# UNIVERSITÀ DEGLI STUDI DI TRIESTE

## FACOLTÀ DI ECONOMIA

*APPUNTI SUGLI ALGORITMI*

**ANNO ACCADEMICO 2007 - 2008**

*Renato Pelessoni*

# ALGORITMI

Un algoritmo è un insieme finito di istruzioni che consente di risolvere una classe di problemi.

Un algoritmo deve essere caratterizzato da

- **generalità**
- **finitezza**
- **non ambiguità**

**Complessità**: "lavoro" eseguito da un algoritmo

**Misura di complessità**: n° di **operazioni di base** eseguite

La complessità di un algoritmo viene in generale espressa in funzione della **dimensione dell'input**.

Un algoritmo può inoltre richiedere dello spazio di memoria aggiuntivo oltre a quello richiesto per memorizzare l'input. L'algoritmo opera **sul posto** (**in place**) se lo spazio usato è costante rispetto alla dimensione dell'input.

Siano

- $S_n$ insieme degli input di dimensione n
- m(x) n° di operazioni di base eseguite per $x \in S_n$
- $[x]=\{y \in S_n | m(y)=m(x)\}$ $(x \in S_n)$
- p([x]) probabilità di un input appartenente a [x]
- $\Gamma=\{[x] \mid x \in S_n\}$ (ipotesi: $\Gamma$ finito)
- $M([x])=m(x)$ $(x \in S_n)$

## Complessità nel caso peggiore (worst-case complexity)

$W(n)=\max\{m(x) \mid x \in S_n\}$

## Complessità media (average complexity)

$$A(n)=\sum_{[x]\in\Gamma} p([x])m(x) = \sum_{[x]\in\Gamma} p([x])M([x])$$

# Ricerca sequenziale

r *variabile contenente l'elemento da cercare*
a *vettore di* n *elementi*

Alcune varianti dell'algoritmo:

```
i = 1
trovato = False
Do While (Not trovato) And (i <= n)
  If a(i) = r Then
     trovato = True
  else
     i = i+1
  End If
Loop                          'rende n+1 se non trova
```

```
i = 1
Do While (i <= n) And (a(i) <> r)
   i = i+1
Loop                                   'rende n+1 se non trova
```

```
i=1
Do While (i<n) And (a(i)<>r)
   i=i+1
Loop
If a(i)<>r Then
   i=n+1
End If       'rende n+1 se non trova
```

# Complessità della ricerca sequenziale

```
i = 1
Do While (i <= n) And (a(i) <> r)
   i = i+1
Loop
If i > n Then i = n+1 EndIf
```

Dimensione dell'input: **n (numero elementi di a)**
Operazione di base: **confronto a(i) <> r**

Caso peggiore: r è in posizione n o non compare

**W(n)=n**

$C_i$ insieme dei vettori di n elementi in cui r è in posizione i (i=1,…,n) $\Longrightarrow$ $M(C_i)=i$

$C_{n+1}$ insieme di vettori di n elementi in cui r non compare $\Longrightarrow$ $M(C_{n+1})=n$

$$p(C_i) = q/n \,(i = 1,\ldots,n) \quad p(C_{n+1}) = 1-q \quad (q \in [0,1])$$

$$A(n) = \sum_{i=1}^{n} p(C_i)M(C_i) + p(C_{n+1})M(C_{n+1}) =$$

$$= q\sum_{i=1}^{n} i/n + n(1-q) = q\,(n+1)/2 + n(1-q)$$

# Confronto tra algoritmi

$S=\{h:N \longrightarrow \Re^{+} \cup \{0\}\}$

Sia $f \in S$ una funzione esprimente la complessità di un algoritmo

- $O(f)=\{g \in S \mid \exists k \in \Re+, m \in N \mid g(n) \leq k\, f(n)\ \forall n \geq m\}$
- $\Omega(f)=\{g \in S \mid \exists k \in \Re+, m \in N \mid g(n) \geq k\, f(n)\ \forall n \geq m\}$
- $\Theta(f)= O(f) \cap \Omega(f)$
- $o(f)= O(f)-\Omega(f)$

