Esta seção descreve duas excelentes ordenações. A primeira é a ordenação *Shell*. A segunda, a *quicksort*, normalmente é considerada a melhor rotina de ordenação. Essas ordenações rodam tão rápido que, se você piscar, você as perde de vista!

Ordenação Shell

A ordenação Shell é assim chamada devido ao seu inventor, D. L. Shell. Porém, o nome provavelmente pegou porque seu método de operação é freqüentemente descrito como conchas do mar empilhadas umas sobre as outras.

O método geral é derivado da ordenação por inserção e é baseado na diminuição dos incrementos. Considere o diagrama da Figura 19.2. Primeiro, todos os elementos que estão três posições afastados um do outro são ordenados. Em seguida, todos os elementos que estão duas posições afastados são ordenados. Finalmente, todos os elementos adjacentes são ordenados.

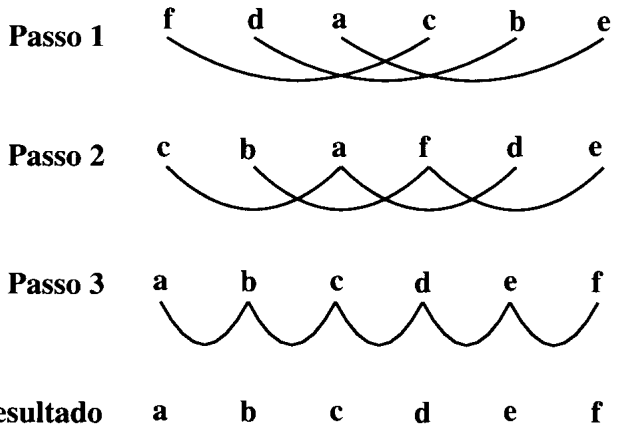

Figura 19.2 A ordenação Shell.

Não é fácil perceber que esse método conduz a bons resultados ou mesmo que ordene a matriz. Mas ele executa ambas as funções. Cada passo da ordenação envolve relativamente poucos elementos ou elementos que já estão razoavelmente em ordem, logo, a ordenação Shell é eficiente e cada passo aumenta a ordenação dos dados.

A seqüência exata para os incrementos pode mudar. A única regra é que o último incremento deve ser 1. Por exemplo, a seqüência

9, 5, 3, 2, 1

funciona bem e é usada na ordenação Shell mostrada aqui. Evite seqüências que são potências de 2 — por razões matemáticas complexas, elas reduzem a eficiência do algoritmo de ordenação (mas a ordenação ainda funciona).

```
/* A ordenação Shell. */
void shell(char *item, int count)
{

  register int i, j, gap, k;
  char x, a[5];

  a[0]=9; a[1]=5; a[2]=3; a[3]=2; a[4]=1;

  for(k=0; k<5; k++) {
    gap = a[k];
    for(i=gap; i<count; ++i) {
      x = item[i];
      for(j=i-gap; x<item[j] && j>=0; j=j-gap)
        item[j+gap] = item[j];
      item[j+gap] = x;
    }
  }
}
```

Você deve ter observado que o laço **for** mais interno tem duas condições de teste. A comparação **x<item[j]** é obviamente necessária para o processo de ordenação. O teste **j>=0** evita que os limites da matriz **item** sejam ultrapassados. Essas verificações extras degenerarão até certo ponto o desempenho da ordenação Shell.

Versões um pouco diferentes da ordenação Shell empregam elementos especiais de matriz, chamados sentinelas, que não fazem parte realmente da matriz a ser ordenada. *Sentinelas* guardam valores especiais de terminação, que indicam o menor e o maior elemento possível. Dessa forma, as verificações dos limites são desnecessárias. No entanto, usar sentinelas requer um conhecimento específico dos dados, o que limita a generalização da função de ordenação.

A análise da ordenação Shell apresenta alguns problemas matemáticos que estão além do objetivo desta discussão. O tempo de execução é proporcional a

$$n^{1.2}$$

para se ordenar n elementos. Essa é uma redução significativa com relação às ordenações n-quadrado. Para entender o quanto essa ordenação é melhor, observe a Figura 19.3, que mostra os gráficos das ordenações n^2 e $n^{1.2}$. Porém, antes de se decidir pela ordenação Shell, você deve saber que a ordenação quicksort é ainda melhor.

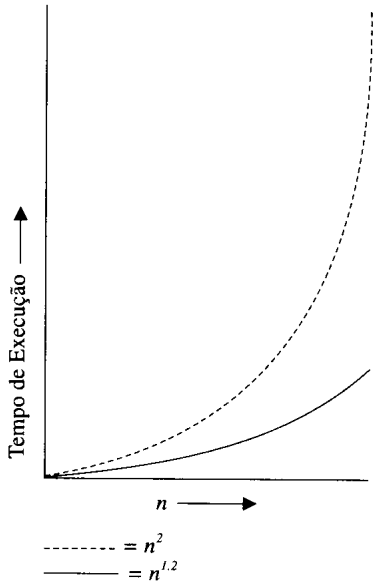

Figura 19.3 As curvas n^2 e $n^{1.2}$.

Quicksort

A quicksort, inventada e denominada por C.A.R. Hoare, é superior a todas as outras ordenações deste livro, e geralmente é considerada o melhor algoritmo de ordenação de propósito geral atualmente disponível. É baseada no método de ordenação por trocas. Isso é surpreendente, quando se considera o terrível desempenho da ordenação bolha!

A quicksort é baseada na idéia de partições. O procedimento geral é selecionar um valor, chamado de *comparando*, e, então, fazer a partição da matriz em duas seções, com todos os elementos maiores ou iguais ao valor da partição de um lado e os menores do outro. Esse processo é repetido para cada seção restante até que a matriz esteja ordenada. Por exemplo, dada a matriz **fedacb** e usando o valor **d** para a partição, o primeiro passo da quicksort rearranja a matriz como segue:

início	f e d a c b
passo1	b c a d e f

Esse processo é, então, repetido para cada seção — isto é, **bca** e **def**. Como você pode ver, o processo é essencialmente recursivo por natureza e, certamente, as implementações mais claras da quicksort são algoritmos recursivos.