EXERCÍCIOS DE COMPUTAÇÃO II
1o. SEMESTRE DE 2002

1. Exercícios 6.5 e 6.6.[1] {*Data de entrega*: 5/4/2002}
2. Exercícios 6.14, 6.18, e 6.22. {*Data de entrega*: 5/4/2002}
3. Exercício 8.1. {*Data de entrega*: 5/4/2002}
4. Suponha que executamos o Programa 6.2 (ordenação por seleção) sobre um vetor aleatório com n elementos. Procure estabelecer uma estimativa para o número (médio) de vezes que a atribuição `min = j` é executada. (Você pode fazer isso experimentalmente. Sugestão: considere a função $n \log n$, onde o logaritmo é na base $e = 2.718\ldots$.) {*Data de entrega*: 12/4/2002}
5. Exercício 6.7 (veja a definição de algoritmo de ordenação *estável* na Definição 6.1 de Sedgewick). Você pode achar útil estudar a Seção 6.7 do Sedgewick. {*Data de entrega*: 12/4/2002}
6. Escreva um algoritmo que recebe como entrada um vetor de n elementos e um inteiro $1 \leq k \leq n$, e devolve o k-ésimo menor elemento do vetor. O seu algoritmo deve ser tal que, em média, ele leva tempo $O(n)$ qualquer que seja o valor de k. Você não precisa provar formalmente esta propriedade de seu algoritmo, mas você deve apresentar um argumento intuitivo. {*Data de entrega*: 9/4/2002}.
7. Suponha que alteramos o trecho de `prog7.1.c`

```
      while (scanf("%d", &a[N]) == 1) N++;
    }
    quicksort(a, 0, N-1);
    for (i = 0; i < N; i++) printf("%3d ", a[i]);
```

   para

```
      while (scanf("%d", &a[N]) == 1) N++;
    }
    quicksort(a, 0, N-1);
    quicksort(a, 0, N-1);
    for (i = 0; i < N; i++) printf("%3d ", a[i]);
```

   Ademais, suponha que alteramos o trecho de `prog7.4.c`

```
    }
    sort(a, 0, N-1);
    for (i = 0; i < N; i++) printf("%3d ", a[i]);
```

   para

Data: Versão de 21 de junho de 2002.

[1] A menos de menção explícita em contrário, estes exercícios são do Sedgewick, Algorithms in C, Partes 1 a 4.

```
}
sort(a, 0, N-1);
sort(a, 0, N-1);
for (i = 0; i < N; i++) printf("%3d ", a[i]);
```

Sejam `prog7.1.b.c` e `prog7.4.b.c` os programas assim obtidos. Execute estes programas para valores grandes de `N`, e meça os respectivos tempos de execução. Por que `prog7.1.b.c` é tão mais demorado? {*Data de entrega*: 12/4/2002}

8. Dizemos que uma palavra s é um *anagrama* de uma palavra t se s pode ser obtida de t pela permutação de suas letras. Por exemplo, as palavras *triangle*, *relating*, *integral*, *altering*, e *alerting* são todas anagramas umas das outras. Neste exercício, você é convidado a encontrar anagramas 'interessantes' nas listas de palavras que você pode encontrar em `http://www.ime.usp.br/~yoshi/dicios`. Por exemplo, você pode procurar pares (s,t) de anagramas compridos (isto é, s e t longos), você pode procurar conjuntos grandes de anagramas $\{s_1,\ldots,s_k\}$ (isto é, k grande), etc. [*Sugestão.* Chame de *assinatura* de uma palavra s a cadeia de caracteres que obtemos ao ordenar as letras em s. Por exemplo, a assinatura de 'assinatura' é 'aaainrsstu'. Use este conceito. Você também pode achar o utilitário `uniq` do Unix útil.] {*Data de entrega*: 26/4/2002}
9. Seja $R(N)$ a soma das alturas dos nós de uma árvore binária completa com N nós. Para um inteiro não-negativo N, escrevermos $b_1(N)$ para o números de bits 1 na representação de N na base 2. Prove que $R(N) = N - b_1(N)$. Em particular, $R(N) \leq N$ para todo N. {*Data de entrega*: 30/4/2002}
10. Explique porque podemos omitir o primeiro `if` e a primeira chamada recursiva na função `radixMSD()` (`prog10.2.c`) quando estamos ordenando `strings`. {*Data de entrega*: 3/5/2002}
11. Escreva um programa como segue. O seu programa deve receber na linha de comando dois inteiros, digamos M e N, ambos $\leq 10^9$, e deve devolver uma lista aleatória de N inteiros *distintos*, todos não-negativos e menores que M. Você pode supor que o seu programa será usado para valores de M e N com N substancialmente menor que M, digamos $N \leq M/2$. O seu programa deve executar em poucos minutos para $M = 10^9$ e $N = 5 \times 10^8$. {*Data de entrega*: 7/5/02}
12. Escreva um módulo `radixL` (`radixL.c` e `radixL.h`) que pode ser usado para ordenar uma lista ligada com nós do tipo

```
typedef struct node_struct {
  long key;
  struct node_struct *link;
} node;
```

