Ordenamento e Pesquisa de um elemento numa lista de elementos [a]

A pesquisa de informação é uma das tarefas mais importantes quando se lida com listas de elementos:

- lista de endereços (agenda electrónica)
- lista de clientes (gestão de encomendas)
- lista de CDs

entre outras aplicações.

Para efectuar a pesquisa de um elemento numa lista temos duas estratégias possíveis.

Pesquisa Sequencial – começa-se numa "ponta" e vai-se até ao fim da lista, ou até se encontrar o elemento.

+ aplica-se a qualquer tipo de lista, ordenada ou não.
+ muito fácil de implementar.
− muito pouco eficiente.

[a] Donald Knuth, *The Art of Computer Programming - Vol 3, Sorting and Searching*, Addison-Wesley, Reading USA,1973

Pesquisa Binária – procura-se no meio da lista, se esse for o elemento pretendido pára-se, caso contrário se esse elemento for menor do que o elemento pretendido vai-se para a direita, ignorando os restantes elementos, caso contrário vai-se para a esquerda (ordem crescente).

\+ muito eficiente.

± fácil de implementar.

− só se aplica a listas ordenadas.

Vejamos então como implementar estes dois métodos.

Pesquisa Sequencial

Especificação

⟶ n,l,elem

com l uma lista com n elementos, e $elem$ o elemento a pesquisar.

⟵ posição, pertence

Isto é a posição do elemento na lista e uma variável lógica com a informação se o elemento pertence ou não à lista.

Algoritmo

Percorrer a lista até encontrar o elemento, ou até chegar ao fim da lista. Neste último caso a variável lógica pertence deverá ter o valor de *false*, sendo que a variável posição não será, neste caso, significativa.

```
alg
  i<-1
  achou<-falso
  enquanto (i<=n ∧ not(achou))
    se  (l(i) = elem) então
      posicao<-i
      achou<-verdade
    fimse
    i<-i+1
  fimenquanto
fimalg
```

Pesquisa Binária

Especificação:

→ v (vector ordenado), dim, elem

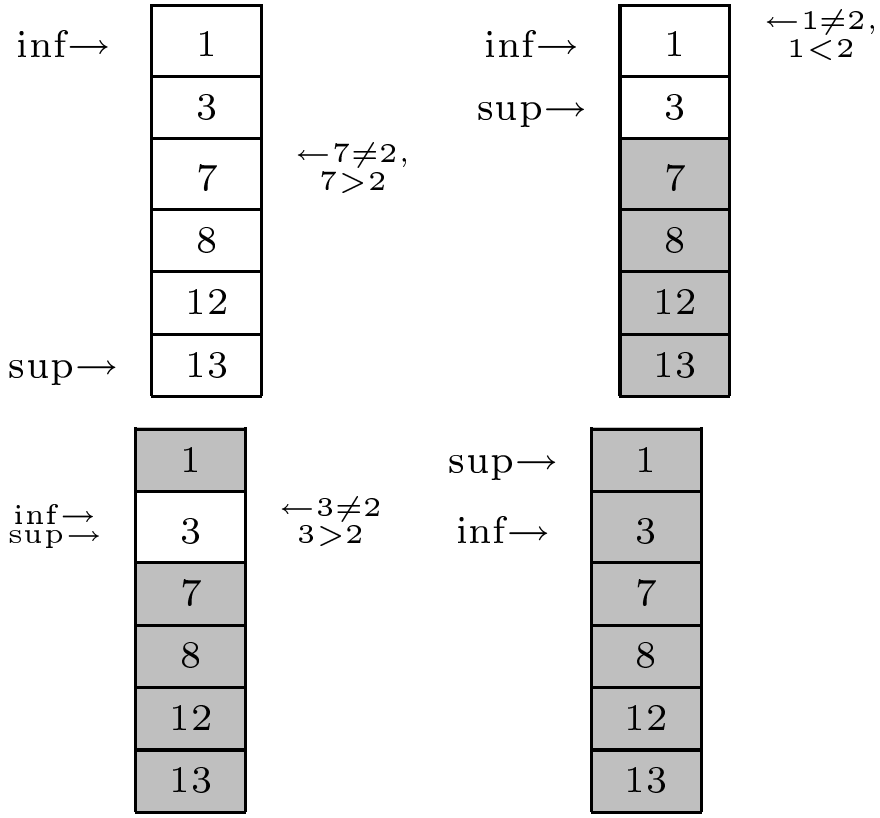
← pertence, posição

Algoritmo:

```
pertence <- falso
enquanto não pertence faz
 se  elem é igual ao elemento do meio então
   pertence <- verdade
   posição <- meio
 senão
   se  elem é maior do que o elemento do meio então
     procura para a direita
   senão
     procura para a esquerda
   fimse
 fimse
fimenquanto
```

Exemplo:

elem=2; v=(1,3,7,8,12,13); dim=6



Chegou-se ao fim do vector, 2 não pertence ao vector.

