ENGENHARIA INFORMÁTICA E DE COMPUTADORES

Algoritmos e Estruturas de Dados

(parte 2 – Algoritmos de Ordenação Elementares)

2° Semestre 2022/2023
Instituto Superior de Engenharia de Lisboa
Paula Graça

ORDENAÇÃO

- Algoritmos de ordenação elementares
  - *Selection Sort, Insertion Sort, Bubble Sort*

- Utilização
  - Algoritmos adequados a entradas *N* de pequena dimensão
- Motivação
  - Se o número de itens a ordenar não é muito grande (alguns milhares), os algoritmos de ordenação mais sofisticados podem introduzir uma maior sobrecarga que os torna mais lentos
  - Em listas quase ordenadas ou com muitas chaves duplicadas, os algoritmos elementares são particularmente eficientes
  - São também utilizados para melhorar a eficiência de algoritmos mais complexos

BUBBLE SORT

- É o método mais usado pela sua simplicidade
- Designa-se por *Bubble Sort*, devido à perceção de que os itens se elevam como bolhas de ar
- O menos densos sobem mais
- Funcionamento
  - Consiste numa passagem sequencial na lista, começando pelo último elemento, trocando os elementos adjacentes que estejam fora de ordem
  - Por cada passagem, fica um elemento arrumado na posição final
  - O processo é repetido até a lista ficar toda ordenada

BUBBLE SORT

A S O R T I N G E X A M P L E

A A S O R T I N G E X E M P L

A A E S O R T I N G E X L M P

A A E E S O R T I N G L X M P

A A E E G S O R T I N L M X P

A A E E G I S O R T L N M P X

A A E E G I L S O R T M N P X

A A E E G I L M S O R T N P X

A A E E G I L M N S O R T P X

A A E E G I L M N O S P R T X

...

A A E E G I L M N O P R S T X

BUBBLE SORT

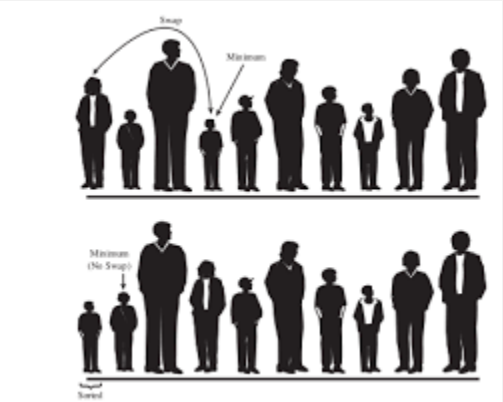
```
fun bubbleSort(table: IntArray, left: Int, right: Int) {
    for (i in left..right) {
        for (j in right downTo i + 1) if (table[j] < table[j - 1])
            exchange(table, j, j - 1)
    }
}
```

```
fun exchange(t: IntArray, i: Int, j: Int) {
    val x = t[i]
    t[i] = t[j]
    t[j] = x
}
```

Para uma entrada de N valores,
usa aproximadamente $N^2/2$ comparações e $N^2/2$ trocas no caso médio e pior caso

SELECTION SORT

- É o algoritmo mais simples



- Designa-se por *Selection Sort*, porque funciona pela seleção repetida do elemento menor de entre os restantes

- Funcionamento
  - É procurado primeiro o elemento menor na lista, sendo trocado com o elemento da primeira posição.
  - A seguir é procurado o segundo elemento menor, sendo trocado com o elemento da segunda posição
  - Assim sucessivamente até a lista estar toda ordenada

SELECTION SORT

- O processo de procurar em cada passagem o próximo elemento mais pequeno, não dá nenhuma informação sobre o elemento seguinte a procurar

- O tempo de ordenação é praticamente o mesmo no caso de a lista estar quase ordenada, tenha muitos elementos duplicados, ou arrumados aleatoriamente

- É o algoritmo de escolha para a ordenação de listas com elementos de dimensão muito grande, pois é o que implica menos deslocamentos

SELECTION SORT

A S O R T I N G E X A M P L E

A S O R T I N G E X A M P L E

A A O R T I N G E X S M P L E

A A E R T I N G O X S M P L E

A A E E T I N G O X S M P L R

A A E E G I N T O X S M P L R

A A E E G I N T O X S M P L R

A A E E G I L T O X S M P N R

A A E E G I L M O X S T P N R

A A E E G I L M N X S T P O R

...

