BCC202 – Estruturas de Dados I (2015-01)

Departamento de Computação - Universidade Federal de Ouro Preto - MG

Prova 01: 10 pontos (15/04/2015 às 10:10) **Nota**:_____
Nome: _____ **Matrícula**: _____

Leia com atenção as instruções abaixo antes de iniciar a solução da prova:

- Esta prova é ***individual***, ***sem consulta*** e tem duração de 1 hora e 40 minutos (das 10:10 às 11:50).
- Preencha o ***nome*** e o ***número de matrícula*** nos locais indicados da folha de prova e nas folhas de resposta.
- A solução das questões pode ser feita a ***tinta ou lápis***, nas folhas de respostas.
- A interpretação das questões faz parte da prova. Leia com atenção e mantenha a calma. Caso alguma questão pareça dúbia, escreva junto à resposta qual foi a sua interpretação e as decisões tomadas.
- ***Boa Prova!***

Questão 1 (2.5 pontos)
Implemente um TAD (Tipo Abstrato de Dado) para representar um ponto no R^2. O conjunto de operações que operam sobre esse tipo é lista a seguir:

- *cria*: operação que cria e aloca dinamicamento um ponto com coordenadas *x* e *y*.
- *libera*: operação que libera a memória alocada por um ponto;
- *acessa*: operação que devolve as coordenadas de um ponto;
- *atribui*: operação que atribui novos valores às coordenadas de um ponto;
- *distancia*: operação que calcula a distância entre dois pontos.
- *copia*: operação que efetua uma cópia de um ponto p_1 para um ponto p_2.

Importante: Não é necessário implementar uma função *main.*

Questão 2 (1 ponto**)**
A *pesquisa* ou *busca binária* (em inglês, *binary search algorithm* ou *binary chop*) é um algoritmo de busca emvetores que segue o paradigma de divisão e conquista. Ela parte do pressuposto de que o vetor está ordenado e realiza sucessivas divisões do espaço de busca comparando o elemento buscado (chave) com o elemento no meio do vetor. Se o elemento do meio do vetor for a chave, a busca termina com sucesso. Caso contrário, se o elemento do meio vier antes do elemento buscado, então a busca continua na metade posterior do vetor. E finalmente, se o elemento do meio vier depois da chave, a busca continua na metade anterior do vetor.

a- (1.0) Implemente a busca binária usando uma única função recursiva. Considere que seu vetor está devidamente ordenado.
b- (*0.5 extra*) Formule e resolva a equação de recorrência do seu algoritmo e defina a ordem de complexidade da sua função.

Importante: Não é necessário implementar uma função *main*. Implemente somente as funções correspondentes considerando seus retornos e possíveis parâmetros.

Questão 3 (2 pontos)
Exponenciação ou *potenciação* é uma operação matemática, escrita como $\boldsymbol{x^n}$, envolvendo dois números: a base *x* e o expoente *n*. Quando *n* é um número natural maior do que *1*, a potência x^n indica a multiplicação da base ***x*** por ela mesma tantas vezes quanto indicar o expoente ***n***. Uma simples implementação recursiva para exponenciação é ilustrada a seguir e sua complexidade de tempo é ***O(n)***.

```
power( x, n)
{
        if ( n==0)
                return 1;
        if(n==1)
                return x;
        else
                return ( x * power( x , n – 1 ) ;
}
```

Ainda sobre exponenciação, considere as afirmações abaixo.

- $x^n = [(x^2)^{n/2}]$, se n é par.
- $x^n = x * [(x^2)^{(n-1)/2}]$, se n é ímpar

a- (1.0) Com base nas afirmações anteriores, implemente uma nova função recursiva para o cálculo da exponenciação.
b- (0,5) Formule a equação de recorrência para o seu algoritmo.
c- (0,5) Resolva a equação definida e indique a complexidade do novo algoritmo para exponenciação e aponte qual algoritmo é mais eficiente (*i.e.*, o algoritmo apresentado ou sua nova solução).

Importante: Não é necessário implementar uma função *main*. Implemente somente as funções correspondentes considerando seus retornos e possíveis parâmetros.

Questão 4 (2.5 pontos)
Indique e justifique se as afirmações a seguir são verdadeiras ou falsas.

a) $T(n) = n^3 + 20n + 1$ é $O(n^2)$
b) Se g = O(f) e h = O(f) então g = O(h)
c) Considere a função definida pela recorrência:

- f(n) = 2f(n−1)
- f(0) = 1

f(n) = O (n) pode ser verificado da seguinte forma:

f(n) = 2f(n-1) => 2 O (n-1) => 2 O(n) => O (n).

d) Você possui um algoritmo A para multiplicação de matrizes que é $f(n) = O(n^3)$, onde n representa o número de linhas e colunas na matriz. Suponha que exista um algoritmo B para a mesma aplicação no qual seja $g(n)=O(2^n)$. A melhor escolha é sempre o algoritmo A, pois f(n) = O(g(n)).