

PUC-Rio
Departamento de Informática
Prof. Marcus Vinicius S. Poggi de Aragão
Período: 2006.2
4 de setembro de 2006
Horário: 2as-feiras e 4as-feiras de 9 às 11 horas

ANÁLISE DE ALGORITMOS (INF 1721)

1ª Lista de Exercícios

1. Considere as seguintes funções:

- (a) $10.n^\pi$
- (b) $\log n$
- (c) $\log(n^2)$
- (d) $30.n^3.\log n$
- (e) $50.n.\log^2 n$
- (f) $(\log n)^{\log n}$
- (g) $\frac{n}{\log n}$
- (h) $70.n$
- (i) $\log \log n$
- (j) $90.n^2.\log n + n^3.\log n$

- Coloque as funções acima em ordem de crescimento assintótico, i.e. valor quando $n \rightarrow \infty$.
- Utilize pelo menos três vezes cada um dos símbolos O , Ω , Θ , o , e ω para indicar a relação existente entre pares das funções acima (não vale recíprocos).

2. Considere uma matriz quadrada M contendo n números inteiros (você pode assumir que $n = (2^k)^2$ para algum k inteiro). Esta matriz é dada ordenada tanto nas linhas como nas colunas (veja o exemplo abaixo, para $n = 64$, k seria 3). Considere agora o seguinte problema: Dado um número inteiro i , deseja-se saber se este se encontra na matriz M . Analise a complexidade (de pior caso) dos algoritmos abaixo para resolver este problema. A complexidade deve ser em função de n .

(a) **Algoritmo I**

Passo 0: Seja l uma linha de M . Faça $l = 0$.

Passo 1: Execute uma Busca Binária para encontrar i na linha l de M . Se i for encontrado, responda SIM e PARE. Senão, incremente l .

Passo 2 Se l é uma linha de M vá para o passo 1. Senão, responda NÃO e PARE.

$$\begin{bmatrix} 4 & 10 & 21 & 34 & 39 & 40 & 57 & 62 \\ 9 & 17 & 28 & 35 & 41 & 42 & 58 & 72 \\ 19 & 20 & 30 & 47 & 49 & 49 & 65 & 73 \\ 27 & 39 & 40 & 48 & 52 & 60 & 66 & 79 \\ 33 & 43 & 44 & 58 & 59 & 61 & 77 & 81 \\ 46 & 46 & 60 & 63 & 69 & 71 & 79 & 88 \\ 56 & 61 & 62 & 68 & 72 & 76 & 87 & 88 \\ 64 & 67 & 69 & 73 & 83 & 89 & 90 & 91 \end{bmatrix}$$

(b) **Algoritmo II**

Passo 0: Seja (p, q) a posição central de M , que divide M em $M1$, $M2$, $M3$ e $M4$, onde $M1$ é a submatriz superior esquerda, $M2$ a superior direita, $M3$ a inferior esquerda e $M4$ a inferior direita, de tamanhos iguais. Se $M1$ for vazia, Responda NÃO e PARE. Senão vá para o passo 1.

Passo 1 Se i for igual a $M(p, q)$, responda SIM e PARE. Senão vá para o passo 2.

Passo 2: Se i for menor que $M(p, q)$ execute Alg. II para as matrizes $M1$, $M2$, e $M3$. Senão execute Alg. II para $M2$, $M3$ e $M4$.

(c) **Algoritmo III**

Passo 0: Seja (p, p) uma posição da diagonal de M .

Passo 1: Busque sequencialmente na diagonal a posição p tal que $M(p, p) < i < M(p+1, p+1)$ utilize essa posição para obter quatro matrizes a partir de M : $M1$, $M2$, $M3$ e $M4$ (observe que o pior caso é o que tem estas 4 matrizes de tamanhos iguais). Se $M1$, $M2$, $M3$ e $M4$ forem vazias, Responda NÃO e PARE. Senão vá para o passo 2.

Passo 2 Se i for igual a $M(p, p)$ ou $M(p+1, p+1)$, responda SIM e PARE. Senão vá para o passo 3.

Passo 3: Execute Alg. III para as matrizes $M2$ e $M3$.

(d) **Algoritmo IV**

Passo 0: Seja (p, p) uma posição da diagonal de M .

Passo 1: Utilize *Busca Binária* na diagonal para encontrar a posição p tal que $M(p, p) < i < M(p+1, p+1)$ utilize essa posição para obter quatro matrizes a partir de M : $M1$, $M2$, $M3$ e $M4$ (observe que o pior caso é o que tem estas 4 matrizes de tamanhos iguais). Se $M1$, $M2$, $M3$ e $M4$ forem vazias, Responda NÃO e PARE. Senão vá para o passo 2.