$g \in O(f) \iff$ g ha ordine minore od uguale a f

$g \in \Omega(f) \iff$ g ha ordine maggiore od uguale a f

$g \in \Theta(f) \iff$ g ha lo stesso ordine di f

$g \in o(f) \iff$ g ha ordine minore di f

## Proprietà

- $\lim_{n \to +\infty} g(n)/f(n) = 0 \quad \Rightarrow \quad g \in O(f)$
- $\lim_{n \to +\infty} g(n)/f(n) = u > 0 \quad \Rightarrow \quad g \in \Theta(f)$
- $\lim_{n \to +\infty} g(n)/f(n) = +\infty \quad \Rightarrow \quad g \in \Omega(f)$
- $\lim_{n \to +\infty} g(n)/f(n) = 0 \quad \Rightarrow \quad g \in o(f)$
- $f \in O(g),\ g \in O(h) \Rightarrow f \in O(h)$ (anche per $\Omega$, $\Theta$, o)
- $f \in O(g) \Leftrightarrow g \in \Omega(f)$
- $f \in \Theta(g) \Leftrightarrow g \in \Theta(f)$
- $O(f+g)=O(\max\{f,g\})$
- $\Omega(f+g)=\Omega(\max\{f,g\})$
- $\Theta(f+g)=\Theta(\max\{f,g\})$

La relazione $\Theta$ definita dalla

$$\mathbf{f \Theta g \Leftrightarrow f \in \Theta(g)}$$

è una relazione di equivalenza le cui classi di equivalenza sono dette **classi di complessità**.

Qualunque funzione appartenente ad una classe di complessità può essere scelta per rappresentare la classe stessa. In particolare

- $\Theta(\log(n))$ classe di complessità **logaritmica**
- $\Theta(n)$ classe di complessità **lineare**
- $\Theta(n^2)$ classe di complessità **quadratica**
- $\Theta(n^3)$ classe di complessità **cubica**
- $\Theta(k^n)$ $(k>1)$ classe di complessità **esponenziale**

Si ha

$\log(n) \in o(n^\alpha)\ \forall \alpha>0$

$n^\alpha \in o(k^n)\ \forall \alpha>0,\ k>1$

La preferenza andrà in generale ad algoritmi la cui complessità ha (appartiene a classi di) ordine inferiore.

# Complessità ed incremento della velocità di esecuzione

Indicando:

- Complessità algoritmo: f(n)
- Max ampiezza input "trattabile" nel tempo T: m
- N° operazioni eseguite nel tempo T: f(m)
- Fattore di incremento velocità calcolatore: k

Per determinare la nuova ampiezza max dell'input "trattabile" nel tempo T, risolvere l'equazione in n:

$f(n)=k \cdot f(m)$

| Complessità f(n) | Max ampiezza input dopo incremento | |
|---|---|---|
| $\log_a(n)$ | $m^k$ | |
| $n$ | $km$ | |
| $n^2$ | $k^{½}m$ | |
| $a^n$ | $m+ \log_a(k)$ | $(a>1)$ |

# Ricerca dei due elementi più piccoli in un vettore

a vettore di n (n≥2) elementi reali
Problema: trovare i due valori più piccoli in a.
Operazione di base: confronto tra valori

## 1° Algoritmo

Idea: cercare il minimo del vettore e di seguito il minimo tra gli elementi rimanenti.

```
min1 = a(1)
indmin = 1                  'Inizializzazione di min1 e dell'indice
For i = 2 To n              'Determinazione del minimo del vettore
    If a(i) < min1 Then
          min1= a(i)
          indmin = i
    End If
Next i                      'Segue inizializzazione della variabile min2
If indmin <> 1 Then min2 = a(1) Else min2 = a(2) EndIf
For i = 2 To n              'Determinazione del minimo tra i rimanenti
    If (i <> indmin) And (a(i) < min2) Then
       min2 = a(i)
    End If
Next i
```

| | | |
|---|---|---|
| N° confronti: | n-1 | per determinare il minimo |
| | 1 | per il test indmin <> 1 |
| | 2(n-1) | per il 2° valore più piccolo |
| Totale | 3n-2 | confronti |

