Lista 1 - PMR2300

Fabio G. Cozman

3 de abril de 2013

1. Qual String é impressa pelo programa:

```
public class What {
   public static void f(int x) {
      x = 2;
   }
   public static void main(String args[]) {
      int x = 0;
      f(x);
      System.out.println("Hey: " + x);
   }
}
```

2. Considere o seguinte exemplo de alteração de um arranjo passado por referência para um método chamado `doIt`:

```
public static void doIt(int h[]) {
      int k[] = new int[2 * h.length];
      for (int i=0; i<h.length; i++) {
            k[i] = h[i];
            k[i+h.length] = h[i];
      }
      h = k;
      k[0] = 2;
}
```

   Esse código contém um erro. Reescreva-o de forma a garantir que o arranjo `h` na chamada da função é modificado (seu tamanho e conteúdo são duplicados, e seu elemento 0 recebe o valor 2).

3. Escreva um pequeno método em Java que receba dois arranjos a e b de tamanho igual, contendo valores do tipo double, e retorne o produto escalar de a e b.

4. Elabore um programa recursivo que calcule o Mínimo Divisor Comum de dois inteiros usando o seguinte algoritmo, conhecido como algoritmo de Euclides: Divida o número maior pelo menor e pegue o resto; então divida o divisor pelo resto e assim sucessivamente até que o resto seja zero — o quociente da última divisão é o divisor comum. Por exemplo, tome 928 e 100: dividimos 928 por 100, obtemos resto 28; dividimos 100 por 28, obtemos resto 16; dividimos 28 por 16, obtemos resto 12; dividimos 16 por 12, obtemos resto 4; dividimos 12 por 4 e obtemos resto zero — o resultado é 4.

5. Escreva um algoritmo recursivo que transforme um número inteiro decimal em um número binário.

6. Considere a função $f(n)$ definida recursivamente como segue: $f(0) = 0$; $f(n) = f(n/2)$ se n é par; $f(n) = 1 + f(n-1)$ se n é ímpar. Codifique essa função em Java e obtenha o valor de $f(100)$.

7. Considere o seguinte algoritmo:

```
int exercicio1(int x) {
    if (x < 5)
        return(3*x);
    else
        return(2 * exercicio1(x-5) + 7);
}
```

   Qual é o resultado de `exercicio1(4)` e `exercicio1(20)`? Desenhe a árvore de chamadas recursivas para cada caso.

8. Considere o seguinte algoritmo:

```
int exercicio2(int x, int y) {
    if (x > y)
        return(-1);
    else {
        if (x == y)
            return(1);
        else
            return(x * exercicio2(x+1, y)
                     + exercicio2(2*x,y))
    }
}
```

   Qual é o resultado de chamadas `exercicio2(10,4)`, `exercicio2(4,3)`, `exercicio2(4,7)`? Desenhe a árvore de chamadas recursivas para cada caso.

9. Dado o fragmento de código abaixo, desenhe a árvore de recursão para $f(1, 10)$ e obtenha o valor de retorno.

```
double f(double x, double y) {
    if (x >= y)
        return((x+y)/2);
    else
        return(f(f(x+2,y-1),f(x+1,y-2)));
}
```

10. Coloque em ordem crescente de complexidade: $\mathcal{O}(2^n)$, $\mathcal{O}(n!)$, $\mathcal{O}(n^5)$, $\mathcal{O}(n \log n)$. Qual é a complexidade em notação BigOh de um programa que utiliza $3n^4 + n \log n$ operações?

11. Considere um polinômio $p(x) = \sum_{i=0}^{N} a_i x^i$, e suponha que o valor desse polinômio para um particular valor x é calculado usando o método de Horner:

$$p(x) = a_0 + x(a_1 + x(a_2 + x(a_3 + \cdots + x(a_{n-1} + xa_n)\ldots).$$

Caracterize em notação BigOh o número de operações aritméticas (adições e multiplicações) executadas nesse método.

