Universidade Federal de Minas Gerais
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Algoritmos e Estruturas de Dados II
2º Semestre de 2013

Lista de Exercícios: Busca e Ordenação

1 – Faça um programa que leia n nomes inserindo-os em uma lista de forma ordenada utilizando a ideia do **algoritmo da inserção**. No final, o programa deve mostrar todos os nomes ordenados alfabeticamente.

2 – Crie um programa que dado uma string, coloque as letras dela em ordem crescente pelo **algoritmo da bolha.**

3 – Faça um programa que leia n nomes e ordene-os pelo tamanho utilizando o **algoritmo da seleção**. No final, o algoritmo deve mostrar todos os nomes ordenados.

4 – Faça um programa que leia n nomes e ordene-os pelo tamanho utilizando o **algoritmo heap-sort**. No final, o algoritmo deve mostrar todos os nomes ordenados.

5 – Crie um programa que dado uma string, coloque as letras dela em ordem crescente pelo **algoritmo quick-sort.**

6 – Dados três vetores ordenados, implemente uma função que intercala e retorne o vetor resultante ordenado. Implemente uma função `merge3_sort`, que faça ordenação em um vetor utilizando a sua função de intercalação (sugestão: se baseie no algoritmo do merge-sort original). Qual a complexidade desse algoritmo?

7 – Considere a seguinte estrutura:

```
typedef struct _pessoa_ {
      unsigned int Matricula;
      float Nota;
} Aluno;
```

Faça uma função que, dado um valor inteiro N positivo, aloque dinamicamente um vetor de `Aluno`, leia do teclado os N pares de valores (Matricula, Nota) e retorne o vetor alocado. Imprima, **ao final do programa**, duas listas: (1) alunos ordenados por nota final e (2) alunos ordenados por número de matrícula. Utilize algoritmos de ordenação vistos em aula e justifique as suas escolhas.

8 – Faça uma tabela comparativa dos algoritmos vistos em aula considerando os seguintes aspectos: número de comparações (melhor, pior e caso médio), número de movimentações, facilidade de implementação, estabilidade e uso de memória auxiliar.

9 – Refaça as funções de busca sequencial e busca binária assumindo que o vetor possui chaves que podem ocorrer múltiplas vezes no vetor. Neste caso, você deve retornar, em um outro vetor, todas as posições onde a chave foi encontrada. Protótipo:

```
int busca(int vet[], int n, int chave, int posicoes[]);
```

Sua função deve retornar o número de ocorrências da chave no vetor e, para cada uma destas ocorrências, indicar no vetor `posicoes[]`, as posições de `vet` que possuem a `chave`.

10 – Desconsidere o conhecimento da função sqrt na linguagem C. Uma forma de se obter a raiz quadrada de um número qualquer x seria através de busca binária. Assuma que a raiz quadrada de x está entre 0 e x (Se o número for negativo, retorne 0). Para sabermos se um palpite y é a raiz quadrada de x, basta testar se y*y é próximo o suficiente de x ou, em outras palavras, se o módulo da diferença entre eles está dentro de uma tolerância definida. Caso contrário, podemos restringir a busca entre 0 e y ou entre y e x. Escreva a função abaixo que implemente este algoritmo, considerando 10^{-6} como tolerância para o cálculo do resultado.

```
double raiz(double x)
```