



Licenciatura em
Engenharia Informática

Programação I

Algoritmos de ordenação e pesquisa

Estrela Ferreira Cruz

1

Objetivos da aula

Objetivos da aula:

- Apresentação de alguns algoritmos de ordenação:
 - BubbleSort,
 - SelectionSort,
 - InsertionSort e
 - QuickSort
- Apresentação de algoritmos de pesquisa: pesquisa sequencial, ordenada e binária
- Apresentação de exemplos práticos

2

2

Algoritmos de ordenação

Suponha o seguinte array de valores inteiros:

val[0]	val[1]	val[2]	val[3]	val[4]	val[5]	val[6]	val[7]
70	25	34	15	5	60	3	55

Como colocava os valores do array por ordem crescente?

8

3

Algoritmos de ordenação

Um Algoritmo de ordenação é um algoritmo que coloca os elementos de uma dada sequência numa ordem específica.

As ordens mais usadas para as ordenações são a ordem numéricas e a ordem alfabética.

A principal razão para se ordenar uma sequência é o aumento da velocidade de acesso aos dados.

Os algoritmos de ordenação podem ser classificados em dois tipos:

Internos – ordenação de informação armazenada em sequências (arrays);

Externa – ordenação de informação armazenada em ficheiros.

4

4

Algoritmos de ordenação

Os algoritmos de ordenação **devem ser eficientes** quer em termos de **velocidade de execução** como em termos de **espaço ocupado**.

Existem vários algoritmos de ordenação:

- Uns de **implementação simples**, outros de implementação mais complexa.
- Uns **mais eficientes** para sequências de grandes dimensões, outros mais eficientes para sequências de pequenas dimensões.
- Alguns algoritmos de ordenação (os mais simples) **ordenam os elementos no seu próprio vetor** (array), fazendo para isso um rearranjo interno dos seus elementos. Outros usam vetores auxiliares.

5

5

Algoritmos de ordenação

Os algoritmos de ordenação mais usados são:

- Bubble sort
- Selection sort
- Insertion sort
- Quick sort
- Shell sort
- Merge sort
- Radix sort
- Shaker sort
- ...

6

6

Algoritmos de ordenação

A descrição dos algoritmos de ordenação que usam a própria sequência para a ordenação assentam nos seguintes pressupostos:

- A entrada é um vetor (array) cujos elementos precisam ser ordenados
- A saída é o mesmo vetor com os seus elementos ordenados
- O espaço que pode ser utilizado é o espaço do próprio vetor, por vezes, com auxílio de variáveis temporárias auxiliares.

7

7

Algoritmos de ordenação

Bubble sort

- Bubble sort ou ordenação “bolha”, é um algoritmo de ordenação simples. O seu nome deve-se ao facto de os elementos maiores subirem para as suas posições corretas formando uma imagem tipo “bolha”.
- A estratégia do algoritmo é a seguinte:
 - Percorre-se o vetor da esquerda para a direita comparando os elementos consecutivos e trocando os elementos que estão fora de ordem.
 - Desta forma garante-se que o elemento com maior valor será levado para a última posição do vetor.
 - Repete-se o processo até que o vetor fique ordenado (ou quantas vezes quanto o número de elementos do array).

8

8

Algoritmos de ordenação

Bubble sort – colocar por ordem **crescente**

val[0] val[1] val[2] val[3] val[4] val[5] val[6] val[7]

25	34	15	5	60	3	55	70
----	----	----	---	----	---	----	----

```
int i,aux=0;
.....
for (i=0; i<qtd-1; i++) {
    if( val[i]>val[i+1]) {
        aux=val[i];
        val[i]=val[i+1];
        val[i+1]=aux;
    }
}
```

9

9

Algoritmos de ordenação

Bubble sort

val[0] val[1] val[2] val[3] val[4] val[5] val[6] val[7]