12. Escreva uma rotina recursiva que resolve com custo $\mathcal{O}\left(2^N\right)$ o seguinte problema: Dado um conjunto de N inteiros A_1 a A_N, e um inteiro K maior que todos os A_i, descubra se existe um subconjunto dos inteiros A_1 a A_N cuja soma é exatamente K. (Nota: Esse problema é chamado Problema da Soma de Subconjuntos; não existe nenhuma solução conhecida para esse problema que seja polinomial em N. A descoberta de uma solução polinomial para esse problema é um dos problemas abertos mais importantes em matemática, e o problema mais importante em computação teórica.)

13. Considere a sequência: [26,34,9,0,4,89,6,15,27,44]. Desenhe esquematicamente os passos realizados por ordenação por inserção e por união (mergesort) para ordenar essa sequência de forma crescente.

14. Qual é a ordem de complexidade em notação BigOh para ordenamento por inserção, seleção, união (mergesort) nos seguintes casos:

    - O arranjo a ser ordenado está ordenado em ordem crescente.
    - O arranjo a ser ordenado está ordenado em ordem decrescente.

    Considere que o resultado final deve ser o arranjo ordenado em ordem crescente.

15. Escreva uma função que faça ordenamento de quatro inteiros distintos com no máximo cinco comparações.

16. Prove que o seguinte algoritmo ordena corretamente um arranjo `a` entre índices `i` e `j` (inclusive), e obtenha a complexidade do algoritmo em notação BigOh.

```
void ordena(int a[], int i, int j) {
    if (a[i] > a[j])
        troque a[i] e a[j]
    if ((i+1) >= j)
        return;
    int k = (i-j+1)/3;
    ordena(a,i,j-k);
    ordena(a,i+k,j);
    ordena(a,i,j-k);
}
```

17. Considere o método:

```
1 void prova(int a[], int b[], int c) {
2    BinarySearch s = new BinarySearch(a);
3    int i = s.search(c);
4    for (int j = 0; j<a.length; j++)
5       for (int k=0; k<b.length; k++)
6          System.out.println( j + s.search(b[k]) );
7 }
```

    onde BinarySearch é uma classe pública que executa busca binária em arranjos; o construtor de `BinarySearch` recebe um arranjo de inteiros e o método público

`search(int c)` procura o inteiro `c` no arranjo, retornando o índice de `c` no arranjo. Suponha que `c` e que todos os elementos de `b` sempre estão em `a`. Considere que `a` e `b` podem ter tamanho `N` e no pior caso os dois tem tamanho `N`.

(a) Qual a complexidade do programa em notação BigOh no pior caso? Justifique.

(b) Suponha que a terceira linha seja removida; qual a complexidade do programa resultante em notação BigOh, no pior caso? Justifique.

(c) Suponha que a sexta linha seja substituída por `System.out.println( j + k )`. Qual a complexidade do programa resultante em notação BigOh no pior caso? Justifique.

(d) Suponha que a quarta e quinta linhas sejam substituídas por

```
for (j = i; j<a.length; j++)
    for (k = j; k<b.length; k++)
```

Qual a complexidade do programa resultante em notação BigOh no pior caso? Justifique.

18. Considere o código abaixo, assumindo que n é maior que zero:

```
1   int exercicio1(int n, int x) {
2       int sum = 0;
3       if (x < 0)  return(-1);
4       if (x >= n)  x = n;
5       for (int i = 0; i < n; i++) {
6           sum++;
7           for (int j = 0; j < n; j++) {
8               auxiliar(sum, n);
9               for (int k = j; k < n * x; k++)
10                  sum = sum + 2;
11          }
12      }
13      return(sum);
14  }
15  void auxiliar(int sum, int n) {
16      sum = n * sum;
17  }
```

(a) Obtenha o valor de retorno da função `exercicio1` em termos de n e x. Justifique. [1.0]

(b) Obtenha a complexidade em notação BigOh em função *apenas* de n no pior caso. Justifique. [1.0]

(c) Apresente esquematicamente (através de um desenho) o conteúdo do Stack e do Heap do programa ao fim da primeira execução da linha 6, assumindo que tanto Stack quanto Heap estão vazios na chamada de `exercicio1(10,1)`. [0.5]

