Exame Preliminar para o Doutorado

IME–USP, Fevereiro, 2003

Prova de Análise de Algoritmos

Instruções:

(i) O candidato pode resolver todas as questões.

(ii) Esta prova contém quatro questões de um ponto, duas de dois pontos e duas de três pontos.

(iii) A banca considerará questões cujos valores somem até 10 pontos de modo que a soma total das notas obtidas seja máxima.

(iv) Mencione os teoremas e propriedades usados para justificar suas afirmações.

Questão 1 [1 ponto]
Foram implementados três algoritmos: `algoritmo A`, `B` e `C`. As tabelas abaixo mostram o resultado de um estudo experimental dessas implementações. Nas tabelas, para alguns tamanhos n de instâncias, é indicado o respectivo consumo de tempo do algoritmo, em segundos. Apresente funções $t_A(n), t_B(n)$ e $t_C(n)$ para as quais você suspeita que os `algoritmos A`, `B` e `C` gastam tempo $\Theta(t_A(n)), \Theta(t_B(n))$ e $\Theta(t_C(n))$, respectivamente. Justifique a sua resposta.

n	`algoritmo A`
256	0.00
512	0.01
1024	0.03
2048	0.09
4096	0.37
8192	1.70
16384	7.08
32768	28.54
65536	113.55
131072	493.08

n	`algoritmo B`
256	0.00
512	0.00
1024	0.00
2048	0.00
4096	0.01
8192	0.01
16384	0.02
32768	0.04
65536	0.10
131072	0.23
262144	0.54
524288	1.31
1048576	3.06
2097152	7.07
4194304	15.84
8388608	35.85

n	`algoritmo C`
1000000	0.06
2000000	0.11
3000000	0.16
6000000	0.32
8000000	0.43
10000000	0.55
15000000	0.82
20000000	1.08
25000000	1.37
30000000	1.70

Questão 2 [1 ponto]
Considere o problema de, dado um vetor a com n elementos e um natural k, imprimir os k menores elementos de a em ordem não-decrescente. Descreva um algoritmo que resolva esse problema em tempo $O(n + k \log n)$. Justifique a sua resposta.

Questão 3 [1 ponto]
Abaixo está um extrato da página de manual da função `readdir` em Linux.

(a) Escreva uma função em `C`-*like* que, tendo como parâmetros um nome de arquivo e um apontador para uma estrutura tipo DIR, devolve:

- um apontador para uma `dirent` com esse mesmo nome de arquivo, se existir (o campo `d_name` é um string terminado por `\0`), ou
- um apontador nulo caso não exista tal entrada.

(b) Suponha que `dir` aponte para n arquivos e que cada chamada de `readdir` gaste tempo constante. Qual a complexidade assintótica do seu algoritmo em termos de comparações de caracteres?

```
NAME
       readdir - read a directory

SYNOPSIS

       struct dirent *readdir(DIR *dir);

DESCRIPTION
       The  readdir()  function  returns  a  pointer  to a dirent
       structure representing the next  directory  entry  in  the
       directory  stream  pointed  to by dir.  It returns NULL on
       reaching the end-of-file or if an error occurred.

       The data returned by readdir() is overwritten by subsequent
       calls to readdir() for the same directory stream.

       The dirent structure is defined as follows:

              struct dirent {
                      long            d_ino;  /* inode number */
                      off_t           d_off;  /* offset to the next dirent */
                      unsigned short  d_reclen;/* length of this record */
                      char            d_name[256];    /* filename */
              };

RETURN VALUE
       The  readdir()  function  returns  a  pointer  to a dirent
       structure, or NULL if an error occurs  or  end-of-file  is
       reached.
```

Questão 4 [1 ponto]
Um grafo é *numerado* se seus vértices são $1, 2, \ldots, n$. Suponha que *jóia* é uma propriedade de grafos numerados, ou seja, é algo que pode depender da numeração dos vértices. Por exemplo: a propriedade "$\langle 1, 2, \ldots, n \rangle$ é um caminho no grafo".

Claro que renumerar os vértices de um grafo (respeitando as adjacências) pode afetar a validade da propriedade jóia. Temos então dois problemas:

JÓIA: Dado um grafo numerado G, decidir se G é jóia.

pJÓIA: Dado um grafo numerado G, decidir se existe uma renumeração de G que é jóia.

(a) Se JÓIA está em P, pJÓIA está em P?

(b) Se JÓIA está em NP, pJÓIA está em NP?

Justifique suas respostas.