# 2° Algoritmo

Idea: modificare la seconda parte del 1° algoritmo spostando il minimo al primo posto nel vettore.

```
min1= a(1)              'Inizializzazione della variabile min1
indmin = 1              'Inizializzazione dell'indice del minimo
For i = 2 To n          'Determinazione del minimo del vettore
    If a(i) < min1 Then
        min1 = a(i)
        indmin = i
    End If
Next i

comodo = a(1)           'Spostamento del minimo in prima posizione
a(1) = min1
a(indmin) = comodo
min2 = a(2)
For i = 3 To n          'Determinazione del minimo tra i rimanenti
    If a(i) < min2 Then
        min2 = a(i)
    End If
Next i
```

| | | |
|---|---|---|
| N° confronti: | n-1 | per determinare il minimo |
| | n-2 | per il 2° valore più piccolo |
| Totale | 2n-3 | confronti |

# 3° Algoritmo

Idea: effettuare un solo ciclo aggiornando due variabili min1 e min2 all'interno di esso.

```
If a(1) < a(2) Then       'Inizializzazione delle variabili min1 e min2
    min1 = a(1)
    min2 = a(2)
Else
    min1 = a(2)
    min2 = a(1)
End If
For i = 3 To n            'Determinazione dei due valori più piccoli
    If a(i) < min2 Then
        If a(i) < min1 Then
           min2 = min1
           min1 = a(i)
        Else
           min2 = a(i)
        End If
   End If
Next i
```

| | | |
|---|---|---|
| N° confronti: | 1 | per l'inizializzazione |
| | $n-2 \leq N \leq 2(n-2)$ | nel ciclo |
| Totale | $n-1 \leq N \leq 2n-3$ | confronti |

# 4° Algoritmo

Idea: considerare coppie di elementi successivi ed ordinare ogni coppia; poi confrontare in sequenza le coppie. Per semplicità si supporrà n pari.

```
i = 1
Do While i < n                          'Ordinamento di ogni coppia
    If a(i+1) < a(i) Then
        com = a(i)     a(i) = a(i+1)   a(i+1) = com
        i = i+2
    End If
Loop
min1 = a(1)
min2 = a(2)
i = 3                                   'Inizializzazione di min1, min2 ed i
Do While i < n                          'Confronto delle coppie
  If a(i) < min1 Then
      If a(i+1) < min1 Then
          min1 = a(i)      min2 = a(i+1)
      Else
          min2 = min1    min1 = a(i)
      End If
  Else
      If a(i) < min2 Then
          min2 = a(i)
      End If
  End If
  i = i+2
Loop
```

| | | |
|---|---|---|
| N° confronti: | n/2 | ordinamento elem. delle coppie |
| | n-2 | confronto delle coppie |
| Totale | 1.5n-2 confronti | |

Riepilogando:

| | W(n) | A(n) |
|---|---|---|
| 1° algoritmo | 3n-2 | 3n-2 |
| 2° algoritmo | 2n-3 | 2n-3 |
| 3° algoritmo | 2n-3 | * |
| 4° algoritmo | 1.5n-2 | 1.5n-2 |

* dipende dalla distribuzione di probabilità dell'input

# Complessità worst case della ricerca sequenziale in un vettore ordinato

r *variabile contenente l'elemento da cercare*
a *vettore ordinato di* n *elementi*

Il ciclo di ricerca può essere interrotto se l'elemento corrente è maggiore od uguale ad r.

```
i = 1
Do While (i<n) And (a(i) < r)
    i = i+1
Loop
If  a(i) <> r Then
    i = n+1                 'Rende n+1 se non trova
End If
```

Operazione di base: **confronto a(i) < r**

| Posizione di r | N° confronti |
|---|---|
| 1) r = a(i) (i = 1, …, n) | i |
| 2) r < a(1) | 1 |
| 3) a(i-1) < r < a(i) (i = 2, …, n) | i |
| 4) a(n) < r | n |