25	34	15	5	60	3	55	70
----	----	----	---	----	---	----	----

25	34	15	5	60	3	55	70
----	----	----	---	----	---	----	----

Iteração 1

```
int i,aux=0;
.....
for (i=0; i<qtd-1; i++) {
    if( val[i]>val[i+1]) {
        aux=val[i];
        val[i]=val[i+1];
        val[i+1]=aux;
    }
}
```

10

10

Algoritmos de ordenação

val[0]	val[1]	val[2]	val[3]	val[4]	val[5]	val[6]	val[7]	
70	25	34	15	5	60	3	55	Iteração 1
25	34	15	5	60	3	55	70	Iteração 2
25	15	5	34	3	55	60	70	Iteração 3
15	5	25	3	34	55	60	70	Iteração 4
5	15	3	25	34	55	60	70	Iteração 5
5	3	15	25	34	55	60	70	Iteração 6
3	5	15	25	34	55	60	70	Iteração 7
3	5	15	25	34	55	60	70	

11

11

Algoritmos de ordenação

Função que implementa o algoritmo **bubble sort**.

A função seguinte ordena um array v (recebido como parâmetro na função) por ordem crescente.

```
int bubbleSort(int val[], int qtd) {
    int i=0, x=0, aux=0;
    for(x=0; x<qtd; x++) {
        for (i=0; i<qtd-1; i++) {
            if( val[i]>val[i+1]) {
                aux=val[i];
                val[i]=val[i+1];
                val[i+1]=aux;
            }
        }
    }
    return 0;
}
```

12

12

Algoritmos de ordenação

A função pode ser usada para ordenar qualquer array de valores inteiros.

```
int main() {
    int vect[50], i=0;
    printf("introduza 50 numeros inteiros:");
    for(i=0; i<50; i++) {
        scanf("%d",&vect[i]);
    }
    bubbleSort(vect,50);
    printf("Numero ordenados:\n");
    for(i=0; i<50; i++) {
        printf("%d\n",vect[i]);
    }
    return 0;
}
```

13

13

Algoritmos de ordenação

Exercícios:

1. Implemente uma função que recebe como parâmetro um array de nomes (com 100 caracteres cada string) e a quantidade de nomes e coloca os nomes por ordem alfabética.
2. Implemente um programa que receba do utilizador o nome de 10 capitais europeias, invoque a função criada anteriormente e apresenta o resultado para o ecrã.

14

14

Algoritmos de ordenação

A função pode ser usada para ordenar um array de strings.

```
void bubbleSort(char nomes[][100], int qtd) {
    int x=0,j=0;
    char temp[100];
    for (x=0; x < qtd; x++) {
        for (j=0; j < qtd-1 ; j++) {
            if (strcmp(nomes[j],nomes[j+1]) > 0) {
                strcpy(temp,nomes[j]);
                strcpy(nomes[j],nomes[j+1]);
                strcpy(nomes[j+1],temp);
            }
        }
    }
}
```

15

15

Algoritmos de ordenação

O programa que recebe o nome das cidades, invoca a função que ordena o array e imprime para o ecrã o array ordenado:

```
int main() {
    char nomes[10][100]; //vector de 10 strings
    int i=0;
    for(i=0; i < 10;i++) {
        printf("Introduza o nome de uma capital europeia:\n");
        gets (nomes[i]);
    }
    bubbleSort(nomes,10);
    printf("Nomes das capitais ordenados:\n");
    for(i=0; i < 10; i++) {
        printf("%s\n",nomes[i]);
    }
    return 0;
}
```

16

16

Algoritmos de ordenação e pesquisa

- Para a ordenação bubble, o número de comparações é sempre o mesmo porque os dois ciclos serão repetidos um número determinado de vezes, estando ou não a lista inicialmente ordenada. **Para um vetor com n elementos, o algoritmo de bubble sort executa sempre n-1 passagens pelos elementos do vetor.**
- O algoritmo de bubble sort apresenta uma variante que evita a execução de todas as passagens (n-1) dos elementos do vetor sempre que este fique ordenado prematuramente.
- Para evitar que o processo continue mesmo depois do vetor estar ordenado, o algoritmo bubble sort modificado (ou otimizado) interrompe o processo quando houver uma passagem inteira sem trocas.
- Para isso usa uma variável para verificar se existe ou não trocas, como se pode ver no diapositivo seguinte.