Questão 5 [2 ponto]
Suponha que você tenha um algoritmo $\Theta(n^2)$ que resolva um problema com entrada de tamanho n.

Suponha que, para o mesmo problema, exista um outro algoritmo, do tipo *Divisão e Conquista*, que divida a entrada em dois sub-problemas de tamanhos $\lceil n/2 \rceil$ e $\lfloor n/2 \rfloor$ em tempo $D(n) = n \log_2 n$ e depois use tempo $C(n) = n \log_2 n$ para combinar as duas soluções obtidas. Esse algoritmo leva tempo constante para resolver o problema quando $n \leq 1$.

Qual dos dois algoritmos é assintoticamente mais eficiente? Considere na sua resposta que n é sempre potência de 2.

Questão 6 [2 pontos]
A função de Pascal pode ser computada pela seguinte rotina recursiva:

```
int Pascal (int a, int b) {
  if (a == 0 || b == 0) return 1;
  else return Pascal(a-1,b) + Pascal (a,b-1);
}
```

(a) Ache uma boa delimitação inferior para o consumo de tempo da rotina acima para o caso em que $a = b$. (Sua delimitação deve ser uma função de a.)

(b) Usando programação dinâmica, mostre como reescrever tal rotina para que o tempo consumido seja $\Theta(ab)$ e o espaço $\Theta(\min(a, b))$.

Questão 7 [3 pontos]
Dizemos que um vetor a de comprimento n é *quase ordenado com erros de tamanho* k para $k < n$ se, para quaisquer i e j, se $j - i > k$ então $a[j] \leq a[i]$. Assim o vetor não tem que estar completamente ordenado, mas quaisquer dois elementos no vetor que estejam fora de ordem crescente não podem distar mais do que k posições. Por exemplo, a lista

5, 8, 1, 6, 15, 12, 11, 20, 19, 25, 30, 35, 32

é quase ordenada com erros de tamanho 2. Em particular, $a[3] = 1 < 5 = a[1]$ e $3 - 1 = 2$, mas não há dois elementos fora de ordem distando três ou mais posições um do outro. Justifique suas respostas.

a) Mostre como o `quicksort` pode ser modificado para produzir uma lista que é quase ordenada com erros de tamanho k. Qual é a complexidade de pior caso para tal versão modificada do `quicksort`? E de melhor caso?

b) Se o vetor de entrada é quase ordenado com erros de tamanho k, qual é a complexidade do algoritmo de ordenação conhecido como `insertion sort`?

c) Descreva um algoritmo $O(n)$ que, dado um vetor a de comprimento n e um inteiro positivo k, verifique se a é quase ordenado com erros de tamanho k.

Questão 8 [3 pontos]
Considere um tipo abstrato de dados para o qual está definida a comparação

> `compare(x,y)`: devolve 1, caso `x<y`, 0 caso `x=y` e -1 caso `x>y`

e a atribuição de valores a variáveis.

Chamamos de *estoque* o tipo abstrato de dados para elementos deste tipo, que permite as seguintes operações:

> `insira(x)`: a inserção deve ser efetuada mesmo que `x` já esteja na estrutura de dados. Em outras palavras, a estrutura de dados deve permitir duplicatas;
>
> `remova()`: remova um elemento arbitrário da estrutura de dados e o devolva. Se há várias cópias do mesmo elemento, apenas uma deve ser removida.

Tal tipo é útil para armazenar tarefas, por exemplo. Novas tarefas são geradas e inseridas no estoque e, quando um trabalhador fica disponível, uma tarefa é removida do estoque. Suponha que a operação `compare(x,y)` é dada e que consome tempo $O(1)$. Justifique as suas respostas.

(a) Descreva uma estrutura de dados para armazenar um estoque em que as operações `insira` e `remova` possam ser implementadas de forma a consumir tempo $O(1)$.

(b) E se duplicatas não são permitidas? Ou seja, se uma inserção é efetuada apenas se o elemento não está na estrutura de dados? Que estrutura de dados você utilizaria neste caso e qual seria o pior caso do tempo consumido por cada uma das operações `insira` e `remova`?

(c) Suponha que os elementos a serem armazenados na estrutura de dados são inteiros no intervalo $[0..n]$, onde n é pequeno o suficiente para que se possa gastar espaço $O(n)$ e que duplicatas não sejam admitidas, como no item b). Descreva uma estrutura de dados para este caso em que as operações `insira` e `remova` possam ser implementadas de forma a consumir tempo $O(1)$.

BOA SORTE!