O algoritmo "binária" (como sub-programa)

```
procedimento binária(elem,dim,v,pertence,pos)
alg
  pertence<-falso
  inf<-1
  sup<-dim
  enquanto inf ≤ sup ∧ ¬ achou faz
    calcular ponto médio (med)
    se  elem = v(med) então
      pertence<-verdade
      pos<-med
    senão
      se  elem > v(med) então
        inf<-med+1
      senão
        sup<-med-1
      fimse
    fimse
  fimenquanto
fimalg
```

O ponto médio é dado pelo número de elementos da secção do vector que se está a considerar a dividir por dois (divisão inteira) mais o ajuste ao primeiro elemento da secção do vector que se está a considerar.

```
  med<-(sup-inf) / 2 + inf
```

```
Program pesquisa
  Implicit None
  Integer, Dimension(100) :: v
  Integer :: n,elem,pos
  Logical :: pertence
  Integer :: i

  Write(*,*) 'Quantos elementos?'
  Read(*,*) n
  Write(*,*) 'Introduza os elementos'
  Read(*,*) (v(i),i=1,n)
  Write(*,*) 'Elemento a procurar?'
  Read(*,*) elem
  Call binaria(n,elem,v,pertence,pos)
  If (pertence) Then
     Write(*,*) 'O elemento está na posi¸cão',pos
  Else
     Write(*,*) 'O elemento não pertence ao vector'
  Endif
endprogram pesquisa

Subroutine binaria(dim,elem,v,pertence,pos)
  Implicit None
  Integer, Dimension(dim),Intent(in) :: v
  Integer, Intent(in) :: dim,elem
  Logical, Intent(out) :: pertence
  Integer, Intent(out) :: pos
  Integer :: inf,sup,med

  pertence = .False.
  inf = 1
  sup = dim
  Do While ((inf <= sup) .And. (.Not. pertence))
     med = (sup-inf) / 2 + inf
     If (elem == v(med)) Then
        pertence=.True.
        pos=med
     Else
        If (elem > v(med)) Then
           inf=med+1
        Else
           sup=med-1
        End If
     End If
  End Do
End Subroutine binaria
```

O algoritmo de pesquisa binária tem uma complexidade $O(\log_2 n)$ isto é o número de operações (médias) que tem de fazer está relacionado com a dimensão do vector n em termos de um logaritmo de base 2. O algoritmo de pesquisa sequencial tem uma complexidade de $O(n)$.

Em conclusão: se a operação de pesquisa é importante há que ordenar o vector antes de proceder a uma qualquer pesquisa.

Existem muitos algoritmos de ordenação de vectores, alguns muito fáceis de implementar embora não muito eficientes, por exemplo o *Borbulhagem* ("Bubble Sort"), o *Ordenamento por Inserção* ("Insertion Sort"), e o *Ordenamento por Selecção* ("Selection Sort"). Outros mais delicados de implementar, mas também muito mais eficientes são o *Ordenamento por fusão*, e o "QuickSort", este último é dos algoritmos mais eficientes para o efeito.

De seguida vai-se descrever o algoritmo de *Ordenamento por Inserção*.

