$$\sum_{i=1}^{P}\left(n - \frac{n}{s^i}\right) = P \cdot n - \frac{2 - n}{1 - s} = O(n \lg n)$$

O pior caso é $O(n^2)$, devido à segunda fase (Drozdek, 2005).

Ensaios experimentais indicam que a melhoria é realmente dramática, e o desempenho impressionante da ordenação tipo pente é comparável ao da ordenação rápida (Figura 9.19).

9.2 Árvores de decisão

Os três métodos de ordenação analisados nas seções anteriores não foram muito eficientes. Isto leva a várias questões: Pode-se esperar algum nível melhor de eficiência para um algoritmo de ordenação? Podem os algoritmos, pelo menos teoricamente, ser mais eficientes quando executados mais rápido? Neste caso, quando podemos ficar satisfeitos com um algoritmo e estar seguros da improbabilidade de a velocidade de ordenação aumentar? Necessitamos de uma medida quantitativa para estimar um *limite inferior* da velocidade de ordenação.

Esta seção foca as comparações de dois elementos, e não seu intercâmbio. As questões são: Na média, quantas comparações têm que ser feitas para se ordenar n elementos? Qual é a melhor estimativa do número de comparações de itens se uma matriz é assumida como ordenada aleatoriamente?

Todo algoritmo de ordenação pode ser expresso em termos de uma árvore binária na qual os arcos carregam rótulos S(im) ou N(ão). Os nós não terminais da árvore contêm condições ou perguntas para os rótulos, e as folhas têm todas as possíveis ordenações da matriz para a qual o algoritmo é aplicado. Este tipo de árvore é chamado *árvore de decisão*. Como a ordenação inicial não pode ser prevista, todas as possibilidades têm que ser listadas na árvore para que o procedimento de ordenação manipule qualquer matriz e qualquer possível ordem inicial de dados. Esta ordem inicial determina qual caminho é tomado pelo algoritmo e qual sequência de comparações é realmente escolhida. Note que diferentes árvores têm que ser desenhadas para matrizes de diferentes comprimentos.

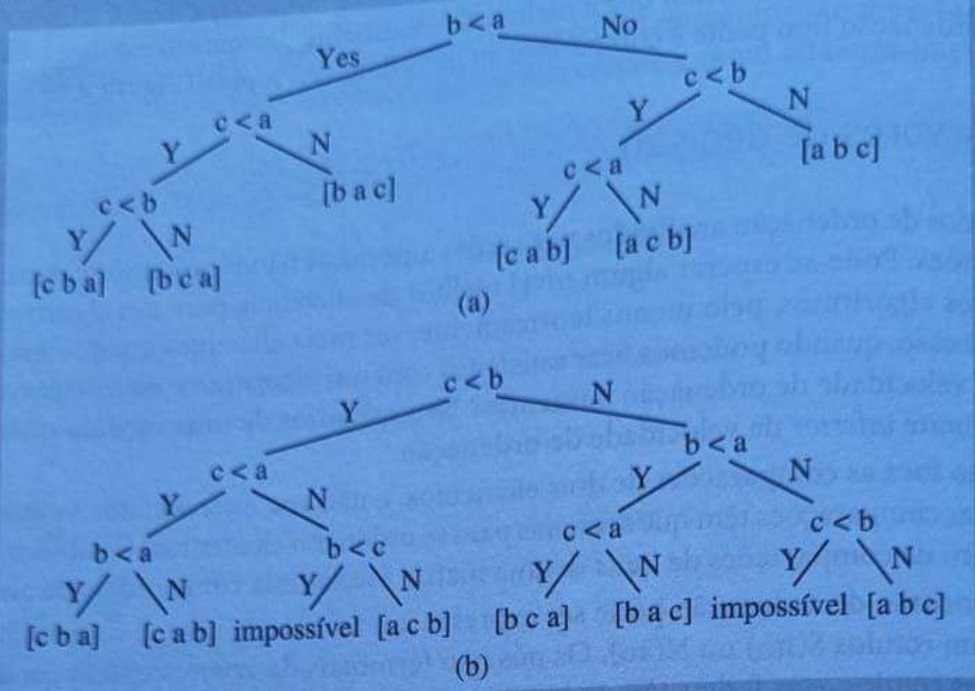
A Figura 9.4 ilustra as árvores de decisão para as ordenações por inserção e por borbulhamento para uma matriz [a b c]. A árvore para a ordenação por inserção tem seis folhas, e a por borbulhamento oito. Quantas folhas uma árvore para uma matriz de n elementos tem? Tal matriz pode ser ordenada em $n!$ modos diferentes, tantas quantas as possíveis permutações dos elementos da matriz, e todas essas ordenações têm que ser estocadas em folhas da árvore de decisão. Assim, a árvore para a ordenação por inserção tem seis folhas, porque $n = 3$ e $3! = 6$.

