Edital de Seleção 071/2023-PROPESP/UFAM

1. Considere o código abaixo cujo objetivo é ordenar um vetor de inteiros com n elementos de maneira crescente:

```
void algoritmo(int v[], int n) {
  int i,j,k, tmp;

 for(i = 0; i < n-1; i++) {
  k = i;
    for(j = 0; j < n; j++) {
        if(v[k] > v[j]) {  k = j  }
    }
    tmp = v[k];
    v[k] = v[i];
    v[i] = tmp;
}
```

A) O código ordena o vetor por seleção de maneira crescente
B) O código não ordena o vetor por seleção e não pode ser modificado para ordenar por seleção de maneira crescente com correção em uma única linha de código.
C) O código pode ser alterado em uma única linha para ordenar o vetor de maneira crescente por seleção
D) O código ordena corretamente o vetor, mas não usa o algoritmo de seleção

2. Sobre o *heapsort*, é correto afirmar que:

I) Seu custo no pior caso é O(n log n), onde n é o tamanho do vetor
II) O algoritmo pode ser dividido em duas etapas, sendo a primeira usada para construir um *heap* e a segunda para usar o *heap* como forma eficiente de executar uma estratégia de ordenação por seleção
III) Tem custo máximo quadrático no pior caso

A) Apenas I e II estão corretas
B) Apenas II e III estão corretas
C) Apenas I está correta
D) Apenas III está correta

3. Em um *heap* binário de mínimo contendo N elementos e implementado em um vetor, sabendo-se que o elemento mínimo está na posição zero do vetor, a operação de remover o elemento da raiz (o da posição zero) e deixar a estrutura do *heap* correta ao final, com o restante das chaves formando um *heap*, pode ser implementada, na melhor maneira possível, a custo:

A) N log N no pior caso
B) Log N no pior caso, mesmo sabendo-se que o custo de mover todos os elementos de um vetor para o lado é N
C) N no pior caso, mesmo sabendo-se que o custo de mover todos os elementos de um vetor para o lado é N
D) A custo O(1) no pior caso, pois sabemos que o elemento está na posição zero

4. Se um programador fez testes para avaliar o tempo de execução para entradas de tamanho 100 em um programa $\Theta(n^3)$ e obteve tempo de 1 minuto, quanto tempo demoraria para uma entrada de tamanho 300? (Considere que 100 é suficientemente grande para desprezar fatores menores na função de custo, e que o custo unitário por operação não variou com o tamanho do problema entre os dois experimentos.)

A) 3 minutos
B) 9 minutos
C) 27 minutos
D) 300 minutos

5. Sobre as afirmativas abaixo:

I) Todo algoritmo que não seja vazio tem custo $\Omega(1)$
II) Todo algoritmo tem custo $O(n^n)$
III) Nenhum algoritmo que não seja vazio tem custo o(2) – "ó pequeno de 2"

A) Apenas I e II estão corretas
B) Apenas I e III estão corretas
C) Apenas II e III estão corretas
D) Todas estão corretas

6. Considere o vetor abaixo, o qual é apresentado em uma tabela onde a primeira linha indica a posição dos elementos e a segunda linha os valores armazenados.

0	1	2	3	4	5	6	7	8
2	4	6	8	10	12	14	16	18

Uma busca binária nesse vetor compararia a chave 9 com os seguintes valores do vetor:

A) 2, 6, 10, 14 e 18
B) 10, 4, 6 e 8
C) Nenhuma porque a chave não existe no vetor
D) 10, 14 e 18

7. Sobre o custo das operações de busca por uma chave em árvores vermelho e preto, sabe-se que:

   I) Uma árvore vermelho e preto tem altura máxima log n, onde n é o número de chaves, pois a mesma é balanceada
   II) Uma árvore vermelho e preto tem altura máxima inferior a 3 log n, onde n é o número de chaves, pois a mesma é balanceada
   III) Uma árvore vermelho e preto pode ser degenerada no pior caso, tendo custo de busca crescendo linearmente com o número de elementos, mas em situações muito raras, pois a mesma é balanceada

   A) Apenas I é verdadeira
   B) Apenas II é verdadeira
   C) Apenas III é verdadeira
   D) Apenas I e II são verdadeiras

