Corso di Programmazione
Algoritmi Fondamentali

Prof.ssa Teresa Roselli
roselli@di.uniba.it

Minimo fra 3 valori

- Trovare il minore fra tre numeri reali a, b, c
  - Esempi:
    - $a = 1$, $b = 2$, $c = 3$
    - $a = 20$, $b = 50$, $c = 55$
    - $a = 2$, $b = 1$, $c = 3$
    - $a = 7$, $b = 34$, $c = 13$
- Operatore di confronto $x \leq y$
  - È verificato se il valore associato alla variabile x è minore di quello associato alla variabile y

Minimo fra 3 valori

Algoritmo

se $a \leq b$

allora se $a \leq c$

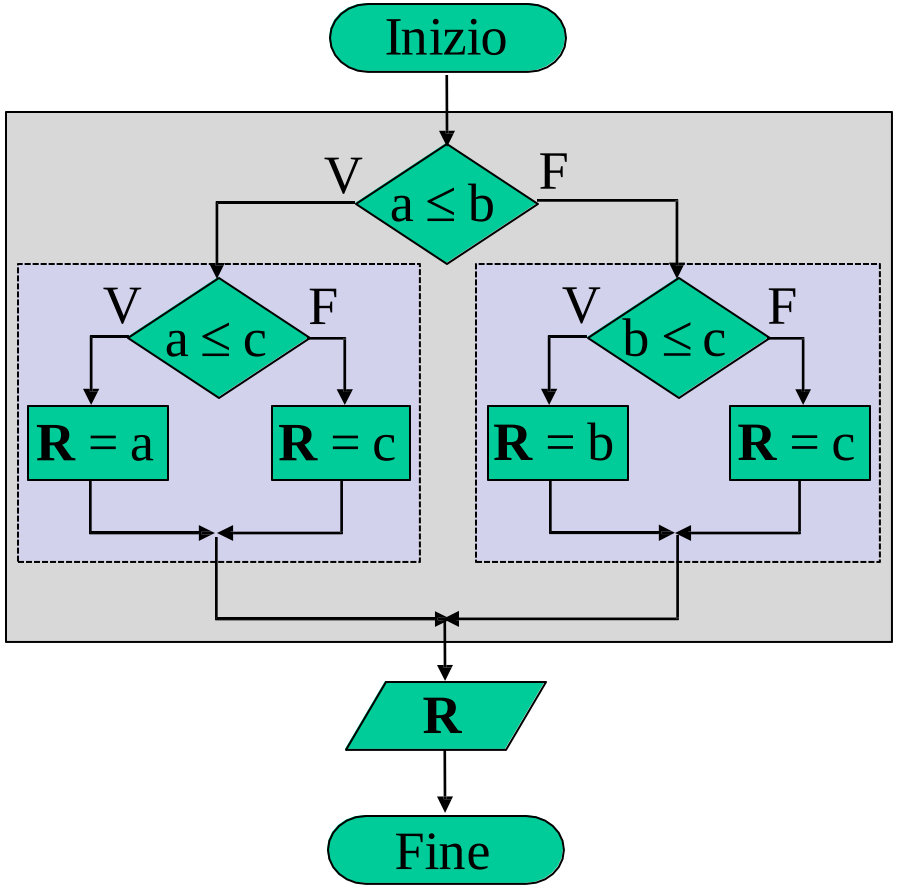
allora soluzione = a

altrimenti soluzione = c

altrimenti se $b \leq c$

allora soluzione = b

altrimenti soluzione = c



Minimo fra 3 valori
Considerazioni

- Analogo per qualunque dominio ordinale
  - Necessità di un operatore di confronto
  - Esempi:
    - Date 3 lettere, stabilire qual è la prima in ordine alfabetico
    - Dati 3 giorni della settimana, stabilire quale viene prima dal lunedì alla domenica

Scambio di valori

- Date due variabili *a* e *b*, scambiare i valori ad esse assegnati
  - Esempio:
    - $a = 12, b = 54 \rightarrow a = 54, b = 12$
- Operatore di assegnamento $x \leftarrow y$
  - Copia il valore associato alla variabile *y* nella memoria associata alla variabile *x*
    - Al termine i valori di *x* ed *y* coincidono
    - Il vecchio valore di *x* viene perso

Scambio di valori
Algoritmo (tentativo)

