MAC 122 - PDA
2. Semestre de 2014 - prof. Marcilio – IME USP BMAC
Segunda Prova – 27 de Novembro de 2014

Questão	1	2	3	4	5	6	Total
Valor	1	1	2	2	2	2	10,0
Nota							

Nome:_____ NUSP:_____

Questão 1 (1 ponto) :

Nas funções abaixo supor **N ≥ 0, M ≥ 0** e **0 ≤ V[i] ≤ M (i = 0, 1, ..., N-1)** Para cada uma das funções abaixo, diga qual a sua ordem (função **O(..)**) em função de **N** e **M** e justifique:

a) **O(N)**

```
int F(int V[],int N) {
    int k = N, s = 1;
    while (--k >= 0) s = s * V[k];    (1)
    k = N;
    while (--k >= 0) s = s - V[k];    (2)
    return s;
}
```

(1) while executado N vezes
(2) while executado N vezes
Tempo proporcional a N+N = 2N.
Portanto O(2N) = O(N)

b) **O(N.log M)**

```
int G(int V[], int N) {
    int i, s = 1;
    for (i = 0; i < N; i++) {          (1)
       int k = 1;
       while (k < V[i]) k = k * 2;     (2)
       s = s + k;
       k = 1;
       while (k < V[i]) k = k * 10;    (3)
       s = s + k;
    }
    return s;
}
```

(1) for executado N vezes
(2) while dentro do for executado no máximo $\log_2$ M vezes
(3) while dentro do for executado no máximo $\log_{10}$ M vezes
Tempo proporcional a N (log M + log M) = 2N log M. Portanto O(N.log M)

Questão 2 (1 ponto)

Considere o vetor `VET[11]` onde cada elemento contém um algarismo do seu NUSP. O vetor está em ordem crescente e está precedido de dois zeros e seguido por dois noves (veja o exemplo abaixo). Se o seu NUSP tiver menos de 7 dígitos considere zeros à esquerda.
Exemplo: Se o seu NUSP é 2473794 então:

i	0	1	2	3	4	5	6	7	8	9	10
VET[i]	**0**	**0**	2	3	4	4	7	7	9	**9**	**9**

Usando o algoritmo de busca binária, diga com quantos números haverá comparação para se procurar:

Cada aluno deve usar o seu NUSP – portanto a resposta será diferente para cada um.
Abaixo o número de trocas para o exemplo acima.

a) O primeiro dígito do seu NUSP (No exemplo acima seria o 2)

digito 2 = 2 comparações

b) O último dígito (No exemplo acima seria o 4)

digito 4 = 1 comparação

c) Qualquer dígito que não pertença ao seu NUSP (No exemplo acima poderia ser o 6)

Pode escolher qualquer um entre 0, 1, 5, 6 e 8. Abaixo todos eles.

digito 0 = 3 comparações
digito 1 = 4 comparações
digito 5 = 3 comparações
digito 6 = 3 comparações
digito 8 = 4 comparações

d) O dígito 0 (zero)

digito 0 = 3 comparações

Questão 3 (2 pontos)

A matriz **`int data[][3]`** contém em cada linha uma data:
`data[i][0]` – dia
`data[i][1]` – mês
`data[i][2]` – ano
O objetivo é classificá-la em ordem crescente de datas.

a) Escreva uma função **`int ComparaDatas(int data[][3], int i, int j)`** que compara as datas da linha **`i`** e da linha **`j`**, devolvendo **`1`** se a data da linha **`i`** for maior que a data da linha **`j`**, **`0`** se forem iguais e **`-1`** se for menor.

```
int ComparaDatas (int data[][3], int i, int j) {
   /* compara as datas nas linhas i e j da matriz data[][3] */
   /* devolve 1 data[i][] > data[j][] e -1 caso contrário */
   if (data[i][2] > data[j][2]) return 1;
   else  if (data[i][2] < data[j][2]) return -1;
   if (data[i][1] > data[j][1]) return 1
   else if (data[i][1] < data[j][1]) return -1;
   if (data[i][0] > data[j][0]) return 1;
   else  if (data[i][0] < data[j][0]) return -1;
   return 0;
}
```

b) Escreva uma função **`void ClassificaDatas(int data[][3], int n)`** que recebe uma matriz **`data`** com **`n`** linhas e classifica a matriz em ordem crescente de datas usando o algoritmo da Bolha. Use obrigatoriamente a função **`ComparaDatas`** acima para comparar duas datas.

