ORDENAÇÃO ÓTIMA

José de Siqueira

DCC - UFMG

2º semestre de 2004

Ordenação Ótima

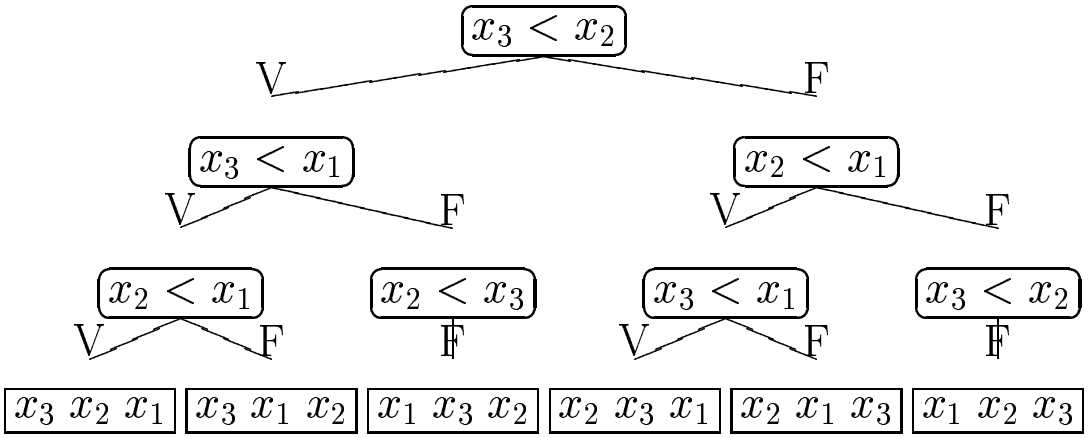
- Existe algum método de ordenação que seja "melhor" que todos os outros?
- Quais critérios de comparação de métodos e para quais métodos esses critérios são significativos?
- Restrição à classe de algoritmos de ordenação por comparação (OC) de chaves entre si, 2 a 2, de itens a serem ordenados.
- **Hipótese:** as chaves são distintas umas das outras.
- *A complexidade, em nº de comparações, tanto na média, quanto no pior caso, de todo algoritmo da classe OC é de ordem de grandeza igual ou superior a* $n \log_2 n$.
- Assim, os algoritmos de ordenação rápida e ordenação com monte são **ótimos** na média.
- Ordenação com monte é também **ótimo** no pior caso.
- O nº de movimentações de itens durante a ordenação é critério importante, além das comparações.

- O gasto em memória necessária para a execução do algoritmo também é um critério importante.
- A ordenação dicotômica é ótima, no pior caso, em nº de comparações. Mas o nº de movimentações é $\Theta(n^2)$.
- Na prática, o tempo de execução desse algoritmo é proporcional a n^2.
- Como provar o limite inferior para o nº de comparações dos algoritmos da classe OC?

Árvores de decisão

- A análise das **árvores de decisão** permite calcular a complexidade ótima da classe OC em nº de comparações média e no pior caso.

- Árvore de decisão da ordenação da bolha (bubble sort) para uma lista de 3 itens x_1, x_2, x_3, armazenados em v[1],v[2],v[3]:



- Como a ordenação da bolha não é ótima, certas comparações podem ser efetuadas várias vezes durante a execução do algoritmo.
- **Lema:** *A árvore binária de decisão associada a um algoritmo de ordenação por comparações de n chaves tem exatamente $n!$ folhas.*
- As complexidades ótimas no pior caso de toda classe de algoritmos que podem ser representados por árvores de decisão se exprimem em função da altura dessas árvores.

- Além disso, sabe-se que o conhecimento do nº de folhas de uma árvore binária permite determinar os limites inferiores de sua altura e de sua profundidade média.
- **Notação:** $\mathrm{OC}_{\max}(n)$ e $\mathrm{OC}_{\mathrm{med}}(n)$ denotam o nº mínimo de comparações para n chaves que deve efetuar, no pior caso (resp. no caso médio), todo algoritmo da classe OC.

