MAC 122 - PDA
2. Semestre de 2015 - prof. Marcilio – IME USP BMAC
Segunda Prova – 26 de Novembro de 2015

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

Nome:_____ NUSP:_____

Questão 1 (1 ponto):

O comando **`x = x + i`** sob os **`for`** abaixo é executados certo número de vezes. Diga quantas vezes em função de N e M e qual a ordem (função O(..)) para esse número de repetições.

a) _____ - _____ **N vezes – O(N)**

```
int i, N, M, x = 0;
for (i = 1; i <=  N; i++) x = x + i;
```

b) _____ - _____ **1+$\log_2$ N (truncado) vezes – O(log N)**

```
int i, N, M, x = 0;
for (i = 1, i <=  N; i = i * 2) x = x + i;
```

c) _____ - _____**1+ $\log_{10}$ N (truncado) vezes – O(log N)**

```
int i, N, M, x = 0;
for (i = N, i > 0; i = i / 10) x = x + i;
```

d) _____ - _____ **N.M vezes – O(N.M)**

```
int i, N, M, x = 0;
for (i = 1, i <=  N; i++)
   for (j = 1, j <= M; j++) x = x + i;
```

e) _____ - _____ **N.(N+1)/2 vezes – O(N^2)**

```
int i, N, M, x = 0;
for (i = 1, i <=  N; i++)
   for (j = 1, j <= i; j++) x = x + i;
```

f) _____ - _____ **N.(1+$\log_2$ N (truncado))– O(N.log N)**

```
int i, N, M, x = 0;
for (i = 1, i <=  N; i = i * 2)
   for (j = 1, j <= N; j++) x = x + i;
```

Questão 2 (2 pontos):

Considere o algoritmo de Busca Binária em tabela ordenada.

a) Escreva uma função **não recursiva** `int BB(int A[], int X, int N)` que procura **X** nos **N** elementos do vetor **A**, de **A[0]** até **A[N-1]**. Retorna **0** (zero) se encontrou ou **-1** se não encontrou.

```
int BB(int A[], int X, int N) {
   int m, i = 0, f = N - 1;
   while (i <= f) {
       m = (i + f) / 2;
       if (A[m] == X) return 0;
       if (X < A[m]) f = m - 1;
       else i = m + 1;
   }
   return -1;
}
```

b) Idem de forma **recursiva**. A função deve ter os mesmos parâmetros, ou seja `int BB(int A[], int X, int N)`

```
int BBRec(int A[], int X, int N) {
   int m, NB, NC;
   // se n chegou a 0 - não achou
   if (N == 0) return -1;
   // meio da tabela
   m = (N - 1) / 2;
   if (X == A[m]) return 0; // achou
   // está acima ou abaixo?
   if (X < A[m]) {
      // está acima - número de elementos depende se N para ou impar
      if (N % 2 == 0) NC = N / 2 - 1; else NC = N / 2;
      return BB(A, X, NC);
   }
   // está abaixo - número de elementos é sempre N / 2;
   // tem que alterar também a base do vetor
   return BB(&A[m + 1], X, N / 2);
   // pode ser também return BB(A + m + 1, X, N / 2);
}
```

Questão 3 (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;}
```

a) Escreva uma função **`int MenorValor(link R)`**, que devolve o valor do menor elemento (campo **`info`**) da ABB apontada por **`R`**. Pode supor que **`R`** diferente de **`NULL`**. Escreva a função na forma **recursiva**.

```
int MenorValor(link R) {
   if (R -> eprox == NULL) return R -> info;
   return MenorValor (R -> eprox);
}
```

b) Idem na forma **não recursiva**.

```
int MenorValor(link R) {
   // varre a parte esquerda até encontrar nó sem filho esquerdo
   while (R -> eprox != NULL) R = R -> eprox;
   // esse é o menor
   return R -> info);
}
```

Questão 4 (2 pontos):

Considere os métodos de classificação Merge, Quick e Heap.
Diga a qual ou a quais deles se aplica cada uma das afirmações abaixo:

a) A complexidade é O(N.log N)

Heap, Quick e Merge

b) Precisa de uma tabela auxiliar do mesmo tamanho que a tabela original

Merge

c) Realiza trocas mesmo se a tabela já estiver classificada

Heap

d) Usa um algoritmo que divide a sequência em duas partes – Menores a esquerda e maiores a direita do elemento que dividiu a sequência.

Quick

e) Usa algoritmo de intercalação de duas sequências já ordenadas

Merge

Considere os métodos de classificação Seleção, Bolha e Shell.
Diga a qual ou a quais deles se aplica cada uma das afirmações abaixo:

a) A complexidade é sempre $O(N^2)$.

Seleção ou (Seleção e Bolha) – considerei as duas respostas

b) Dependendo da sequência de passos a ser usada em cada repetição, a complexidade pode ser menor que $O(N^2)$.

Shell

c) Quando a sequência está invertida, ocorre o maior número de trocas.

Seleção é Bolha

d) Quando a sequência já está classificada não são efetuadas trocas.

Seleção, Bolha e Shell

e) O número de comparações é constante independente da sequência.

Seleção

Questão 5 (1 ponto) :

O Algoritmo de Boyer-Moore (versão 1) para procurar uma palavra **A** em um texto **B**, verifica próximo elemento de **B** para decidir qual o deslocamento. O algoritmo conta quantas vezes **A** aparece em **B**.
O número de **tentativas** é a quantidade de vezes que **A** é comparada com um **novo trecho** de **B**.

Considere agora a palavra **A** como o seu **NUSP** formado por 7 dígitos. Vamos procurá-lo dentro de um texto B qualquer usando esse algoritmo.

Qual o deslocamento necessário se o próximo caractere de B for:

Supondo NUSP = 4367356

Caractere	Deslocamento
0	**8**
1	**8**
2	**8**
3	**3**
4	**7**
5	**2**
6	**1**
7	**4**
8	**8**
9	**8**
Qualquer outro	**8**

Questão 6 (2 pontos):

Um carregamento é composto de **N** pacotes.
O peso de cada pacote está no vetor **Peso[1..N]** (**Peso[i] =** Peso do pacote **i**)
O meio de transporte suporta no máximo um peso **PesoMax**.

Escreva a função **void QuaisPacotes (int N, double Peso[], double PesoMax)** que lista (imprime) quais os possíveis subconjuntos de pacotes que podem ser transportados.

Pode supor que exista a função **int ProxLex(int S[], int K, int N)** que gera uma **subsequência** de 1..N na ordem lexicográfica a partir da anterior.

Ou seja, o trecho abaixo, gera todas as sequências:

```
k = 0;
while (1) {
           /* gera a próxima sequência */
           k = ProxLex(Seq, k, N);
           if (k == 0) return; /* última sequência */
           .......
}
```

Temos quer gerar todos os subconjuntos possíveis de pacotes e para cada um deles testar a soma dos pesos. Vamos usar a função ProxLex para gerar esses subconjuntos.

```
void QuaisPacotes (int N, double Peso[], double PesoMax) {
     int seq[100], i, k = 0;
     double pt;
     while (1) {
         /* gera a próxima sequência */
         k = ProxLex(seq, k, N);
         if (k == 0) return; /* última sequência */
         /* testa peso total */
         pt = 0;
         for (i = 1; i <= k; i++) pt = pt + Peso[seq[i]];
         if (pt <= PesoMax) Imprima(seq, k);
     }
}
```