**W(n)=n** Complessità worst-case lineare

Supponendo che sia 0.5 la probabilità che l'elemento cercato appartenga al vettore e supponendo che:

probabilità casi di tipo 1
(sono in tutto n casi) $\frac{1}{2n}$

probabilità casi di tipo 2, 3, 4
(sono in tutto n+1 casi) $\frac{1}{2(n+1)}$

$$\mathbf{A(n) = \sum_{i=1}^{n} \frac{i}{2n} + \sum_{i=1}^{n} \frac{i}{2(n+1)} + \frac{n}{2(n+1)} \cong \frac{n}{2}}$$

In queste ipotesi la complessità media è quindi lineare.

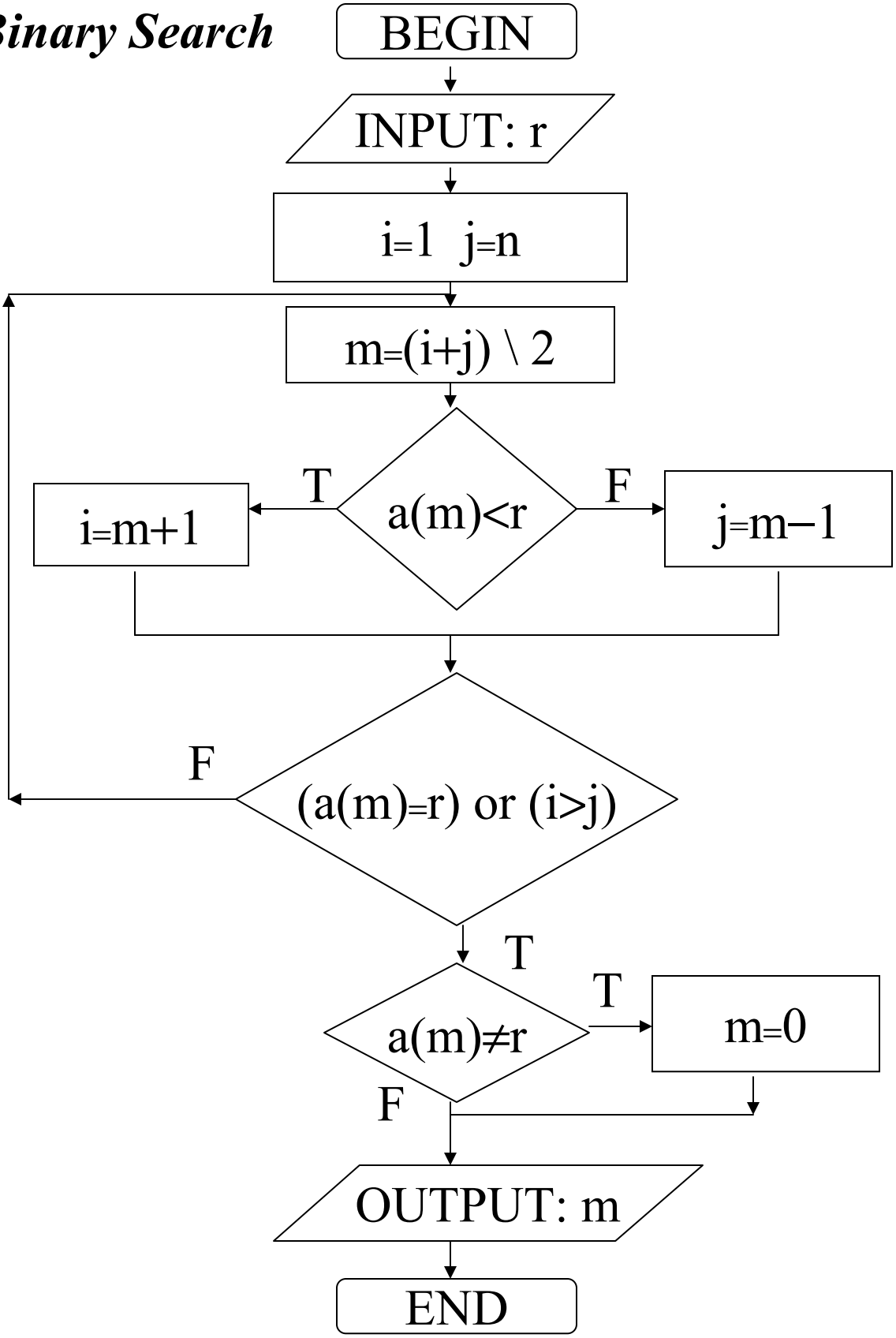
# Algoritmo di ricerca binaria

r *variabile contenente l'elemento da cercare*
a *vettore* <u>*ordinato*</u> *di* n *elementi*