8. Considere uma pilha que inicialmente estava vazia e na qual inseriu-se as chaves 2, 4 e 6, 8 e 10 nessa ordem. Sobre uma operação de remoção de elemento da mesma (retirada padrão de uma pilha), considerando-se que a estrutura de dados usada é uma lista encadeada, pode-se afirmar que:

   I) A remoção retira a chave 2
   II) A remoção da chave 10
   III) Custo computacional proporcional ao tamanho da pilha
   IV) Custo computacional constante, não importando o tamanho da pilha

   A) Apenas I e III estão corretas
   B) Apenas I e IV estão corretas
   C) Apenas II e III estão corretas
   D) Apenas II e IV estão corretas

9. Sobre um vetor que se mantém de forma ordenada a cada inserção, remoção ou alteração feita, qual a alternativa verdadeira ao usarmos os algoritmos mais eficientes para cada operação:

   A) A inserção tem custo linear no pior caso, a remoção tem custo linear no pior caso e a busca mais eficiente possível tem custo linear
   B) A inserção tem custo linear no pior caso, a remoção tem custo linear no pior caso e a busca tem custo logarítmico no pior caso
   C) A inserção tem custo logarítmico no pior caso, a remoção tem custo logarítmico no pior caso e a busca mais eficiente possível tem custo constante
   D) A inserção tem custo linear no pior caso, a remoção tem custo logarítmico no pior caso e a busca mais eficiente possível tem custo logarítmico

10. Sobre os algoritmos de ordenação *quicksort, mergesort* e inserção, é correto afirmar que:

I) O *quicksort* é o que apresenta melhor desempenho no pior caso dentre os três algoritmos, sendo por isso o algoritmo mais usado
II) A ordenação por inserção tende a ser a mais rápida em casos de ordenação de um vetor onde o número de elementos fora de ordem é menor que um décimo de log N, sendo N o número de elementos do vetor e N um valor grande
III) O *mergesort* é o que tem menor custo assintótico no pior caso quando comparamos os três algoritmos

A) Apenas I é verdadeira
B) Apenas II é verdadeira
C) Apenas III é verdadeira
D) As alternativas A, B e C não são corretas

11. Marque a alternativa correta sobre os custos, no pior caso, dos melhores algoritmos conhecidos para inserir um elemento, remover um elemento e pesquisar por uma chave em um vetor que contenha n elementos, sendo a busca feita pelo campo usado para ordenar o vetor. Considere que o vetor sempre esteja ordenado antes da operação e permaneça ordenado após a operação. Pode-se dizer que as operações de inserção, remoção e busca nesse cenário têm, respectivamente, custo no pior caso:

A) O(n. log(n)), O(n. log(n)) e O(log (n))
B) O(n. log(n)), O(n. log(n)) e O(n)
C) O(n), O(n) e O(n)
D) O(n), O(n) e O(log (n))

12. Considere que um vetor de 8 elementos está sendo ordenado de forma crescente pelo algoritmo *quicksort*, e que após a escolha do *pivot* (elemento usado como chave para o particionamento) e ao final do primeiro particionamento, o vetor tenha ficado com os seguintes valores:

2	5	9	1	7	2	1	9

Dado o resultado, pode-se afirmar que:

A) O valor 2 pode ter sido o pivô que gerou a partição
B) O valor 9 pode ter sido o pivô que gerou a partição
C) O valor 1 pode ter sido o pivô que gerou a partição
D) O valor 7 pode ter sido o pivô que gerou a partição

13. Considere uma tabela *hash* por encadeamento onde os elementos inseridos são números inteiros positivos. Considere que a tabela tem tamanho 5 e que a função de espalhamento (função de *hash*) calcula o resto da divisão do valor do elemento por 5 para determinar a posição de inserção na tabela. Quantas colisões de chaves teremos se a tabela estiver inicialmente vazia e inserirmos as chaves 2, 5, 9, 20 e 7?