Limite inferior para a complexidade do pior caso

- Toda árvore binária com k folhas tem uma altura superior ou igual a $\lceil \log_2 k \rceil$.
- Resulta que o nº máximo de comparações para ordenar n itens é:

$$\mathrm{OC}_{\max}(n) \geq \lceil \log_2(n!) \rceil$$

- Existe algum algoritmo para o qual o limite inferior é atingido?
- Podemos nos contentar com um algoritmo com uma complexidade no pior caso da mesma ordem de grandeza de $\lceil \log_2(n!) \rceil$.
- É a complexidade ótima em *ordem de grandeza*.

- Podemos também procurar um algoritmo que efetue no pior caso exatamente $\lceil \log_2(n!) \rceil$ comparações.
- É a complexidade ótima exata.

Complexidade ótima em

ordem de grandeza

- Desenvolvimento assintótico de $n!$ dado pela fórmula de Stirling:

$$n! = \sqrt{2\pi n}(\frac{n}{e})^n.(1 + 1/12n + o(1/n))$$

$$\begin{aligned} \log_2(n!) &= \log_2 \sqrt{2\pi n} + n \log_2(\tfrac{n}{e}) + \\ &\quad + \log_2(1 + 1/12n + o(1/n)) \end{aligned}$$

$$\begin{aligned} \log_2(n!) &= n \log_2 n - n \log_2 e + \\ &\quad + \tfrac{1}{2} \log_2 n + \tfrac{1}{2} \log_2 2\pi + \\ &\quad + \log_2(1 + 1/12n + o(1/n)) \end{aligned}$$

- Logo, $\lceil \log_2(n!) \rceil = \Theta(n \log n)$.

Complexidade ótima exata

- Princípio da ordenação por fusão (mergesort):
  1. Divide-se a lista em duas partes.
  2. Ordena-se cada uma das partes.
  3. Funde-se as duas sublistas ordenadas em uma só.
- Esse algoritmo utiliza memória extra para a fusão. É usado para ordenação externa.
- No pior caso, a operação de fusão necessita $n-1$ comparações.
- O nº máximo de comparações para ordenar n itens com este algoritmo verifica a eq. de recorrência abaixo:

$$\begin{cases} F(n) = n - 1 + F(\lfloor n/2 \rfloor) + F(\lceil n/2 \rceil) \\ F(0) = 0 \end{cases}$$

