SEMANAS:
- INICIO/APRESENTAÇÃO
- INTRODUÇÃO AOS ALGORITMOS E ESTRUTURAS DE DADOS
- ESTRUTURAS DE CONTROLO DE FLUXO EM C#
- CONCEITO DE CLASSE E OBJETO
- SEPARAÇÃO ENTRE INTERFACE PÚBLICA E IMPLEMENTAÇÃO PRIVADA
- ESTRUTURAS DE DADOS – INTRODUÇÃO
- ANÁLISE DE ALGORITMOS E COMPLEXIDADE
- ALGORITMOS DE ORDENAÇÃO
- ALGORITMOS DE ORDENAÇÃO (PARTE 2)
- PROVA DE AVALIAÇÃO
- LISTAS E FILAS
- TABELAS DE HASH
- TRATAMENTO DE COLISÕES
- ÁRVORES BINÁRIAS
- TIPOS GENÉRICOS E TRATAMENTO DE EXCEPÇÕES
- PROVA DE AVALIAÇÃO

Algoritmos e Estruturas de Dados

CTeSP - Tecnologias e Programação de Sistemas de Informação 2020/21

Docente: Ricardo Henriques.

Sumário:

- Utilização de `List<T>` em `ContasBancarias`.
- Algoritmos de ordenação - definições
- Algoritmos:
  - `Selection Sort`
  - `Insertion Sort`
  - `Bubble Sort`
- Exemplo

Algoritmos de ordenação elementares

A maioria das modernas linguagens de programação dispõem já de eficientes implementações de ordenação. Daí que não seja consensual a relevância desta matéria. Contudo, não estudar os fundamentos do raciocínio de como se pensa um algoritmo, qual a estratégia na procura de melhores implementações, e dessa forma perceber os seus raciocínios, seria como um mecânico não querer saber como se constrói um motor, e quais as razões que têm levado à introdução das modernas opções tecnológicas. O resultado seria um mau mecânico... e nós, um mau programador!

Vamos num primeiro instante apresentar algumas definições gerais teóricas e posteriormente iremos prosseguir com o estudo da implementação de algumas soluções de ordenação mais conhecidas, para além da apresentada na aula passada.

Estável

*Um algoritmo de ordenação é dito **estável** se preserva a ordem relativa dos itens com chaves repetidas.*

Por exemplo, ordenar a lista de alunos por ano de graduação quando já estava ordenada alfabeticamente pelo nome:

- É possível estabilizar um algoritmo alterando a sua chave (tem um custo adicional).
- Algoritmos básicos são quase todos estáveis, mas poucos algoritmos avançados são estáveis.

Interno/Externo

*Um algoritmo de ordenação é dito **interno** se o conjunto de todos os dados a*

*ordenar estiver na memória (volátil); caso contrário diz-se **externo**.*

Direto/Indirecto

*Um algoritmo de ordenação diz-se **direto** se os dados são acedidos diretamente nas operações de comparação e troca; diz-se **indireto** caso contrário.*

Insertion Sort

A ideia por detrás deste é algoritmo passa por considerar os elementos um a um e inseri-los no seu lugar entre os elementos já tratados, mantendo a ordenação - exemplo: ordenar um baralho de cartas.

- Inserção implica arranjar um novo espaço ou seja mover um número elevado de elementos numa posição;
- Inserir os elementos um a um a começar pelo da 1º posição.
- Os elementos à esquerda do índice atual estão ordenados, mas não necessariamente na sua posição final - podem ainda ter de ser deslocados (para a direita) para dar lugar a elementos menores encontrados posteriormente.

```
public static class InsertionSort {
    1 reference
    public static string[] Sort(string[] vector) {
        if (vector != null && vector.Length > 1)
            for (int i = 1; i < vector.Length; i++) {
                for (int j = i; j > 0; j--)
                    MetodosAuxiliares.ComplexSwap(ref vector[j - 1], ref vector[j]);
            }
        return vector;
    }
}
```

InsertionSort

```
internal static void ComplexSwap(ref string a, ref string b) {
    if (Less(b, a))
        Swap(ref a, ref b);
}
internal static bool Less(string a, string b) {
    return a.CompareTo(b) < 0;
}
```

A implementação por inserção é ineficiente: o código é simples, mas pouco eficiente e pode ser melhorado. Representa pois aquilo que se deseja que retenha, sob a forma de pensar:

- Ilustra um bom raciocício;
- A solução encontrada é simples;
- É fácil estudar o seu funcionamento;
- Permite, através de pequenas transformações melhorar o desempenho.