Passo 2 Se i for igual a $M(p, p)$ ou $M(p+1, p+1)$, responda SIM e PARE. Senão vá para o passo 3.

Passo 3: Execute Alg. IV para as matrizes $M2$ e $M3$.

3. Seja $f_i(n)$ o número de operações realizadas pelo programa i abaixo (\log logaritmo na base 2).

- Obtenha $f_i(n)$ em função de n e de grupos de operações que demandam tempo constante (para os quais você pode utilizar símbolos c_1, c_2, \dots para indicar esse número constante de operações). Caso existam valores indeterminados, considere-os como variáveis e dê o intervalo de valores que esta pode assumir.

- Encontre funções $l_i(n)$ e $u_i(n)$ tais que $f_i(n) = \Omega(l_i(n))$ (a maior possível) e $f(n) = O(u_i(n))$ (a menor possível).
- Indique quando existe uma função $h_i(n)$ tal que $f_i(n) = \Theta(h_i(n))$.
- Caso o algoritmo seja recursivo, escreva sua relação de recorrência. Caso seja iterativo e apareçam somatórios, escreva-os.

***** A1 *****

Chamada: A1 (n,k)

```
void A1 (int n, int k)
```

```
{
  int i,f,m;
  i = 0;
  f = (int)(log(n)/2);
  while ( 1 )
  {
    m = (int)((i+f)/2);
    p = 2**m;
    if (k < p){
      if (k >= (p/2) )
      {
        return(p/2);
      }
      f = m;
    }
    else {
      if (k < 2*p)
      {
        return(p);
      }
      i = m;
    }
  }
}
```

***** A2 *****

Chamada: A2 (1,n,k)

```
int A2 (int i, int f, int k)
```

```
{
  int a,b,p;
  if (i < f)
  {
    a = (int)log(i);
    b = (int)log(f);
    h = (int)((a+b)/2);
    p = 2**h;
    if (k < p)
    {
      if (k >= (p/2) )
      {
        return(p/2);
      }
    }
  }
}
```

```

        }
    else
    {
        return(A2(i, p/2, k));
    }
}
else
{
    if (k < 2*p)
    {
        return(p);
    }
    else
    {
        return(A2(2*p, f, k));
    }
}
}
}

```

***** A3 *****

Chamada: A3 (1, n)

void A3 (int i, int f)

```

{
    int j, continue;
    if (i <= f)
    {
        continue = 1;
        j = i;
        while ( (j <= f) && continue)
        {
            if (V[i] > V[j])
            {
                V[i] = V[j];
                continue = 0;
            }
            j = j + 1;
        }
        A3(i, (int)(f/2));
    }
}
}

```

***** A4 *****

Chamada: A4 (1, n)

void A4 (int i, int f)

```

{
    int j, continue;
    if (i <= f)
    {
        continue = 1;
        j = i;

```

```

while ( (j <= f) && continue)
{
    if (V[i] > V[j])
    {
        V[i] = V[j];
        continue = 0;
    }
    j = j + 1;
}
m = (int)((f-i)/5);
A4(i, i+m);
A4(i+m+1, i+2*m);
A4(i+2*m+1, i+3*m);
}
}

```

4. Dê uma descrição precisa dos algoritmos de ordenação abaixo e analise suas respectivas complexidades de pior caso. A entrada é um conjunto de n inteiros.
- Ordenação por intercalação (*merge sort*)
 - Ordenação por rápida (*quick sort*)
 - Ordenação por inserção:
 - Utilizando um vetor como estrutura abstrata de dados (*insertion sort*);
 - Utilizando uma árvore balanceada de busca como estrutura abstrata de dados: descreva as etapas no algoritmo que são fase de inserção e de preenchimento do vetor ordenado;
 - Utilizando uma *Heap* como estrutura abstrata de dados: descreva as etapas no algoritmo que são fase de inserção e de preenchimento do vetor ordenado (*Heapsort*);
 - Discute as 3 versões acima.
 - Ordenação por radicais (*radix sort*), neste último considere que os inteiros são positivos e menores que 100000.
5. Seja o problema de ordenar um conjunto de n inteiros I onde qualquer que seja $x \in I$, $0 \leq x \leq n^3$. Apresente um algoritmo $O(n)$ para ordenar esse conjunto I . Explique como isso é possível, uma vez que sabemos que a ordenação de um conjunto de n elementos tem limite inferior de complexidade $n \log n$, i.e. $\Omega(n \log n)$.
6. Dado um vetor V (ou tupla, i.e. elementos onde a ordem em que aparecem é relevante) de n elementos de \mathcal{Z} , uma **inversão** ocorre quando $\exists i, j$ tais que $i < j \leq n$ e $V[i] > V[j]$. Diz-se que (i, j) é uma inversão de V .
- Escreva um algoritmo iterativo (não-recursivo) que calcule o número de inversões de V .
 - Utilize divisão e conquista para obter um algoritmo recursivo. Dica: Este algoritmo deve ser semelhante ao MergeSort.
 - Escreva sua relação de recorrência.
 - Descreva as estruturas de dados e analise as respectivas complexidades dos algoritmos que você propôs acima.

