UNIVERSIDADE DE SÃO PAULO - USP
INSTITUTO DE CIÊNCIAS MATEMÁTICAS E DE COMPUTAÇÃO - ICMC

SCE5832 – Teoria da Computação – 1º. Semestre 2008

Docente Responsável:
Graça Nunes

LISTA DE EXERCÍCIOS I

1. Ordene as seguintes funções por suas taxas de crescimento: n, $\sqrt{n}$, $n^{1,5}$, n^2, $n\log(n)$, $n\log(\log(n))$, $n(\log(n))^2$, $n\log(n^2)$, $2/n$, 2^n, $2^{n/2}$, 37, $n^2\log(n)$, n^3.
Indique quais funções têm taxa de crescimento iguais.

2. Suponha $T_1(n) = O(f(n))$ e $T_2(n) = O(f(n))$. Quais das seguintes afirmações são verdadeiras?
   a) $T_1(n) + T_2(n) = O(f(n))$
   b) $T_1(n) - T_2(n) = o(f(n))$
   c) $T_1(n) / T_2(n) = O(1)$
   d) $T_1(n) = O(T_2(n))$

3. Vamos supor que estamos comparando implementações de ordenação por inserção e ordenação por intercalação na mesma máquina. Para entradas de tamanho *n,* a ordenação por inserção é executada em $8n^2$ etapas, enquanto a ordenação por intercalação é executada em $64\ n\ log\ n$ etapas. Para que valores de *n* a ordenação por inserção supera a ordenação por intercalação?

4. Para cada um dos seguintes trechos de algoritmos abaixo:
   a) Analise o tempo estimado de execução;
   b) Implemente o código e execute-o para vários valores de *n*;
   c) Compare sua análise com os tempos obtidos.

(1)
```
...
soma ← 0;
Para i ← 0 até n-1
   Faça soma ← soma + 1;
...
```

(2)
```
...
soma ← 0;
Para i ← 0 até n-1
   Faça Para j ← 0 até n-1
        Faça soma ← soma + 1;
...
```

(3)
```
...
soma ← 0;
Para i ← 0 até n-1
   Faça Para j ← 0 até n*n-1
      Faça soma ← soma + 1;
...
```

(4)
```
...
soma ← 0;
Para i ← 0 até n-1
   Faça Para j ← 0 até i-1
      Faça soma ← soma + 1;
...
```

(5)
```
...
soma ← 0;
Para i ← 0 até n-1
   Faça Para j ← 0 até i*i-1
      Faça Para k ← 0 até j-1
         Faça soma ← soma + 1;
...
```

(6)
```
...
soma ← 0;
Para i ← 1 até n
   Faça Para j ← 1 até i*i
      Faça Se resto(j,i) = 0
         Então Para k ← 0 até j-1
            Faça soma ← soma + 1;
...
```

5. Suponha que você precisa gerar permutações aleatórias dos n primeiros números inteiros. Por exemplo, {4, 3, 1, 5, 2} e {3, 1, 4, 2, 5} são permutações corretas, mas {5, 4, 1, 2, 1} não é, porque o número 1 está duplicado e o número 3 não aparece. Esta rotina é usada com freqüência em simulações. Assumimos a existência de um gerador de números aleatórios `gera_aleat(i,j)`, que gera inteiros entre `i` e `j` com igual probabilidade. Considere os três algoritmos:

a) Preencha um arranjo `A` de `A[0]` até `A[n-1]` como a seguir: Para preencher `A[i]` gere um número aleatório até encontrar um que não esteja em `A[0]`, `A[1]`, ..., `A[i-1]`.
b) Igual ao algoritmo (a), mas mantendo um outro arranjo chamado `Usados`. Quando um número aleatório `aleat` é gerado, antes de colocá-lo no arranjo `A` faça `Usados[aleat] = 1`. Isso significa que quando preenchemos o arranjo `A` com um número aleatório, pode-se testar em um único passo se o número já foi ou não usado, ao invés do algoritmo (a) que pode ter que testar `i` vezes.
c) Preencha o arranjo tal que `A[i] = i+1`. Então, `Para i ← 0 até n-1` faça `troque(A[i], gera_aleat(0,i))`.

De acordo com os algoritmos (a), (b) e (c):

- Verifique se os três algoritmos geram somente permutações corretas;
- Faça a análise assintótica do tempo estimado de cada um deles;
- Escreva programas para executar cada algoritmo 10 vezes para pegar uma boa estimativa. Rode o programa do algoritmo (a) para n = 250, 500, 1000, 2000. Rode o programa do algoritmo (b) para n = 2500, 5000, 10000, 20000, 40000, 80000. Rode o programa do algoritmo (c) para n = 10000, 20000, 40000, 80000, 160000, 320000, 640000;
- Compare suas análises com os tempos de execuções obtidos.

6. Qual o tempo requerido para calcular $f(x) = \sum_{i=0}^{n} a_i x^i$:

a) Usando uma rotina simples para calcular a exponenciação;
b) Usando a Regra de Hornes a seguir:

```
poli ← 0;
Para i ← n (passo −1) até 0
   Faça poli ← x * poli + A[i];
```

7. (a) É verdade que $2^{n+1} = O(2^n)$?

(b) É verdade que $2^{2n} = O(2^n)$?

8. Escreva um algoritmo para determinar se existe um inteiro `k` tal que `a[i] = k` num arranjo de inteiros `a[1] < a[2]< a[3] < ... < a[n]`. Faça a análise do seu algoritmo e verifique junto aos seus colegas se alguém conseguiu um algoritmo mais rápido.

9. Mostre que:

- *$1/2n(n+1)$ é $O(n^2)$;*
- *$n + \sqrt{n}$ é $O(n)$;*
- *$n/1000$ não é $O(1)$;*
- *$1/2n^2$ não é $O(n)$;*
- *$1/2n^2 - 3n$ é $O(n^2)$.*

10. Escreva um algoritmo e faça a análise para determinar se um inteiro positivo n é primo. Procure as diferentes implementações do algoritmo e compare com a sua.

11. Mostre que x^{62} pode ser calculado com apenas oito multiplicações.