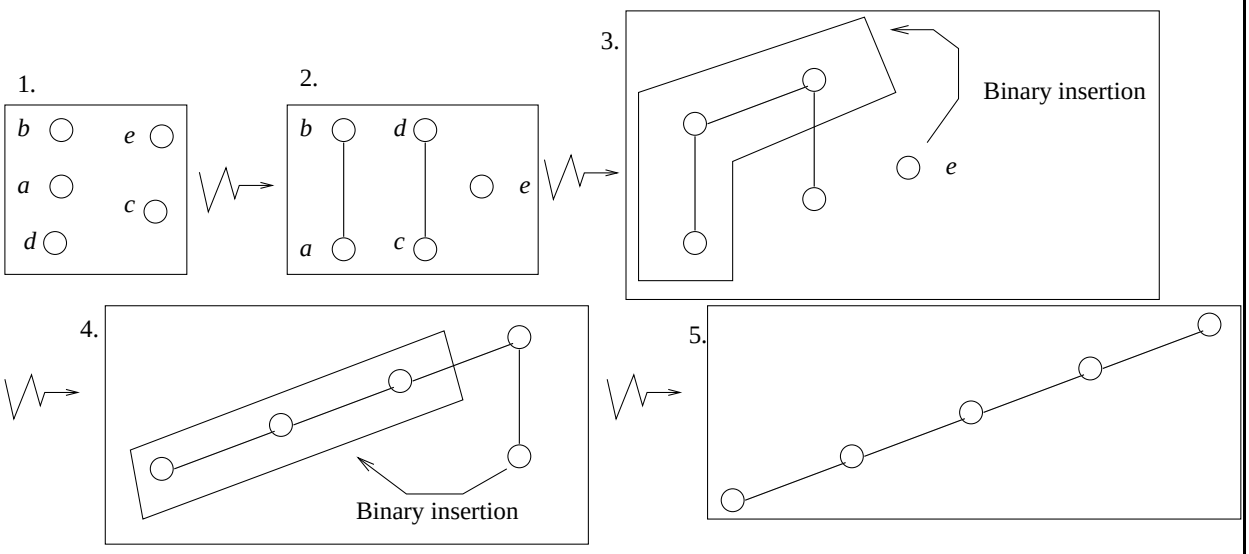
  cuja solução é

$$F(n) = n\lceil \log_2 n \rceil - 2^{\lceil \log_2 n \rceil} + 1$$

$$F(n) = n\lceil \log_2 n \rceil - 2^{\lceil \log_2 n \rceil} + 1$$

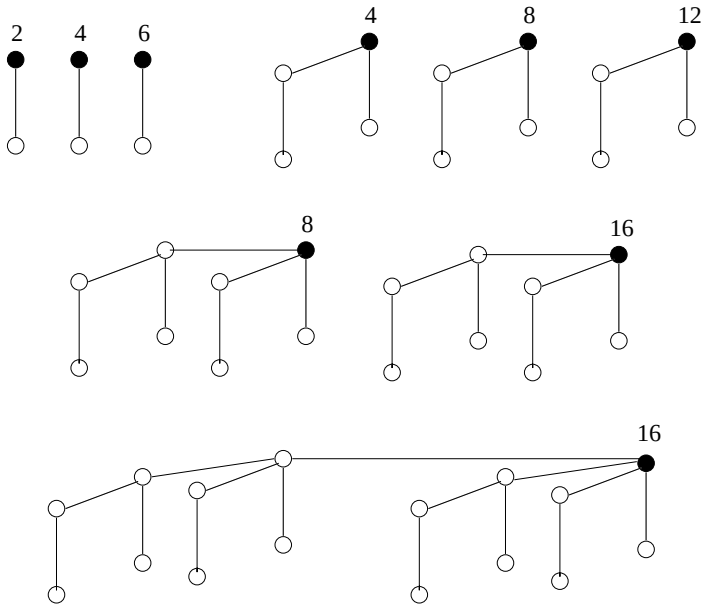
n	=	1	2	3	4	5	6	7	8	9	10	11	12
$\lceil \log_2 n! \rceil$	=	0	1	3	5	7	10	13	16	19	22	26	29
$F(n)$	=	0	1	3	5	8	11	14	17	21	25	29	33

- Mas é possível ordenar 5 itens com 7 comparações, no pior caso!

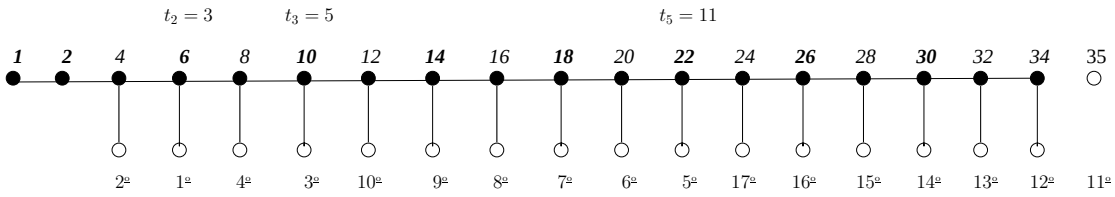


- A generalização deste algoritmo foi proposta for Ford e Johnson em 1959.
- Knuth o chama de *merge insertion*.

- Para ordenar n elementos:
  1. Forme $\lfloor n/2 \rfloor$ pares de elementos comparando-os dois a dois. (Se n é ímpar, deixe um elemento de fora.)



