# Prova I - Teoria

**Entrega** 31 mar em 10:30 **Pontos** 11,5 **Perguntas** 13
**Disponível** 31 mar em 8:40 - 31 mar em 10:30 aproximadamente 2 horas
**Limite de tempo** 110 Minutos

# Instruções

Nossa primeira prova de AEDs II tem duas partes: teórica e prática. Cada uma vale 10 pontos. A prova teórica será realizada no Canvas e a prática, no Verde. A teórica será neste momento. A parte prática foi realizada na aula de laboratório desta semana.

A prova teórica terá 11 questões fechadas no valor de 0,5 pontos e, em seguida, duas abertas no valor de 3 pontos cada. No total, distribuímos 1,5 pontos a mais. Após terminar uma questão, o aluno **não** terá a opção de retornar à mesma. No caso das questões abertas, o aluno poderá enviar um arquivo contendo sua resposta. Essa resposta pode ser uma imagem (foto de boa resolução) ou um código fonte. No horário da aula, você pode acessar a Prova I através do menu "Testes / Prova I".

**Após a prova**, para cada questão aberta, o aluno deve fazer um vídeo de no MÁXIMO 2 minutos explicando sua resposta. Vídeos com mais de 2 minutos serão penalizados. Cada vídeo deverá ser postado no YouTube e seu link submetido no Canvas até às 23:59 do dia da prova. Vídeos alterados após essa data/horário serão zerados. Questões sem o vídeo serão penalizadas. Você pode acessar a submissão dos vídeos através do menu "Testes / Prova I - submissão de vídeos".

Este teste foi travado 31 mar em 10:30.

## Histórico de tentativas

| | Tentativa | Tempo | Pontuação |
|---|---|---|---|
| **MAIS RECENTE** | Tentativa 1 | 98 minutos | 11 de 11,5 |

ⓘ As respostas corretas não estão mais disponíveis.

Pontuação deste teste: **11** de 11,5
Enviado 31 mar em 10:18
Esta tentativa levou 98 minutos.

| **Pergunta 1** | **0,5 / 0,5 pts** |
|---|---|

Um desafio no projeto de algoritmos é a obtenção de um custo computacional reduzido. Para isso, uma habilidade do projetista é contar o número de operações realizadas pelo algoritmo. O trecho de código abaixo realiza algumas operações.

```
for (int i = 1; i < n; i <<= 1){
  a += 3;
}
```

Considerando o código acima, assinale a opção que apresenta a função de complexidade *f(n)* para o melhor e pior caso considerando a operação de adição.

- $f(n) = lg(n)$
- $f(n) = \lceil lg(n) \rceil + 1$
- $f(n) = \lceil lg(n) \rceil$
- $f(n) = n$
- $f(n) = n - 1$

## Pergunta 2

**0,5 / 0,5 pts**

Um desafio no projeto de algoritmos é a obtenção de um custo computacional reduzido. Para isso, o projetista de algoritmos deve ser capaz de contar o número de operações realizadas em seus algoritmos. O trecho de código abaixo realiza algumas operações.

```
for (int i = 0; i < n ; i++){
    for (int j = 0; j < n - 1; j++){
        l = a * 2 + b * 5;
    }
}
```

Considerando o código acima, assinale a opção que apresenta a função de complexidade *f(n)* para o melhor e pior caso considerando a operação número de multiplicações.

- $f\left(n\right) = 2 \times n \times (n-1)$

- $f(n) = 2 \times n \times (n+1)$
- $f(n) = 2 \times n \times lg(n)$
- $f(n) = 2 \times (n-2) \times (n-1)$
- $f(n) = 2 \times n^2$

Incorreta

## Pergunta 3

**0 / 0,5 pts**

A contagem do número de operações realizadas por um algoritmo é uma tarefa fundamental para identificar seu custo computacional. O trecho de código abaixo realiza algumas operações.

```
Random gerador = new Random();
gerador.setSeed(4);
int x = Math.abs(gerador.nextInt());

for (int i = 0; i < n-4; i++){
    if( x % 9 < 4) {
        a *= 2; b *= 3; l *= 2;
    } else if (x % 9 == 5) {
        a *= 2; l *= 3;
    } else  {
        a *= 2;
    }
}
```

Considerando o código acima, marque a opção que apresenta o pior e melhor caso para o número de multiplicações, respectivamente.