(d) Suponha que a linha 5 seja substituída pela linha

```
5                   for (int i = x; i < n; i++) {
```

Obtenha o valor de retorno em função de n e x; assumindo que n é par, obtenha a complexidade em notação BigOh em função apenas de n no pior caso. Justifique. [1.5]

19. Considere o código abaixo:

```
1  int exercicio2(int a) {
2    return( exercicio2(a, 0, a.length) );
3  }
4  int exercicio2(int a, int x, int y) {
5    int n = y-x;
6    if (n < 2) return(0);
7    int z = 0;
8    for (int i=0; i<7; i++) {
9      Ordena.ordenaInsercao(a, i * n/7,
10                                (i+1) * (n/7) - 1);
11     z = z + exercicio2(a, i * n/7,
12                                (i+1) * (n/7) - 1);
13   }
14 return(z);
15 }
```

A função `Ordena.ordenaInsercao(a, x, y)` ordena o sub-arranjo de a entre indices x e y (inclusive) por inserção. Para simplificar, assuma que n é uma potência de um número que for apropriado.

(a) Apresente uma relação de recorrência que rege o custo $T(n)$ desta função em termos do comprimento n do arranjo de entrada. Justifique. [1.0]

(b) Resolva a relação de recorrência obtida no item anterior, justificando cada passo. [1.5]

(c) Suponha que a linha 9 seja substuída por ordenação por união. Qual é a relação de recorrência resultante? Justifique. [0.5]

20. Considere uma busca sequencial de um elemento `X` em um arranjo de N elementos. Normalmente assumimos que cada comparação utilizada na busca tem um custo fixo. Suponha porém que cada comparação exija uma busca binária no arranjo de N elementos. Qual é a complexidade da busca de `X` em notação BigOh no pior caso?

21. Para o código a seguir, indique o trecho crítico do programa (isto é, as linhas que se repetem mais vezes). Determine o valor da variável `sum` ao final. Indique a complexidade no pior caso em notação BigOh. Justifique.

```
int sum = 0;
for (int i=0; i<n; i++)
    for (int j = 0; j<(n*i); j++)
        for (int k=0; k<j; k++)
            sum++;
```

DICA: O resultado é $\mathcal{O}\left(N^5\right)$, pois o valor de `sum` atinge $N^5/6 - N^4/4 - N^3/6 + N^2/4$.

22. Para o código a seguir, indique o trecho crítico do programa. Determine o valor da variável `sum` ao final. Indique a complexidade no pior caso em notação BigOh. Justifique.

```
int sum = 0;
for (int i=0; i<n; i++)
    sum++;
for (int i=0; i<n; i++)
    for (int j=0; j<(i*i); j++)
        if ((j%i) == 0)
            for (int k=0; k<j; k++)
                sum++;
```

DICA: O resultado é $\mathcal{O}\left(N^4\right)$, pois o valor de `sum` atinge $N^4/8 - 5N^3/12 + 3N^2/8 - 11N/12$. Este valor é obtido pela expressão

$$\sum_{i=0}^{N-1} 1 + \sum_{i=1}^{N-1} \sum_{j=0}^{i-1} \sum_{k=0}^{ij-1} 1.$$

Note o efeito da operação % na complexidade do programa.

23. Demonstre que o seguinte programa leva a um loop infinito para n maior que 2.

```
public boolean verify(int x, int y, int z, int n) {
    boolean flag = false;
    int auxX, auxY, auxZ;
    while (flag == false) {
        auxX = 1; auxY = 1; auxZ = 1;
        for (int i=0; i<n; i++) {
            auxX *= x;
            auxY *= y;
            auxZ *= z;
        }
        if (auxX + auxY == auxZ)
            flag = true;
        n++;
    }
}
```

DICA: Esse problema é extremamente simples! Verifique a expressão matemática produzida para cada valor de n, e argumente que a expressão não tem solução usando um resultado conhecido em matemática.