```
i = 1
j = n
Do
   m = (i+j) \ 2;
   If a(m) < r Then
      i = m+1
   Else
      j = m – 1
   End If
Until (a(m) = r) Or (i > j)
If a(m) <> r Then
   m = 0                ‘rende 0 se non trova
End If
```

La complessità nel caso peggiore della ricerca binaria è logaritmica: $\mathbf{W(n) \in \Theta(\log_2(n))}$

***Binary Search***



# Algoritmi di selection sort

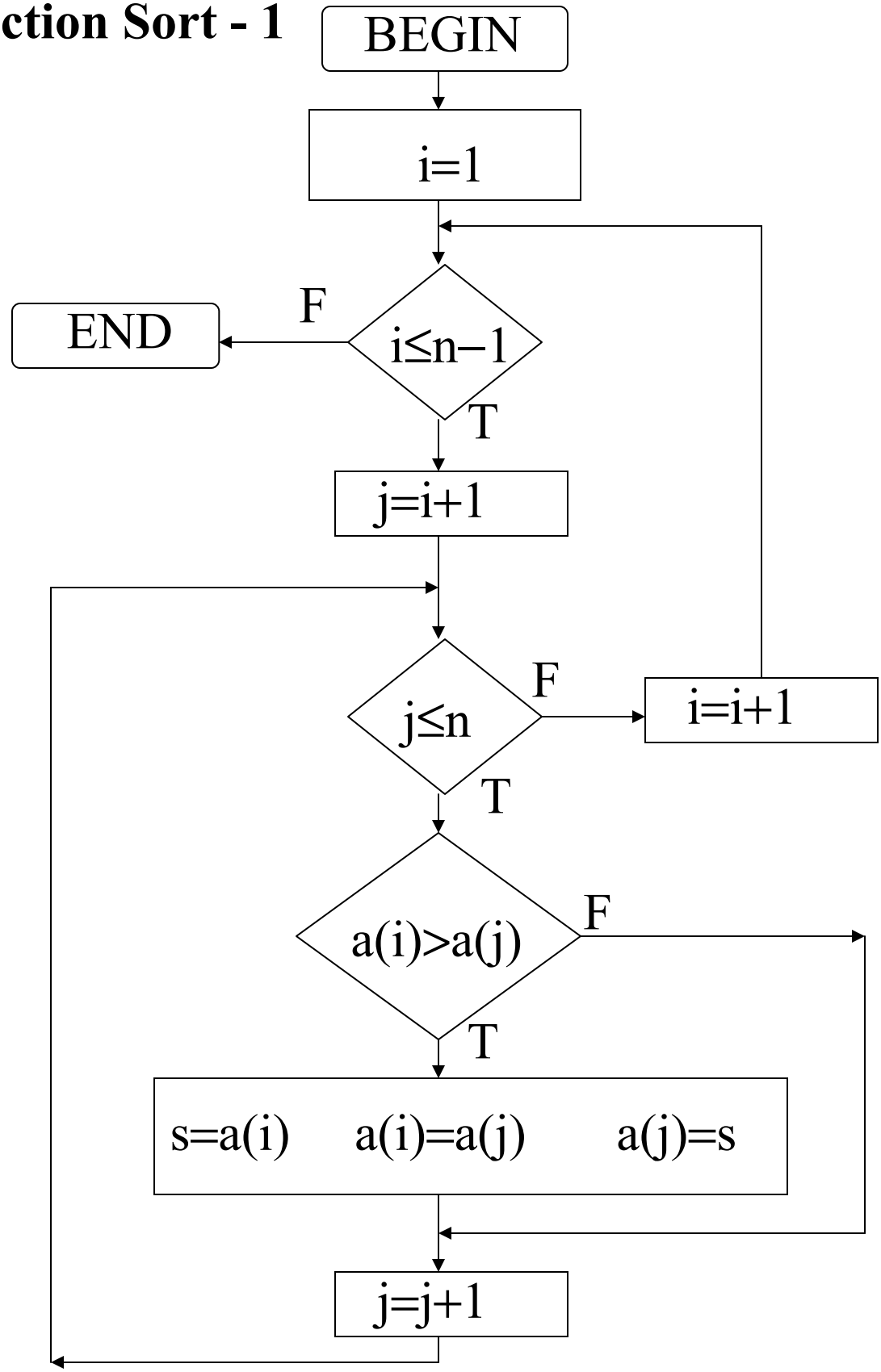
Gli algoritmi operano scorrendo il vettore, determinando via via il minimo degli elementi ancora da spostare e sistemando questo nella posizione corretta, come elemento successivo a quelli già posizionati.