Vejamos alguns pontos de trabalho:

- Existem demasiadas operações de comparação/troca (`complexswap`):
  - Pode travar-se o ciclo se encontramos uma chave que não é maior que a do item a ser inserido (a tabela está ordenada à esquerda) - pode-se sair do ciclo interno;
  - A modificação torna o algoritmo adaptativo;
  - Aumenta o desempenho aproximadamente por um factor de 2.
- Passa a haver duas condições para sair do ciclo:
  - Mudar para um ciclo `while`;
  - Procurar remover instruções irrelevantes - o `complexswap` não é o melhor processo de mover vários dados uma posição para a direita.

```
public static class InsertionSortEnhanced {
    public static string[] Sort(string[] vector) {
        for(int i = 0; i < vector.Length; i++) {
            string aux = vector[i];
            int j = i;
            while(j > 0 && MetodosAuxiliares.Less(aux, vector[j-1])) {
                vector[j] = vector[j - 1];
                j--;
            }
            vector[j] = aux;
        }
        return vector;
    }
}
```

InsertionSort (versão adaptativa)

Em resumo, no *Insertion Sort*:

- Usa aproximadamente $N^2/4$ comparações;
- Faz N^2 trocas (translações ou movimentos) no caso médio e o dobro destes valores no pior caso.

Bubble Sort

Trata-se do algoritmo mais utilizado e o que muitos programadores aprendem. A ideia por detrás é fazer múltiplas passagens pelos dados trocando de cada vez dois elementos adjacentes que estejam fora de ordem, até não haver mais trocas.

- É de implementação muito fácil;
- Na maioria dos casos é mais lento que os dois métodos anteriores.

```
public static string[] Sort(string[] vector) {
    for(int i = 0; i < vector.Length; i++)
        for(int j = vector.Length-1; j > i; j-- )
            MetodosAuxiliares.ComplexSwap(ref vector[j - 1], ref vector[j]);
    return vector;
}
```

BubbleSort

- Movendo da direita para a esquerda na entrada de dados
  - Quando o elemento mais pequeno é encontrado na primeira passagem 1) é

sucessivamente trocado com todos à sua esquerda 2) acaba por ficar na primeira posição.
    - Na segunda passagem o 2º elemento mais pequeno é colocado na sua posição e assim por diante;
    - ***N*** passagens pelos dados são suficientes.
- É semelhante ao método de seleção: mas tem maior esforço para colocar cada elemento na sua posição final - todas as trocas sucessivas até chegar à posição certa.

Em resumo, no *Bubble Sort*:

- Usa aproximadamente $N^2/2$ comparações e $N^2/2$ trocas
- A *i*-ésima passagem de *bubble sort* requer ***N-i*** operações de comparação/troca, logo é semelhante ao *selection sort*.

Exercício 7:

Considere o seguinte vector:

```
string[] nomes = { "Serafim", "Manuel", "Frederica", "Ana", "Luís", "Maria", "Carlos" };
```

1. Faça um *dry-running* da ordenação do vector para o algoritmo *Insertion Sort Enhanced*.
2. Faça um *dry-running* da ordenação do vector para o algoritmo *Bubble Sort*.

Sugestão: utilize uma folha de cálculo para cada *dry-running*.

Comparação entre os elementares algoritmos de ordenação

Confira, com base nos *dry-runnings*, a seguinte comparação de complexidade:

	Selection	Insertion	Bubble
Comparações (aprox.)	$N^2/2$	$N^2/4$	$N^2/2$
Trocas (aprox.)	N	$N^2/4$	$N^2/2$

(continua)

Exercício 8:

Deseja-se que cada grupo aplique os conceitos de ordenação apresentados através de uma boa consolidação prática da matéria.

Assim cada grupo deve usar como base o trabalho de grupo anterior (@Moodle: Trabalho de Grupo 2021 Abr 09) submetido. Deve ter por base a implementação do conjunto dos funcionários nas classes `Sistema` ou `Empresa` à custa de um *array*.

Nota: os grupos que não implementaram dessa maneira devem fazê-lo agora, sem excepção. Se tiverem dúvidas devem colocar na aula de apresentação do enunciado (2021 Abr 23).

Em avaliação sugere-se que siga os seguintes passos:

1. Escolha livre de dois dos três algoritmos apresentados:
    - `Selection Sort`
    - `Insertion Sort`
    - `Bubble Sort`
2. Estude bem como cada um se define.
3. Escreva dois métodos que devolvam (**sem modificar o *array* original**):
    - o conjunto dos funcionários ordenado pelo nome (pelo primeiro algoritmo que escolheram);
    - o conjunto dos funcionários ordenado pelo número de dependentes que tem (com recurso ao segundo algoritmo que escolheram).
    - majora-se a cotação para os grupos que ordenem internamente pelo nome nos casos em que o número de dependentes é o mesmo.

Os pormenores da entrega deste exercício serão indicados através da plataforma *moodle* associada à disciplina.

MRH, 2020/21