24. Considere os seguintes métodos:

```
public cube(int n) {
    aux = 0;
    for (int i=1; i<=n; i++)
        aux += i*i*i;
    return(aux);
}
public square(int n) {
```

```
    aux =0;
    for (int i=1; i<=n; i++)
        aux += i;
    return(aux * aux);
}
```

(a) Indique a complexidade dos dois métodos em notação BigOh.

(b) Demonstre que os dois métodos retornam o mesmo valor.

(c) Escreva um método que retorna o mesmo valor em tempo $\mathcal{O}(1)$.

25. Considere o seguinte teorema (conhecido como Teorema do Limite Superior): um politopo com N vértices em D dimensões tem $KN^{D/2}$ faces para uma constante $K > 0$ e para qualquer N maior que um certo M. A partir desse teorema, prove que a ordenação e impressão das faces de um politopo com N vértices em 4 dimensões pode ser feita em $\mathcal{O}\left(N^2 \log N\right)$. Justifique a partir da definição da notação BigOh.

    DICA: Lembre-se que um politopo é um objeto geométrico definido por hiperplanos em um espaço com D dimensões (para $D = 2$, temos poliedros).

26. Considere um programa com dois métodos. O primeiro método recebe um número N e produz um arranjo `a` de tamanho $N!$, onde cada elemento do arranjo é uma permutação do arranjo `b` igual a $[1, 2, ..., N]$ (existem $N!$ permutações de uma lista com N elementos). Suponha que `a` seja ordenado de alguma forma, e o segundo método realiza uma busca binária em `a`. Prove que a complexidade do segundo método é $\mathcal{O}(N \log N)$, usando a aproximação de Stirling:

$$N! = \sqrt{2N\pi}(N/e)^N(1 + \mathcal{O}(1/N)).$$

27. Considere os dois programas abaixo:

```
int doIt1(int a[]) {
    int n = a.length;
    int sum = 0;
    for (int i=0; i<=n; i++)
       for (int j=0; j<=i; j++)
          sum += a[i] * a[j];
    return(sum);
}
int doIt2(int a[]) {
    int n = a.length;
    int sum = 0;
    int sum2 = 0;
    for (int i=0; i<=n; i++)  {
       sum += a[i];
       sum2 += a[i] * a[i];
    }
    return( (sum * sum + sum2)/2 );
}
```

(a) Determine a complexidade de cada método no pior caso em notação BigOh.

(b) Prove que os dois métodos retornam o mesmo resultado. (Dica: Use indução finita em n, o tamanho do vetor `a`. Note que os dois métodos retornam o mesmo resultado para $n = 1$, assuma que os resultados são iguais para n genérico e prove que são iguais para $(n + 1)$.)

28. Considere o seguinte método:

```
public void doIt(int a[], int b[]) {
    Ordena.ordena(a);
    for (int i=0; i<b.length; i++) {
        int aux = BinarySearch.search(a, b[i]);
        for (int j=aux; j<a.length; j++)
            System.out.println(a[j]);
    }
}
```

onde:

- `void ordena(int a[])` é um método de classe (da classe `Ordena`) que faz o ordenamento do arranjo `a`).
- `int search(int a[], int x)` é um método de classe (da classe `BinarySearch`) que faz busca binária do elemento `x` no arranjo `a` e retorna o índice de `x` em `a`.
- `a` e `b` são arranjos de tamanho máximo N.
- todos os elementos de `b` estão em `a` (os elementos de `b` podem ser repetidos).

Qual a complexidade em notação BigOh no pior caso em função de N para (justifique!):

(a) método `ordena` implementado com ordenação por inserção.

(b) método `ordena` implementado com mergesort.