    A) Não há colisões
    B) Uma colisão
    C) Duas colisões
    D) Três colisões

14. Considere uma árvore binária de pesquisa sem balanceamento contendo como dado números inteiros, que começou vazia, e onde foram inseridos posteriormente, nesta ordem, os valores 10, 5, 20, 1, 30, 7 e 15. Uma chave de busca nessa árvore sem balanceamento terá que comparar a chave de busca, na pior das hipóteses, com:

    A) 7 nós da árvore
    B) 6 nós da árvore
    C) 2 nós da árvore
    D) 3 nós da árvore

15. Dado o seguinte pseudocódigo Python de busca binária recursiva, indique qual das opções abaixo é impressa pela linha final print().

```
def pesquisa_binaria_recursiva(A, esquerda, direita, item):
    if direita < esquerda:
        return -1
    meio = (esquerda + direita) // 2
    if A[meio] == item:
        return meio
    elif A[meio] > item:
        return pesquisa_binaria_recursiva(A, esquerda, meio - 1, item)
    else:
        return pesquisa_binaria_recursiva(A, meio + 1, direita, item)

A = [0, 100, 20, 30, 40, 50, 60, 70]
print("Pesquisa com sucesso:", pesquisa_binaria_recursiva(A, 0, len(A)
 - 1, 100))
```

A) Pesquisa com sucesso: 100
B) Pesquisa com sucesso: -1
C) Pesquisa com sucesso: 1
D) Pesquisa com sucesso: 2

16. Considere a seguinte afirmação:

    Considerando o pior caso no custo das operações, em uma tabela *hash* com encadeamento na qual uma lista encadeada é usada para tratamento de colisões e uma chave no *hash* não pode se repetir, uma inserção tem custo de __, enquanto que uma busca tem custo de __. Entretanto, se tirarmos a lista encadeada e usarmos uma árvore rubro-negra para o tratamento de colisões, o custo de inserção na tabela *hash* passa a ser __ e o custo de busca passa a ser __.

    Assinale a opção que preenche corretamente os campos em branco:

    A) O(n), O(n), O(log n), O(log n)
    B) O(1), O(n), O(1), O(log n)
    C) O(n), O(1), O(log n), O(1)
    D) O(1), O(1), O(1), O(1)

17. Dado o seguinte pseudocódigo Python de busca binária recursiva, indique quantas vezes ocorre a chamada da função pesquisa_binaria_recursiva():

```
def pesquisa_binaria_recursiva(A, esquerda, direita, item):
    if direita < esquerda:
        return -1
    meio = (esquerda + direita) // 2
    if A[meio] == item:
        return meio
    elif A[meio] > item:
        return pesquisa_binaria_recursiva(A, esquerda, meio - 1, item)
    else:
        return pesquisa_binaria_recursiva(A, meio + 1, direita, item)

A = [0, 100, 20, 30, 40, 50, 60, 70]
print("Pesquisa com sucesso:", pesquisa_binaria_recursiva(A, 0, len(A)
 - 1, 70))
```

A) 0 vezes
B) 3 vezes
C) 4 vezes
D) 5 vezes

18. Dado o seguinte pseudocódigo Python, indique a saída correta que será impressa na tela do computador:

```
import numpy as np

A = [[0, 2, 1],[1, 0, 0]]
B = [[7, 0],[0, 10],[11, 0]]

def operacao_com_matrizes():
    resultado = np.zeros((2,2))
    for i in range(len(A)):
        for j in range(len(B[0])):
            for k in range(len(B)):
                resultado[i][j] += A[i][k] * B[k][j]
    return resultado

print(operacao_com_matrizes())
```

A) [[0. 2. 1.]
[1. 0. 0.]]
B) [[7. 0.]
[0. 10.]
[11. 0.]]
C) [[7. 0.]
[11. 20.]]
D) [[11. 20.]
[7. 0.]]

19. Dado o seguinte pseudocódigo Python, indique a saída correta que será impressa na tela do computador quando a seguinte sequência de comandos é executada:

```
class Stack:
    def __init__(self):
        self.stack = []

    def pushit(self,value):
        self.stack.append(value)

    def popit(self):
        try:
            self.stack.pop()
        except:
            print('empty stack')
            self.stack = []
```

Os comandos executados são:

```
stack = Stack(); stack.pushit(1); stack.pushit(3); stack.popit(); st
ack.popit(); stack.popit(); stack.pushit(2); stack.pushit(1); stack.
popit(); print(stack.stack)
```

A) [2, 1]

B) empty stack
[2]

C) empty stack
[]

D) []

20. Dado o seguinte pseudocódigo Python, indique qual seria a saída correta no terminal do computador:

```
def func1(fator):
    def func2(x):
        return x * fator
    return func2

funcao = func1(3)
print(funcao(5))
```

A) A saída é um erro pois a primeira função retorna outra função mas não passou o argumento 'fator'
B) A saída é 15
C) A saída é 3
D) A saída é 0