MAC0122 – Princípios de Desenvolvimento de Algoritmos

IME - Segundo Semestre de 2007

Segunda Lista de Exercícios

Exercícios para a 2a prova

1. **Busca Binária**

   (a) Suponha que o vetor v tem 511 elementos e que x não está no vetor. Quantas comparações a função buscaBinaria fará entre x e elementos do vetor?

   (b) Se preciso de t segundos para fazer uma busca binária em um vetor com n elementos, de quando tempo preciso para fazer uma busca em n^2 elementos?

   (c) Confira a validade da seguinte afirmação: quando n+1 é uma potência de 2, a expressão $(e + d)$ é sempre divisível por 2, quaisquer que sejam v e x, onde e e d são os índices esquerdo e direito respectivamente, como visto no algoritmo visto em aula.

   (d) Escreva uma função que receba um vetor inteiro estritamente crescente v[0..n-1] e devolva um índice i entre 0 e n-1 tal que v[i] == i ; se tal i não existe, a função deve devolver -1. O seu algoritmo não deve fazer mais que lg(n) comparações envolvendo elementos de v.

   (e) Escreva uma função eficiente que receba inteiros positivos k e n e calcule k^n (k elevado a n). Quantas multiplicações sua função executa?

2. **Ordenação:**

   (a) escreva uma função em C que verifique se um vetor v[0..n-1] está em ordem crescente.

   (b) Escreva um função que ordene uma lista encadeada. Inspire-se no algoritmo de inserção para vetores. Faça duas versões: uma para lista com cabeça e outra para lista sem cabeça. (Sua função precisa devolver alguma coisa?).

   (c) Escreva um função para ordenar uma lista encadeada. Imite o algoritmo de seleção para vetores. Faça duas versões: uma para lista com cabeça e outra para lista sem cabeça. (Sua função precisa devolver alguma coisa?).

3. **Quicksort**

   (a) Escreva uma função que rearranje um vetor $v[p \ldots r]$ de inteiros de modo que tenhamos $v[p \ldots j-1] <= 0$ e $v[j \ldots r] > 0$ para algum j em $[p \ldots r+1]$. A expressão "$v[p \ldots j-1] <= 0$"significa $v[i] <= 0$ para todo i no conjunto $[p \ldots j-1]$.
   Faz sentido exigir que j esteja em $[p \ldots r]$?
   Não use vetor auxiliar. Procure fazer uma função rápida. Repita o exercício depois de trocar $v[j \ldots r] > 0$ por $v[j \ldots r] >= 0$.
   Faz sentido exigir que $v[j]$ seja 0?

   (b) Um vetor $v[p \ldots r]$ está arrumado se existe j em $[p \ldots r]$ tal que $v[p \ldots j-1] <= v[j] < v[j+1 \ldots r]$.
   Escreva um algoritmo que decida se $v[p \ldots r]$ está arrumado. Em caso afirmativo, o seu algoritmo deve devolver o valor de j.

   (c) Um programador inexperiente afirma que a seguinte implementação da função de separação rearranja o vetor $v[p \ldots r]$ e devolve um índice j em $p \ldots r-1$ tal que $v[p \ldots j] <= v[j+1 \ldots r]$.

```
int separa (int v[], int p, int r) {
   int j;
   j = (p + r) / 2;
   do {
```

```
      while (v[p] < v[j]) p++;
      while (v[r] > v[j]) r--;
      if (p <= r) {
         troca (v[p], v[r]);
         p++, r--;
      }
   } while (p <= r)
   return p;
}
```

Mostre um exemplo onde essa função não dá o resultado esperado. E se trocarmos "return p"por "return p-1"? É possível fazer algumas poucas correções de modo que a função dê o resultado esperado?

(d) Suponha dada uma lista encadeada que armazena números inteiros. Cada célula da lista tem a estrutura abaixo.

```
struct celula {
   int            chave;
   struct celula *prox;
};
```

Escreva uma função que transforme a lista em duas: a primeira contendo as células cuja chave é par e a segunda aquelas cuja chave é ímpar.

(e) escreva uma versão não recursiva do Quicksort.

4. **Mergesort**

(a) A função mergesort vista em aula chama as funções malloc e free muitas vezes (as chamadas acontecem dentro de intercala). Escreva uma versão da função que contenha o código da função de intercalação e chame malloc uma só vez.

(b) Escreva uma versão recursiva do algoritmo Mergesort que rearranje um vetor dado v[p . . . r-1] em ordem decrescente. Sua versão deve ser integrada com o código da intercalação, ou seja, deve incluir o código de intercalação. A intercalação deve começar com v[p . . . q-1] e v[q . . . r-1] decrescentes e terminar com v[p . . . r-1] decrescente.

5. **heapsort**

(a) Use o heapsort para ordenar o vetor

16	15	14	13	12	11	10	9	8	7	6	5	4

.

(b) Suponha que o vetor $v[1 \ldots n]$ é um max-heap. O seguinte fragmento de código rearranja o vetor em ordem crescente?

```
for (m = n; m >= 2; --m) {
   int x = v[1];
   for (j = 1; j < m; ++j) v[j] = v[j+1];
   v[m] = x;
}
```

6. **Árvores**

(a) Escreva uma função que calcule o número de células de uma árvore binária.

(b) Escreva uma função que imprima, em ordem e-r-d, as chaves das folhas de uma árvore binária.

(c) Dada uma árvore binária, encontrar uma célula da árvore cuja chave tenha um certo valor k.

(d) Escreva uma função que faça varredura r-e-d (= preorder traversal) de uma árvore binária.

(e) Escreva uma função que faça varredura e-d-r (= postorder traversal) de uma árvore binária.

(f) Escreva uma função que decida se uma dada árvore binária é ou não é de busca.

(g) Suponha dada uma árvore binária com a seguinte propriedade:
para cada célula x tem-se
$x \to$ esq $\to$ chave $<= x \to$ chave $<= x \to$ dir $\to$ chave. Essa árvore é de busca?

(h) Escreva uma função que transforme um vetor crescente em uma árvore de busca balanceada.