Assegna il valore di *b* ad *a*

Assegna il valore di *a* a *b*

- Trace:
  - *a* = 12, *b* = 54
  - *a* = 54, *b* = 54
  - *a* = 54, *b* = 54

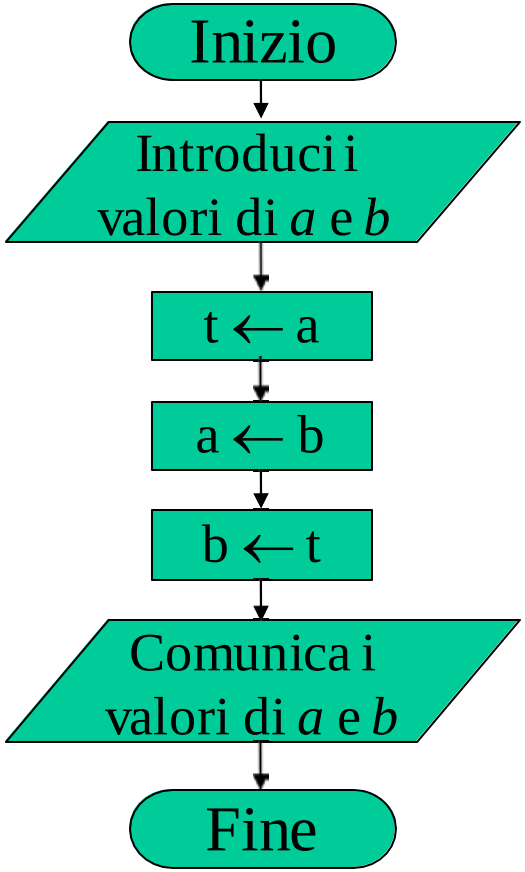
- ***Non funziona!***
  - Dopo il primo passo il valore originario di *a* viene perso
  - Serve una variabile temporanea *t* in cui copiare il valore di *a* prima che sia sovrascritto

Scambio di valori
Algoritmo (corretto)

Assegna il valore di *a* a *t*

Assegna il valore di *b* ad *a*

Assegna il valore di *t* a *b*



– Trace:
  - *a* = 12, *t* = ??, *b* = 54
  - *a* = 12, *t* = 12, *b* = 54
  - *a* = 54, *t* = 12, *b* = 54
  - *a* = 54, *t* = 12, *b* = 12

Scambio di valori
Considerazioni

- Ad ogni passo dell'algoritmo una variabile assume sempre il valore definito dalla assegnazione più recente per essa
- L'applicazione di un algoritmo a casi particolari può aiutare a scoprire gli errori
  - Scelta oculata degli esempi critici
    - Non valori uguali per le due variabili
- Quante variabili temporanee servono per una rotazione dei valori contenuti in *n* variabili?

Conteggio

- Dato un insieme di *n* valori, contare il numero di valori maggiori di una soglia *s*
  - Esempio:
    Studenti che hanno superato un esame (*s* = 17)
    - Voti 22 30 17 25 4 18 27
- Accumulatore
  - Area di memoria interna contenente i risultati parziali dei calcoli

Conteggio

- Ad ogni passo, se il voto supera la soglia, si compiono le seguenti azioni:

  conteggio attuale ← conteggio precedente + 1

  conteggio precedente ← conteggio attuale

  – Alla destra del primo assegnamento si ha un'espressione e non una variabile
    - Il risultato andrà nell'accumulatore in attesa di assegnarlo alla variabile a sinistra

Conteggio
Considerazioni

- L'assegnamento è diverso dalla relazione matematica di uguaglianza
  - Il simbolo a sinistra dell'assegnamento deve essere un identificatore
    - Lo stesso simbolo, posto a destra, indica il valore prima dell'assegnamento
      - $x \leftarrow x + 1$ ha senso
      - $x + 1 \leftarrow x$ non ha senso

Conteggio
Algoritmo

leggi il numero *n* di voti da elaborare

inizializza il contatore a 0

mentre ci sono ancora voti da esaminare **esegui**

    leggi il voto successivo

    se voto > 17

     allora somma 1 al contatore

comunica il numero totale di promossi

Somma

- Dato un insieme di *n* numeri, progettare un algoritmo che li sommi e restituisca il totale risultante
  - Ipotesi: $n \geq 0$
  - Esempio:
    - Sommare 46, 2 e 284
      - somma ← 46 + 2 + 284
        Troppo specifico sui valori!
      - somma ← a + b + c
        Troppo specifico sul numero di valori!

Somma

- Non è possibile scrivere un'assegnazione che valga per qualunque insieme di *n* valori, ma:
  - Gli elaboratori sono adatti ai compiti ripetitivi
  - Il sommatore ha capienza di 2 numeri per volta

Somma

- Si può usare un accumulatore per contenere le somme parziali:
  - $s \leftarrow a_1 + a_2$
  - $s \leftarrow s + a_3$
  - …
  - $s \leftarrow s + a_n$
- Compattando i vari passi:

$s \leftarrow a_1 + a_2$

finché ci sono ancora numeri da sommare

leggi il numero successivo a_{i+1}

$s \leftarrow s + a_{i+1}$

- Non funziona per la somma di 0 o 1 valore

Somma
Algoritmo

considera il numero *n* di
addendi

$s \leftarrow 0$

finché ci sono ancora numeri
da sommare esegui
leggi il numero successivo
a_{i+1}
$s \leftarrow s + a_{i+1}$

comunica che la somma
finale è s

Somma
Considerazioni

- Per sommare *n* numeri sono necessarie *n* – 1 addizioni
  - Se ne usano *n* per generalità
- Ad ogni iterazione *s* rappresenta la somma dei primi *i* valori
  - Per *i* = *n* si ha la somma totale
- Il valore di *i* cresce ad ogni iterazione
  - Prima o poi *i* = *n* e si uscirà dal ciclo (Terminazione)
- Non è assicurata la precisione della somma finale
  - Sommare i numeri per valore assoluto crescente