  2. Ordene os $\lfloor n/2 \rfloor$ elementos maiores (max), encontrados no passo (1), recursivamente:



  3. Insira os elementos *pendentes* (bolinhas brancas) na cadeia principal (bolinhas pretas), segundo a ordem da seqüência abaixo:

$$(t_1, t_2, t_3, t_4, \ldots) = (1, 3, 5, 11, \ldots)$$

- Os t_k são definidos por

$$t_k = (2^{k+1} + (-1)^k)/3$$

- Seja $F(n)$ o nº de comparações necessárias para ordenar n elementos pelo algoritmo FJ.
- $F(n) = \lfloor n/2 \rfloor + F(\lfloor n/2 \rfloor) + G(\lceil n/2 \rceil)$.
- $G(\lceil n/2 \rceil)$ é o número de comparações necessárias para inserir elementos pendentes na cadeia principal.
-

$$F(n) = \sum_{k=1}^{n} \lceil log_2(\tfrac{3}{4}k) \rceil$$

- Desta equação, construímos a seguinte tabela para $F(n)$:

n	=	1	2	3	4	5	6	7	8	9	10	11	12
$\lceil \log_2 n! \rceil$	=	0	1	3	5	7	10	13	16	19	22	26	29
$F(n)$	=	0	1	3	5	7	10	13	16	19	22	26	30

n	=	13	14	15	16	17	18	19	20	21	22	23
$\lceil \log_2 n! \rceil$	=	33	37	41	45	49	53	57	62	66	70	75
$F(n)$	=	34	38	42	46	50	54	58	62	66	71	76

- No entanto, é impossível ordenar 12 itens em menos de 30 comparações, enquanto que $\log_2(12!) = 29$ (Wells, 1965).
- Não pode existir método de ordenação que faça $\lceil \log_2 n! \rceil$ comparações para todo n.
- Mas o algoritmo FJ é ótimo?
- FJ foi proposto em 1959, mas só em 1979 é que Manacher provou que não.
- Mas e para $n = 13$ a $n = 22$?
- Peczarski mostrou em 2004 que o número mínimo de comparações para $n = 13$ é 34, para $n = 14$ é 38 e que para $n = 22$ é 71.
- Logo FJ é ótimo para esses valores, mas não é ótimo para um número infinito de valores...
- Mas existe algum algoritmo de ordenação que seja ótimo, isto é, que sempre faça o número mínimo de comparações?
- Manacher, Bui e Mai (1989) propõem combinações ótimas do algoritmo de FJ com o algoritmo ótimo de fusão (merge) proposto por Christen (1978).

Algoritmo $_4$FJ

- Algoritmo de FJ:
  1. Forme $\lfloor n/2 \rfloor$ pares de elementos comparando-os dois a dois. (Se n é ímpar, deixe um elemento de fora.)
  2. Ordene os $\lfloor n/2 \rfloor$ elementos maiores (max), encontrados no passo (1), recursivamente.
  3. Insira os elementos *pendentes* (bolinhas brancas) na cadeia principal (bolinhas pretas), segundo a ordem da seqüência dada acima.
- A recursividade gasta memória e tempo no gerenciamento da pilha da máquina virtual.
- **Proposta**: Em vez de formar duplas de início, porque não formar quádruplas? Idéia básica do Algoritmo $_4$FJ.
- **Vantagem**: ganho em memória.
- Para executar o algoritmo $_4$FJ para n elementos, $_4$FJ é chamado recursivamente para uma lista de tamanho $\lfloor n/4 \rfloor$, uma de tamanho $\lfloor n/4^2 \rfloor$, etc.

