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?

3

3

Algoritmos de ordenação

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

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 auxilio de **variáveis temporárias auxiliares**.

7

7

Algoritmos de ordenação

Bubble sort

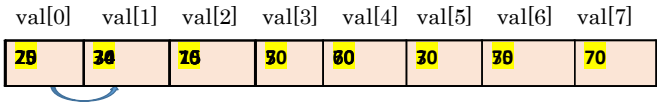
- Bubble sort ou ordenação "bolha", é **um algoritmo de ordenação simples**. O seu nome deve-se ao facto de os elementos maiores subirem para as suas posições corretas formando uma imagem tipo "bolha".
- A estratégia do algoritmo é a seguinte:
  - **Percorre-se o vetor da esquerda para a direita comparando 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**



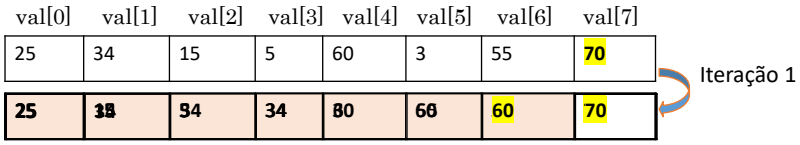

```
int i,aux=0;
.....
for (i=0; i<qtd-1; i++) {
        if( val[i]>val[i+1]) {
                aux=val[i];
                val[i]=val[i+1];
                val[i+1]=aux;
        }
}
```

9

9

Algoritmos de ordenação

Bubble sort

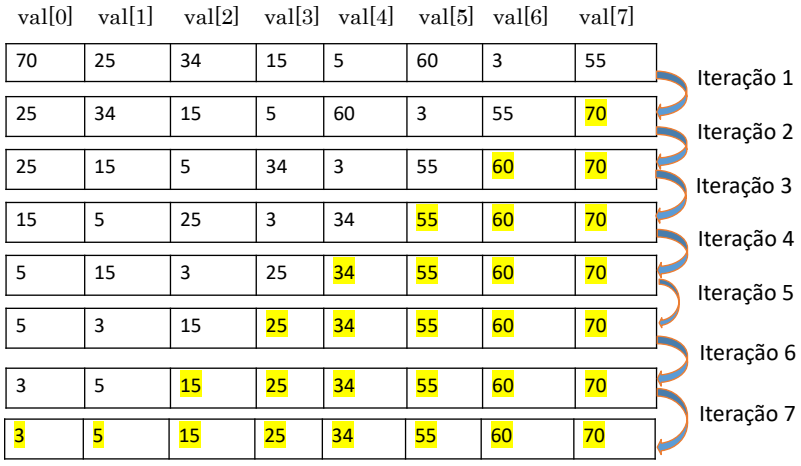



```
int i,aux=0;
.....
for (i=0; i<qtd-1; i++) {
        if( val[i]>val[i+1]) {
                aux=val[i];
                val[i]=val[i+1];
                val[i+1]=aux;
        }
}
```

10

10

Algoritmos de ordenação



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 carateres 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 apresentar 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 bubllesort 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

Minimo

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=i,
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

Exercícios:

1. Implemente uma função que recebe como parâmetro um array de strings (max 100 caracteres) e a respetiva quantidade e coloca as strings por ordem alfabética, recorrendo ao uso do algoritmo selectionsort.
2. Implemente um programa que receba do utilizador o nome de 10 pessoas, invoque a função criada anteriormente e apresenta o resultado para o ecrã.

28

28

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 (nº 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.

29

29

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);
}
```

30

30

Algoritmos de ordenação

O programa seguinte lê do teclado os valores a ordenar, invoca a função de ordenação e apresentar 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;
}
```

31

31

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.

32

32

Algoritmos de ordenação

O algoritmo Insertion sort ou ordenação por inserção usa a seguinte esquema de ordenação:

- Ordena inicialmente os dois primeiros elementos do array.
- De seguida, é inserido o terceiro elemento na posição ordenada em relação aos dois primeiros elementos.
- O quarto elemento é inserido na lista dos três elementos e assim sucessivamente até que todos os elementos tenham sido ordenados.

