Universidade Federal de Ouro Preto
Instituto de Ciências Exatas e Biológicas
Programa de Pós-Graduação em Ciência da Computação
Projeto e Análise de Algoritmos - 1ºsemestre de 2010
Professor: David Menotti

1ªLista de Exercícios

Valor: 1,0 pontos (10% da nota total) Data de Entrega: 18/04/2010

1. Como podemos modificar quase que qualquer algoritmo para ter um bom tempo de execução para o melhor caso?

2. Observe que o laço **while** das linhas 5-7 do procedimento INSERTION-SORT (apresentado abaixo) usa uma pesquisa linear para percorrer de trás para frente o vetor ordenado $A[1..j-1]$.

   (a) Pode-se usar a pesquisa binária para melhorar o tempo de execução no pior caso do INSERTION-SORT para $\Theta(n \log n)$.

   (b) Justifique sua resposta?

```
INSERTION-SORT(A, n)
1  for j = 2 to length(A)
2      key = A[j]
3      // Insert A[j] into the sorted sequence A[1 . . j − 1].
4      i = j − 1
5      while i > 0 and A[i] > key
6          A[i + 1] = A[i]
7          i = i − 1
8      A[i + 1] = key
```

3. 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.

4. * Sejam $f(n)$ e $g(n)$ funções assintoticamente não-negativas. Usando a definição básica da notação Θ, prove que

$$max(f(n), g(n)) = \Theta(f(n) + g(n)). \quad (1)$$

5. * Mostre que para quaisquer constantes reais a e b, tal que $b > 0$,

$$(n + a)^b = \Theta(n^b). \tag{2}$$

6. Mostre se:

   (a) $2^{n+1} = O(2^n)$;

   (b) $2^{2n} = O(2^n)$.

7. Pode-se estender a notação assintótica para o caso de dois parâmetros n e m que vão para inifinito independentemente com taxas diferentes. Para uma dada função $g(n, m)$, denota-se $O(g(n, m))$ o conjunto de funções

$$O(g(n,m)) = \left\{ \begin{array}{ll} f(n,m): & \text{existem constantes positivas } c,\ n_0 \text{ e } m_0 \\ & \text{tais que } 0 \leq f(n,m) \leq cg(n,m) \\ & \text{para todo } n \geq n_o em \geq m_o \end{array} \right\}.$$

   Apresente as definições correspondes para:

   (a) $\Omega(g(n, m))$;

   (b) $\Theta(g(n, m))$.

8. * Mostre se:

   (a) A função $\lceil \log n \rceil !$ é limitada polinomialmente;

   (b) A função $\lceil \log \log n \rceil !$ é limitada polinomialmente.

9. * (*Crescimento assintótico relativo*) Indique para cada par de expressões (A, B) na tabela abaixo, se A é O, o, Ω, ω, ou $Theta$ de B. Assuma que $k \geq 1$, $\epsilon > 0$ e $c > 1$ são constantes. Sua resposta deve ser na forma SIM ou NÃO acompanhada da justificativa.

	A	B	O	o	Ω	ω	Θ
(a)	$\log^k n$	n^ϵ					
(b)	n^k	c^n					
(c)	$\sqrt{n}$	$n^{\sin n}$					
(d)	2^n	$2^{n/2}$					
(e)	$n^{\log m}$	$m^{\log n}$					
(f)	$\log(n!)$	$\log(n^n)$					

10. * (*Ordenação por taxa de crescimento assintótico*)

    (a) Classifique as funções abaixo pela ordem de crescimento, ou seja, ache uma permutação das funções $g_1, g_2, ..., g_{30}$ que satisfaça a relação $g_1 = \Omega(g_2), g_2 = \Omega(g_3), ..., g_{29} = \Omega(g_{30})$. Particione a lista em classes de equivalência tais que $f(n)$ e $g(n)$ estão na mesma classe se, e somente se, $f(n) = \Theta(g(n))$.

    A notação $\log^* n$ representa a função logarítmica "iterada" conforme definido em [3, Iteração funcional, Capítulo 3].