Mas, como o exemplo da árvore de decisão para a ordenação por borbulhamento indica, o número de folhas não tem que se igualar a $n!$. De fato, nunca é menor do que $n!$, o que significa que pode ser maior. Esta é uma consequência do fato de uma árvore de decisão poder ter folhas que correspondem a falhas, não somente a possíveis ordenações. Os nós de falhas são alcançados por uma sequência inconsistente de operações. Além disso, o número total de folhas pode ser maior do que $n!$, porque algumas ordenações (permutações) podem ocorrer em mais de uma folha, já que as comparações podem ser repetidas.

Uma das interessantes propriedades das árvores de decisão é o número médio de arcos cruzados, a partir da raiz, para atingir uma folha. Como um arco representa uma comparação, o número médio de arcos reflete-se no número médio de comparações de chaves quando se executa o algoritmo de ordenação.

Como já vimos no Capítulo 6, uma árvore de decisão completa de i-níveis tem 2^{i-1} folhas, $2^{i-1} - 1$ nós não terminais (para $i \geq 1$) e $2^i - 1$ nós totais. Como todas as árvores não completas com o mesmo número i de níveis têm muito menos nós do que aquela, $k + m \leq 2^i - 1$, onde m é o número de folhas e k o de não folhas. Além disso, $k \leq 2^{i-1} - 1$ e $m \leq 2^{i-1}$ (Seção 6.1 e Figura 6.5). A última desigualdade é usada como uma aproximação para m. Por isso, em uma árvore de decisão de nível i, existem no máximo 2^{i-1} folhas.

FIGURA 9.4 Árvores de decisão (a) ordenação por inserção e (b) ordenação por borbulhamento, conforme aplicadas à matriz [a b c].



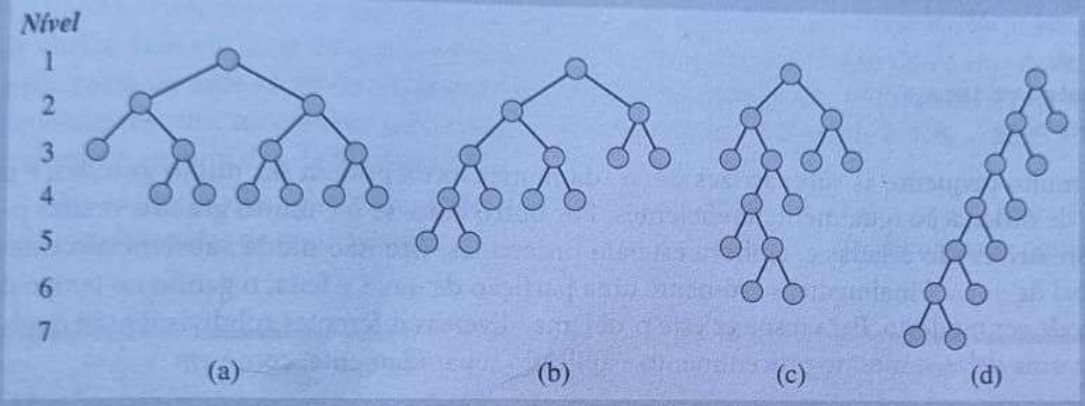
Agora surge uma pergunta: Qual é a relação entre o número de folhas de uma árvore de decisão e o número de todas as possíveis ordenações de uma matriz de n elementos? Existem $n!$ possíveis ordenações, e cada uma delas é representada por uma folha em uma árvore de decisão. Mas a árvore também tem alguns nós extras devido às repetições e falhas. Em consequência, $n! \leq m \leq 2^{i-1}$, ou $2^{i-1} \geq n!$. Esta desigualdade responde à seguinte pergunta: Quantas comparações são realizadas quando se usa um algoritmo de ordenação para uma matriz de n elementos no pior caso? Ou, melhor, no pior caso espera-se que qual número seja mais baixo ou melhor? Note que esta análise pertence ao pior caso. Assumimos que i é um nível de uma árvore independentemente de ela estar completa ou não; i sempre se refere ao caminho mais longo que leva da raiz da árvore para seu nível mais baixo, que é também o maior número de comparações necessárias para atingir uma configuração ordenada da matriz armazenada na raiz. Primeiro, a desigualdade $2^{i-1} \geq n!$ é transformada em $i - 1 \geq \lg(n!)$, o que significa que o comprimento do caminho em uma árvore de decisão com pelo menos $n!$ folhas precisa ser pelo menos $\lg(n!)$, ou, melhor, ser $\lceil \lg(n!) \rceil$, onde $\lceil x \rceil$ é um inteiro não menor do que x. Veja o exemplo da Figura 9.5.