17

17

Algoritmos de ordenação

Função que implementa o algoritmo bubble sort otimizado.

```
int bubbleSortOpt (int v[], int qtd) {
    int x, j, troca=0, aux=0;

    for(x=0; x<qtd && troca==0; x++) {
        troca=1;
        for (j=0; j<qtd-1-x; j++) {
            if( v[j]>v[j+1]) {
                aux=v[j];
                v[j]=v[j+1];
                v[j+1]=aux;
                troca=0;
            }
        }
    }
    return 0;
}
```

18

18

Algoritmos de ordenação

Função que implementa outra versão do algoritmo bubble sort otimizado.

```
int bubble_opt(int v[], int qtd) {
    int i, trocou=0, aux=0;
    do {
        qtd--;
        trocou = 0;
        for(i=0; i<qtd; i++) {
            if( v[i]>v[i+1]) {
                aux=v[i];
                v[i]=v[i+1];
                v[i+1]=aux;
                trocou = 1;
            }
        }
    } while(trocou==1);
    return 0;
}
```

19

19

Algoritmos de ordenação

O programa seguinte lê do teclado os valores a ordenar, invoca a função de ordenação e apresenta os valores ordenados para o ecrã.

```
int main() {
    int vect[20], temp=0, i;
    printf("Digite 20 numeros inteiros:");
    for(i=0; i<20; i++) {
        scanf("%d",&vect[i]);
    }
    bubble_opt(vect, 20);
    printf("Vector ordenado:\n");
    for(i=0; i<20; i++) {
        printf("Na posição %d fica o valor %d\n",i+1,vect[i]);
    }
    return 0;
}
```

20

20

Algoritmos de ordenação

Exercícios:

1. Implemente uma função que recebe como parâmetro um array de inteiros e a respetiva quantidade valores e coloca os valores por ordem crescente, recorrendo ao uso do algoritmo **bubblesort** otimizado.
2. Implemente um programa que receba do utilizador os valores, invoque a função criada anteriormente e apresenta o resultado para o ecrã.

21

21

Algoritmos de ordenação

O algoritmo **selection Sort** ou ordenação por seleção, consiste em:

- Seleciona-se repetidamente o menor elemento do vetor e coloca-o na sua posição correta dentro do futuro vetor ordenado.

Ou seja, o algoritmo **selection Sort**:

- Seleciona, na primeira iteração, o elemento de menor valor e troca-o com o primeiro elemento.
- Repete o processo para os $N-1$ elementos restantes: o elemento com o menor valor é encontrado e trocado com o segundo elemento, e assim sucessivamente até aos dois últimos elementos.

22

22

Algoritmos de ordenação

O algoritmo selection Sort:

- Durante a aplicação deste algoritmo, um vetor de N elementos fica decomposto em dois (sub)vetores: um que contém os **elementos já ordenados** e outro que contém os restantes, **ainda não ordenados**.
- De início o (sub)vetor ordenado estará **vazio**, o outro (sub)vetor conterá os elementos na ordem original.
- No final, o (sub)vetor ordenado conterá N-1 elementos ordenados, o outro (sub)vetor conterá apenas um valor.

