IMD0029 - Estrutura de Dados Básicas 1 –2023.1 – Prova 01
Prof. Eiji Adachi M. Barbosa

Nome:_____

Matrícula: _____

ANTES DE COMEÇAR A PROVA, leia atentamente as seguintes instruções:

- Esta é uma prova escrita de caráter individual e sem consultas a pessoas ou material (impresso ou eletrônico).
- A prova vale 5,0 pontos na Unidade I e o valor de cada questão é informado no seu enunciado.
- Preze por respostas legíveis, bem organizadas e simples.
- As respostas devem ser em caneta. Respostas em lápis serão aceitas, mas eventuais questionamentos sobre a correção não serão aceitos.
- Celulares e outros dispositivos eletrônicos devem permanecer desligados durante toda a prova.
- **Desvios éticos ou de honestidade levarão a nota igual a zero na Unidade 1.**

Questão 1: (1,5ponto) Considere um array A contendo N inteiros, com possíveis repetições, já ordenado em ordem crescente. Nesta questão, faça uma função que recebe como entrada um inteiro K e retorna um inteiro i que indica o índice no array A do primeiro elemento que seja maior ou igual ao valor de K. Por exemplo: considerando o array A = {0, 2, 2, 2, 3, 8, 8, 8, 10} e K = 6, sua função deve retornar i = 5, pois este é o índice do primeiro elemento maior ou igual a K que está no array de entrada. Sua função deverá obrigatoriamente ser **recursiva**, ter complexidade **Θ(lg(n))** e seguir a assinatura:

```
int acharMaiorOuIgual(int a[], int tamanho, int k)
```

Obs.: Nesta questão, não podem ser usadas instruções para realizar repetição, como for, while e do-while. Ou seja, você deverá construir sua solução apenas com chamadas recursivas.

Questão2: (1,0 ponto) Nesta questão, implemente uma função **recursiva** para contar quantos números ímpares um array de entrada contém. Sua função deverá seguir a assinatura:

```
int contarImpares(int array, int tamanho)
```

Obs.: Um número inteiro X é ímpar se x % 2 == 1.

Questão3: (1,0 ponto) O algoritmo de ordenação por inserção funciona mantendo num array A de tamanho N duas regiões distintas: uma região que fica entre [0...i], onde ficam os elementos já ordenados, e uma outra região entre [i+1...N-1], onde ficam os elementos ainda não ordenados. A cada iteração, o algoritmo pega o primeiro elemento na região não ordenada, busca a posição correta de onde inserir o elemento na região ordenada e insere-o na região ordenada. Considere uma versão do algoritmo de ordenação por inserção que usa a busca binária para encontrar a posição correta de onde o elemento deve ser inserido na região ordenada. O uso da busca binária melhora a eficiência do algoritmo de ordenação por inserção? Justifique, considerando o pior caso.

Questão4: (1,5ponto) Para cada uma das afirmações a seguir, marque V (verdadeiro) ou F (falso), **justificando sucintamente** sua resposta. Marcações de V ou F **sem justificativas não serão aceitas.**

1 – ( ) Se o array de entrada já estiver ordenado, então o algoritmo de ordenação Merge Sort cairá em seu pior caso, que tem complexidade assintótica $\Theta(n^2)$.

2 – ( ) Se o array de entrada já estiver ordenado, então o algoritmo de ordenação Quick Sort cairá em seu pior caso, que tem complexidade assintótica $\Theta(n^2)$.

3 – ( ) No melhor caso, a complexidade assintótica do Selection Sort é menor do que a complexidade dos algoritmos Merge Sort e Quick Sort.

4 – ( ) No melhor caso, a complexidade assintótica do Insertion Sort é menor do que a complexidade dos algoritmos Merge Sort e Quick Sort.

5 – ( ) Os algoritmos de ordenação Quick Sort e Merge Sort possuem a mesma complexidade assintótica para o pior caso.

6 – ( ) Os algoritmos de ordenação Quick Sort e Merge Sort possuem a mesma complexidade assintótica para o melhor caso.

7 – ( ) Os algoritmos de ordenação Quick Sort e Merge Sort gastam a mesma quantidade de memória para ordenar um array de tamanho N.

8 – ( ) O algoritmo de intercalação usado pelo Merge Sort possui complexidade $\Theta(n)$ no pior caso e $\Theta(\lg(n))$ no melhor caso.