MINISTÉRIO DA EDUCAÇÃO E DO DESPORTO
Universidade Federal de Ouro Preto - UFOP
Instituto de Ciências Exatas e Biológicas - ICEB
Programa de Pós-Graduação em Ciência da Computação
Disciplina: Projeto e Análise de Algoritmos
Professor: David Menotti Gomes

1ªLista de Exercícios

Valor: 1,0 pontos (10% da nota total) Data de Entrega: 18/04/2011

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 $\Theta(n \lg n)$.

   (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. O algoritmo de Ordenação por Inserção pode ser expresso como um procedimento recursivo da seguinte forma: para ordenar $A[1..n]$, ordena-se recursivamente $A[1..n-1]$ e então insere-se $A[n]$ no vetor ordenado $A[1..n-1]$.

   (a) Apresente um algoritmo em pseudo-código que descreva o algoritmo acima;

(b) Escreva uma equação de recorrência para o tempo de execução dessa versão recursiva;

(c) Resolva essa equação de recorrência, ou seja apresente a solução em forma fechada (função de complexidade)

(d) Determine a ordem de complexidade desse algoritmo por meio do Teorema Mestre.

4. Descreva um algoritmo e implemente uma função em C/C++ com complexidade de tempo $\Theta(n \lg n)$ que, dado um conjunto S de n inteiros e um outro inteiro x, determina se existe ou não dois elementos de S cuja soma é exatamente x.

5. (*Limite Inferior*) Para cada um dos problemas elencados abaixo: (1) Apresente um algoritmo ótimo para resolver esse problema; (2) Prove que o algoritmo apresentado é ótimo.

   (a) Considere um arranjo A com n elementos não ordenados. O problema é encontrar o maior valor dentre estes n elementos.

   (b) Considere um arranjo A com n elementos não ordenados. O problema é encontrar o maior e o menor valores dentres estes n elementos.

   (c) Considere um arranjo A com n elementos não ordenados. O problema é ordenar estes n elementos usando somente comparações entre chaves.

   (d) São dados $2n$ elementos distintos distribuídos em dois arranjos A e B, cada um com n elementos ordenados, tal que $A[0] < A[1] < ... < A[n-2] < A[n-1]$ e $B[0] < B[1] < ... < B[n-2] < B[n-1]$. O problema é encontrar o n-ésimo maior valor dentre estes $2n$ elementos.

6. 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.

7. (*Ordenação por Inserção em pequenos vetores no Mergesort*) Sabe-se que o algoritmo Mergesort executa no pior caso em tempo $\Theta(n \lg n)$ e o algoritmo de ordenação por Inserção no pior caso em tempo $\Theta(n^2)$. No entanto, os fatores constantes do Inserção o tornam mais rápido que o Mergesort para um pequeno valor de n. Assim, faz sentido usar o algoritmo de Ordenação por Inserção quando os sub-problemas tornam-se suficientemente pequenos. Considere a seguinte modificação no Mergesort: n/k sub-listas de comprimento k são ordenadas usando o algoritmo de ordenação por Inserção e então combinadas/intercaladas (*merged*) usando o mecanismo do Mergesort, sendo k o valor a ser determinado.

   (a) Mostre que se as n/k sub-listas, cada uma de comprimento k, podem ser ordenadas pelo algoritmo de Ordenação por Inserção no pior caso em tempo $\Theta(nk)$

(b) Mostre que as sub-listas podem ser combinadas (*merged*) no pior caso em tempo $\Theta(n \lg n/k)$

(c) Dado que o algoritmo modificado executa no pior caso em tempo $\Theta(nk + n \lg n/k)$, qual é o maior valor assintótico (usando notação Θ de k como uma função de n para o qual o algoritmo modificado tem o mesmo tempo de execução assintótico do Mergesort padrão?

(d) Na prática, como o valor de k seria escolhido?

8. (*Inversões*) Seja $A[1..n]$ um vetor com n número distintos. Se $i < j$ e $A[i] > A[j]$, então o par (i, j) é chamado de uma inversão de A.

   (a) Liste as cinco inversões do vetor $< 2, 3, 8, 6, 1 >$;

   (b) Que vetor com elementos do conjunto $\{1, 2, ..., n\}$ tem o maior número de inversões? Quantas inversões existem?

   (c) Qual é a relação entre o tempo de execução do algoritmo de ordenação por Inserção e o número de inversões do vetor de entrada? Justifique sua resposta.

   (d) Apresente um algorimto que determina o número de inversões em qualquer permutação de n elementos no pior caso em tempo $\Theta(n \lg n)$. (Dica: modifique o Mergesort).

9. Apresente um algoritmo que calcule a^n, onde n é inteiro, em $\Theta(\lg n)$ passos.

10. Dadas n variáveis booleanas, faça um algoritmo que gere todas as combinações possíveis. Por exemplo, para três variáveis deverá ser gerado:

$$000, 001, 010, 011, 100, 101, 110, 111.$$

11. Mostre se:

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

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

12. Diga se é falso ou verdadeiro e justifique?

   (a) Sempre que $f = O(h)$ e $g = O(h)$, $f = O(g)$.

   (b) Sempre que $f = O(g)$ e $g = O(h)$, $f = O(h)$.

   (c) As funções $n \lg n$ e $n \lg(n^2)$ possuem a mesma ordem de complexidade.

   (d) $\lg(n^c) = \Theta(\lg(n))$, $c > 0$.

   (e) $2^{100} = O(1)$.

   (f) $2^{n-1} = O(2^n)$.

   (g) $2^n = \omega(2^n - 1)$.

