DCC/ICEx/UFMG **Lista de Exercícios 1**
DCC 205 - Estruturas de Dados
1º semestre de 2024

---

1. Para cada uma das afirmativas abaixo, diga se ela é verdadeira (V) ou falsa (F). Em todas as afirmativas, justifique a sua resposta.

   (a) Considere um programa P que faz uma série de operações de custo constante, chama uma função $F1$ com complexidade dada por $f(n)$ e depois chama uma função $F2$ com complexidade dada por $g(n)$, onde $g(n) = 1000f(n)$. Pode-se afirmar que o programa P é $O(f(n))$.

   (b) Considere um programa cuja função de complexidade é $f(n) = 3log(n)$. É correto afirmar que esse programa é $O(log(n))$, mas não é $O(n^2)$.

   (c) Um programa recursivo com equação de recorrência $T(n) = n + T(n\check{}1)$, sendo $T(0) = 1$, tem ordem de complexidade menor que a de um programa que implementa dois loops aninhados com n passos cada.

   (d) Sejam duas funções de complexidade $g(n) = 5n^2 + 3n + 4$ e $f(n) = 95n^2 + n + 15$. É correto afirmar que um programa $P1$ cuja complexidade é $g(n)$ é mais rápido que um programa $P2$, com complexidade $f(n)$.

   (e) Se $f(n) = O(g(n))$ e $g(n) = \Omega(f(n))$ então $f(n) = \Theta(g(n))$.

   (f) Se $f(n) = O(g(n))$ e $g(n) = O(f(n))$ então $f(n) = \Theta(g(n))$

   (g) Se $f(n) = O(g(n))$ e $g(n) = \Omega(h(n))$ então $f(n) = O(h(n))$.

   (h) Se $f(n) = O(g(n))$ e $g(n) = O(h(n))$ então $f(n) = O(h(n))$.

   (i) Se $f(n) + g(n) = O(g(n))$ então $f(n) < g(n)$ para todos os valores de n.

   (j) Seja $T(n) = 3n^3 + n^2$ a função que descreve o tempo de execução de um programa A com uma entrada de tamanho n. Temos que $T(n) \in O(n^3)$.

   (k) Seja $T(n) = 3T(\frac{n}{2}) + n$, $T(0) = 0$ a relação de recorrência que descreve o tempo de execução de um programa B com uma entrada de tamanho n. Temos que $T(n) \in \Theta(nlog(n))$.

   (l) Sejam $f(n)$ e $g(n)$ funções **positivas**. Se $f(n) + g(n) \in O(h(n))$, então $h(n) \in \Omega(f(n))$

   (m) Sejam $f(n)$ e $g(n)$ funções **positivas**. Se $f(n)g(n) \in O(g(n))$, então $f(n) < g(n)$ para todos os valores de n.

2. Dado o código do programa abaixo, pergunta-se:

```
float FazAlgo(int a, int b) {
    if (b == 0)
        return 1;
    else
        return a * FazAlgo(a, b-1);
}
```

   (a) O que ele faz?

   (b) Qual sua ordem de complexidade? Para isso, determine e resolva a equação de recorrência correspondente.

   (c) Qual seria a complexidade de uma implementação não recursiva dessa mesma função? Qual das duas implementações vocês escolheria? Justifique a sua resposta.

3. Para cada um dos algoritmos a seguir indique sua complexidade assintótica utilizando o teorema mestre

   (a)

```
int REC(int n) {
    for(int i = 0; i < n; i++) printf("TEOREMA_MESTRE!");
    if(n > 0) return REC(n/2) + REC(n/2);
}
```

(b)
```c
int REC(int n) {
    for(int i = 0; i < n; i++) printf("TEOREMA_MESTRE!");
    if(n > 0) return REC(n/4) + REC(n/5);
}
```

DICA: Tente determinar um limite superior e um limite inferior em que o teorema mestre seja aplicável

t

5. Assinale com **V** para verdadeiro e **F** para falso as afirmativas abaixo. Não é necessário justificar suas afirmativas, no entanto cada afirmativa incorreta irá anular uma correta.