- 3(n-4), (n-4)
- n, n
- **3(n-4), n**
- 3(n-3), n
- 3(n-3), (n-3)

## Pergunta 4

**0,5 / 0,5 pts**

O comando condicional *if-else* possibilita a escolha de um grupo de ações a serem executadas quando determinadas condições de entrada são ou não satisfeitas. O trecho de código abaixo contém uma estrutura condicional.

```
if (n < a + 3 || n > b + 4 || n > c + 1){
    l+= 5;
} else {
    l+= 2; k+=3; m+=7; x += 8;
}

if (n > c + 1){
    l+= 2; k+=3; m+=7; x += 8;
} else {
    l+= 5;
}
```

Considerando o código acima, marque a opção que apresenta o melhor e pior caso, respectivamente, para o número de adições.

- 6 e 9
- 4 e 12
- 4 e 9 (selecionada)
- 6 e 12
- 5 e 9

O pior caso tem nove adições e acontece quando as três condições do primeiro if são falsas e, consequentemente, a condição única do segundo if é falsa dado que ela é igual a primeira condição do primeiro if. Dessa forma, o teste do primeiro if realiza 3 adições e sua lista de comandos, quatro. O teste do segundo if realiza uma adição e mais uma na lista do else.

O melhor tem quatro adições e isso acontece quando a primeira condição é verdadeira e a segunda é falsa. Nesse caso, o teste do primeiro if realiza uma adição e sua lista de comandos, uma. No segundo if, temos uma adição do teste e mais uma da lista do else.

## Pergunta 5

**0,5 / 0,5 pts**

Uma das principais estruturas de dados é a matriz que possui diversas aplicações na Computação como, por exemplo, em processamento de imagens e vídeos, otimização de sistemas e teoria dos grafos. O código abaixo realiza a multiplicação entre duas matrizes quadradas.

```
for (i = 0; i < n; i++)
   for (j = 0; j < n; j++)
      for (k = 0; k < n; k++)
         c[i][j] += a[i][k] * b[k][j];
```

Sobre o código acima é correto afirmar:

I. Se alterarmos o primeiro laço para *for(i = n - 1; i >= 5; i--)*, a função de complexidade do algoritmo será mantida.
II. Se alterarmos o primeiro laço para *for(i = n - 1; i >= 5; i--)*, a ordem de complexidade do algoritmo será mantida.
III. Sua função de complexidade para o número de multiplicações envolvendo os elementos do vetor é $f(n) = n^3$.

É correto o que se afirma em

- I, II e III.

I, apenas.

I e II, apenas

III, apenas.

II e III, apenas.

A função de complexidade para o número de multiplicações envolvendo os elementos do vetor é $f(n) = n^3 = O(n^3)$.

I. ERRADA - A alteração fará com que a função de complexidade seja $f(n) = (n-5)n^2$

II. CORRETA - A alteração fará com que a função de complexidade seja $f(n) = (n-5)n^2 = O(n^3)$

III. CORRETA.

Incorreta

## Pergunta 6

0,5 / 0,5 pts

A pesquisa é um dos problemas mais importantes na Computação. Nesse caso, dado um conjunto de elementos e cada um desses identificado por uma chave, precisa-se buscar nesse conjunto uma chave específica. A respeito da pesquisa, avalie as afirmações a seguir:

O algoritmo de pesquisa binária pode ser aplicado para qualquer disposição dos elementos do vetor.

A função de complexidade do algoritmo de pesquisa binária para o número de comparações envolvendo os elementos do vetor é $f(n) = \lg(n)$.

No pior caso, o algoritmo de pesquisa sequencial realiza menos comparações entre elementos do vetor do que o pior caso do algoritmo de pesquisa binária.

No melhor caso, o algoritmo de pesquisa binária realiza menos comparações entre elementos do vetor do que o melhor caso do algoritmo de pesquisa sequencial.

Dado um vetor com elementos ordenados, o algoritmo de pesquisa sequencial pode realizar menos comparações entre elementos do vetor do que a pesquisa binária dependendo o valor a ser pesquisado.

ERRADA: O algoritmo de pesquisa binária pode ser aplicado para qualquer disposição dos elementos do vetor. Ele só pode ser aplicado quando os elementos estiverem ordenados.

ERRADA: A função de complexidade do algoritmo de pesquisa binária para o número de comparações envolvendo os elementos do vetor é $f(n) = \lg(n)$.. A ordem de complexidade é O(lg(n)).