$\log(\log^* n)$	$2^{\log^* n}$	$(\sqrt{2})^{\log n}$	n^2	$n!$	$(\log n)!$
$(3/2)^n$	n^3	$\log^2 n$	$\log(n!)$	2^{2^n}	$n^{1/\log n}$
$\ln\ln n$	$\log^* n$	$n.2^n$	$n^{\log\log n}$	$\log n$	1
$2^{\log n}$	$(\log n)^{\log n}$	e^n	$4^{\log n}$	$(n+1)!$	$\sqrt{\log n}$
$\log^*(\log n)$	$2^{\sqrt{2\log n}}$	n	2^n	$n\log n$	$2^{2^{n+1}}$

(b) Apresente um exemplo de uma função $f(n)$ não-negativa tal que para todas as funções $g_i(n)$ da letra anterior, $f(n)$ não é $O(g_i(n))$ nem $\Omega(g_i(n))$.

11. Use uma árvore de recursão para apresentar uma solução assintoticamente firme para a recorrência

$$T(n) = T(\alpha n) + T((1-\alpha)n) + cn, \tag{3}$$

onde α é uma constante na faixa $0 < \alpha < 1$ e $c > 0$ é também uma constante.

12. Use o Teorema Mestre, se for possível, para apresesentar limites assintóticos firmes para as seguintes recorrências, e quando não for possível utilize outro método de resolução de equações de recorrência.

(a) $T(n) = 4T(n/2) + n$;
(b) $T(n) = 4T(n/2) + n^2$;
(c) $T(n) = 4T(n/2) + n^3$.

13. A recorrência $T(n) = 7T(n/2) + n^2$ descreve o tempo de execução de um algoritmo A. Um algoritmo alternativo A' tem um tmepo de execução $T'(n) = aT'(n/4) + n^2$. Qual é o maior número inteiro a que faz com que A' seja assintoticamente mais rápido que A?

14. O Teorema Mestre pode ser aplicado à recorrência $T(n) = 4T(n/2) + n^2\log n$? Justifique sua resposta. Apresente um limite superior assintótico para essa recorrência.

15. (*Exemplos de recorrência*) Apresente os limites assintóticos superior e inferior para $T(n)$ em cada uma das recorrências abaixo. Assuma que $T(n)$ é constante para $n \leq 2$. Tente determinar cada limite tão firme quanto possível e justifique a sua resposta.

(a) $T(n) = 2T(n/2) + n^3$;
(b) $T(n) = T(9n/10) + n$;
(c) $T(n) = 16T(n/4) + n^2$;
(d) $T(n) = 7T(n/3) + n^2$;
(e) $T(n) = 7T(n/2) + n^2$;
(f) $T(n) = 2T(n/4) + \sqrt{n}$;

(g) $T(n) = T(n-1) + n$;

(h) $T(n) = T(\sqrt{n}) + 1$;

16. (*Algoritmos não-recursivos*) Determine a função de complexidade (no pior e melhor caso e no caso médio), das funções implementadas em C, apresentadas abaixo, fazendo as considerações pertinentes.

(a)

```
int Max(int* A, int n)
{
  int i, Temp;

  Temp = A[0];
  for (i = 1; i < n; i++)
    if (Temp < A[i])
      Temp = A[i];
  return Temp;
}
```

Programa 1: Max

(b)

```
void MaxMin1(int* A, int n, int* pMax, int* pMin)
{
  int i;

  *pMax = A[0];
  *pMin = A[0];
  for (i = 1; i < n; i++)
  {
    if (A[i] > *pMax) *pMax = A[i];
    if (A[i] < *pMin) *pMin = A[i];
  }
}
```

Programa 2: MaxMin1

(c)

```
void MaxMin2(int* A, int n, int* pMax, int* pMin)
{
  *pMax = A[0];
  *pMin = A[0];
  for (i = 1; i < n; i++)
  {
    if (A[i] > *pMax) *pMax = A[i];
    else if (A[i] < *pMin) *pMin = A[i];
  }
}
```

Programa 3: MaxMin2

(d)

```
void MaxMin3(int* A, int n, int* pMax, int* pMin)
{
  int i, FimDoAnel;

  if ((n % 2) > 0)
  {
    A[n] = A[n - 1];
```

```
    FimDoAnel = n;
  }
  else FimDoAnel = n - 1;

  if (A[0] > A[1]) { *pMax = A[0]; *pMin = A[1]; }
  else             { *pMax = A[1]; *pMin = A[0]; }

  i = 3;
  while (i <= FimDoAnel)
  {
    if (A[i - 1] > A[i])
    {
      if (A[i - 1] > *pMax) *pMax = A[i - 1];
      if (A[i]     < *pMin) *pMin = A[i];
    }
    else
    {
      if (A[i - 1] < *pMin) *pMin = A[i - 1];
      if (A[i]     > *pMax) *pMax = A[i];
    }
    i += 2;
  }
}
```