a *vettore di* n *elementi*

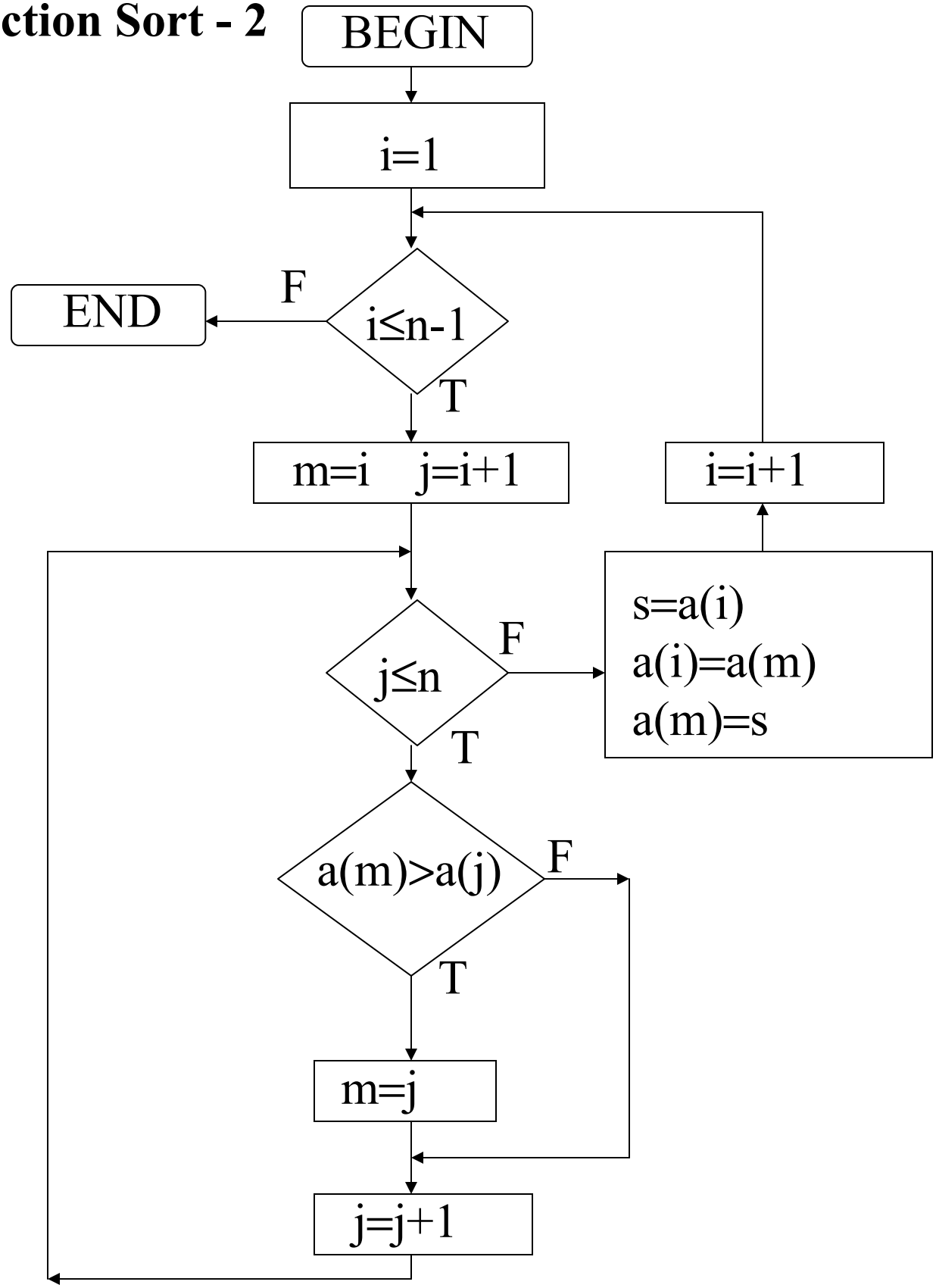
```
For i = 1 To n – 1                  ‘versione 1
   For j = i + 1 To n
      If a(i) > a(j) Then
         s = a(i)                   ‘scambio degli elementi
         a(i) = a(j)
         a(j) = s
      End If
   Next j
Next i

For i = 1 To n – 1                  ‘versione 2
   m = i
   For j = i + 1 To n
      If a(m) > a(j) Then
         m = j                      ‘salvataggio dell’indice
      End If
   Next j
   s = a(i)                         ‘scambio degli elementi
   a(i) = a(m)
   a(m) = s
Next i
```

