MAC122/BCC **SEGUNDA PROVA** *8/12/98*

Q1. [2 pontos] Lembre que os números de Fibonacci $(F_n)_{n\geq 0}$ são definidos pondo-se $F_0 = 0$, $F_1 = 1$, e

$$F_n = F_{n-2} + F_{n-1}$$

para todo $n \geq 2$.

(i) Escreva a função *recursiva* `fr()` óbvia da definição acima que recebe um `int n` e devolve F_n.

(ii) Escreva uma função *não-recursiva* `fnr()` que recebe um `int n` e devolve F_n (basta se usar um "`for`").

(iii) Qual das duas funções acima é mais rápida?

(iv) Justifique a sua resposta para (iii).

Q2. [2 pontos] Nesta questão, estamos interessados no mergesort para listas ligadas. Suponha que temos uma lista cujos elementos têm chaves M, E, R, G, E, nesta ordem. Execute o mergesort para listas visto em sala nesta lista. Mostre o estado de todas as listas que são obtidas durante o processo.

Q3. [2 pontos] Nesta questão, estamos interessados em filas de prioridades implementadas por *heaps* (as chaves serão inteiros e as chaves maiores têm maior prioridade).

(i) Suponha que inserimos os seguintes elementos, nesta ordem, em uma fila inicialmente vazia: 3, 1, 4, 5, 9, 2, 6. Desenhe o heap resultante, tanto como um vetor e como uma árvore.

(ii) Remova agora o elemento máximo de seu heap. Desenhe o heap resultante, novamente como um vetor e como uma árvore.

(iii) Por que a operação de remoção do elemento máximo de um heap com n elementos resulta em no máximo $O(\log n)$ comparações entre chaves?

(iv) Quantas comparações são feitas entre chaves quando ordenamos um vetor usando heapsort? Justifique.

Q4. [3 pontos] Suponha que implementamos listas ligadas com elementos do tipo `Elemento`, definido assim:

```
typedef struct el_struct {
  char *chave;
  struct el_struct *prox;
} Elemento;
```

Suponha que sejam dadas duas listas ligadas (simples, sem cabeça de lista ou cauda), cujos elementos iniciais são apontados por `p` e `q`. Escreva uma função em `C` de protótipo

```
Elemento *produto(Elemento *p, Elemento *q);
```

que recebe estes ponteiros e devolve um ponteiro para (o elemento inicial de) uma lista ligada simples dos elementos que formam o "produto cartesiano" dos elementos de `p` e `q`. Por exemplo, se as chaves da lista `p` são os strings `ana`, `bete` e `carolina`, e as chaves dos elementos da lista `q` são `xilofone` e `yahoo!`, então as chaves dos 6 elementos da lista devolvida por sua função devem ser os strings `anaxilofone`, `anayahoo!`, `betexilofone`, `beteyahoo!`, `carolinaxilofone`, `carolinayahoo!`.

As listas `p` e `q` não devem ser destruídas. Caso uma das listas `p` ou `q` seja vazia, a lista resultante deve ser vazia.

Q5. [3 pontos] Considere a seguinte função:

```
int ind_particao(int a[], int l, int r)
{ int v, i, j, t;
  v=a[r]; i=l-1; j=r;
  while (1) {
    while (a[++i]<v) ;
    while (a[--j]>v) ;
    if (i>=j) break;
    t=a[i]; a[i]=a[j]; a[j]=t;
  }
  t=a[i]; a[i]=a[r]; a[r]=t;
  return i;
}
```

(i) Escreva uma função em `C` que implementa o quicksort baseado na função acima.

(ii) Suponha que executamos a função acima sobre um vetor `a[]`, com parâmetros `l` e `r`. Seja `i` o valor devolvido pela função. É verdade que para todo `k`, com $\mathtt{l} \leq \mathtt{k} < \mathtt{i}$, e todo `m`, com $\mathtt{i} < \mathtt{m} \leq \mathtt{r}$, temos que

$$\mathtt{a}[\mathtt{k}] \leq \mathtt{a}[\mathtt{i}] \leq \mathtt{a}[\mathtt{m}] \quad ?$$

Justifique.

(iii) Existe um caso em que uma sentinela é, na verdade, imprescindível para a função acima funcionar corretamente. Qual é este caso? Leve esta observação em conta para responder (i).