Universidade Federal do Ceará – UFC
Centro de Ciências – CC
Mestrado e Doutorado em Ciências da Computação - MDCC
Estruturas de Dados Avançadas

Exercício: Complexidade de Algoritmos
Objetivos: Exercitar os conceitos de Complexidade de Algoritmos.
Data da Entrega: 04/04/2017

OBS 1: Exercício Indivudual.
OBS 2: A entrega desta lista deverá ser executada via SIGAA.

NOME: Pedro Roger Magalhães Vasconcelos MATRÍCULA: 394492

<u>Questão 1</u>

As funções *f(n)* mostradas abaixo fornecem o tempo de processamento *T(n)* de um algoritmo resolvendo um problema de tamanho *n*. Complete a tabela abaixo colocando, para cada algoritmo, sua complexidade (*O maiúsculo*) e a ordem do mais eficiente para o menos eficiente. Em caso de empate repita a ordem (por exemplo: 1º ,2º, 2º, ...).

f (n)	$O(\dots)$	*ordem*
$5 + 0.001n^3 + 0.025n$	$O(n^3)$	9
$500n + 100n^{1.5} + 50n\log_{10} n$	$O(n^{1.5})$	5
$0.3n + 5n^{1.5} + 2.5 \cdot n^{1.75}$	O(n^1.75)	6
$n^2 \log_2 n + n(\log_2 n)^2$	O(n^2 log n)	8
$n\log_3 n + n\log_2 n$	O(n log n)	2
$3\log_8 n + \log_2\log_2\log_2 n$	O(log n)	1
$100n + 0.01n^2$	O(n^2)	7
$0.01n + 100n^2$	O(n^2)	7
$2n + n^{0.5} + 0.5n^{1.25}$	O(n^1,25)	4
$0.01n\log_2 n + n(\log_2 n)^2$	O(n(log n)^2)	3
$100n\log_3 n + n^3 + 100n$	O(n^3)	9
$0.003\log_4 n + \log_2\log_2 n$	O(log n)	1

Questão 2

Os algoritmos abaixo são usados para resolver problemas de tamanho n. Descreva e informe para cada algoritmo sua complexidade no pior caso (O maiúsculo/Ômicron). Tente entender o problema antes de apresentar uma resposta.

a)

```
for ( i=1; i < n; i *= 2 ) {
     for ( j = n; j > 0; j /= 2 ) {
          for ( k = j; k < n; k += 2 )
               { sum += (-j * k) <<
               i/2;
          }
     }
}
```

Dado o laço for mais externo, deduzimos que a execução é de tempo logarítmico, pois dobre-se os passos a cada iteração. O laço do meio é executado em tempo constante, já que será executado exatamente quatro vezes, devido a iteração '/= 2' sobre 16. O laço interno leva tempo O(n) para executar. Da multiplicação das complexidades obtemos a complexidade final O(n log n)

b)

```
Leia(n);
x ← 0
Para i ← 1 até n faça
      Para j ← i+1 até n faça
            Para k ← 1 até j-i faça
                  x ← x + 1
```

Os três laços irão executar em ordem de n, mudando apenas a constante. Assim, a complexidade será de $O(n^3)$

Questão 3

Suponha um algoritmo A e um algoritmo B com funções de complexidade de tempo $a(n) = n^2 - n + 549$ e $b(n) = 49n + 49$, respectivamente. Determine quais são os valores de n pertencentes ao conjunto dos números naturais para os quais A leva menos tempo para executar do que B.

Realizamos a comparação A < B das funções que definem A e B para encontrarmos o intervalo no qual A leva menos tempo para executar que B. Assim temos:

nˆ2 – n + 549 < 49n + 49

n^2- 50n + 500 < 0

Achando as raízes:

n = (50 +/- √ (50^2 – 4 * 1 * 500) /2

n1 = 13.8

n2 = 36.1

(50 +/- 22.3)/2

Questão 4

O Casamento de Padrões é um problema clássico em ciência da computação e é aplicado em áreas diversas como pesquisa genética, editoração de textos, buscas na internet, etc. Basicamente, ele consiste em encontrar as ocorrências de um padrão P de tamanho m em um texto T de tamanho n. Por exemplo, no texto T = “PROVA DE AEDSII” o padrão P = “OVA” é encontrado na posição 3 enquanto o padrão P =“OVO” não é encontrado. O algoritmo mais simples para o casamento de padrões é o algoritmo da “Força Bruta”, mostrado abaixo. Analise esse algoritmo e responda: Qual é a função de complexidade do número de comparações de caracteres efetuadas no melhor caso e no pior caso. Dê exemplos de entradas que levam a esses dois casos. Explique sua resposta!

