Universidade Federal de Ouro Preto – UFOP
Instituto de Ciências Exatas e Biológicas – ICEB
Departamento de Computação – DECOM
Disciplina: Algoritmos e Estruturas de Dados I – CIC102
Professor: David Menotti (menottid@gmail.com)

Lista de Exercícios sobre Ordenação

1) Faça uma comparação entre todos os métodos de ordenação estudados em aula com relação a estabilidade (preservar ordem lexicográfica), ordem de complexidade levando em consideração comparações e movimentações.

2) Dada a sequência de números: 3 4 9 2 5 8 2 1 7 4 6 2 9 8 5 1, ordene-a em ordem não decrescente segundo os seguintes algoritmos, apresentando a sequência obtida após cada passo do algoritmo:
   a. MergeSort
   b. QuickSort
   c. HeapSort

3) João diz ter desenvolvido um algoritmo que é capaz de ordenar qualquer conjunto de n números reais, fazendo apenas $O(n^{3/2})$ comparações. Você compraria este algoritmo? Justifique.

4) Um amigo lhe diz que é capz de ordenar qualuqer conjunto de 6 números com no máximo 8 comparações. O seu amigo está falando a verdade ou mentindo? Justifique.

5) Dado um conjunto de n inteiros distintos e um inteiro positivo $k \leq n$:
   a. Proponha um algoritmo que imprime os k menos elementos do conjunto (em qualuqer ordem) em tempo O(n).
   b. Suponha agora que queremos imprimir os k menos em ordem crescente. É ainda possível fazer isso em tempo O(n) para quaisquer valores $k \leq$ n?

6) Dado um vetor com n elementos, projete um algoritmo linear para encontrar todos os elementos que se repetem pelo menos $n/3$ vezes.

7) Reescreva a função *BubbleSort* apresentada em aula com sucessivas passagens em direções opostas.

8) Uma ***ordenação por contagem*** de um vetor x de tamanho n é executada da seguinte forma: declare um vetor *count* e defina *count*[i] como o número de elementos menores que x[i]. Em seguida, coloque x[i] na posição *count*[i] de um vetor de saída (leve em consideração a possibilidade de elemntos iguais). Escreva uma função para ordenar um vetor x de tamanho n usando esse método.

9) Presuma que um vetor contém inteiros entre a e b, inclusive, com vários números repetidos diversas vezes. Uma ***ordenação por distribuição*** (*BucketSort*) ocorre da seguinte maneira: declare um vetor *number* de tamanho $b - a + 1$, defina *number*[i-a] como o número de vezes que o inteiro i aparece no vetor e, em seguida, redefina os valores no vetor concomitantemente, Escreva uma função para ordenar um vetor x de tamanho n contendo inteiros entre a e b, inclusive, com esse método.

10) A ***ordenação por transposição de par-impar*** ocorre da seguinte maneira. Percorra o vetor várias vezes. Na primeira passagem compare $x[i]$ com $x[i+1]$ para todo i ímpar. Na segunda passagem compare $x[i]$ com $x[i+1]$ para todo i par. Todav vez que $x[i] > x[i+1]$ troque os dois. Continue alternando dessa maneira até que o vetor esteja ordenado.
    a. Qual a condição para o término da ordenação?
    b. Escreva uma função para implementar essa ordenação?
    c. Qual é o custo médio dessa ordenação?

11) Modifique a função *partition* (partição) do método *QuickSort* de modo que o valor do meio (mediano) de $x[menor]$, $x[maior]$ e $x[meio]$ (onde $meio = (maior + meio) / 2$) seja usado para particionar o vetor. Em que casos o quicksort usará esse método com mais eficiência do que a versão apresentada em aula? Em que casos ele será menos eficiente?

12) Considere a seguinte **ordenação por seleção quadrática**: divida os n elementos do vetor em $\sqrt{n}$ grupos de $\sqrt{n}$ elementos casa. Encontre o maior elemento de cada grupo e insira-o num vetor auxiliar. Encontre o maior elemento nesse vetor auxiliar. Esse será o maior elemento do vetor. Em seguida, substitua esse elemento dentro do vetor pelo maior elemento seguinte do grupo a que ele pertencia. Ache novamente o maior elemento do vetor auxiliar. Esse será o segundo maior elemento do vetor. Repita o processo até que o arquivo esteja classificado. Escreva uma função para implementar uma ordenação por seleção quadrática o mais eficiente possível.