23

23

Algoritmos de ordenação

 Mínimo
 Valores já ordenados

selection sort – colocar por ordem **crescente**

val[0]	val[1]	val[2]	val[3]	val[4]	val[5]	val[6]	val[7]
70	25	34	15	5	60	3	55
3	25	34	15	5	60	70	55
3	5	34	15	25	60	70	55
3	5	15	34	25	60	70	55
3	5	15	25	34	60	70	55
3	5	15	25	34	60	70	55
3	5	15	25	34	55	70	60
3	5	15	25	34	55	60	70

24

24

Algoritmos de ordenação

Obter a posição do valor mínimo da parte do array que falta ordenar. → a posição **i** é a posição que está a ser tratada

```
int j=0, min=0,
for (j=i;j<qtd;j++) {
    if(v[j]<v[min]) {
        min=j;
    }
}
```

Trocar o menor para a posição correta → **i**

```
if(min!=i){
    aux=v[i];
    v[i]=v[min];
    v[min]=aux;
}
```

25

25

Algoritmos de ordenação

Função que implementa o algoritmo selection sort:

```
int selectionSort(int v[], int qtd) {
    int i=0, j=0, min=0, aux=0;
    for(i=0; i<qtd-1; i++) {
        min=i; // procura a posição do menor valor
        for (j=i;j<qtd;j++) {
            if(v[j]<v[min])
                min=j;
        }
        if(min!=i){ // troca: coloca-o na posição correta
            aux=v[i];
            v[i]=v[min];
            v[min]=aux;
        }
    }
    return 0;
}
```

26

26

Algoritmos de ordenação

Exercícios:

1. Implemente uma função que recebe como parâmetro um array de inteiros e a respetiva quantidade valores e coloca os valores por ordem crescente, recorrendo ao uso do algoritmo **selectionsort**.
2. Implemente um programa que receba do utilizador os valores, invoque a função criada anteriormente e apresenta o resultado para o ecrã.

27

27

Algoritmos de ordenação

O algoritmo **Quick Sort** ou ordenação rápida, tal como o bubble sort é algoritmo de ordenação por troca.

A estratégia de funcionamento é a seguinte:

- **Seleciona um elemento pivot**, normalmente o último, o primeiro ou o elemento do meio do vetor;
- **Coloca o elemento selecionado para pivot na sua posição correta**, ou seja, a sua posição definitiva no vetor.
- **Coloca à esquerda do elemento pivot os elementos menores que este**, e **à direita os elementos maiores que este**. Assim, supondo que o elemento pivot fica na posição x , todos os elementos entre 0 e $x-1$ são menores que o pivot e todos os elementos entre $x+1$ e n (n º de elementos do vetor) são maiores de o pivot.
- Em seguida **repete-se o procedimento para cada uma das partes do vetor** (esquerda e direita), usando para isso a recursividade.

28

28

Algoritmos de ordenação

Função que implementa o algoritmo quick sort recursivo:

```
void quicksort(int v[], int left, int right) {
    int i,j,p=0,aux=0;
    i = left; j = right;
    p = v[(left+right)/2];
    do {
        while(v[i] < p && i < right) { i++; }
        while(p < v[j] && j > left) { j--; }
        if (i <= j) { //troca
            aux = v[i];
            v[i] = v[j];
            v[j] = aux;
            i++;
            j--;
        }
    } while(i <= j);
    if(left < j) quicksort(v, left, j);
    if(i < right) quicksort(v, i, right);
}
```

29

29

Algoritmos de ordenação

O programa seguinte lê do teclado os valores a ordenar, invoca a função de ordenação e apresenta os valores ordenados para o ecrã.

```
int main() {
    int vect[20], temp=0, i;
    printf("Digite 20 numeros inteiros:");
    for(i=0; i<20; i++) {
        scanf("%d",&vect[i]);
    }
    quicksort(vect,0,19);
    printf("Vector ordenado:\n");
    for(i=0; i<20; i++) {
        printf("Na posição %d fica o valor %d\n",i+1,vect[i]);
    }
    return 0;
}
```

30

30

Algoritmos de ordenação

Exercícios:

1. Implemente uma função que recebe como parâmetro um array de inteiros e a respetiva quantidade valores e coloca os valores por ordem crescente, recorrendo ao uso do algoritmo **quicksort**.
2. Implemente um programa que receba do utilizador os valores, invoque a função criada anteriormente e apresenta o resultado para o ecrã.

31