33

33

Algoritmos de ordenação

Valor a tratar

Valores já ordenados

Insertion sort – 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
25	70	34	15	5	60	3	55
25	34	70	15	5	60	3	55
15	25	34	70	5	60	3	55
5	15	25	34	70	60	3	55
5	15	25	34	60	70	3	55
3	5	15	25	34	60	70	55
3	5	15	25	34	55	60	70

34

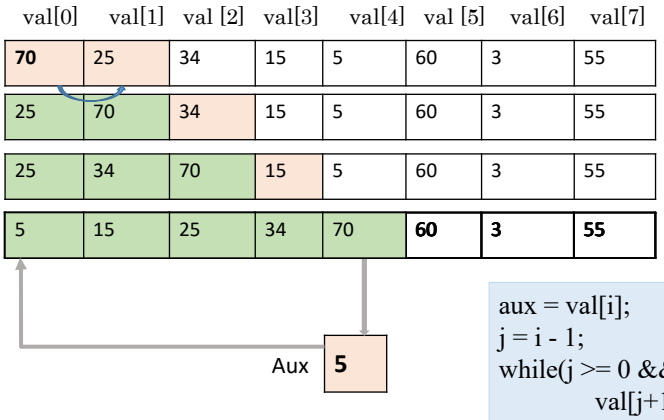
34

Algoritmos de ordenação

Valor a tratar

Valores já ordenados

Insertion sort




```
aux = val[i];
j = i - 1;
while(j >= 0 && val[j]>aux) {
        val[j+1] = val[j];
        j--;
}
val[j+1] = aux;
```

35

Algoritmos de ordenação

Função que implementa o algoritmo insertion sort:

```
void insertionSort(int val[], int tam) {
  int i=0, j=0, aux=0;
  for(i=1;i<tam;i++) {
        aux = val[i];
        j = i - 1;
        while(j >= 0 && val[j]>aux) {
              val[j+1] = val[j];
              j--;
        }
        val[j+1] = aux;
  }
}
```

36

36

Algoritmos de ordenação

O programa seguinte recebe do teclado 16 valores inteiros, invoca a função insertionSort e apresenta os valores ordenados para o ecrã.

```
int main() {
int vect[16], i=0;
printf("Introduza 16 numeros inteiros:");
for(i=0; i<16; i++) {
        scanf("%d",&vect[i]);
}
insertionSort(vect,16);
printf("Valores ordenados:\n");
for(i=0; i<16; i++) {
            printf("Na posição %d fica o valor %d\n",i+1,vect[i]);
 }
return 0;
}
```

37

37

Algoritmos de ordenação

Função que implementa o algoritmo insertion sort () que ordena um array de strings (nomes)

```
void insertionSort(char nomes[][100], int tam){
    int i=0,j=0;
    char aux[100];
    for(i=1; i<tam;i++){
        strcpy(aux,nomes[i]);
        j=i-1;
        while(j>=0 && strcasecmp(aux, nomes[j])<0){
            strcpy(nomes[j+1], nomes[j]);
            j--;
        }
        strcpy(nomes[j+1], aux);
    }
}
```

38

38

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 insertionSort.
2. Implemente um programa que receba do utilizador os valores, invoque a função criada anteriormente e apresenta o resultado para o ecrã.

39

39

Algoritmos de pesquisa

Algoritmos de pesquisa

- Sabendo que a operação de pesquisa de um elemento é uma operação muito frequente nas aplicações informáticas, é de extrema importância a escolha da estratégia mais eficiente para a efetuar, principalmente quando envolve um grande volume de dados.
- Para uma primeira abordagem considera-se que os dados armazenados estão desordenados.
- Posteriormente, serão consideradas estruturas com dados ordenados.

40

40

Algoritmos de ordenação

Pesquisa sequencial linear

val[0]	val[1]	val[2]	val[3]	val[4]	val[5]	val[6]	val[7]
89	**22**	**40**	15	**20**	**5**	**33**	**7**

É = 15?

Existe na posição 4

41

41

Algoritmos de ordenação

Pesquisa sequencial linear

val[0]	val[1]	val[2]	val[3]	val[4]	val[5]	val[6]	val[7]
89	**22**	**40**	**15**	**20**	**5**	33	7

É = 17?