ERRADA: No melhor caso, o algoritmo de pesquisa binária realiza menos comparações entre elementos do vetor do que o melhor caso do algoritmo de pesquisa sequencial. Em ambas as pesquisas, o melhor caso realiza uma comparação.

ERRADA: No pior caso, o algoritmo de pesquisa sequencial realiza menos comparações entre elementos do vetor do que o pior caso do algoritmo de pesquisa binária. O pior caso da pesquisa sequencial realiza X e da binária, Y.

CORRETA: Dado um vetor com elementos ordenados, o algoritmo de pesquisa sequencial pode realizar menos comparações entre elementos do vetor do que a pesquisa binária dependendo o valor a ser pesquisado. Isso acontece, por exemplo, quando o elemento desejado é a primeira posição procurada pela pesquisa sequencial.

Incorreta

## Pergunta 7

**0,5 / 0,5 pts**

A ordenação interna é um problema clássico na Computação. Considerando-o, avalie as asserções que se seguem:

I. O algoritmo Countingsort ordena um vetor com custo linear.

**PORQUE**

II. O limite inferior do problema de ordenação interna é $\Theta(n \times lg\, n)$ para a comparação entre registros.

A respeito dessas asserções, assinale a opção correta.

- As asserções I e II são proposições verdadeiras, mas a segunda não é uma justificativa correta da primeira
- **A asserção I é uma proposição verdadeira, e a asserção II é uma proposição falsa**
- As asserções I e II são proposições falsas
- A asserção I é uma proposição falsa, e a asserção II é uma proposição verdadeira
- As asserções I e II são proposições verdadeiras, e a segunda é uma justificativa correta da primeira

I - CORRETA: O algoritmo Countingsort efetua em tempo linear $\Theta(n)$ a ordenação dos elementos de um vetor. Ele considera três vetores: entrada, contagem e saída. O primeiro passo é em $\Theta(n)$ é criar o vetor de contagem de tal forma que cada posição tenha o número de elementos menores ou iguais aquela posição. O segundo passo é copiar cada elemento do vetor de entrada para o de saída mapeando de tal forma que a posição do elemento no vetor de saída será mapeada a partir do vetor de contagem.

II - CORRETA: É impossível ordenar um vetor com menos do que $\Theta(n \times lg\, n)$ comparações entre os elementos do vetor. O Countingsort não se aplica a tal regra porque ele triplica o espaço de memória e não funciona para qualquer tipo de elemento.

As duas afirmações são independentes.

Incorreta

## Pergunta 8

**0,5 / 0,5 pts**

O Heapsort é um algoritmo de ordenação clássico que utiliza uma estrutura de *heap* invertido para ordenar os elementos do vetor. A primeira etapa desse algoritmo organiza todos os elementos do vetor em um *heap* invertido, estrutura de dados na qual todos os elementos são maiores ou iguais aos seus filhos. A segunda etapa remove a cabeça do *heap* (maior elemento), reduzindo o tamanho do mesmo em uma unidade. Em seguida, a posição que ficou livre recebe o elemento removido. A segunda etapa repete esse processo até que o *heap* tenha somente um elemento, o menor de todos. Considerando a descrição anterior e seus conhecimentos sobre algoritmos de ordenação, avalie as asserções que se seguem:

I. A ordem de complexidade do Heapsort é $O(n \times lg\, n)$ para todos os casos.

**PORQUE**

II. A ordem de complexidade da primeira etapa é $O(n)$ no melhor caso e $O(n \times lg\, n)$ no pior caso e a da segunda etapa é $O(n \times lg\, n)$ para todos os casos.

A respeito dessas asserções, assinale a opção correta.

- (●) A asserção I é uma proposição verdadeira, e a asserção II é uma proposição falsa.
- ( ) As asserções I e II são proposições verdadeiras, mas a segunda não é uma justificativa correta da primeira.
- ( ) A asserção I é uma proposição falsa, e a asserção II é uma proposição verdadeira.
- ( ) As asserções I e II são proposições verdadeiras, e a segunda é uma justificativa correta da primeira.
- ( ) As asserções I e II são proposições falsas.

As duas afirmações são verdadeiras e a segunda, realmente, justifica a primeira.