- Considerando que se use apontadores para o ambiente a ser empilhado, um total de $\Sigma_{i=1}^{\lceil log_4 n \rceil - 1} \lfloor n/4^i \rfloor$ células extras de memória serão usadas.
- Para o algoritmo FJ também, utilizamos $\Sigma_{i=1}^{\lceil log_2 n \rceil - 1} \lfloor n/2^i \rfloor$ células extras para empilhamento a cada chamada rcursiva.
- Para estimar a diferença, suponhamos $n = 4^k$.
- Logo, Para o algoritmo ${}_4$FJ temos

$$\sum_{i=1}^{\lceil log_4 n \rceil - 1} \lfloor \frac{n}{4^i} \rfloor = \sum_{j=1}^{k-1} 4^j \frac{1}{3}(4^k - 1) - 1 = \frac{1}{3}n - \frac{4}{3}$$

- Do mesmo modo, para o algoritmo FJ, obtemos

$$\sum_{i=1}^{\lceil log_2 n \rceil - 1} \lfloor \frac{n}{2^i} \rfloor = \sum_{j=1}^{2k-1} 2^j = 2^{2k} - 1 - 1 = n - 2$$

- Isto significa que o algoritmo ${}_4$FJ utiliza somente 33% do espaço extra utilizado pelo algoritmo FJ!