13) Um **torneio** é uma árvore estritamente binária quase completa na qual cada nó não-folha contém o maior de dois elemetnos em seus dois filhos. Por conseguinte, o conteúdo das folhas de um torneio determina totalmente o conteúdo de todos os seus nós. Um torneio com n folhas representa um conjunto de n elementos.
    a. Desenvolva um algoritmo *Insira*(t,n,elt) para incluir um novo elemnto elt num torneio contendo n folhas representadas implemente por um vetor t.
    b. Desenvolva um algoritmo *RemoveMax*(t,n) para eliminar o elemento máximo de um torneio com n elementos, substituindo a folha contendo o elemento máximo por um valor artificial menor que qualquer elemento possível (por exemplo, -1 num torneio de inteiros não-negativos) e reajustando, em seguida, todos os valores no caminho a partir dessa folha até a raiz.

14) Desenvolva um algoritmo usando um heap de k elementos para encontrar os maiores k números num grande vetor não classificado de n números (onde $n >> k$).

15) A **ordenação por inserção bidirecional** é uma modificação da ordenação por inserção simples da seguinte forma: um vetor de saída separado, de tamanho n, é reservado. Esse vetor de saída atua como uma estrutura circular. $x[0]$ é posicionado no elemento do meio do vetor. Assim que um grupo contíguo de elementos estiver no vetor, será aberto espaço para um novo elemento deslocando todos os elementos menores um passo para a esquerda ou todos os elementos maiores um passo a direita.

A escolha do deslocamento é feita para provocar o menor número de movimentações. Escreva uma função para implementar essa técnica.

16) A **ordenação por inserção intercalada** é a seguinte:
Passo 1: Para todo i par entre 0 e n-2, compare $x[i]$ a $x[i+1]$. Posicione o maior na próxima posição de um vetor *large* e o menor na próxima posição de um vetor *small*. Se n for ímpar, posicione $x[n-1]$ na última posição do vetor *small*. (*large* é de tamanho *ind*, onde *ind* = (n - 1)/2; *small* é de tamanho *ind* ou *ind*+1, dependendo de n ser ímpar ou par.).
Passo 2: Ordene o vetor *large* usando a inserção intercalada recursivamente. Sempre que um elemento $large[j]$ for transferido para $large[k]$, $small[j]$ será também movido para $small[k]$. (No final desse passo, $large[i] \leq large[i+1]$ para todo i menor que *ind*, e $small[i] \leq large[i]$ para todo i menor ou igual a *ind*.
Passo 3: Copie $small[0]$ e todos os elementos de *large* em $x[0]$ a $x[ind]$.
Passo 4: Defina o inteiro $num[i]$ como $(2^{i+1} + (-1)^i)/3$. Começando com $i = 0$ e continuando de 1 em 1 enquanto $num[i] \leq (n/2) + 1$, insira os elementos $small[num[i+1]]$ até $small[num[i]+1]$ em x, por vez, usando a inserção binária. (Por exemplo, se $n = 20$, os sucessivos valores de *num* são $num[0] = 1$, $num[1] = 1$, $num[2] = 3$, $num[3] = 5$ e $num[4] = 11$, que é igual a $(n/2) + 1$. Dessa forma, os elementos de *small* serão inseridos na seguinte ordem: $small[2]$, $small[1]$; em seguida, $small[4]$, $small[3]$; depois $small[9]$, $small[8]$, $small[7]$, $small[6]$, $small[5]$. Neste exemplo, não existe $small[10]$.
Escreva uma função para implementar essa técnica.

17) Explique por que é necessário escolher todos os incrementos do método ShellSort de modo que eles sejam primos entre si.

18) Escreva um algoritmo que *merge*(x,$c1$,$f1$,$c2$,$f2$) que presuponha que $x[c1]$ até $x[f1]$ e de $x[c2]$ até $x[f2]$ estão ordenados e intercale os dois intervalos em $x[c1]$ até $x[f2]$.

19) Considere a seguinte versão recursiva da ordenação por intercalação que usa a função merge do exercício anterior. Inicialmente, ela é chamada por *msort2*(x,0,n-1). Reescreva a função eliminando a recursividade e simplificando-a. Qual a diferença entre a função resultante e a apresentada aqui?

```
msort2(int x, int c, int f)
{
  If (c != f)
  {
     m = (f+c)/2;
    msort2(x,c,m);
    msort2(x,m+1,f);
    merge(x,0,m,m+1,f);
  }
}
```

20) Escreva um programa que imprima todos os conjuntos de seis inteiros positivos, $a1$, $a2$, $a3$, $a4$, $a5$ e $a6$, tais que:

$$a1 \leq a2 \leq a3 \leq 20$$
$$a1 < a4 \leq a5 \leq a6 \leq 20$$

e a soma dos quadrados de $a1$, $a2$ e $a3$ seja igual à soma dos quadrados de $a4$, $a5$ e $a6$. (Dica: Gere todas as somas possíveis de três quadrados e use um procedimento de ordenação para localizar repetições).

Exercícios extraídos de (Referências)

[1] Aaron M. Tenenbaum, Yedidyah Langsam, Moshe J. Augenstein, *Estruturas de Dados Usando C*, Makron Books/Pearson Education, 1995.