Não existe

42

42

Algoritmos de pesquisa

Pesquisa sequencial (linear)

- É a forma mais simples e óbvia de pesquisa num vetor e consiste em percorrer o vetor, elemento a elemento, verificando se o valor objeto de procura é igual ao elemento do vetor.
- Se chegar ao fim do vetor sem encontrar o elemento é sinal que o elemento não existe.
- O numero máximo de iterações é o numero de elementos do vetor.

```
int pesquisaSeq(int v[], int tam, int x) {
int i;
for(i=0;i<tam;i++) {
        if (v[i] == x)  { return(i+1); }
}
return -1;
}
```

43

43

Algoritmos de pesquisa

Teste o algoritmo anterior pedindo os valores ao utilizador, bem como o elemento a procurar.

```
int main() {
int vect[20],i, aux=0, elem;
printf("Digite 20 numeros inteiros:");
for(i=0; i<20; i++) {
    scanf("%d",&vect[i]);
}
printf("Introduza o elemento a procurar:\n");
scanf("%d",&elem);
aux=pesquisaSeq(vect,20,elem);
if (aux==-1) {
        printf("Não encontrou o elementos\n");
}
else {
        printf("O elemento encontra-se na posição %d\n", aux);
}
return 0;
}
```

44

Algoritmos de pesquisa

Algoritmos de pesquisa linear ordenada:

- Se os elementos do vetor estiverem **ordenados**, é possível melhorar o algoritmo de pesquisa.
- Se o elemento a procurar não existir no vetor, e este estiver ordenado por ordem crescente, **logo que se encontre um elemento maior pode-se terminar a procura**, evitando assim percorrer todo o vetor.
- Para estes casos, o algoritmo que se apresenta a seguir é mais eficiente.

45

45

Algoritmos de ordenação

Pesquisa sequencial ordenada

val[0]	val[1]	val[2]	val[3]	val[4]	val[5]	val[6]	val[7]
2	**5**	**10**	**15**	**20**	**22**	**33**	40

É = 17?

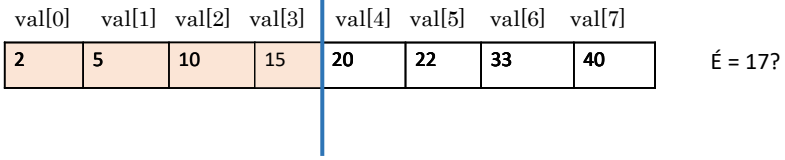
Não existe!

46

46

Algoritmos de ordenação

Pesquisa sequencial ordenada



Não existe!

47

47

Algoritmos de pesquisa

Pesquisa linear ordenada: a função seguinte verifica se o numero passado no parâmetro (x) se encontra no array v (1º parâmetro da função).

```
int pesquisaOrdenada(int v[], int tam, int x) {
int i;
for(i=0;i<tam && v[i]<=x;i++) {
        if (v[i] == x) {
                return(i+1);
        }
}
return -1;
}
```

48

48

Algoritmos de pesquisa

O programa seguinte testa o algoritmo anterior, pedindo os valores ao utilizador, bem como o elemento a procurar.

```
int main() {
int vect[20],i, aux=0, elem;
printf("Digite 20 numeros inteiros:");
for(i=0; i<20; i++) {
      scanf("%d",&vect[i]);
}
printf("Introduza o elemento a procurar:\n");
scanf("%d",&elem);
bublleSort(vect,20);
aux=pesquisaOrdenada(vect,20,elem);
if (aux==-1) {
    printf("Não encontrou o elementos\n");
}
else {
   printf("O elemento encontra-se na posição %d\n", aux);
}
return 0;
}
```

49

Algoritmos de pesquisa

Algoritmos de pesquisa binária:

Nos casos em que os elementos do vetor estão ordenados, é possível aplicar um algoritmo bem mais eficiente para efetuar a pesquisa, denominado de algoritmo de pesquisa binária.

Este algoritmo usa a seguinte estratégia:

- Divide o vetor a meio.
- Se o elemento a procurar for o elemento do meio,
  - então está encontrado, termina a procura.
- senão,
- se o elemento a procurar é menor que elemento do meio,
  - então procura no sub-vetor da direita,
  - senão procura no sub-vetor da esquerda.

50

50

Algoritmos de ordenação

O 33 existe?

Algoritmo pesquisa binaria

val[0]	val[1]	val[2]	val[3]	val[4]	val[5]	val[6]	val[7]
2	5	10	15	20	22	33	40

É o 15?
É maior
ou menor?

2	5	10	15	20	22	33	40

É o 22?
É maior
ou menor?

2	5	10	15	20	22	33	40

É o 33?

Sim --- 33!!