Ordenamento por Inserção

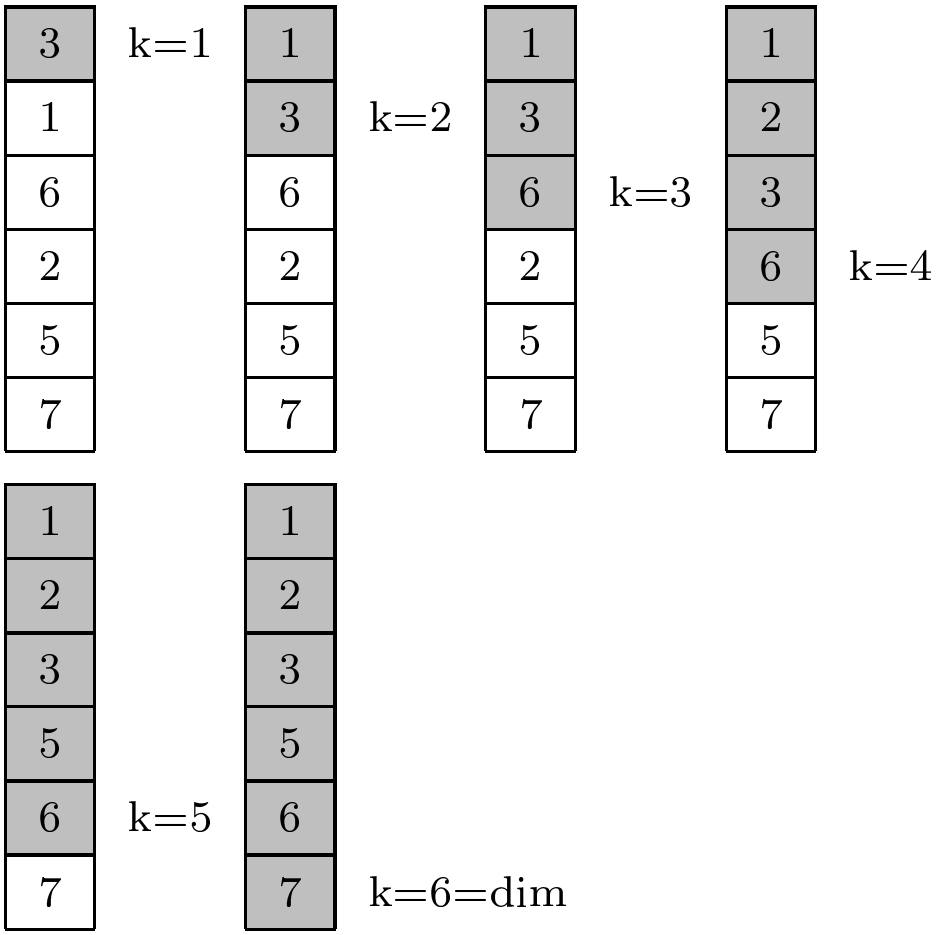
Se os k primeiros elementos de um vector estão ordenados e inserirmos o elemento na posição $k+1$ na sua posição correcta passaremos a ter $k+1$ elementos ordenamos.

$+$ fácil de implementar.

$\pm$ não é muito eficiente.

- Pode-se aplicar à ordenação de um vector.
- Pode-se aplicar à construção ordenada de um vector.

Um exemplo:



A inserção dos diferentes elementos é feita do seguinte modo: comparar o elemento com o de ordem inferior, se necessário trocar a ordem desses dois elementos, repetir o procedimento descrito até não haver motivo para nova troca, ou até se atingir o princípio do vector.

Este método é eficaz em vectores com poucos elementos, assim como em vectores quase-ordenados.

Especificação:

→ v (vector não ordenado), dim

← v (vector ordenado), dim

Algoritmo:

```
para k<-1 até n-i faz
  se  v(k+1) < v(k) então
    colocar o elemento v(k) na sua posição
  fimse
fimpara
```

O colocar do elemento na posição correcta:

```
i<-k
enquanto v(i+1)<v(i) ∧ i≥1 faz
  aux<-v(i)
  v(i)<-v(j)
  v(j)<-aux
  i<-i-1
fimenquanto
```

O Programa em FORTRAN

```
Program ordena

  Implicit None

  Integer, Dimension(100) :: v
  Integer :: n
  Integer :: i

  Write(*,*) 'Quantos elementos?'
  Read(*,*) n
  Write(*,*) 'Introduza os elementos'
  Read(*,*) (v(i),i=1,n)
  Write(*,*) 'Vector não ordenado'
  Write(*,*) (v(i),i=1,n)
  Call insercao(n,v)
  Write(*,*) 'Vector ordenado'
  Write(*,*) (v(i),i=1,n)
endprogram ordena

Subroutine insercao(dim,v)

  Implicit None

  Integer, Intent(in) :: dim
  Integer, Dimension(dim),Intent(inout) :: v

  Integer :: k,i,aux

  Do k=1,dim-1
     If (v(k+1) < v(k)) Then
        i=k
        Do While (v(i+1) < v(i) .And. i>=1)
           aux=v(i)
           v(i)=v(i+1)
           v(i+1)=aux
           i=i-1
        End Do
     End If
  End Do
End Subroutine insercao
```