**1ª Lista de Exercícios de Introdução à Análise de Algoritmos**

**Prof. Glauber Cintra – Entrega: 23/set/2019**

Esta lista deve ser feita por grupos de no **mínimo** 4 e no **máximo** 5 alunos.

1. **(0,5 pontos)** Numere as funções abaixo a partir do 1 em ordem **estritamente** crescente de dominação assintótica. Se f e g são tais que f ∈ o(g) então f deve ter número menor do que g. Se f ∈ Θ(g) então f e g devem ter o mesmo número.

( 7 ) n! ( 1 ) log n ( 5 ) 3n ( 1 ) 2log n3 ( 6 ) 3n+1

( 6 ) fib(n) ( 4 ) n3 ( 2 ) nlog n ( 3 ) 6n2 ( 3 ) 2n + n2

1. **(0,5 pontos)** Explique o significado dos termos *algoritmo*, *algoritmo computacional,* *algoritmo correto,* *algoritmo eficiente* e *tamanho da entrada de um algoritmo*.

**Algoritmo**: Sequência de instruções desenvolvidas para resolver instâncias de um problema.

**Algoritmo Computacional**: Algoritmo criado apenas com instruções bem definidas, não ambíguas.

**Algoritmo Correto**: Algoritmo que resolve corretamente todas as instâncias do problema para qual ele foi desenvolvido.

**Algoritmo Eficiente**: Algoritmo que requer a execução de uma quantidade de instruções elementares limitada por um polinômio no tamanho da entrada.

**Tamanho da Entrada**: quantidade de bits para representar os dados de uma entrada.

1. **(1,5 pontos)** Indique quais são o melhor caso e o pior caso do algoritmo abaixo e sua região crítica. Qual a complexidade temporal desse algoritmo no melhor e no pior caso? Qual a complexidade espacial desse algoritmo? O algoritmo é eficiente? É de cota inferior? Justifique suas respostas. Finalmente, prove que esse algoritmo é correto.

Algoritmo Contido

Entrada: os vetores A e B com m e n posições, respectivamente

Saída: *Sim*, se todos os elementos de A estão contidos em B; *Não*, caso contrário

para i = 0 até m – 1

j = 0

enquanto j < n e A[i] ≠ B[j]

j++

se j >= n

devolva *Não* e pare

devolva *Sim*

**Melhor caso**: O primeiro elemento de A não estar contido em B, pois ao finalizar o laço de comparações, o teste será feito e o algoritmo é finalizado.

**Pior caso**: Apenas o ultimo elemento de A não estar contido em B, pois além de já ter comparado todas as outras posições do Vetor A com Vetor B, na ultima ele irá percorrer todo o vetor B.

**Região crítica**: O laço “enquanto j < n e A[i] ≠ B[j]”, pois acontece que no pior caso ele será feito *(n-1)\*(m-1)*.

**Complexidade temporal / melhor caso**: Θ(n), a instrução crítica nesse caso será a quantidade de vezes que a instrução do laço é feita e é representado por (n-1)

**Complexidade temporal / pior caso**: Θ(nm), a instrução crítica nesse caso será a quantidade de vezes que a instrução do laço é feita e é representado por (n-1)\*(m-1) = nm – n – m +1.

**Complexidade espacial**: Θ(1), pois só se utiliza variaveis escalares.

**Cota inferior**: Assintoticamente sim, pois claramente só é possivel ter a certeza de que A não está contido em B, ou não, após verificar, no minimo, a primeira posição de A em relação a todo o vetor B, se o primeiro elemento de A estiver contido em B o algoritmo continua até ou achar algum elemento que não esteja contido em B ou checar todas a posições de A e mostrar que A está contido em B. Conlui-se qualquer algoritmo que resolva corretamente esse problema gasta tempo Ω(nm), e têm-se que o algoritmo Contido requer tempo O(nm), concluímos que ele é um algoritmo de cota inferior.

**Algoritimo eficiente**: O tamanho da entrada é igual o tamanho do vetor A mais o tamanho do vetor B (n + m). Afirmando com isso que o algoritmo é eficiente.

**Corretude**:

Teorema: O Algoritmo Contido é correto.

A está contido em B: Se a posição A[x0] que ⊇ o elemento k, e k ⊆ B, logo B[y0] ⊇ o elemento k ⇒ que sairá do laço condicional, pois a condição “A[i]≠B[j]” não será satisfeita, e perceba que j será igual y0 e y0 < n. Sendo assim a condição “se j>=n”

nunca séra satisfeita, e devolverá sim, o que é correto.

A não está contido em B: Se a posição A[x0] que ⊇ o elemento k, e k ⊈ B ⇒ que o laço só irá acabar quando a condição “j<n” for insatisfeita, pois ao testar k com todas os elementos em B serão diferentes e somará mais um ao contador j, até que j seja igual a n. Sendo assim a condição “se j>=n” séra satisfeita, devolverá não, o que é correto.