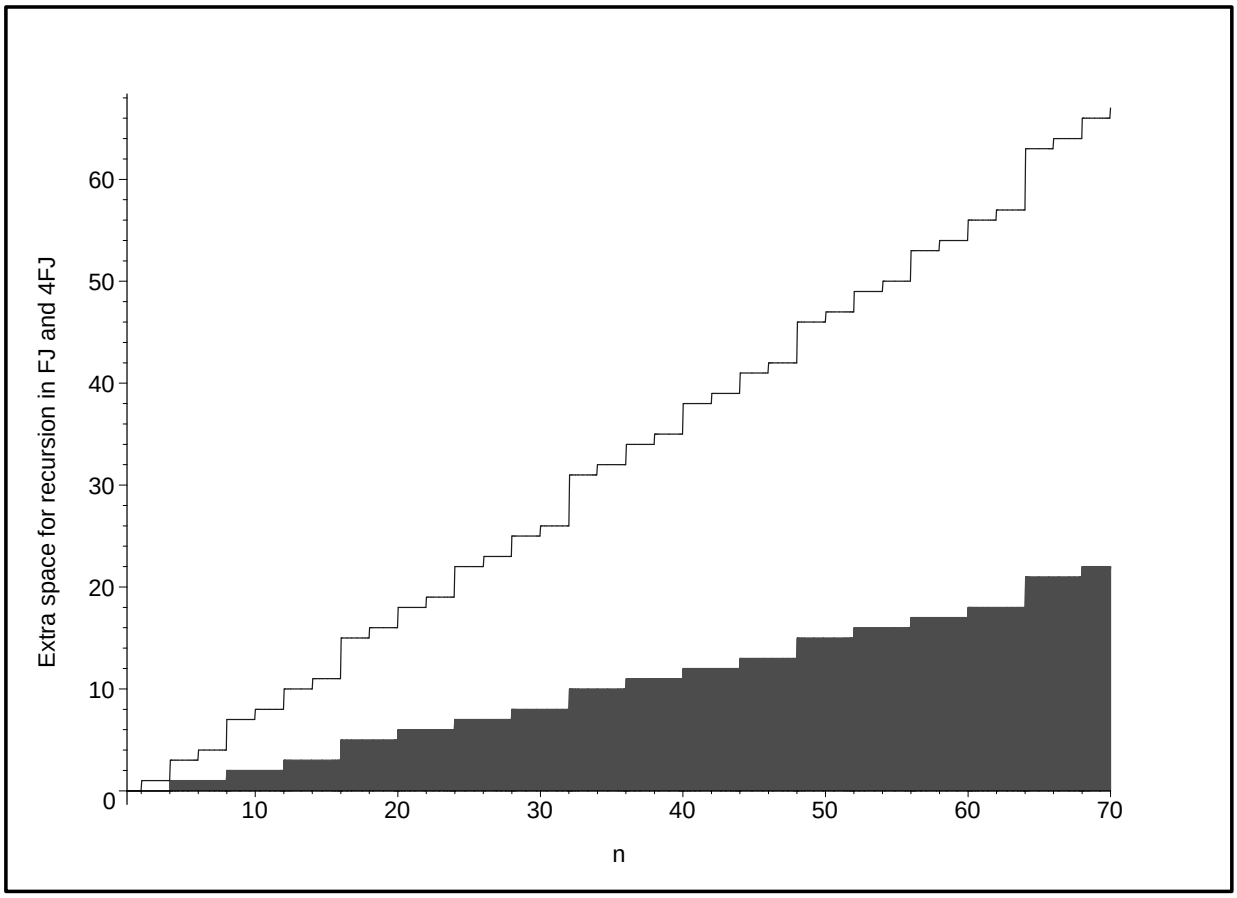


Figure 1: Comparação do espaço extra utilizado pela recursão nos algoritmos FJ e ${}_4$FJ: a área escura representa o espaço utilizado pelo ${}_4$FJ e a area entre as duas curvas representa o ganho relativo (67 %).

- É possível ganhar ainda mais espaço de memória na utilização da pilha?
- Generalização do algoritmo ${}_4$FJ: algoritmo ${}_{2^k}$FJ.
- Se o algoritmo forma a maior tupla possível de início, na prática ele se tornou iterativo!

- Isso permite a eliminação da recursão e, logo, eliminação da necessidade de pilha!
- O **Teorema da Recursão** diz que todo programa recursivo é equivalente a um programa iterativo e vice-versa.

Trabalhos futuros

- O algoritmo de ordenação mais rápido na prática é o algoritmo de ordenação rápida (quicksort).
- Comparar os tempos de execução de ambos (porque a teoria não diferencia qual é o mais rápido deles).
- Melhorar o desempenho do algoritmo ${}_{2^k}$FJ: utilizar indexação indireta.
-

Referências

- M. Ayala-Rincón, B. T. de Abreu, and J. de Siqueira. A Variant of the Ford-Johnson Algorithm that is more Space Efficient. Available `www.mat.unb.br/ ayala/publications.html`, 2004.
- B. T. de Abreu. Buscando uma solução para o problema aberto de Ordenação Ótima Relatório final de Projeto Orientado em Computação II Bacharelado em Ciência da Computação da UFMG 30 de janeiro de 2004
- C. Christen. Improving the Bounds on Optimal Merging. In *Proc. of the 19^{th} Annual IEEE Conference on the Foundations of Computer Science*, pages 259–266. IEEE, 1978.
- H. B. Demuth. *Electronic Data Sorting*. PhD thesis, Stanford University, 1956.
- L. R. Jr. Ford and S. B. Johnson. A tournament problem. *American Mathematical Monthly*, 66(5):387–389, 1959.

- D. E. Knuth. *Sorting and Searching*, volume Volume 3 of The Art of Computer Programming. Reading, Massachusetts: Addison-Wesley, 1973. Also, 2nd edition, 1998.
- D. E. Knuth and E. B. Kaehler. An Experiment in Optimal Sorting. *Information Processing Letters*, 1:173–176, 1972. Reprinted as Chapter 30 in *Selected Papers on Algorithms*, D. E. Knuth, SLSI, 2000.
- G. K. Manacher. The Ford-Johnson sorting algorithm in not optimal. *J. of the ACM*, 26(3):441–456, 1979.
- G. K. Manacher, T. D. Bui, and T. Mai. Optimum Combinations of Sorting and Merging. *J. of the ACM*, 36(2):290–234, 1989.
- J. S. Mönting. Merging of 4 or 5 elements with n elements. *Theoretical Computer Science*, 14:19–37, 1981.
- M. Peczarski. Sorting 13 Elements Requires 34 Comparisons. In *European Symposium on Algorithms - ESA 2002*, volume 2461, pages 785–794. Springer Verlag, 2002.

- M. Peczarski. New Results in Minimum-Comparison Sorting. *Algorithmica*, volume 40, pag. 134-145, 2004.
- M. B. Wells. Applications of a language for computing in combinatorics. In *Proceedings of IFIP Congress 65*, pages 497–498, 1965.