# Selection Sort - 1



# Selection Sort - 2



# Complessità del selection sort -2

Operazione di base: **confronto a(m) > a(j)**

N° confronti: n–i per ogni valore di i (i=1, …, n–1)

Totale: $\sum_{i=1}^{n-1}(n-i)=\sum_{i=1}^{n-1} i=\frac{n(n-1)}{2}$ confronti

**$W(n)=A(n)=\frac{n(n-1)}{2}$**

Operazione di base: **assegnazione per scambio di elementi**

N° assegnazioni: 3 per ogni valore di i (i=1, …, n-1)

Totale: 3(n-1) assegnazioni

**W(n)=A(n)=3(n-1)**

# Merge di vettori

a, b *vettori* *ordinati* *di* n *ed* m *elementi rispettivamente*
c *vettore di* n+m *elementi*

```
i=1   j=1   k=1
Do
   If a(i)<=b(j) then
      c(k)=a(i)
      i=i+1
   Else
      c(k)=b(j)
      j=j+1
   End If
   k=k+1
Until (i>n) Or (j>m)
If (i<=n) Then
   For u=i To n
      c(k)=a(u)
      k=k+1
   Next u
Else
   For u=j To m
      c(k)=b(u)
      k=k+1
   Next u
End If
```

## Complessità del merge di vettori

Operazione di base: confronti tra elementi dei vettori a e b

Ogni confronto comporta la copiatura di un elemento di uno dei due vettori nel vettore c.

Dopo l'ultimo confronto l'elemento più piccolo e tutti gli elementi rimanenti nell'altro vettore vengono copiati nel vettore c. Non si possono quindi effettuare più di n+m-1 confronti.

Il caso peggiore, che comporta proprio n+m-1 confronti, si ha quindi quando a(n) e b(m) verranno copiati nelle ultime due posizioni del vettore c.

**W(n,m)=n+m-1**

# Ricorsione

Un oggetto è detto **ricorsivo** se comprende parzialmente o è definito in termini di sé stesso.

