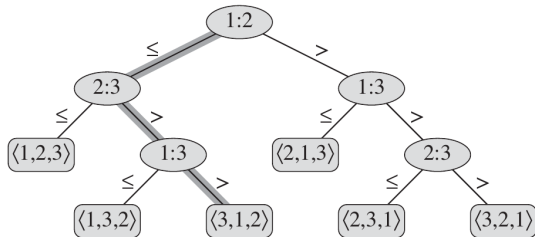


Figura 8.1 de Cormen et al. (2009) — árvore de decisão da inserção para $n = 3$: $3! = 6$ folhas.

Teorema 8.1: $\Omega(n \lg n)$

Todo algoritmo de ordenação por comparação faz $\Omega(n \lg n)$ comparações no pior caso.

Prova. Uma árvore correta precisa de pelo menos $n!$ folhas, uma por permutação. Uma árvore binária de altura h tem no máximo 2^h folhas. Logo

$$n! \leq \ell \leq 2^h \implies h \geq \lg(n!) = \Omega(n \lg n),$$

pois, por Stirling, $\lg(n!) = n \lg n - n \lg e + O(\lg n) \approx n \lg n - 1,44 n$.

Corolário 8.2: heapsort e merge sort são **assintoticamente ótimos** entre os algoritmos de comparação.

O que o limite não diz

- ▶ Vale para **comparações**. Se as chaves são inteiros pequenos, contagem e radix sort ordenam em $O(n)$ (complemento 4).
- ▶ É um limite de **pior caso**, mas também vale para o caso médio: a profundidade média das folhas de uma árvore binária com $n!$ folhas é pelo menos $\lg(n!)$ (problema 8-1).
- ▶ Não diz qual algoritmo é mais rápido na prática: constantes, memória e cache decidem entre heapsort, merge sort e quicksort.
- ▶ **Informação:** cada comparação dá no máximo 1 bit; identificar uma de $n!$ permutações exige $\lg(n!)$ bits.



Prática em C

- ▶ Gera as $n!$ permutações (algoritmo de Heap)
- ▶ Guarda o maior número de comparações do merge sort
- ▶ Compara com $\lceil \lg n! \rceil$, calculado com `lgamma`

A altura da árvore do merge sort (1/3)

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <math.h>
4
5  /* Percorre TODAS as n! permutações (algoritmo de Heap) e guarda o maior número
6     de comparações que o merge sort fez: a altura da sua árvore de decisão. */
7  long comp;
8  int aux[16];
9
10 void merge_sort(int A[], int p, int r) {
11     if (p >= r) return;
12     int q = (p + r) / 2;
13     merge_sort(A, p, q);
14     merge_sort(A, q + 1, r);
```

A altura da árvore do merge sort (2/3)

```
15     for (int k = p; k <= r; k++) aux[k] = A[k];
16     for (int k = p, i = p, j = q + 1; k <= r; k++)
17         if (i > q) A[k] = aux[j++];
18         else if (j > r) A[k] = aux[i++];
19         else A[k] = (comp++, aux[i] <= aux[j]) ? aux[i++] : aux[j++];
20 }
21
22 int main(void) {
23     int n;
24     if (scanf("%d", &n) != 1 || n < 1 || n > 11) return 1;
25     int P[16], c[16] = {0}, A[16];
26     for (int k = 0; k < n; k++) P[k] = k;
27     long pior = 0;
28     for (int i = 0;;) {                                     // algoritmo de Heap (iterativo)
29         for (int k = 0; k < n; k++) A[k] = P[k];
```

A altura da árvore do merge sort (3/3)