51

51

Algoritmos de pesquisa

Função que implementa o algoritmo de pesquisa binária:

```
int pesquisaBin(int v[], int tam, int x) {
int meio=0, ini=0, fim=tam-1;   //fim fica com o último índice
while (ini <= fim) {
        meio=(fim+ini)/2;
        if (x==v[meio]) {
                return (meio+1);
        }
        if (x < v[meio]) {
                fim=meio-1;   // pesquisa na parte esquerda
        }
        else {
                ini=meio+1;   // pesquisa na parte direita
        }
}
return -1;
}
```

52

52

Algoritmos de pesquisa

O programa seguinte testa o algoritmo de pesquisa binária:

```
int main() {
int vect[18]={3,5,7,9,23,45,67,89,90,120, 122, 124,128,130, 140,150,155,160};
int aux=0, elem;
printf("Introduza o elemento a procurar\n");
scanf("%d",&elem);
aux=pesquisaBin (vect,18,elem);
if (aux==-1) {
        printf("Não encontrou o elemento\n");
}
else {
        printf("O elemento encontra-se na posição %d\n", aux);
}
return 0;
}
```

53

53

Algoritmos de pesquisa

Função que implementa o algoritmo de pesquisa binária recursivo:

```
int pesquisaBinRec(int v[], int ini, int fim, int x) {
int meio=0;
if (ini>fim) return -1;
meio=(ini+fim)/2;
if (x==v[meio]) {       // encontrou o elemento a pesquisar
    return (meio+1);
}
if (x < v[meio]) {
     pesquisaBinRec (v, ini, meio-1, x);  // pesquisa na parte direita
}
else {
    pesquisaBinRec (v, meio+1, fim, x);  // pesquisa na parte esquerda
}
}
```

54

54

Algoritmos de pesquisa

O programa seguinte testa o algoritmo de pesquisa binária:

```
int main() {
  int vect[20]={1,3,5,7,8, 9,20,30,45,67,90,120, 122, 124,128,130, 140,150,155,160};
  int aux=0, elem;
  printf("Introduza o elemento a procurar\n");
  scanf("%d",&elem);
  aux=pesquisaBinRec (vect,0,19,elem);
  if (aux==-1) {
            printf("Não encontrou o elemento\n");
   }
  else {
            printf("O elemento encontra-se na posição %d\n", aux);
   }
   return 0;
}
```

55

55

Algoritmos de pesquisa

Algoritmos de pesquisa binária:

O numero de comparações efetuadas por este algoritmo é substancialmente menor que nos anteriormente apresentados, visto que cada iteração reduz para metade o conjunto de valores a pesquisar.

Alguns dados:

- Num conjunto de 100 elementos são necessárias, no máximo, 6 interações;
- Num conjunto de 10 000 elementos são necessárias, no máximo, 13 interações;
- Num conjunto de 1 000 000 elementos são necessárias, no máximo, 19 interações;
- Num conjunto de 1 000 000 000 elementos são necessárias, no máximo, 29 interações;

56

56

Algoritmos de ordenação

Exercícios:

1. Implemente uma função que recebe como parâmetro um array de nomes e a respetiva quantidade e ordena alfabeticamente os nomes.
2. Implemente o algoritmos de pesquisa binária, que verifica se um determinado nome recebido como parâmetro existe no array de nomes.
3. Implemente um programa que receba do utilizador os nomes, invoque a função de ordenação criada anteriormente e apresenta o resultado para o ecrã.
4. Depois de ordenar os nomes, deve pedir ao utilizador um nome e, recorrendo ao algoritmo de pesquisa binária, verificar se o nome existe no array.

57

57

Bibliografia

- Programação Avançada Usando C, António Manuel Adrego da Rocha, ISBN: 978-978-722-546-0.
- Schildt, Herbert: C the complete Reference, McGraw-Hill, 1998.
- Algoritmia e Estruturas de Dados, José Braga de Vasconcelos, João Vidal de Carvalho, ISBN: 989-615-012-5.
- Linguagem C, Luís Manuel Dias Damas, ISBN: 972-722-156-4.
- Elementos de Programação com C - Pedro João Valente D. Guerreiro, 3ª edição, ISBN: 972-722-510-1.
- Introdução à Programação Usando C, António Manuel Adrego da Rocha, ISBN: 972-722-524-1.
- https://bettercodehub.com/

58

58