29. Você foi chamado por uma empresa para analisar um bloco fundamental de programas de controle de fluxo de caixa. Você identificou que o seguinte método é chamado muito frequentemente:

```
public int doIt(int a[]) {
    int i, j, k;
    int sum = 0;
    int aux = 1;
    for (i=0; i<a.length; i++) {
        sum++;
        aux *= a[i];
    }
    for (i=0; i<a.length; i++) {
        for (j=0; j<a.length; j++) {
            sum++;
            aux = aux + (a[i] - a[j]);
        }
    }
```

```
    for (i=0; i<a.length; i++)
        for (j=i; j<(2*i); j++)
            for (k=j; k<(j+5); k++)
                sum++;
    return(sum+aux);
}
```

(a) Determine a complexidade do programa em função de N, o tamanho do vetor de entrada `a`, em notação BigOh. Justifique.

(b) Codifique o programa de forma que o resultado seja o mesmo mas a complexidade seja $\mathcal{O}(N)$. Justifique.

30. Considere o seguinte fragmento de código. A função `proximoArranjo(`M`)` retorna um arranjo de comprimento M preenchido com inteiros lidos a partir de um arquivo pré-especificado (essa operação tem custo $\mathcal{O}(M)$), e a função `ordenaUniao(`a`)` ordena o arranjo a usando ordenação por união. A função `auxiliar(`a`,`x`)` é recursiva: cada chamada onde o arranjo a tem comprimento Q realiza operações de custo $\mathcal{O}(Q)$ seguidas de 3 chamadas recursivas à mesma função `auxiliar`, cada uma delas usando metade do arranjo a.

```
1   public static REC1(int N, int M) {
2      int i, j, a[], sum = 0;
3
4      for (i=0; i<N; i++) {
5         a = proximoArranjo(M);
6         ordenaUniao(a);
7         for (j=0; j<M; j++)
8            if (a[j] == 0)
9               sum = sum + 1;
10           else
11              sum = auxiliar(a,a[j]);
12     }
13     return(sum);
14  }
```

Qual o custo de `REC1` em função de N e M, em notação BigOh no pior caso? Justifique cada passo do seu argumento.

31. Considere o seguinte fragmento de código:

```
1   public static SUB1(Object a[], int b[][],
2                      boolean saveMemory) {
3      int i, j, k, sum = 0;
4
5      for (k=0; k<b.length; k++) {
6         if (saveMemory == true) ordenaSelecao(b[k]);
7         else                    ordenaUniao(b[k]);
8      }
9      boolean flag = false;
10     for (k=0; k<b.length; k++) {
11        if (b[k][0] == 0) flag = true;
```

```
12      }
13      if (flag == true) {
14          for (i=0; i<a.length; i++) {
15              for (j=i+1; j<a.length; j++) {
16                  if (a[i]==a[j]) sum = sum+1;
17              }
18          }
19      }
20      return(sum);
21  }
```

O arranjo `a` tem N linhas e N colunas, e o arranjo b tem N elementos. A função `ordenaSelecao(c)` ordena o arranjo `c` por seleção e a função `ordenaUniao(c)` ordena o arranjo `c` por união. Assuma $N > 1$.

(a) Qual o custo de `SUB1` em função de N, em notação BigOh no pior caso? Justifique.

(b) Recodifique a função de forma que o seu retorno seja igual, mas as funções `ordenaSelecao(c)` e `ordenaUniao(c)` *não* sejam usadas, e o custo seja $\mathcal{O}(N^2)$ em notação BigOh no pior caso. Justifique.

32. Uma função recursiva `SUB2(a,k)` recebe como entrada um arranjo `a` de tamanho N e um inteiro k que indica o nível da recursão. A função é sempre chamada como `SUB2(a,0)`. Uma análise da função indicou que seu custo $T(N)$ é $\mathcal{O}(1)$ para $N = 1$ e ($\alpha > 0$ é uma constante)

$$T(N,k) = \begin{cases} 2T(N/2, k+1) + \alpha N, & \text{para } k \text{ par,} \\ 4T(N/2, k+1) + \alpha N, & \text{para } k \text{ ímpar.} \end{cases}$$

Obtenha o custo da chamada `SUB2(a,0)` assumindo que N é uma potência de 2.