Pode-se provar que, para uma folha aleatoriamente escolhida de uma árvore de decisão de m-folhas, o comprimento do caminho da raiz até a folha não é menor do que $\lg m$ e que, tanto no médio como no pior casos, o número exigido de comparações, $\lg(n!)$, é $O(n \lg n)$ (veja a Seção A.2 no Apêndice A). Portanto, $O(n \lg n)$ é também o melhor que se pode esperar em casos médios.

É interessante comparar esta aproximação com alguns dos números calculados para os métodos de ordenação, especialmente para o médio e o pior casos. Por exemplo, a ordenação por inserção exige somente $n - 1$ comparações no melhor caso, mas, no médio e no pior, ela se torna um algoritmo de ritmo n^2, pois as funções que relacionam o número de comparações com o de elementos são, nestes casos, os O-Grande de n^2. Isto é muito maior do que $n \lg n$, especialmente para números grandes. Em consequência, a ordenação por inserção não é um algoritmo ideal. A busca por melhores métodos pode prosseguir com pelo menos a expectativa de que o número de comparações deve ser aproximado por $n \lg n$, em vez de n^2.

A diferença entre essas duas funções será mais bem vista na Figura 9.6 se o desempenho dos algoritmos analisados até agora for comparado com o esperado $n \lg n$ no caso médio. Os números na tabela da Figura 9.6 mostram que, se 100 itens são ordenados, o algoritmo desejado é quatro vezes mais rápido do que a ordenação por inserção, e oito vezes mais rápido do que as por seleção e por borbulhamento. Para 1.000 itens, é, respectivamente, 25 e 50 vezes mais rápido. Para 10.000, a

FIGURA 9.5 Exemplos de árvores de decisão para uma matriz de três elementos.
Estas são algumas árvores de decisão possíveis para uma matriz de três elementos. Essas árvores precisam ter pelo menos 3! = 6 folhas. No exemplo apresentado, foi assumido que cada árvore tem uma folha adicional (uma repetição ou uma falha). No pior caso e no caso médio, a quantidade de comparações é dada por $i - 1 \geq \lceil \lg(n!) \rceil$. Nesse exemplo, $n = 3$, portanto $i - 1 \geq \lceil \lg 3! \rceil = \lceil \lg 6 \rceil \approx \lceil 2{,}59 \rceil = 3$. Conforme pode ser observado, somente a árvore mais bem balanceada (a) apresenta profundidade média de ramo menor do que três.



As somas dos comprimentos dos percursos a partir da raiz para todas as folhas nas árvores de (a) a (d), bem como seus respectivos comprimentos de percurso médio, são:

(a) $2 + 3 + 3 + 3 + 3 + 3 + 3 = 20$; média $= \frac{20}{7} \approx 2{,}86$

(b) $4 + 4 + 3 + 3 + 3 + 2 + 2 = 21$; média $= \frac{21}{7} = 3$

(c) $2 + 4 + 5 + 5 + 3 + 2 + 2 = 23$; média $= \frac{23}{7} \approx 3{,}29$

(d) $6 + 6 + 5 + 4 + 3 + 2 + 1 = 27$; média $= \frac{27}{7} \approx 3{,}86$

FIGURA 9.6 Número de comparações realizadas pelo método de ordenação simples e por um algoritmo cuja eficiência é estimada pela função $n \lg n$.

