4ª Avaliação Individual
Estruturas de Dados I
Prof. Leandro M. Zatesko
30 de junho de 2015

- Este instrumento avaliativo tem início às 21:10 e fim às 22:40. Durante e apenas durante este tempo, o estudante deve resolver as questões, colocando suas soluções num único arquivo de nome `NomeCompletoDoEstudante.zip` na Área de Trabalho, sendo `NomeCompletoDoEstudante` o nome completo do estudante sem diacríticos nem espaços (e.g. o arquivo de Ciça Guimarães deve se chamar `CicaGuimaraes.zip`).
- Esta folha de questões possui páginas (frente e verso).
- Nenhum material pode ser consultado, embora todos os programas e aplicativos instalados no ambiente que não façam uso da Internet possam ser usados. Apenas as máquinas do Laboratório podem ser usadas.
- Caso o estudante deseje fazer uso das folhas de rascunho, deve solicitar ao professor.
- Sobre a mesa são permitidos apenas lápis, canetas e borrachas. Quaisquer outros objetos, como estojos, capacetes, bolsas e mochilas, devem ser acomodados no chão ou na prateleira sobre a mesa.
- O estudante que precisar ir ao banheiro poderá fazê-lo apenas solicitando ao professor. Contudo, não será permitido que mais de um estudante esteja ausente do Laboratório ao mesmo tempo.
- Após terminar sua avaliação, o estudante deverá, permanecendo sentado, solicitar ao professor a submissão de seu arquivo `.zip`.
- Para operações de leitura e impressão, seus códigos devem sempre considerar os dispositivos padrões de entrada e saída (`stdin` e `stdout`, respectivamente).
- O estudante que tentar de algum modo *hackear* o bloqueio da Internet, reiniciar sua máquina ou sua sessão ou fraudar o instrumento avaliativo de algum modo, ou que descumprir alguma das regras estabelecidas neste cabeçalho, terá sua nota imediatamente anulada.

Questão 1 (2,5 pontos)**.** O código abaixo é uma implementação do método de ordenação por inserção para ordenar um *array* de inteiros.

```
void insertion_sort(int V[], int N) {
  int i, j, pivo;
  for (i = 1; i < N; i++) {
    pivo = V[i];
    for (j = i - 1; j >= 0 && V[j] > pivo; j--)
      V[j + 1] = V[j];
    V[j + 1] = pivo;
  }
}
```

Em cada iteração do laço externo, é procurada através de uma busca sequencial a posição em que deve ser inserido o pivô da iteração no subvetor já ordenado.

(a) Re-escreva o código usando uma busca binária ao invés de uma busca sequencial para encontrar a posição em que deve ser inserido o pivô da iteração no subvetor já ordenado.

(b) Qual a complexidade de tempo do seu algoritmo? Explique.

Apresente suas respostas num único arquivo `questao1.pdf`.

Questão 2 (2,5 pontos)**.** O Algoritmo 1 apresenta um pseudocódigo recursivo para o método de ordenação QuickSort, desconsiderando a implementação da função Particione, a qual não é relevante para esta Questão. O pseudocódigo do Algoritmo 1 pode ser re-escrito de modo iterativo utilizando-se uma pilha S para simular a recursão, como no pseudocódigo do Algoritmo 2. Complete o pseudocódigo do Algoritmo 2.

```
QuickSort(V, inicio, fim):
1  se inicio < fim, então:
2     meio ←Particione(V, inicio, fim);
3     QuickSort(V, inicio, meio − 1);
4     QuickSort(V, meio + 1, fim).
```

Algoritmo 1

```
QuickSort(V, inicio, fim):
1  inicialize uma pilha S com o par ordenado (inicio, fim);
2  enquanto a pilha S _____, faça:
3     desempilhe um par (i, j) de S;
4     se _____, então:
5        meio ←Particione_____;
6        empilhe o par _____ na pilha S;
7        empilhe o par _____ na pilha S.
```

Algoritmo 2

Apresente suas respostas num único arquivo `questao2.pdf`.

Questão 3 (2,5 pontos)**.** Os métodos de ordenação estudados podem ser aplicados em listas duplamente encadeadas, alguns de modo mais imediato, outros nem tanto.

(a) Qual a complexidade de tempo do método de ordenação por borbulhamento (*Bubble Sort*) em listas duplamente encadeadas?

(b) Qual a complexidade do método de ordenação por mescla (*Merge Sort*) em listas duplamente encadeadas?

(c) Explique por que a substituição de um *array* por uma lista duplamente encadeada não afeta a complexidade de tempo do método de ordenação por borbulhamento mas afeta a complexidade de tempo do método de ordenação por mescla.

Apresente suas respostas num único arquivo `questao3.pdf`.

Questão 4 (2,5 pontos)**.** O *Problema da Mediana* é definido como: *Dado um array de inteiros* $\boldsymbol{A}[0..n-1]$ *não necessariamente ordenado, qual é a mediana do array?* Por exemplo, no caso do *array* $\boldsymbol{A} = [4,9,0,1,1]$, a mediana é 1, pois 1 é o 3º menor elemento do *array*. No caso do *array* $\boldsymbol{A} = [5,15,18,0,1,1]$, a mediana é 3, pois 3 é a média entre o 3º e o 4º menores elementos.

Não é necessário ordenarmos todo o *array* $\boldsymbol{A}$ para encontrarmos a mediana: basta que encontremos, no caso de n ser ímpar, o elemento que ficaria na posição $(n-1)/2$ se o *array* fosse ordenado, e, no caso de n ser par, os elementos que ficariam nas posições $\lfloor (n-1)/2 \rfloor$ e $\lceil (n-1)/2 \rceil$ se o *array* fosse ordenado.

Para descobrirmos qual elemento ficaria na posição k de um *array* $\boldsymbol{A}$ sem precisarmos ordenar todo o *array*, podemos usar a função de particionamento do *Quick Sort*. Sorteando um elemento do *array* e tomando-o como pivô da função, descobrimos qual a posição i desse elemento no vetor ordenado. Se a posição i não é ainda a posição k, podemos repetir o procedimento recursivamente para o subvetor à esquerda da posição i ou para o subvetor à direita. Como a função de particionamento roda em tempo linear, a complexidade de tempo esperada deste algoritmo é $O(n+n/2+n/4+\cdots+1) = O(2n) = O(n)$, por incrível que pareça :)

Escreva uma função em C de nome `mediana()` que recebe o *array* $\boldsymbol{A}$ e o inteiro n e implementa o algoritmo descrito acima. Seu arquivo `questao4.c` deve conter a implementação da função `mediana()` e de todas as suas funções que são chamadas direta ou indiretamente pela função `mediana()`.