Classificação por Troca - bubble sort

Existem outras formas de aprimorar o *bubble sort*.

Uma delas é fazer com que as passagens sucessivas percorram o vetor em sentidos opostos, de modo que os elementos de menor magnitude se desloquem mais rapidamente para o início da lista da mesma forma que os de maior magnitude se desloquem para o final (no caso de uma ordenação crescente).

Construa agora, como exercício, um módulo que implemente este método.

UNIVASF

```c
#include <stdio.h>
#define TAMANHO 100
void shaker_sort(int v[], int n) {
    int aux, esq, dir, i, trocou;
    esq = 0;
    dir = n - 1;
    do {
        trocou = 0;
        // Movimento da esquerda para a direita
        for (i = esq; i < dir; i++) {
            if (v[i] > v[i + 1]) {
                aux = v[i];
                v[i] = v[i + 1];
                v[i + 1] = aux;
                trocou = 1;
            }
        }
        dir = dir - 1;
```

UNIVASF

```
        if (trocou) {
            trocou = 0;
            // Movimento da direita para a esquerda
            for (i = dir; i > esq; i--) {
                if (v[i] < v[i - 1]) {
                    aux = v[i];
                    v[i] = v[i - 1];
                    v[i - 1] = aux;
                    trocou = 1;
                }
            }
        }
        esq = esq + 1;
    } while (esq < dir && trocou);
}
```

Classificação por Troca - shaker sort

Esta variante é conhecida como troca alternada ou *shaker sort,* devido a propiciar, por assim dizer, acomodação por “agitação” das chaves.

O ganho obtido se deve a que:

a) vão se formando dois subvetores ordenados que podem sair do processo de comparação e trocas;

b) para vetores pouco desordenados, as chaves menores e maiores são logo posicionadas, detectando-se em seguida o fim do processo.

UNIVASF

Classificação por Troca - shaker sort

De qualquer forma, o ganho é apenas em relação ao número de comparações: as trocas a serem efetuadas são sempre lado a lado, e todos os elementos terão que se movimentar no mesmo número de casas que no método *bubble sort*.

Como as comparações são menos onerosas que as trocas, estas continuam a ditar o desempenho.

UNIVASF

Classificação

Devemos ter em mente que os métodos de classificação podem ser aplicados não apenas sobre listas de inteiros, podemos aplicar sobre listas de reais, de caracteres, etc..

Outro detalhe a ser ressaltado é que estas listas podem ser constituídas não apenas por vetores de elementos dos tipos primitivos, ou seja, podemos classificar, por exemplo, uma lista de valores reais que se encontra armazenada nos elementos de um determinado campo dos registros contidos em um vetor de registros.

UNIVASF

Classificação

Exercício:

Defina um tipo de dado capaz de armazenar o nome, o número de inscrição e o percentual de acerto de um candidato a um concurso.

Em seguida construa um programa capaz de manipular uma lista com no máximo 100 registros de candidatos, onde cada registro é um elemento do tipo de dado definido. A manipulação da lista é feita através dos seguintes módulos: incluir candidato, excluir candidato, classificar lista com base no desempenho dos candidatos (implementar bubble sort), classificar lista com base no nome dos candidatos (implementar shaker sort), e imprimir lista. O programa deve se utilizar de forma satisfatória dos módulos mencionados.

UNIVASF

```c
#include <stdio.h>
#include <string.h>
#define MAX 100
typedef struct {
    char nome[100];
    int inscricao;
    float percentual_de_acerto;
} registro_candidato;
```

```c
void incluir_candidato(registro_candidato v[], int *n) {
    if (*n >= MAX) {
        printf("Lista cheia.\n");
        return;
    }
    printf("Nome: ");
    scanf(" %[^\n]", v[*n].nome);
    printf("Inscricao: ");
    scanf("%d", &v[*n].inscricao);
    printf("Percentual de acerto: ");
    scanf("%f", &v[*n].percentual_de_acerto);
    (*n)++;
    printf("Candidato incluido com sucesso!\n");
}
```

```c
void excluir_candidato(registro_candidato v[], int *n) {
    int inscricao, i, j;
    if (*n == 0) {
        printf("Lista vazia.\n");
        return;
    }
    printf("Informe o numero de inscricao do candidato a excluir: ");
    scanf("%d", &inscricao);
    for (i = 0; i < *n; i++) {
        if (v[i].inscricao == inscricao) {
            for (j = i; j < *n - 1; j++)
                v[j] = v[j + 1];
            (*n)--;
            printf("Candidato excluido com sucesso!\n");
            return;
        }
    }
    printf("Candidato nao encontrado.\n");
}
```

```c
void classificar_com_base_no_desempenho(registro_candidato v[], int n) {
    int i = 1, j;
    int ah_troca = 1;
    registro_candidato aux;
    while (i < n && ah_troca) {
        ah_troca = 0;
        for (j = 0; j < n - i; j++) {
            if (v[j].percentual_de_acerto < v[j + 1].percentual_de_acerto) {
                aux = v[j];
                v[j] = v[j + 1];
                v[j + 1] = aux;
                ah_troca = 1;
            }
        }
        i++;
    }
}
```

```c
void classificar_com_base_no_nome(registro_candidato v[], int n) {
    int esq = 0, dir = n - 1, i;
    int trocou = 0;
    registro_candidato aux;
    do {
        trocou = 0;
        for (i = esq; i < dir; i++)
            if (strcmp(v[i].nome, v[i + 1].nome) > 0) {
                aux = v[i];
                v[i] = v[i + 1];
                v[i + 1] = aux;
                trocou = 1;
            }
        dir--;
```

```
        if (trocou) {
            trocou = 0;
            for (i = dir; i > esq; i--)
                if (strcmp(v[i].nome, v[i - 1].nome) < 0) {
                    aux = v[i];
                    v[i] = v[i - 1];
                    v[i - 1] = aux;
                    trocou = 1;
                }
        }
        esq++;
    } while (esq < dir && trocou);
}
```