   [

   ] O método da bolha é um método estável.
   [ ] O método da seleção executa mais movimentações de registros que os outros métodos quadráticos vistos em sala.
   [ ] O método da inserção é indicado para inserir um novo elemento em um vetor ordenado.
   [ ] A equação de recorrência do tempo de execução do caso recursivo do Mergesort é $T(n) = T(\frac{n}{2}) + n$.
   [ ] O pior caso do quicksort acontece quando os pivôs escolhidos são sempre o menor ou o maior elemento de seu respectivo subvetor.
   [ ] O método counting sort requer espaço adicional proporcional ao tamanho do vetor.
   [ ] O melhor caso do Bucket sort é linear na quantidade de elementos do vetor.
   [ ] O tempo de execução do Radix sort independe dos elementos do vetor.
   [ ] Considere a versão não recursiva do Quicksort. Se você substituir a pilha auxiliar por uma fila auxiliar, o algoritmo continua funcionando.

6. Indique se as afirmações a seguir são Verdadeiras ou Falsas e justifique.

   (a) Inserção é, em geral, mais eficiente que o Quicksort para arquivos quase ordenados.
   (b) O Bucket Sort tem complexidade $O(n^2)$ quando o número de buckets tende a n.
   (c) O Counting Sort requer uma memória auxiliar de tamanho n.
   (d) Qualquer algoritmo de ordenação realiza pelo menos $\Omega(nlog(n))$ operações.
   (e) Não conhecemos nenhum algoritmo de ordenação cujo pior e o melhor caso sejam idênticos em termos de complexidade assintótica.
   (f) Insertionsort e Selectionsort são exemplos de algoritmos de ordenação estáveis.
   (g) QuickSort com escolha do pivô realizada através de sorteio, com todas as posições do vetor de entrada podendo ser escolhidas com mesma probabilidade, é um algoritmo ótimo de ordenação
   (h) Um algoritmo de ordenação que realiza comparações e tem complexidade de $O(nlog(n))$ é um algoritmo ótimo.

7. O algoritmo do Quicksort, apesar de otimizado, ainda realiza uma quantidade significativa de movimentações de registros. O desempenho do algoritmo pode ser impactado por essas movimentações em função do tamanho dos registros. O código a seguir mostra as várias funções do algoritmo Quicksort e a definição da estrutura Item, que tem um tamanho significativo (8 megabytes em uma máquina 64 bits).

```c
typedef struct {
  int Chave;
  int Conteudo [1000000];
} Item;
void Particao(int Esq, int Dir,
 int *i, int *j, Item *A){
  Item x, w;
  *i = Esq; *j = Dir;
  x = A[(*i + *j)/2];
  do {
     while (x.Chave > A[*i].Chave) (*i)++;
     while (x.Chave < A[*j].Chave) (*j)--;
     if (*i <= *j){
       w = A[*i];
A[*i] = A[*j];
A[*j] = w;
         (*i)++;
(*j)--;
     }
   } while (*i <= *j);
}

void Ordena(int Esq, int Dir, Item *A){
  int i, int j;
  Particao(Esq, Dir, &i, &j, A);
  if (Esq < j) Ordena(Esq, j, A);
  if (i < Dir) Ordena(i, Dir, A);
}

void QuickSort(Item *A, int n){
  Ordena(0, n-1, A);
}
```

Uma estratégia para minimizar esse impacto é o chamado Quicksort indireto, onde estrutura Item é desmembrada em duas estruturas, uma contendo a Chave e outra contendo o Conteudo (conforme o exemplo). Reescreva o código apresentado (incluindo definição de estruturas) implementando uma estratégia indireta e minimizando o impacto das movimentações. Explicite quaisquer premissas que você tenha levado em consideração na sua proposta.