1. **(2 pontos)** Escreva um algoritmo que receba um vetor de números (e, naturalmente, seu tamanho) e devolva *Sim* se existir um número par no vetor; *Não*, caso contrário. Caracterize o pior e o melhor caso do seu algoritmo e determine a complexidade temporal no pior e no melhor caso. Determine também a complexidade espacial do algoritmo. O seu algoritmo é eficiente? É de cota inferior? Prove que seu algoritmo é correto.

Entrada: Vetor A com n posições

Saída: *Sim*, se ao menos um elemento de A é par; *Não*, caso contrário

para i = 0 até n - 1

se A[i] for par

devolva *Sim* e pare

devolva Não

**Melhor caso**: Quando o primeiro elemento do vetor for par, logo, a condição será satisfeita.

**Pior caso:** Quando não há números pares no vetor, assim o algoritmo percorrerá todo o vetor

**Região crítica**:

**Complexidade temporal / melhor caso**: Logo na primeira iteração do laço, a condição do Se será verdadeira, logo o tempo requerido para este caso é O(1).

**Complexidade temporal / pior caso**: Já no pior caso, o vetor V será todo percorrido e a condição do Se nunca será satisfeita. Por isso, o tempo requerido no pior caso é θ(n).

**Complexidade espacial**: Θ(1), pois só se utiliza variaveis escalares.

**Cota inferior**: Simpois o tempo requerido pelo algoritmo é (O(n)) e a entrada tem tamanho n, logo, o tempo é delimitado por um polinômio do tamanho da entr ada.

**Algoritimo eficiente**: Sim, pois o tempo requerido pelo algoritmo é (O(n)) e a entrada tem tamanho n, logo, o tempo é delimitado por um polinômio do tamanho da entrada.

**Corretude**:

Teorema: O algoritmo é correto.

Suponha que não haja nenhum elemento par no vetor A. Logo percebemos que a condição do Se nunca será satisfeita e, no final da iteração, o algoritmo devolverá *Não*, o que é correto. Suponha agora que haja um elemento par no vetor A. Seja *k* a posição desse elemento no vetor. Nota-se que *0 ≤ k ≤ n – 1*. Quando *i = k*, a condição do Se será satisfeita, e o algoritmo devolverá *Sim*, o que é correto.

1. **(1 ponto)** Resolva as seguintes fórmulas de recorrência:
   1. T(n) = 2T(⎣n/2⎦) + n, T(1) = 1

Supondo n = 2^k, ∀ k ≠ 0 e ∈ N ⇒ **⎣**n/2**⎦** = **⎣**2(k-1)**⎦** = 2(k-1). Com isso podemos eliminar a função piso pois ela não será utilizada de a cordo com a nossa suposição já que sempre resultará em um numero natural.

Utilizando o método da somas

*Equação 0*: 1\*T(n)= 2T(n/2) + n

*Equação 1*: 2\*T(n/2) = 4T(n/4) + n

*Equação 2*: 4\*T(n/4) = 8T(n/8) + n

*Equação 3*: 8\*T(n/8) = 16T(n/16) + n

....

*Equação k – 1*: 2(k-1) \* T() = T()+ n

*Equação k*: 2k \* T() e como pressupus que n é igual 2k temos que T()= T(1) = 1.

Ao somar todas as equações ficamos com:

T(n) = n + n + n + n +... +2k + 1 ⇒ T(n) = k\*n + 2k

Ainda baseado na suposição de n = 2k ⇒ k = ⇒ T(n) = n\* + n. No final temos que T(n) ∈ O(n)

* 1. T(n) = T(n – 1) + n/2, T(1) = ½

Utilizando o método da somas

*Equação 0*: T(n)= T(n - 1) +

*Equação 1*: T(n - 1) = T(n - 2) +

*Equação 2*: T(n - 2) = T(n - 3) +

*Equação 3*: T(n - 3) = T(n - 4) +

....

*Equação n – 1*: T(2) = T(1) + 1

*Equação n*: T(1) =

Ao somar todas as equações ficamos com:

T(n) = + + + + ... + 1 +

Aplicando a fórmula da soma de uma P.A obtêm-se: ()\*n = () = () ⇒ T(n) ∈ O(n²)

)

1. **(1,5 pontos)** Considere o algoritmo *enigma* descrito a seguir:

Algoritmo enigma

Entrada: um vetor V e a posição n

se (n = 0)

devolva V[n]

se não

devolva V[n] \* enigma(V, n - 1)

Seja L = [8, 3, 5, 2, 9]. Simule o cálculo de *enigma*(L, 4). Determine a complexidade temporal e espacial do algoritmo *enigma* (exiba os cálculos realizados para determinar tais complexidades). Para que serve esse algoritmo? Ele é eficiente? Prove que o algoritmo é correto.