UNIVASF

```c
void imprimir_lista(registro_candidato v[], int n) {
    if (n == 0) {
        printf("Lista vazia.\n");
        return;
    }
    printf("\nLista de candidatos:\n");
    for (int i = 0; i < n; i++) {
        printf("Nome: %s\n", v[i].nome);
        printf("Inscricao: %d\n", v[i].inscricao);
        printf("Percentual de acerto: %.2f%%\n", v[i].percentual_de_acerto);
        printf("--------------------------\n");
    }
}
```

UNIVASF

```c
int main() {
    registro_candidato candidatos[MAX];
    int n = 0, opcao;
    do {
        printf("\nMENU:\n");
        printf("1 - Incluir candidato\n");
        printf("2 - Excluir candidato\n");
        printf("3 - Classificar por desempenho\n");
        printf("4 - Classificar por nome\n");
        printf("5 - Imprimir lista\n");
        printf("0 - Sair\n");
        printf("Escolha uma opcao: ");
        scanf("%d", &opcao);
        switch (opcao) {
            case 1: incluir_candidato(candidatos, &n); break;
            case 2: excluir_candidato(candidatos, &n); break;
            case 3: classificar_com_base_no_desempenho(candidatos, n); break;
            case 4: classificar_com_base_no_nome(candidatos, n); break;
            case 5: imprimir_lista(candidatos, n); break;
            case 0: printf("Encerrando programa.\n"); break;
            default: printf("Opcao invalida!\n");
        }
    } while (opcao != 0);
    return 0;
}
```

Classificação por Seleção - selection sort

Outro método, também simples, de ordenação é a ordenação por seleção.

Princípio de funcionamento:

1. Selecione o menor item do vetor (ou o maior).
2. Troque-o com o item que está na primeira posição do vetor.

Repita estas duas operações com os n-1 itens restantes, depois com os n-2 itens, até que reste apenas um elemento.

UNIVASF

Classificação por Seleção - selection sort

A ordenação por seleção consiste, em cada etapa, em selecionar o maior (ou o menor) elemento e colocá-lo em sua posição correta dentro da futura lista ordenada.

Durante a aplicação do método de seleção a lista com *n* registros fica decomposta em duas sub listas, uma contendo os itens já ordenados e a outra com os restantes ainda não ordenados.

- No início a sub lista ordenada é vazia e a outra contém todos os demais.
- No final do processo a sub lista ordenada apresentará (n-1) itens e a outra apenas 1.

UNIVASF

Classificação por Seleção - selection sort

As etapas (ou varreduras) como já descrito consistem em buscar o maior (ou menor) elemento da lista não ordenada e colocá-lo na lista ordenada.

Para uma melhor compreensão trabalharemos com um exemplo, visando demonstrar o resultado das etapas da ordenação de um vetor de inteiros, consideraremos o ordenamento crescente dos elementos e selecionaremos em cada etapa o maior elemento do subvetor não ordenado.

UNIVASF

Classificação por Seleção - selection sort

Resultado das Etapas					
Etapa	X[1]	X[2]	X[3]	X[4]	X[5]
0	5	9	1	4	3
1	{5	3	1	4}	{9}
2	{4	3	1}	{5	9}
3	{1	3}	{4	5	9}
4	{1}	{3	4	5	9}

Classificação por Seleção - selection sort

Exemplo:

	1	2	3	4	5	6
Chaves Iniciais	O	R	D	E	N	A
I=1	A	R	D	E	N	O
I =2	A	D	R	E	N	O
I=3	A	D	E	R	N	O
I=4	A	D	E	N	R	O
I=5	A	D	E	N	O	R

Nota: As Chaves em vermelho sofreram uma troca entre si

Classificação por Seleção - selection sort

Com base no que foi visto, construa um módulo, que receba como parâmetros um vetor de inteiros e o número de elementos neste vetor. Este módulo deve ordenar o vetor implementando a classificação por seleção.

UNIVASF