13. Mostre se:

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

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

14. (*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)	$\lg^k n$	n^ϵ					
(b)	n^k	c^n					
(c)	$\sqrt{n}$	$n^{\sin n}$					
(d)	2^n	$2^{n/2}$					
(e)	$n^{\lg m}$	$m^{\lg n}$					
(f)	$\lg(n!)$	$\lg(n^n)$					

15. (*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, $f(n) = \Theta(g(n))$.

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

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

    (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 $\Omega(g_i(n))$.

16. Use árvore de recursão para apresentar solução assintoticamente firme para as recorrências abaixo

$$T(n) = T(\alpha n) + T((1-\alpha)n) + cn, T(n) = 4T(n/5) + cn^2, \quad (1)$$

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

17. Para cada uma das funções encontre uma $g(n)$ simples tal que $f = \Theta(g)$:

    (a) $f(n) = 1000(2^n) + 4^n$

    (b) $f(n) = n + n\lg n + \sqrt{n}$

(c) $f(n) = \lg n^{20} + (\lg n)^{10}$

(d) $f(n) = (0.99)^n + n^{100}$

18. Para cada um dos pares de função abaixo, defina uma constante c para a qual $f(n) = O(g(n))$:

(a) $f(n) = n^2 + 2n + 1$, $g(n) = n^2/2$

(b) $f(n) = n\sqrt{n} + n^2$, $g(n) = n^2$

19. Encontre duas funções que satisfaçam o relacionamento ou argumente porque não podem ser encontradas:

(a) $f(n) = o(g(n))$ e $f(n) \neq \Theta(g(n))$

(b) $f(n) = \Theta(g(n))$ e $f(n) \neq o(g(n))$

(c) $f(n) = \Theta(g(n))$ e $f(n) \neq O(g(n))$

(d) $f(n) = \Omega(g(n))$ e $f(n) \neq O(g(n))$

20. Para cada um dos pares a seguir, diga se f é $O(g)$, $\Theta(g)$ ou $\Omega(g)$ e explique brevemente porque.

(a)	$f(n) = \lg n^2$;	$g(n) = \lg n + 5$.
(b)	$f(n) = \sqrt{n}$;	$g(n) = \lg n^2 + 5$.
(c)	$f(n) = \lg^2 n$;	$g(n) = \lg n$.
(d)	$f(n) = n$;	$g(n) = \lg^2 n$.
(e)	$f(n) = n \lg n + n$;	$g(n) = \lg n + 5$.
(f)	$f(n) = 10$;	$g(n) = \log 10$.
(g)	$f(n) = 2^n$;	$g(n) = 10n^2$.

21. 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) $T(n) = T(n/2) + 1$;

(b) $T(n) = 4T(n/2) + n$;

(c) $T(n) = 4T(n/2) + n^2$;

(d) $T(n) = 4T(n/2) + n^3$.

22. Use o método da substituição para apresentar limites assintóticos para as recorrências da questão anterior.

23. Use indução matemática para provar limites superior para as seguintes recorrências:

(a) $T(n) = 2$, se $n = 2$ e $T(n) = 2T(n/2) + n$, se $n = 2^k$, com $k > 1$

(b) $T(n) = 4$, se $n = 1$ e $T(n) = 4T(n/2) + n^2$, se $n = 2^k$, com $k > 1$

(c) $T(n) = 4$, se $n = 1$ e $T(n) = 4T(n/2) + n^2 \lg n$ se $n = 2^k$, com $k > 1$

24. A recorrência $T(n) = 7T(n/2) + n^2$ descreve o tempo de execução de um algoritmo A. Um algoritmo alternativo A' tem um tempo 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?

25. Um colega diz que descobriu um novo algoritmo de pesquisa super-eficiente com implementação recursiva de complexidade $T(n) = 3T(n/3) + n$ e com custo constante para tamanho 1. Você recomendaria que ele implementasse o algoritmo? Porque?

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

27. (*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) $T(n) = T(\sqrt{n}) + 1$;

28. (*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)

```c
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 1: BubbleSort

(b)

```
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 2: BubbleSort2

(c)

```
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 3: SelectionSort

(d)

```
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] ) )
    {
      A[j + 1] = A[j];
      j--;
    }
    A[j + 1] = aux;
```

```
  }
}
```

Programa 4: InsertionSort

(e)

```
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 5: FazAlgo

(f)

```
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 6: FazAlgo2

29. (*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 7: 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 8: 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 9: 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 10: Fibonacci

(e)

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

Programa 11: 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 12: 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 );
    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 13: Sort2

(h)

```
void Misterio2(Item* A, int m, int n,int *i, int *j)
{
  Item x, aux;
  *i = m; *j = n;
  x = A[(*i + *j)/2];
  do
  {
    while (x.Chave > A[*i].Chave) (*i)++;
    while (x.Chave < A[*j].Chave) (*j)--;
    if (*i <= *j)
    {
      aux = A[*i];
      A[*i] = A[*j];
      A[*j] = aux;
      (*i)++; (*j)--;
    }
  } while (*i <= *j);
}

void Misterio1(Item *A,int m, int n)
{
  int i,j;
  Misterio2(A, m, n, &i, &j);
  if (m < j) Misterio1(A, m, j);
  if (i < n) Misterio1(A, i, n);
}
```

Programa 14: Misterio

30. 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;

  for(i = 2 ; i < n ; i++)
  {
    aux = A[i];
    j = i - 1;
    A[0] = aux;
    while( aux < A[j] )
    {
      A[j + 1] = A[j];
      j = j - 1;
    }
    A[j + 1] = aux;
  }
}
```

Programa 15: 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 maioria desses exercícios foram extraídos de [2, 1, 3, 7, 8, 5, 6, 4] e as vezes de alguma 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] R. Sedgewick. *Algorithms*. Addison-Wesley, 2 edition, 1988. ISBN-10: 0201066734.

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

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

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

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