UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO
CENTRO DE CIÊNCIAS AGRÁRIAS – CCA/ UFES
Departamento de Engenharia Rural

Lista de exercícios 5

Disciplina: Computabilidade e Complexidade
Professora: Juliana Pinheiro Campos
Data: 29/11/2011
Assunto: Complexidade de algoritmos

1) Uma das possíveis formas de se descrever a complexidade de um algoritmo é a chamada Notação-Big-Oh, que é definida da seguinte forma: T(n) = O(f(n)) se existem constantes c e n0 tais que T(n) é menor ou igual c.f(n) quando n maior ou igual n0. Explique o que você entendeu por esta definição.

2) No geral, quais tipos de problemas possuem complexidade da ordem de n?

3) Quais problemas possuem geralmente complexidade da ordem de log n?

4) Resolva a operação abaixo usando as propriedades da notação O:

$$[\log n + k + O(1/n)] * [n + O(\sqrt{n})]$$

5) Para as funções f(n) e g(n) dadas abaixo, mostre que f(n) = O(g(n)):

   a) $f(n) = 5n$ e $g(n) = n^2$

   b) $f(n) = 6n^2$ e $g(n) = (n + 1)^2$

   c) $f(n) = (n + 1)^2$ e $g(n) = n^2$

6) Sejam A e B, 2 algoritmos cujas complexidades são respectivamente determinadas pelas funções f(n) e g(n) dadas abaixo. Para cada caso, determine os valores inteiros e positivos de n para os quais o algoritmo A leva "menos tempo" para executar do que o algoritmo B.

   a) $f(n) = n^2 + 1$ e $g(n) = 5n – 5$

   b) $f(n) = n^4 + n^2 + 1$ e $g(n) = 4n^2 + 5$

7) Responda se cada uma das afirmações abaixo é verdadeira ou falsa. Justifique.

   a) Todo algoritmo que é $\Theta(n\log n)$ é $O(n^2)$.

b) Se existe um exemplo que faz com que um algoritmo realize da ordem de n operações, então este algoritmo é $\Theta(n)$.

c) Um algoritmo A possui função de complexidade $f(n) = 2^n$ e um algoritmo B possui função de complexidade $g(n) = n^5$. Podemos dizer que o custo de A sempre será maior que o custo de B.

8) Resolva a seguintes relações de recorrência:

a) $T(n) = T(n-1) + 1$
$T(1) = 1$

b) $T(n) = 2T(n-1) + 1$
$T(1) = 1$

9) Determine a complexidade dos algoritmos abaixo considerando o nº de vezes que as operações assinaladas com → são executadas.

a)
```
void algoritmo1( int n){
        int s, i;
        s = 0;
        for(i = 3; i < n; i ++){
→               if(A[i] = A[i + 1])
                        s = s + 1;
        }
}
```

b)
```
void algoritmo2(int n){
        if(n > 0){
                for(int i = 1; i <= n; i++)
→                       printf("%d", i);
                algoritmo2(n/2);
        }
}
```

Como esse algoritmo é recursivo, o nº de vezes q o printf será executado é dado pela seguinte relação de recorrência:

$T(1) = 1$
$T(n) = n + T(n/2)$

c) OBS: Nesse algoritmo deve ser levado em consideração o nº de comparações envolvendo os elementos do vetor que aparece como parte da linha assinalada.

```
void algoritmo3(int* A, int n){
        int i, j, aux;
        for( j = 2; j <= n; j++){
                aux = A[j];
                i = j - 1;
          →     while (i  > 0 && A[i] > aux){
                        A[i  + 1] = A[i];
                        i = i -1;
                }
                A[i + 1] = aux;
        }
}
```

d)
```
void algoritmo4(int n){
        int n, i;
        if(n > 1){
                for(i = 0; i <=1; i++){
                        A[n] = i;
                        algoritmo4(n-1);
                }
        }
        else
     →          "imprima o vetor A"

}
```

10) Implemente um algoritmo que calcule a média dos elementos de um vetor e faça a análise de complexidade desse algoritmo em função do número de operações realizadas.

11) O algoritmo abaixo cria dinamicamente uma matriz que corresponde a soma das matrizes mat1 e mat2 com n linhas e n colunas. Faça a análise de complexidade desse algoritmo em função do número de operações realizadas.

```
// A matriz soma é a matriz obtida adicionando-se os elementos correspondentes de A e B.
int** adicao (int n, int** mat1, int** mat2){

    int i, j;
    int** soma;

    soma = malloc (n * sizeof (int*));
    for(i = 0; i < n; i++){
       soma[i] = malloc(n * sizeof (int));
    }
```

```
    for(i = 0; i < n; i++){
      for (j = 0; j < n; j++){
         soma[i][j] = mat1[i][j] + mat2[i][j];
      }
     }
   return soma;
}
```