A A E E G I L M N O P R S T X

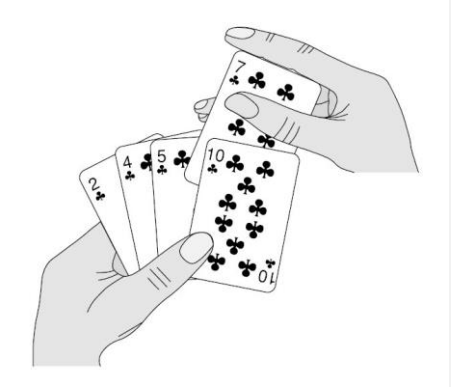
SELECTION SORT

```
fun selectionSort(table: IntArray, left: Int, right: Int) {
    for (i in left..right) {
        var min = i
        for (j in i + 1..right) if (table[j] < table[min]) min = j
            exchange(table, i , min)
    }
}
```

Para uma entrada de N valores
Usa aproximadamente $N^2/2$ comparações e N trocas em qualquer caso

INSERTION SORT

- Usado para ordenar uma mão de um jogo de cartas



- Designa-se por *Insertion Sort*, porque funciona pela inserção de cada elemento na posição correta, de entre os restantes já ordenados

- Funcionamento
  - Os elementos à esquerda do elemento corrente a colocar por ordem, estão sempre ordenados durante o processo, embora ainda não estejam na posição final. Os elementos à direita estão desordenados
  - Durante o processo, os elementos ordenados são movidos uma posição à direita de forma a permitir alojar o elemento corrente a ordenar

INSERTION SORT

- É o algoritmo mais rápido se os elementos da lista estiverem quase ordenados
- É utilizado para como base de ordenação de um algoritmo mais sofisticado designado por *Shell Sort*
- É também utilizado no algoritmo *Quick Sort*, quando as partições ficam suficientemente pequenas

INSERTION SORT

A S O R T I N G E X A M P L E

A S O R T I N G E X A M P L E

A O S R T I N G E X A M P L E

A O R S T I N G E X A M P L E

A O R S T I N G E X A M P L E

A I O R S T N G E X A M P L E

A I N O R S T G E X A M P L E

A G I N O R S T E X A M P L E

A E G I N O R S T X A M P L E

A E G I N O R S T X A M P L E

...

A A E E G I L M N O P R S T X

INSERTION SORT

```
fun insertionSort(table: IntArray, left: Int, right: Int) {
    for (i in left + 1..right)
        val curr = table[i]
        j = i
        while (j > 0 && curr < table[j - 1]) {
            table[j] = table[j – 1]
            j--
        }
        table[j] = curr
}
```

Para uma entrada de N valores,
usa aproximadamente $N^2/4$ comparações e trocas no caso médio e $N^2/2$ no pior caso

ALGORITMOS ADAPTATIVOS

- Os algoritmos de ordenação cujo tempo de execução depende da ordenação inicial dos elementos, designam-se por algoritmos de ordenação adaptativos

- Quais dos algoritmos de ordenação são adaptativos?
  - Bubble sort
  - Selection sort
  - Insertion sort

OTIMIZAÇÕES NOS ALGORITMOS

- *Bubble Sort adaptativo* (utiliza uma flag)
  - Durante o processo de ordenação, quando em determinada passagem pela lista a ordenar (ciclo de repetição interior), não existir nenhuma troca, quer dizer que a lista já está ordenada
  - O ciclo de repetição exterior pode ser então quebrado para terminar a ordenação

BUBBLE SORT (ADAPTATIVO)

```
fun bubbleSortAdaptive(table: IntArray, left: Int, right: Int) {
    var trocas = false
    for (i in left..right) {
        for (j in right downTo i + 1) if (table[j] < table[j - 1]) {
            exchange(table, j, j - 1)
            trocas = true
        }
        trocas = if (trocas) false else break
    }
}
```

Uso de uma flag que indica se houve trocas

Se flag=false, significa que não houve trocas, sendo o ciclo quebrado (break)

INSERTION SORT (ADAPTATIVO)

- *Insertion Sort* adaptativo (utiliza uma sentinela)
  - O algoritmo pode ser melhorado da seguinte forma
    - Durante o processo de ordenação, ao comparar o elemento corrente para o inserir na posição correta da lista, tem que também ser testada quando é que se atinge o início da lista (para não provocar um erro de execução ao aceder a um índice não válido)
    - Se for utilizada uma sentinela (menor valor no início da lista) poupa-se a comparação com o início da lista, pois todos os outros elementos ficarão após o menor

INSERTION SORT (ADAPTATIVO)

```
fun insertionSortAdaptive(table: IntArray, left: Int, right: Int) {
    for (i in right downTo left + 1)
        if (table[i] < table[i - 1]) exchange(table, i, i - 1)
    for (i in left + 2..right) {
        val curr = table[i]
        var j = i
        while (curr < table[j - 1]) {
            table[j] = table[j - 1]
            j--
        }
        table[j] = curr
    }
}
```

Uso de sentinela (menor elemento na 1ª posição da lista)

O uso de sentinela simplifica o teste das condições limite de um algoritmo

ALGORITMOS ESTÁVEIS

- Um algoritmo de ordenação estável mantém a ordem relativa dos elementos que, segundo o critério de comparação, são considerados iguais

- Quais dos algoritmos de ordenação são estáveis?
  - Bubble sort
  - Selection sort
  - Insertion sort