Programa 4: MaxMin3

(e)

```
void BubbleSort(int* A, int n)
{
  int i,j;
  int aux;

  for( j = 0; j < n; j++ )
  {
    for( i = 0; i < n - 1; i++ )
    {
      if( A[i] > A[i+1] )
      {
        aux   = A[i];
        A[i]  = A[i+1];
        A[i+1] = aux;
      }
    }
  }
}
```

Programa 5: BubbleSort

(f)

```
void BubbleSort2(int* A, int n)
{
  int i,troca;
  int aux;

  do
  {
    troca = 0;
    for ( i = 0 ; i < n-1 ; i++ )
    {
      if ( A[i] > A[i+1] )
```

```
        {
          aux     = A[i];
          A[i]    = A[i+1];
          A[i+1] = aux;
          troca = 1;
        }
      }
    } while (troca);
}
```

Programa 6: BubbleSort2

(g)

```
void SelectionSort(int* A,int n)
{
  int i,j,min;
  int aux;

  for ( i = 0 ; i < n - 1 ; i++ )
  {
    min = i;
    for ( j = i + 1 ; j < n ; j++ )
      if ( A[j] < A[min] )
        min = j;

    aux = A[min];
    A[min] = A[i];
    A[i] = aux;
  }
}
```

Programa 7: SelectionSort

(h)

```
void InsertionSort(int* A, int n )
{
  int j;
  for (int i = 1; i < n; i++)
  {
    aux  = A[i];
    j = i - 1;

    while ( ( j >= 0 ) && ( aux < v[j] ) )
    {
      v[j + 1] = v[j];
      j--;
    }
    v[j + 1] = aux;
  }
}
```

Programa 8: InsertionSort

(i)

```
void FazAlgo(int n )
{
  int i,j,k;

  x = 0;
  for( i = 1 ; i <= n - 1 ; i++ )
```

```
  {
    for( j = i + 1 ; j <= n ; j++ )
    {
      for( k = 1 ; k <= j ; k++ )
        x = x + 1;
    }
}
```

Programa 9: FazAlgo

(j)

```
void FazAlgo2(int n)
{
  int i,j,k,x;

  x = 0;
  for( i = 1 ; i <= n ; i ++)
    for( j = i + 1 ; j <= n - 1 ; j++ )
      for( k = 1 ; k <= j ; k++ )
        x = x + 1;
}
```

Programa 10: FazAlgo2

17. (*Algoritmos recursivos*) Determine a função de complexidade, das funções recursivas apresentadas abaixo, fazendo as considerações que considerar pertinente.

(a)

```
void Pesquisa1(int* A,int n)
{
  if (n > 1)
  {
    Inspecione n*n*n elementos; // custo n^3
    Pesquisa (A,n/3);
  }
}
```

Programa 11: Pesquisa1

(b)

```
void Pesquisa2(int* A, int n)
{
  if ( n <= 1 )
    return;
  else
  {
    obtenha o maior elemento dentre os n elementos;
    /* de alguma forma isto permite descartar 2/5 dos*/
    /* elementos e fazer uma chamada recursiva no resto*/
    Pesquisa(A,3*n/5);
  }
}
```

Programa 12: Pesquisa2

(c)

```
void Pesquisa3(int* A, int n);
{
  if ( n <= 1 )
```

```
    return;
  else
  {
    ordena os n elementos;
    /* de alguma forma isto permite descartar 1/3 dos */
    /* elementos e fazer uma chamada recursiva no resto*/
    Pesquisa (2*n/3);
  }
}
```

Programa 13: Pesquisa3

(d)

```
int Fibonacci(int n)
{
        if (n == 0) return 0;
  else if (n == 1) return 1;
  else             return Fibonacci(n-1) + Fibonacci(n-2);
}
```

Programa 14: Fibonacci

(e)

```
void Proc( int n )
{
  if ( n == 0)
    return 1;
  else
    return Proc(n-1) + Proc(n-1);
}
```