## Pergunta 9

**0,5 / 0,5 pts**

Considere o algoritmo de ordenação abaixo (TRT-SP'13, adaptada):

```
for(int i = 0; i < n-1; i++){
   int pos = i;
   for(int j = i + 1; j < n; j++){
      if(v[j] < v[pos){
```

```
            pos = j;
        }
    }
    int aux = v[pos];
    v[pos] = v[j];
    v[j] = aux;
}
```

Sendo $\mathrm{n}$ o número de elementos do vetor, qual é o algoritmo apresentado, quantas comparações e movimentações entre elementos do vetor ele realiza?

- Algoritmo de Inserção, $O\left(n \times \lg(n)\right)$ comparações e $O\left(n \times \lg(n)\right)$ movimentações.
- Algoritmo de Inserção, $O\left(n^2\right)$ comparações e $O\left(n^2\right)$ movimentações
- **Algoritmo de Seleção, $O\left(n^2\right)$ comparações e $O\left(n\right)$ movimentações.**
- Algoritmo de Seleção, $O\left(n^2\right)$ comparações e $O\left(n^2\right)$ movimentações.
- Algoritmo de Inserção, $O\left(n^2\right)$ comparações e $O\left(n\right)$ movimentações.

O algoritmo em questão é o Seleção que realiza $f\left(n\right) = \frac{\{n^2\}}{2} + \frac{\{n\}}{2}$ comparações entre elementos do vetor e $f\left(n\right) = n - 1$ movimentações.

## Pergunta 10

**0,5 / 0,5 pts**

Quais destes algoritmos de ordenação têm a classe de complexidade assintótica, no pior caso, em $O\left(n \times \lg(n)\right)$.

- QuickSort, MergeSort e HeapSort
- QuickSort, MergeSort e SelectionSort
- No answer text provided.
- MergeSort e HeapSort
- QuickSort e BubbleSort
- QuickSort e SelectionSort

O pior caso do Quicksort, Selectionsort e Bubblesort são $O\left(n^2\right)$ comparações entre registros. Por outro lado, no caso do Quicksort, seu melhor e caso médio fazem $O\left(n \times \lg(n)\right)$ comparações entre registros (POSCOMP'15).

## Pergunta 11

**0,5 / 0,5 pts**

Considere o seguinte algoritmo de ordenação de elementos em uma lista (IBGE'13):

I. Escolha um elemento que será chamado o pivot da lista.
II. Reordene a lista de tal forma que os elementos menores que o pivot venham antes dele e os elementos maiores ou iguais ao pivot venham depois dele. Essa operação é chamada de partição, e cria duas sublistas: a de menores que o pivot e a de maiores ou iguais ao pivot.
III. Aplique recursivamente os passos 1 e 2 às sublistas de menores e maiores que o pivot.

O algoritmo acima corresponde ao

- Quicksort, e faz, em média, $O\left(n^2\right)$ comparações para ordenar $n$ itens.

Quicksort, e faz, em média, $O\left(n \times \lg(n)\right)$ comparações para ordenar $n$ itens.

Insertionsort, e faz, em média, $O\left(n\right)$ comparações para ordenar $n$ itens.

Insertionsort, e faz, em média, $O\left(n \times \lg(n)\right)$ comparações para ordenar $n$ itens.

## Pergunta 12

3 / 3 pts

Encontre a fórmula fechada do somatório $\sum_0^n (5i+2)^2$ e, em seguida, prove a usando indução matemática.

Prova01Pergunta12.pdf (https://pucminas.instructure.com/files/3093466/download)

A ser explicada na aula posterior à prova.

## Pergunta 13

3 / 3 pts

Considere a classe **Fila circular** de inteiros vista na sala de aula. Crie o método *void separar(FilaCicular f1, FilaCircular f2)* que recebe dois objetos do tipo **Fila Circular f1** e **f2** e copia todos os números ímpares da Fila corrente para f1 e os pares para f2. Seu método não pode utilizar os métodos de inserir e remover existentes na fila.

***Nesta questão você deve apresentar o código fonte da solução e explicá-lo via vídeo ou áudio. Ao final, junte os dois arquivos em um zip e entregue fazendo upload.***

**Pergunta13.zip**
**(https://pucminas.instructure.com/files/3093800/download)**

Esta questão exige que o aluno desenfileire um elemento da fila principal e teste se o resto da divisão do mesmo por 2 é igual a 0. Assim, ele enfileira em **f1** se for **0** e em **f2** se for **1,** apenas com o ajuste dos valores de ultimo de cada estrutura.

Pontuação do teste: **11** de 11,5