Melhor caso: O padrão é encontrado na primeira posição, dessa forma, o for só será executado 1 vez e o while será executado m vezes, assim o número de comparações executadas será igual a O(m). Exemplo: T = “CIÊNCIA DA COMPUTAÇÃO” ; P = “CIÊ”. Pior caso: O pior caso será quando apenas o último caractere se diferenciar, assim a comparação vai ser realizada até a penúltima posição. Assim o while será executado m vezes para da iteração do for, que por sua vez será executado n-m+1 vezes. Dessa forma, as comparações no pior caso serão da seguinte ordem: (n-m+1)*(m). Assim a complexidade no pior caso é dada por: O(m*n). Exemplo: T : “ESTRUTURA DE DADOS” ; P: “ESTRUTURA DE DAD”.

```c
#define MaxTexto 100
#define MaxPadrao 10

/* Pesquisa o padrao P[1..m] no texto  T[1..n] */
void ForcaBruta(   char T[MaxTexto], int n,
                   char T[MaxPadrao], int m)
{
    int i,j,k;
    for( i = 0 ; i < n - m + 1 ; i++ )
    {
         k = i;
         j = 0;
         while ( ( j <= m ) && ( T[k] == P[j] ) )
         {
              j = j + 1;
              k = k + 1;
         }
         if (j > m)
         {
              printf("Casamento na posicao %d",i);
              break;
         }
    }
}
```

<u>Questão 5</u>

Considere que você tenha um problema para resolver e duas opções de algoritmos. O primeiro algoritmo é quadrático tanto no pior caso quanto no melhor caso. Já o segundo algoritmo, é linear no melhor caso e cúbico no pior caso. Considerando que o melhor caso ocorre 90% das vezes que você executa o programa enquanto o pior caso ocorre apenas 10% das vezes, qual algoritmo você escolheria? Justifique a sua resposta em função do tamanho da entrada.

O melhor algoritmo a ser utilizado é o quadrático, já que para entregas com n a apartir de 1000, a vantagem do algoritmo cúbico se perde, conforme descrito abaixo:

i) 0,9 n^2 + 0,1n^2 = n^2
ii) 0,9n + 0,1n^3

Para n = 1:
i) 1^2 = 1
ii) 0,9 + 0,1 = 1

Para n = 2:
i) 2^2 = 4
ii) 0,9 * 2 + 0,1 * 8 = 1,8 + 0,8 = 2,6

Para n = 3:
i) 3^2 = 9
ii) 0,9 * 3 + 0,1 * 27 = 2,7 + 2,7 = 5,4

Para n = 1000:
i) 1000^2 = 1000000
ii) 0,9 * 1000 + 0,1 * 1000000000 = 900 + 100000000 = 100000900

<u>Questão 6</u>

Perdido em uma terra muito distante, você se encontra em frente a um muro de comprimento infinito para os dois lados (esquerda e direita). Em meio a uma escuridão total, você carrega um lampião que lhe possibilita ver apenas a porção do muro que se encontra exatamente à sua frente (o campo de visão que o lampião lhe proporciona equivale exatamente ao tamanho de um passo seu). Existe uma porta no muro que você deseja atravessar. Supondo que a mesma esteja a n passos de sua posição inicial (não se sabe se à direita ou à esquerda), elabore um algoritmo para caminhar ao longo do muro que encontre a porta em O(n) passos. Considere que n é um valor desconhecido (informação pertencente à instância). Considere que a ação composta por dar um passo e verificar a posição do muro correspondente custa O(1).

O algoritmo resolve o problema em O(1)

```
i = 1
Enquanto não encontrar a porta
      Ande até a posição i.
      Se encontrar a porta no caminho: pare
      Ande até a posição -2*i.
      Se encontrar a porta: pare
      i++
Fim Enquanto
```

Assim o algoritmo vai executar um passo de cada vez, sendo que a verificação é feita em O(1), a complexidade é dada pelo loop interno, executado no máximo 2n vezes até encontrar a porta. Desprezando a constante, a complexidade desse algoritmo é dada por O(n).