Esempi:

1) Numeri naturali
   - 1 è un numero naturale
   - il successore di un numero naturale è un numero naturale
2) Funzione fattoriale (per $n \geq 0$)
   - $0!=1$
   - $n!=n(n-1)!\ \forall n>0$
3) Numeri di Fibonacci
   ($Fib_n$ n-esimo numero di Fibonacci)
   - $Fib_0=0$
   - $Fib_1=1$
   - $Fib_n= Fib_{n-1}+Fib_{n-2}$

Nella programmazione si possono avere due tipi di ricorsione:

**Ricorsione diretta**

```
procedura A
……..
A
……..
```

**Ricorsione indiretta**

```
procedura A
……..
B
……..

procedura B
……..
A
……..
```

Le regole di visibilità di variabili locali e parametri impediscono il verificarsi di conflitti di nome.

Una **condizione di arresto** porrà termine alla successione di chiamate ricorsive.

# Calcolo del fattoriale

*Versione iterativa*

```
Private Function iterFatt(n As Integer) As Double
Dim i As Integer
Dim com As Double
com = 1
For i = 2 To n
   com = com * i
   iterFatt = com
Next i
End Function
```

*Versione ricorsiva*

```
Private Function ricFatt(n As Integer) As Double
If n = 0 Then
   ricfatt = 1
Else
   ricfatt = n * ricFatt(n – 1)
End If
End Function
```

# Calcolo dei numeri di Fibonacci

*Versione ricorsiva*

```
Private Function ricFib(n As Integer) As Integer
If n = 0 Then
   ricFib = 0
ElseIf n = 1 Then
   ricFib = 1
 Else
   ricFib = ricFib(n – 1) + ricFib(n – 2)
End If
End Function
```

*Versione iterativa*

```
Private Function iterFib (n As Integer) As Integer
Dim x As Integer
Dim y As Integer
Dim z As Integer
Dim i As Integer

If n = 0 Then
   itFib = 0
Else
   x = 1    y = 0
   For i = 1 To n – 1
       z = x     x = x + y     y = z
   Next i
   itFib = x
End If
End Function
```

# Torre di Hanoi

Sono dati tre pioli, su uno dei quali sono infilati *n* dischi, ciascuno di dimensioni inferiori a quello sotto di lui.

Scopo: portare tutti i dischi su un altro piolo

Regole:

- un solo disco può essere mosso per volta
- nessuno disco può essere sistemato sopra un disco più piccolo

Soluzione:

- spostare gli *n-1* dischi più piccoli su un piolo
- spostare il disco più grande sul piolo libero
- spostare gli *n-1* dischi più piccoli sul più grande

⇨ Sposto gli *n* dischi spostando una volta un disco e due volte *n-1* dischi

```
Private Sub scriviMosse(n As Integer, part As Integer, dest As Integer,
interm As Integer)

If n = 1 Then
   lstMess.Items.Add("Muovi disco da" & " " & part  & " a " & dest)
Else
   scriviMosse(n – 1, part, interm, dest)
   lstMess.Items.Add("Muovi da" & " " & part & " a " & dest)
   scriviMosse(n – 1, interm, dest, part)
End If

End Sub
```

Se n è la variabile che contiene il numero di dischi da muovere dal piolo 1 al piolo 2, la prima chiamata della routine sarà:

```
lstMess.Items.Add("Per muovere " & n & " dischi tra i pioli 1 e 2: ")
scriviMosse(n, 1, 2, 3)
```

# Esempio di output del programma:

Per muovere 3 dischi dal piolo 1 al piolo 2

Muovi da 1 a 2
Muovi da 1 a 3
Muovi da 2 a 3
Muovi da 1 a 2
Muovi da 3 a 1
Muovi da 3 a 2
Muovi da 1 a 2

f(n)=numero di mosse per spostare n dischi

$$f(n) = 1 + 2f(n-1) \qquad (n \geq 2)$$

$\Longrightarrow$

$$f(1) = 1$$

Soluzione: $f(n) = 2^n - 1$

Esempio:

$n = 64$ $\qquad f(64) = 2^{64} - 1 \cong 1.84 \cdot 10^{19}$

$1 \text{ anno} = 31536 \cdot 10^3 \text{ sec}$

1 mossa al sec $\Longrightarrow$ $\cong 5.85 \cdot 10^{11}$ anni