7. Considere d seqüências ordenadas de números inteiros. O número total de elementos é n . Descreva um algoritmo que obtenha uma única seqüência ordenada com os n elementos em $O(n \log d)$.
8. Considere duas seqüências ordenadas S e T , onde $|S| = n$, $|T| = k$ e $k < n$. T é uma subseqüência de S se todo elemento de T está em S . Proponha um algoritmo $O(n)$ para determinar se T é ou não é uma subseqüência de S .
9. Seja x_1, x_2, \dots, x_n uma seqüência de números reais (não necessariamente positivos). Escreva um algoritmo de complexidade $O(n)$ para encontrar a subseqüência x_i, x_{i+1}, \dots, x_j (de elementos consecutivos) tal que o produto $x_1 * x_2 * \dots * x_n$ seja máximo (dentre todas as subseqüências de números consecutivos). O produto de uma subseqüência vazia é definido como 1.
10. Suponha que você tenha um algoritmo que funcione como uma *caixa preta* (você não pode ver como ele foi construído) e que tem as seguintes propriedades: se você insere qualquer seqüência de números reais e um inteiro k , o algoritmo retorna "Sim" ou "Não", indicando se existe um subconjunto dos números reais cuja soma seja exatamente k . Mostre como usar esta *caixa preta* para encontrar o subconjunto cuja soma seja k . Você pode usar a "caixa preta" $O(n)$ vezes (onde n é o tamanho da seqüência de números reais).
11. Escreva um algoritmo de "busca binária" que não divida o conjunto de elementos em 2 subconjuntos de tamanhos (aproximadamente) iguais, mas sim em um subconjunto de tamanho um terço do original e outro de tamanho dois terços do original. Compare a complexidade deste algoritmo com a do algoritmo de busca binária.
12. Escreva um algoritmo de "busca ternária" que primeiro testa se o elemento da posição $n/3$ é igual ao elemento procurado x e então, se necessário, testa o elemento da posição $2n/3$ para verificar a igualdade com o elemento x e, possivelmente, reduzir o tamanho do conjunto para um terço do original. Compare a complexidade deste algoritmo com a do algoritmo de busca binária.
13. Sejam x_1, x_2, \dots, x_n e y_1, y_2, \dots, y_n seqüências de inteiros ordenadas de forma não decrescente. Escreva um algoritmo que encontre a mediana da intercalação dos $2n$ elementos. (Dica: use busca binária).
14. Sejam u e v dois números de n bits (considere que n é potência de 2). A multiplicação tradicional de u por v utiliza $O(n^2)$ operações. Um algoritmo baseado em divisão e conquista divide os números em duas partes iguais, calculando o produto como: $uv = (a2^{n/2} + b)(c2^{n/2} + d) = ac2^n + (ad + bc)2^{n/2} + bd$. As multiplicações ac , ad , bc e bd são feitas usando este mesmo algoritmo recursivamente.
 - Determine a complexidade deste algoritmo; e
 - Qual é a complexidade se $ad + bc$ é calculado como $(a + b)(c + d) - ac - bd$?
15. Considere dois heaps h_1 e h_2 de tamanhos n e m , respectivamente. Escreva um algoritmo para construir um heap que contenha todos os elementos dos heaps h_1 e h_2 . A complexidade do algoritmo deve ser $O(\log(m + n))$ no pior caso.
16. Prove que se k é uma constante não negativa e n é potência de 2, então $T(n) = 3kn^{\log_2 3} - 2kn$ é a solução da recorrência:

$$T(n) = \begin{cases} k, & n = 1 \\ 3T(n/2) + kn, & n > 1 \end{cases}$$

17. • A equação abaixo utiliza o fato de que a soma $\sum_{i=0}^{\infty} (i/2^i)$ converge e é menor do que 2. Prove este fato.

$$\sum_{1 \leq i \leq k} 2^{i-1}(k-i) = \sum_{1 \leq i \leq k-1} i2^{k-i-i} \leq \sum_{1 \leq i \leq k-1} i/2^i < 2n = O(n)$$

- Use indução para mostrar que $\sum_{i=1}^k 2^{i-1}(k-i) = 2^k - k - 1$, $k \geq 1$.