33. Considere uma recursão em que cada problema de tamanho N é dividido recursivamente em 4 sub-problemas. Cada um desses sub-problemas tem tamanho $N/2$. Assuma que N é um número cujo logaritmo em uma base conveniente é um número inteiro. Qual é a complexidade dessa recursão para um problema de tamanho N, se o "custo" de unir as soluções dos quatro sub-problemas de um único problema de tamanho N é exatamente N^2, e o "custo" de resolver um problema com $N = 1$ é exatamente 1.

DICA: Considere o primeiro caso. A recursão leva à seguinte expressão:

$$T(N) = 4T(N/2) + N^2.$$

Podemos escrever

$$T(N/2) = 4T(N/4) + N^2/4$$

e portanto:

$$T(N) = 16T(N/4) + N^2 + N^2.$$

Da mesma forma:

$$T(N/4) = 4T(N/8) + N^2/16 \Rightarrow T(N) = 64T(N/8) + N^2 + N^2 + N^2.$$

Seguindo nesse raciocínio, atingimos:

$$T(N) = 4^k T(N/2^k) + kN^2.$$

Para $k = \log_2 N$ (pois a árvore de recursão terá tamanho $\log_2 N$):

$$T(N) = N^2 T(1) + N^2 \log_2 N.$$

Portanto a complexidade é $\mathcal{O}\left(N^2 \log N\right)$.

Tente resolver para o caso em que o "custo" de unir soluções é N^3; a solução é $\mathcal{O}\left(N^3\right)$. Lembre-se da expressão:

$$\sum_{k=0}^{n} a^k = (a^{(n+1)} - 1)/(a - 1).$$

34. Resolva a seguinte equação, resultado da análise de um algoritmo recursivo:

$$T(N) = aT(N/b) + N \log N,$$

para:

(a) `a` igual a 3 e `b` igual a 2.

(b) `a` igual a 2 e `b` igual a 2.

(c) `a` igual a 2 e `b` igual a 3.

Assuma que N é uma potência na base mais conveniente e que $T(1) = 1$.

35. Considere o seguinte programa recursivo:

```
public int doIt(int a[], int i, int j) {
    int n = j - i - 1;
    if (n <= 1)
        return(1);
    else
    return( doIt(a, i, i+n/2-1) +
            doIt(a, i+n/4, i+3*n/4-1) +
            doIt(a, i+n/2, j) );
}
```

Considere agora o seguinte método:

```
public int do(int a[]) {
    return( doIt(a, 0, a.length - 1) );
}
```

Qual é a ordem de complexidade da chamada `do(a)` para um vetor `a` de tamanho N, em notação BigOh? Assuma que N é uma potência de 2.

36. Considere o seguinte fragmento de código, onde `a` e `b` são arranjos de mesmo tamanho N (onde $N > 2$) e a função
`Sort.insertionsort(int a[], int o, int f)`
ordena o arranjo `a` entre `o` e `f`, usando ordenação por inserção.

```
public void doIt(int a[], int b[], int o, int f) {
    if ( (f-o) < 2 )
        return(a[f]);
    else
        Sort.insertionsort(a, o, f);
    if (a[0] < b[0])
        doIt(a, b, o, (f+o)/2);
    else
        doIt(a, b, (f+o)/2, f);
    if (a[1] < b[1])
        doIt(a, b, o + (f-o)/4, o + 3*(f-o)/4);
    doIt(a, b, o, o + (f-o)/3);
}
```

Escreva a equação recursiva que deve ser resolvida para obter a complexidade deste fragmento de código em notação BigOh, considerando o pior caso. Indique também as condições de contorno que devem ser atendidas pela solução (isto é, qual a complexidade da parada da recursão). Justifique a resposta. [Note: não há necessidade de resolver a equação!]

37. Resolva as seguintes equações recursivas, expressando o resultado em notação BigOh e assumindo que N é uma potência na base mais conveniente:

    - $T(1) = 1, T(N) = 3T(N/2) + N$.
    - $T(1) = 1, T(N) = 7T(N/3) + N^2$.
    - $T(1) = 1, T(N) = 2T(N-1) + N$.