A linha

```
lista = ordene(lista);
```

em um programa cliente deve ordenar os elementos da lista ligada cujo primeiro elemento é apontado por `lista`, em ordem crescente de seus campos `key`. Use o LSD Radix Sort fornecido em `http://www.ime.usp.br/~yoshi/2002i/CompII/exx/LTSs/CWEB/`
{*Data de entrega*: 17/5/2002}

13. Seja T_n o número de árvores binárias com n nós internos. Por exemplo, $T_0 = T_1 = 1$, $T_2 = 2$, $T_3 = 5$, etc.
    (*i*) Prove que, para todo $n > 0$, temos $T_n = \sum_{0\le k<n} T_k T_{n-k-1}$.
    (*ii*) Quantos times (mistos) de futebol podemos escalar se temos m meninos e n meninas?
    (*iii*) Prove a *identidade de Vandermonde* para coeficientes binomiais: para todo m, n, e $k \ge 0$ inteiros, temos
$$\sum_j \binom{m}{j}\binom{n}{k-j} = \binom{m+n}{k}, \tag{1}$$
    onde a soma do lado esquerdo é sobre todo j inteiro. Se $x \in \mathbb{R}$ e $k \in \mathbb{Z}$, definimos
$$\binom{x}{k} = \begin{cases} 0 & \text{se } k < 0 \\ \frac{1}{k!}x(x-1)\dots(x-k+1) & \text{se } k \ge 0. \end{cases}$$
    (Note que a definição acima coincide com a definição usual de coeficientes binomiais para x e k inteiros não-negativos.)
    (*iv*) Prove que (1) vale também na forma
$$\sum_{j\in\mathbb{Z}} \binom{x}{j}\binom{y}{k-j} = \binom{x+y}{k}, \tag{2}$$
    para todo x e y reais. [*Sugestão.* Considere inicialmente o caso em que x é um inteiro fixo. Neste caso, tanto o lado esquerdo como o lado direito de (2) são polinômios na variável real y; use este fato.]
    (*v*) Prove a identidade
$$\binom{-1/2}{n} = \left(\frac{-1}{4}\right)^n \binom{2n}{n},$$
    válida para todo n inteiro, e deduza que
$$\frac{1}{n+1}\binom{2n}{n} = 2(-4)^n \binom{1/2}{n+1}$$
    para todo n inteiro.
    (*vi*) O *n-ésimo número de Catalan* C_n é dado por
$$C_n = \frac{1}{n+1}\binom{2n}{n}.$$
    Faça uma tabela dos C_n para valores pequenos de n.

(*vii*) Prove que $T_n = C_n$ para todo n. [*Sugestão.* Note que basta provar que T_n e C_n satisfazem uma mesma recorrência.] {*Data de entrega*: 21/5/2002}

14. (*i*) Escreva uma função de cabeçalho

```
int int_path_length(link h);
```

que, ao ser chamada com parâmetro `h`, determina o comprimento interno da árvore binária cuja raiz é apontada por `h`. (*ii*) Escreva uma função de cabeçalho

```
int ext_path_length(link h);
```

que, ao ser chamada com parâmetro `h`, determina o comprimento externo da árvore binária cuja raiz é apontada por `h`. {*Data de entrega*: 21/5/2002}

15. Escreva um programa (ou programas) para estudar a altura 'típica' de árvores binárias de busca (ABBs). Você deve considerar duas formas de se gerar tais árvores:
   (*i*) gere N chaves distintas em ordem aleatória e as insira em uma ABB inicialmente vazia;
   (*ii*) considere as N primeiras palavras distintas de um texto longo e as insira em uma ABB inicialmente vazia. Você pode achar útil o utilitário `tr` (experimente aplicar `tr -d ".,;:"` em um texto).

   Você deve fazer estes experimentos para vários valores de N e então usar o `GNUPLOT` para comparar a altura destas árvores com a função $c \log N$ (a idéia é você determinar a constante c, e também ver se esta função aproxima bem a altura). {*Data de entrega*: 24/5/2002}
16. Prove a Propriedade 13.3 do Sedgewick considerando apenas seqüências de 3 inserções. {*Data de entrega*: 28/5/2002}
17. Faça o exercício 14.10 do Sedgewick. {*Data de entrega*: 4/6/2002}
18. Escreva um programa para resolver o seguinte problema. Dado um texto e um inteiro K, determine as K palavras (distintas) mais freqüentes no texto. O seu programa deve imprimir estas K palavras em ordem decrescente de freqüência. Cada palavra deve ser impressa juntamente com o número de vezes que ela ocorre no texto.

   O seu programa deve ser eficiente para que determinar as 1000 palavras mais comuns nos livros que você pode encontrar no Projeto Gutenberg seja 'trivial'.

   O natural para se resolver este problema é usar tabelas de símbolos. Experimente várias implementações de tais tabelas (pelo menos, você deve usar ABBs e ABBs aleatórias; tente também usar tabelhas de espalhamento). {*Data de entrega*: 7/6/2002}
19. Resolva o problema do estacionamento enunciado em sala na aula de 7/6/2002. {*Data de entrega*: 14/6/2002}
20. Altere o programa os programas em

    `http://www.ime.usp.br/~yoshi/2002i/CompII/exx/g6` para que, dados dois vértices do grafo gerado, um caminho de comprimento mínimo entre eles é determinado e impresso. {*Data de entrega*: 28/6/2002}