Programa 15: Proc

(f)

```
/* n é uma potencia de 2 */
void Sort (int* A,int i, int j)
{
  if ( i < j )
  {
    m = (i + j - 1)/2;

    Sort( A , i , m );        /* custo = T(N/2) */
    Sort( A , m + 1 , j );    /* custo = T(N/2) */
    Merge( A , i , m , j );   /* custo = N-1
    /* comparacoes no pior caso */
    /* Merge intercala A[i..m] e A[m+1..j] em A[i..j] */
  }
}
```

Programa 16: Sort

(g)

```
/* n uma potencia de 3 */
void Sort2 (int* A, int i, int j)
{
  if ( i < j )
  {
    m = ( (j - i) + 1 )/3;
    Sort2( A , i , i + m - 1 );
    Sort2( A , i + m , i + 2*m - 1 );
    Sort2( A , i + 2*m , j );
```

```
10        Merge( A , i , i + m , i + 2*m , j );
          /* Merge intercala A[i..(i+m-1)], A[(i+m)..(i+2m-1) e A[i
             +2m..j] em A[i..j] a um custo ( (5n/3) -2 ) */
      }
  }
```

Programa 17: Sort2

18. Considere a função em C abaixo.

    (a) O que ela faz?

    (b) Qual é a função de complexidade do número de comparações no melhor e no pior caso?

    (c) Que configuração do vetor de entrada A leva a essas duas situações?

```
void FazAlgo3(int* A, int n)
{
  int i, j;
  int aux;
5
  for(i = 2 ; i < n ; i++)
  {
    aux = A[i];
    j = i - 1;
10  A[0] = aux;
    while( aux < A[j] )
    {
      A[j + 1] = A[j];
      j = j - 1;
15  }
    A[j + 1] = aux;
  }
}
```

Programa 18: FazAlgo3

O que deve ser entregue

- Solução dos exercícios propostos.

Como deve ser apresentada a solução dos exercícios

1. Em LaTeX: Caso seja utilizado algum editor eletrônico de texto, a solução da lista de exercícios deve ser elaborada obrigatoriamente em LaTeX. Nenhum outro formato será aceito. Veja modelo em latex: `http://www.decom.ufop.br/menotti/paa101/tps/modelo.zip`

2. Caso os exercícios sejam resolvidos à mão, em folha de papel almaço pautado ou sulfite, usando exclusivamente caneta azul ou preta, e não-ambas, estas folhas devem ser escaneados e encapsuladas em um único arquivo PDF;

3. Formato final: mandatoriamente em PDF (`http://www.pdf995.com/`).

Como deve ser feita a entrega

A entrega DEVE ser feita via Moodle (`www.decom.ufop.br/moodle`) na forma de um único arquivo zipado, contendo a documentação da solução das questões e arquivos que foram utilizados para elaborá-la. Também deve ser entregue a documentação impressa na próxima aula teórica após a data de entrega.

Comentários Gerais

- A entrega dos exercícios marcados com * (asterico) é opcional e vale pontos extra.
- A maioria desses exercícios foram extraídos de [2, 1, 3, 6, 7, 4, 5] e as vezes de alguam forma alterados;
- Comece a resolver esta liga logo, enquanto o problema está fresco na memória e o prazo para terminá-lo está tão longe quanto jamais poderá estar;
- A solução da lista é individual (grupo de UM aluno);
- Soluções copiadas (e FONTE) terão nota zero;
- Soluções entregues em atraso serão aceitas, todavia a nota atribuída a solução será zero;
- Evite discussões inócuas com o professor em tentar postergar a data de entrega.

Referências

[1] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein. *Algoritmos: Teoria e Prática*. Editora Campus, 2001. Tradução da 2a. edição americana.

[2] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press/McGraw-Hill, 2nd edition, 2001.

[3] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press/McGraw-Hill, 3rd edition, 2009.

[4] A.M. Tenenbaum, Y. Langsam, and M.J. Augenstein. *Data Strcutures Using C*. Prentice-Hall International Editions, 1995.

[5] A.M. Tenenbaum, Y. Langsam, and M.J. Augenstein. *Estruturas de Dados Usando C*. Makron Books/Pearson Education, 1995.

[6] N. Ziviani. *Projeto de Algoritmos: com implementações em Pascal e C*. Cengage Learning (Thomson / Pioneira), São Paulo, 2nd edition, 2004.

[7] N. Ziviani. *Projeto de Algoritmos com implementações em Java e C++*. Cengage Learning (Thomson / Pioneira), São Paulo, 1st edition, 2006.