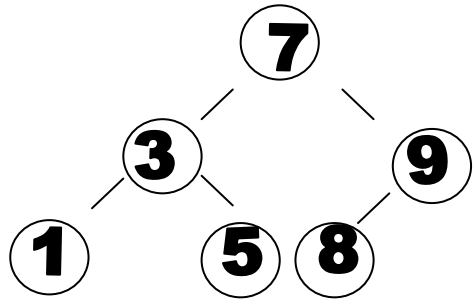
```
void TrocaLinhas(int k1, int k2) {
  /* troca o contéudo das linhas data[k1][] com data[k2][] */
  int a = data[k1][0], int b = data[k1][1], int a = data[k1][2];
  data[k1][0] = data[k2][0]; data[k1][1] = data[k2][1];
  data[k1][2] = data[k2][2];
  data[k2][0] = a; data[k2][1] = b; data[k2][2] = c;
}

void ClassificaDatas(int data[][3], int n) {
   int i, j;
   for (i = 1; i < n; i++) {
      j = i;
      while (j > 0 && ComparaDatas(j, j - 1) == -1) {
           TrocaLinhas(j, j - 1);
           j--;
      }
   }
}
```

Questão 4 (2 pontos)

Considere uma Árvore Binária de Busca (ABB).
Cada nó tem a seguinte estrutura:

```
typedef struct elemento * link;
struct elemento {int info; link eprox, dprox;}
```

Escreva uma função **`int MaiorDosMenores(link R)`**, que devolve o valor do campo **`info`** do maior dos elementos dentre os menores que o indicado por **`R`**. No exemplo abaixo, se **`R`** indicar o elemento com 7 seria devolvido 5, se R indicar o elemento com 9 seria devolvido 8, se **`R`** indicar o elemento com 5 seria devolvido -1 (não existe tal valor).




```
int MaiorDosMenores(link R) {
   link p;
   /* verifica se tal elemento existe */
   if (R == NULL or R -> eprox == NULL) return -1;
   /* procura o elemento
      será aquele que estiver mais à direita de eprox */
   p = R -> eprox;
   while (p -> dprox != NULL) p = p -> dprox;
   return p -> info;
}
```

Questão 5 (2 pontos):

Sobre o método de classificação Quick escreva V (Verdadeiro) ou F (Falso) em frente de cada uma das afirmações abaixo:

a) Quando a tabela está invertida o número de trocas do Quick é máximo.

b) A complexidade do método Quick é **`O(N.log N)`**.

c) O método Quick não precisa de uma tabela auxiliar para fazer a classificação

d) Quando a sequência já está classificada, o Quick não efetua trocas entre os elementos.

e) Se a sequência tem muitas inversões, o Quick é o melhor dos métodos.

Sobre o método de classificação Heap escreva V (Verdadeiro) ou F (Falso) em frente de cada uma das afirmações abaixo:

a) A complexidade do método Heap é **`O(N.log N)`**.

b) O Heap precisa de uma tabela auxiliar para efetuar a classificação

c) Quando a sequência já está classificada o Heap não efetua trocas entre os elementos.

d) Se a sequência tem muitas inversões, o Heap é o melhor dos métodos.

e) Mesmo quando a sequência já está classificada, o Heap efetua trocas entre os elementos.

Questão 6 (2 pontos)

Considere o algoritmo de Boyer-Moore versões 1 e 2 para procurar uma palavra **A** em um texto **B:**
versão 1 - verifica próximo elemento de B para decidir qual o deslocamento.
versão 2 – baseado no trecho final que coincide, verifica se existe um trecho igual dentro de **A** e decide qual o deslocamento. Ambos os algoritmos só dependem de **A**.

Supondo **A = b a b c a b** , diga qual o deslocamento em cada uma das versões do algoritmo:

a) versão 1

Próximo caractere de B	Deslocamento
a	2
b	1
c	3
qualquer outro	7

b) versão 2

Trecho coincidente	Deslocamento
b	3
a b	3
c a b	5
b c a b	5
a b c a b	5
b a b c a b	5
nenhum trecho	1

Idem para **A = c b a c b a**

a) versão 1

Próximo caractere de B	Deslocamento
a	1
b	2
c	3
qualquer outro	7

b) versão 2

Trecho coincidente	Deslocamento
a	3
b a	3
c b a	3
a c b a	3
b a c b a	3
c b a c b a	3
nenhum trecho	1