**Simulação**: O algorirmo enigma pegará o elemento da posição V[4] e multiplicará com a posição V[3], onde o mesmo será multiplicado por V[2] e assim sucessivamente: V[4] \* ( V[3] \* ( V[2] \* ( V[1] \* ( V[0] ) ) ) ) = 9 \* ( 2 \* ( 5 \* ( 3 \* ( 8 ) ) ) ) = 18 \* ( 5 \* ( 3 \* ( 8 ) ) ) = 90 \* ( 3 \* ( 8 ) ) = 270 \* ( 8 ) = 2160

**Complexidade temporal**: Como o algoritmo utliza-se da recursão podemos usar a formula de recorrência para o mesmo.

T(n) = nT(n – 1) + c, T(0) = c

Utilizando o método da soma

*Equação 0*: T(n)= T(n – 1) + c

*Equação 1*: T(n - 1) = T(n - 2) + c

*Equação 2*: T(n - 2) = T(n - 3) + c

*Equação 3*: T(n - 3) = T(n - 4) + c

....

*Equação n – 1*: T(1) = T(0) + c

*Equação n*: T(0) = c

Ao somar todas as equações ficamos com:

T(n) = c(n-1) = cn -1 ⇒ T(n) ∈ O(n)

**Complexidade espacial**:

**Serventia do algoritmo**: Multiplicar todos os elementos de um vetor começando pela n do vetor até a primeira.

**Algoritimo eficiente**:

**Corretude**:

1. **(1 ponto)** Escreva um algoritmo **recursivo** que receba um número *a* e um número natural *b* e devolva *ab*. Prove que seu algoritmo é correto e determine a complexidade temporal e espacial do algoritmo (bônus de **0,5 pontos** se o algoritmo for eficiente).

Algoritmo potencia

Entrada:um numero *a* e um numero

Saída: ab

aux =

Se b é par

Devolva potencia(a, aux) \* potencia(a, aux)

Senão

Devolva potencia(a, aux) \* potencia(a, aux) \* a

**Complexidade temporal**:

A complexidade temporal é dada por T(n) = 2T() + c, T(0) = 1

Supondo n = 2^k, ∀ k ≠ 0 e ∈ N ⇒ **⎣**n/2**⎦** = **⎣**2(k-1)**⎦** = 2(k-1). Com isso podemos eliminar a função piso pois ela não será utilizada de a cordo com a nossa suposição já que sempre resultará em um numero natural.

Utilizando o método da somas

*Equação 0*: 1\* T(n) = 2T() + c

*Equação 1*: 2\* T() = 4T() + c

*Equação 2*: 4\* T() = 8T() + c

*Equação 3*: 8\* T() = 16T()+ c

....

*Equação k – 1*: 2(k-1) \*T () = T(1) + c

*Equação k*: T(1) = c

Ao somar todas as equações ficamos com: T(n) = k \* c + c e como k = ⇒ T(n) = \*c + c ⇒ T(n) O()

**Complexidade espacial**: A complexidade espacial é igual a temporal isso quer dizer que O()

**Corretude**:

**Algoritimo eficiente**:

1. **(0,5 pontos)** Cite o nome de um algoritmo de *cota superior* para o *problema de encontrar uma Trilha de Euler* (*Eulerian Trail*). Este algoritmo também é de cota inferior? Justifique.
2. **(1 ponto)** Escreva um algoritmo de *cota inferior* que receba uma matriz de número com *n* linhas e *m* colunas e devolva a soma dos números contidos na matriz. Determine a complexidade temporal e espacial do algoritmo e justifique porque ele é um algoritmo de cota inferior.

Algoritmo: somaMatriz

Entrada: uma matriz A com n linhas e n colunas

Saída: a soma dos elementos da matriz

soma = 0

Para i = 0 até n – 1

Para j = 0 até m – 1

soma = soma + M[i, j]

Devolva soma

- Complexidade Temporal: o algoritmo percorre as n linhas e m colunas da matriz, logo a complexidade temporal é O(nm).

- Complexidade Espacial: o espaço requerido é O(1), pois há somente declarações de variáveis para os laços e uma variável usada para guardar a soma dos elementos dos vetores.

- O algoritmo é de cota inferior pois é fácil perceber que todo algoritmo correto para esse problema requer tempo Ω(nm) e o algoritmo em questão requer tempo O(nm).

1. **(2 pontos)** Podemos implementar uma fila *F* usando duas pilhas *P1* e *P2*. A operação *insereNaFila(x, F)* é feita empilhando-se *x* em *P1*. A operação *removeDaFila(F)* é feita como indicado abaixo:

Algoritmo removeDaFila

Entrada: uma fila F implementada através das pilhas P1 e P2

Saída: remove um elemento de F

Se P2 for vazia

Enquanto P1 não for vazia

empilha(desempilha(P1), P2)

devolva desempilha(P2) // pode haver um erro (underflow) se P2 for vazia

Utilizando dois dos métodos de análise amortizada que estudamos, mostre que o custo de uma sequência de *n* operações *insereNaFila* e *removeDaFila* é Θ *(n)* e que o *custo amortizado* da operação *removeDaFila* é *O(1)*, considerando que inicialmente a fila está vazia.

A classe de complexidade BPP (bounded-error probabillistic polynomial time) é definida como a classe dos problemas de decisão para os quais existem soluções algorítmicas polinomiais não-determinísticas que produzem a resposta errada com probabilidade inferior à 1/3 para qualquer instância