Universidade Federal do ABC
MCTA028-15 - Programação Estruturada
2018.Q3

Lista de Exercícios 7

Professores Emílio Francesquini e Carla Negri Lintzmayer

21 de novembro de 2018

1. Altere os algoritmos de ordenação vistos em aula para que eles ordenem um vetor de inteiros em ordem decrescente ao invés de ordem crescente.
2. Altere os algoritmos de ordenação vistos em aula para que eles lidem com números repetidos.
3. Considere os algorimos de ordenação vistos em aula (SelectionSort, InsertionSort, MergeSort e BubbleSort) e responda:
   (a) Quantos acessos ao vetor cada um dos algoritmos faz?
   (b) Quantas comparações entre elementos do vetor cada um dos algoritmos faz?
   (c) Quantas trocas de elementos do vetor cada um dos algortimos faz?
4. O SelectionSort visto em aula escolhe o menor elemento do vetor a partir da posição i e o troca pelo elemento na posição i para colocar o vetor em ordem crescente. Escreva uma variação do SelectionSort que em vez de colocar os menores elementos no início do vetor para deixá-lo em ordem, o faça colocando os maiores no final do vetor. Mais especificamente seu algoritmo deve encontrar o maior elemento do vetor até a posição $n - i - 1$ e colocá-lo na posição $n - i - 1$.
5. Considere a seguinte estrutura:

```c
struct aluno {
    unsigned int RA;
    double nota;
    char nome[50]
};
typedef struct aluno Aluno;
```

Faça uma programa que, dado um valor ‘n‘, aloque um vetor de ‘Aluno‘ dinamicamente e leia do teclado todas as informações sobre ‘n‘ alunos (nome, nota, RA). Imprima três listas de alunos, sendo cada linha de uma lista no formato "‘Nome - RA - Nota‘". A primeira lista deve estar ordenada de acordo com o nome. A segunda deve estar ordenada de acordo com o RA. A terceira deve estar ordenada de acordo com a nota.

6. [PF] Escreva uma função que verifica se um vetor ‘v[0..n-1]‘ está em ordem crescente.

7. [PF] A seguinte função promete verificar se um vetor ‘v[0..n-1]‘ está em ordem crescente. Ela funciona?

```c
int verifica(int v[], int n) {
    int sim;
    for (int i = 1; i < n; i++)
        if (v[i-1] <= v[i])
            sim = 1;
        else {
            sim = 0;
            break;
        }
    return sim;
}
```

8. [PF] Escreva uma versão recursiva do algoritmo Selection Sort.

9. [PF] Uma palavra é anagrama de outra se a sequência de letras de uma é permutação da sequência de letras da outra. Por exemplo, ‘aberto‘ é anagrama de ‘rebato‘. Dizemos que duas palavras são *equivalentes* se uma é anagrama da outra. Uma *classe de equivalência* de palavras é um conjunto de palavras duas a duas equivalentes. Escreva um programa que receba um arquivo ASCII contendo palavras (uma por linha) e extraia desse arquivo uma classe de equivalência máxima. Aplique o seu programa a um arquivo que contém todas as palavras do português brasileiro com diacríticos removidos. (O arquivo é grande; portanto, o seu programa deverá ser muito eficiente.)

10. Implemente todos os algoritmos vistos em aula e compare o desempenho entre eles. Para fazê-lo, utilize o código abaixo (baixe aqui) para gerar um vetor de tamanho n passado como parâmetro.

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
```

```c
#include <time.h>

void bubbleSort(int *v, int tam);
void selectionSort(int *v, int tam);
void insertionSort(int *v, int tam);
void mergeSort(int *v, int tam);

void geraVetor(int *v, int tamanho) {
    int i;
    srand(clock());
    for (i = 0; i < tamanho; i++)
        v[i] = rand();
}

int main () {
    int tamanho, *vetorOriginal, *vetorAuxiliar;
    clock_t inicio;
    double tempo;

    scanf("%d", &tamanho);
    vetorOriginal = malloc(sizeof(int) * tamanho);
    vetorAuxiliar = malloc(sizeof(int) * tamanho);

    geraVetor(vetorOriginal, tamanho);

    memcpy(vetorAuxiliar, vetorOriginal, sizeof(int) * tamanho);
    inicio = clock();
    bubbleSort(vetorAuxiliar, tamanho);
    tempo = (double)(clock() - inicio) / CLOCKS_PER_SEC;
    printf ("Tempo BubbleSort: %lf\n", tempo);

    memcpy(vetorAuxiliar, vetorOriginal, sizeof(int) * tamanho);
    inicio = clock();
    selectionSort(vetorAuxiliar, tamanho);
    tempo = (double)(clock() - inicio) / CLOCKS_PER_SEC;
    printf ("Tempo SelectionSort: %lf\n", tempo);

    memcpy(vetorAuxiliar, vetorOriginal, sizeof(int) * tamanho);
    inicio = clock();
    insertionSort(vetorAuxiliar, tamanho);
    tempo = (double)(clock() - inicio) / CLOCKS_PER_SEC;
    printf ("Tempo InsertionSort: %lf\n", tempo);

    memcpy(vetorAuxiliar, vetorOriginal, sizeof(int) * tamanho);
```

```
    inicio = clock();
    mergeSort(vetorAuxiliar, tamanho);
    tempo = (double)(clock() - inicio) / CLOCKS_PER_SEC;
    printf ("Tempo MergeSort: %lf\n", tempo);

    free(vetorAuxiliar);
    free(vetorOriginal);
    return 0;
}
```