tipo de ordenação	n	100	1.000	10,000
inserção	$\frac{n^2 + n - 2}{4}$	2.524,5	250.249,5	25.002.499,5
seleção, borbulhamento	$\frac{n(n-1)}{2}$	4.950	499,500	49.995.000
esperado	$n \lg n$	664	9.966	132.877

diferença no desempenho difere por fatores de 188 e 376, respectivamente. Isto pode servir somente para encorajar a procura por um algoritmo que incorpore o desempenho da função $n \lg n$.

9.3 Algoritmos de ordenação eficientes

9.3.1 Ordenação de Shell

O limite $O(n^2)$ para um método de ordenação é muito grande e precisa ser quebrado para melhorar a eficiência e diminuir o tempo de execução. Como isto pode ser feito? O problema é que, usualmente, o tempo exigido para ordenar uma matriz pelos três algoritmos cresce mais rápido do que o tamanho da matriz. De fato, é geralmente uma função quadrática do tamanho. Pode ser útil ordenar partes da matriz original primeiro e então, caso estejam pelo menos parcialmente ordenadas, ordenar a matriz inteira. Se as submatrizes já estiverem ordenadas, estaremos muito mais perto do melhor caso de uma matriz ordenada. Um delineamento geral deste procedimento é este:

```
divida data em h submatrizes;
for i = 1 até h
   ordene a submatriz data_i;
ordene a matriz data;
```

Se h é muito pequeno, as submatrizes $\texttt{data}_i$ da matriz `data` podem ser muito grandes, e os algoritmos de ordenação igualmente ineficientes. Por outro lado, se h é muito grande, muitas pequenas submatrizes são criadas e, embora estejam ordenadas, isto não muda substancialmente a ordem global de `data`. Finalmente, se somente uma partição de `data` é feita, o ganho no tempo de execução pode ser modesto. Para resolver este problema, diversas diferentes subdivisões são usadas e, para cada uma delas, o mesmo procedimento é aplicado separadamente, como em

```
determine os números h_t...h_1 de modos de dividir a matriz data em submatrizes;
for (h=h_t; t > 1; t--, h=h_t)
   divida data em h submatrizes;
   for i = 1 até h
      ordene a submatriz data_i;
ordene a matriz data;
```

Esta ideia é a base da *ordenação por diminuição do incremento*, também chamada *ordenação de Shell*, em homenagem a Donald L. Shell, que concebeu esta técnica. Note que esse pseudocódigo não identifica um método específico de ordenação para a ordenação das submatrizes; pode ser qualquer método simples. Usualmente, no entanto, a ordenação de Shell usa o método de inserção.

O coração da ordenação de Shell é uma divisão engenhosa da matriz `data` em diversas submatrizes. O truque é comparar primeiro os elementos espaçados mais afastados, depois os mais perto uns dos outros, e assim por diante, até os elementos adjacentes ser comparados em uma última passada. A matriz original é logicamente subdividida em submatrizes escolhendo-se cada h_t-ésimo elemento como parte de uma submatriz. Em consequência, existem h_t submatrizes e, para cada $h = 1, \ldots, h_t$,

$$\texttt{data}_{h_t h}[i] = \texttt{data}[h_t \cdot i + (h-1)]$$

Por exemplo, se $h_t = 3$, a matriz `data` é subdividida em três submatrizes, $\texttt{data}_1$, $\texttt{data}_2$ e $\texttt{data}_3$, de modo que

$$\texttt{data}_{31}[0] = \texttt{data}[0],\ \texttt{data}_{31}[1] = \texttt{data}[3],\ \ldots,\ \texttt{data}_{31}[i] = \texttt{data}[3*i],\ \ldots.$$
$$\texttt{data}_{32}[0] = \texttt{data}[1],\ \texttt{data}_{32}[1] = \texttt{data}[4],\ \ldots,\ \texttt{data}_{31}[i] = \texttt{data}[3*i+1]$$
$$\texttt{data}_{33}[0] = \texttt{data}[2],\ \texttt{data}_{33}[1] = \texttt{data}[5]$$