```
30     comp = 0;
31     merge_sort(A, 0, n - 1);
32     if (comp > pior) pior = comp;
33     while (i < n && c[i] >= i) c[i++] = 0;    // próxima permutação
34     if (i >= n) break;
35     int t, j = i % 2 ? c[i] : 0;
36     t = P[j]; P[j] = P[i]; P[i] = t;
37     c[i]++;
38     i = 0;
39 }
40 double lg_fat = lgamma(n + 1) / log(2);      // lg n! = ln Γ(n+1) / ln 2
41 printf("n = %d: limite %.0f, merge sort %ld\n", n, ceil(lg_fat - 1e-9), pior);
42 return 0;
43 }
```



Experimento

- ▶ Pior caso exato: todas as $n!$ permutações, $n = 2..10$
- ▶ Média para n até 2^{20} contra $\lg n!$
- ▶ Nenhum algoritmo pode ficar abaixo do limite

O merge sort perto do limite

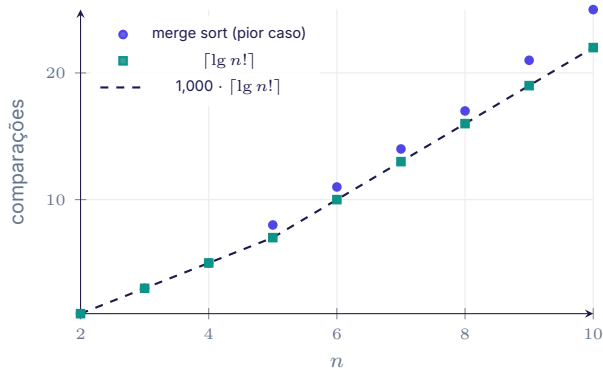
Hipótese. Para todo n , o pior caso do merge sort é $\geq \lceil \lg n! \rceil$ (teorema 8.1). Para n grande, o merge sort faz em média $n \lg n - 1,26 n$, e $\lg n! \approx n \lg n - 1,44 n$: a diferença deve ser de apenas $\approx 0,18 n$, menos de 1% para $n = 2^{20}$.

Método

1. $n = 2..10$: todas as permutações (até 3.628.800)
2. $n = 2^{10}..2^{20}$: média de 3 vetores aleatórios
3. $\lg n! = \ln \Gamma(n+1) / \ln 2$ via `lgamma`
4. Verificação: medido $\geq$ limite em todos os n

🧪 EXPERIMENTO

Pior caso exato contra o limite



<i>n</i>	medido	razão
2,000	1,000	1,000
3,000	3,000	1,000
4,000	5,000	1,000
5,000	8,000	1,143
6,000	11,0	1,100
10,0	25,0	1,136

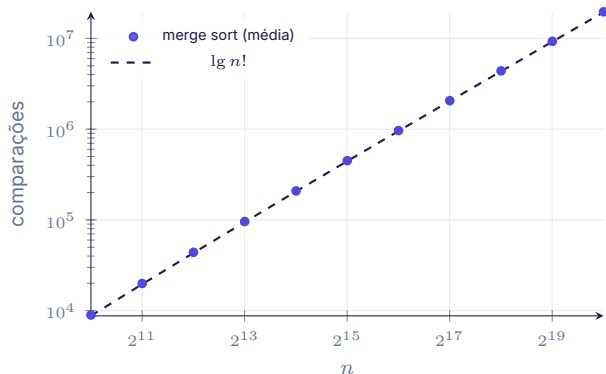
razão = medido $\div$ $\lceil \lg n! \rceil$

✓ sempre $\geq$ o limite

Até $n = 4$, o merge sort é ótimo. Para $n = 10$, faz 25 comparações no pior caso; o limite é 22.

🧪 EXPERIMENTO

Média do merge sort contra $\lg n!$



n	medido	razão
1.024	8.952	1,021
4.096	43.985	1,017
16.384	208.612	1,014
65.536	965.684	1,012
262.144	4.387.099	1,011
1.048.576	19.646.487	1,010

razão = medido $\div$ $\lg n!$

✓ desvio máx. 1,3%

A razão medido/ $\lg n!$ cai para 1,01: o merge sort está a 1% do ótimo teórico.

EXPERIMENTO

O que o experimento mostrou

Em todos os n testados, o pior caso do merge sort ficou **acima** de $\lceil \lg n! \rceil$, como o teorema exige, e para $n \leq 4$ ele atinge o limite. Para $n = 2^{20}$, o merge sort faz só 1% mais comparações que $\lg n!$. Não há como fazer muito melhor **comparando**; para ir além, é preciso usar a estrutura das chaves.

➤ Reproduza: `gcc -O2 compl3_experimento.c -o exp -lm && ./exp`



Conexões

- ▶ Teoria da informação
- ▶ Compressão
- ▶ Criptografia
- ▶ Computação quântica

Onde isso aparece na computação

Teoria da informação

O argumento é de Shannon: uma comparação gera no máximo 1 bit. O mesmo raciocínio dá limites para busca ($\lg n$) e para jogos como "vinte perguntas".

Compressão

Nenhum compressor sem perdas reduz todos os arquivos: há 2^n arquivos de n bits e menos códigos menores. É o argumento de contagem das folhas.

Criptografia

Uma chave de 128 bits tem 2^{128} possibilidades: um ataque de força bruta precisa de $\Omega(2^{127})$ tentativas em média. Limites inferiores são a base da segurança.

Computação quântica

O algoritmo de Grover busca em $O(\sqrt{n})$, e há prova de que nenhum algoritmo quântico faz melhor: limites inferiores também valem no modelo quântico.

Para casa

Exercícios

1. (8.1-1) Qual a menor profundidade possível de uma folha numa árvore de decisão de ordenação?
2. (8.1-3) Mostre que não existe ordenação por comparação linear para metade das $n!$ entradas.
3. (8.1-4) Limite inferior para ordenar n/k subsequências de k elementos cada.
4. Use `aula13_arvore_decisao.c` para achar o pior caso da **inserção** até $n = 10$.
5. Por que $\lceil \lg 5! \rceil = 7$, mas o merge sort precisa de 8? Existe algoritmo com 7? (Pesquise "merge insertion de Ford e Johnson.")

Leitura

Cormen et al., seção 8.1; aproximação de Stirling (equação 3.18).

Código

josedihego.net/estruturas