Successione di Fibonacci

- Generare i primi *n* termini della successione di Fibonacci (n ≥ 1)
  - I primi due termini sono 0 e 1
  - Ogni termine successivo è ottenuto come somma degli ultimi 2 termini
    - 1, 2, 3, 5, 8, 13, 21, 34, …

Successione di Fibonacci

- Necessario un ciclo di generazioni
  - In ogni istante, bisogna conoscere gli ultimi due termini generati per generare il successivo
    - Ultimo
    - Penultimo
  - Dopo la generazione
    - Il termine appena generato diventa l'ultimo
    - Il termine che prima era l'ultimo diventa penultimo

Successione di Fibonacci
Algoritmo

Acquisire il numero di termini da generare *n*
primo (e penultimo) termine ← 0
secondo (e ultimo) termine ← 1
termini generati ← 2
fintantoché termini generati < *n* **esegui**
    nuovo termine ← penultimo + ultimo
    penultimo ← ultimo
    ultimo ← nuovo termine
    termini generati ← termini generati + 1

Inversione delle Cifre di un Intero

- Progettare un algoritmo che accetta un intero positivo e ne inverte l'ordine delle cifre
  - Esempio
    $27953 \rightarrow 35972$
- Operatori DIV e MOD
  - Calcolano, rispettivamente, il quoziente intero ed il resto di una divisione fra interi

Inversione delle Cifre di un Intero

- Serve accedere alle singole cifre del numero
  - Non è nota a priori la lunghezza del numero
    - Iniziare dalla cifra meno significativa
- Eliminare progressivamente le cifre dal numero originario
  - Procedere da destra verso sinistra
- Accodarle progressivamente al numero invertito
  - Far scalare le precedenti verso sinistra

Inversione delle Cifre di un Intero

- Individuazione della cifra meno significativa
  - c = n MOD 10
- Rimozione della cifra meno significativa
  - n' = n DIV 10
- Scalare verso sinistra l'inversione parziale per aggiungere la nuova cifra
  - m * 10 + c
- Ripetere fino a quando non ci sono più cifre
  - Quoziente della divisione pari a 0

Inversione delle Cifre di un Intero Algoritmo

Stabilire *n*

Predisporre a 0 l'intero rovesciato

Finché l'intero da rovesciare è maggiore di 0

- Estrarre dall'intero da rovesciare la cifra meno significativa come resto della divisione per 10
- Aggiornare l'intero rovesciato moltiplicandolo per 10 e aggiungendo la cifra estratta
- Eliminare dall'intero da rovesciare la cifra estratta dividendolo per 10

Inversione delle Cifre di un Intero
Algoritmo

read *n*

inverso:= 0

while *inverso* > 0 do

    begin

    c:= *n* MOD 10

    inverso:= *inverso* * 10 + *c*

    n:= n DIV 10

    end

write *inverso*

Algoritmi su Array

- Algoritmi di base
- Algoritmi di supporto
  - Partizionamento
  - Fusione
- Ricerca
  - Basata su un ordinamento
- Ordinamento
  - Finalizzato alla ricerca

Acquisizione di un Array

- Dichiarazione del tipo opportuno
  - Dimensione fissa
    - Sufficiente all'uso previsto
    - Indice massimo attualmente usato

inizializza l'indice
fintantoché c'è un nuovo dato e l'array non è pieno
    incrementa l'indice
    leggi il nuovo dato e inseriscilo nell'array
  - Al termine l'indice specifica il numero di elementi
- N.B.: NON è possibile leggere una stringa in un unico passo

Massimo

- Trovare il massimo valore in un insieme di *n* elementi
  - Il computer non può guardare globalmente i valori nell'insieme
    - Analizzarli uno per volta
      - Necessità di esaminarli tutti per trovare il massimo
    - Ricordare in ogni momento il massimo parziale
      - Guardando solo il primo, è il più alto che si sia visto
      - Guardando i successivi, si aggiorna il massimo ogni volta che se ne trova uno maggiore del massimo parziale

Massimo
Algoritmo

- Inserire il numero di valori
- Leggere il primo
- Porre il massimo al primo
- Mentre non si sono letti tutti gli elementi
  - Leggere il successivo
  - Se è maggiore del massimo
    - Porre il massimo a questo numero
- Comunicare il massimo

Massimo
Considerazioni

- Necessari $n - 1$ confronti
- Si devono analizzare tutti gli elementi fino all'ultimo
  - Noto se l'insieme è conservato in un array
  - Ignoto se la sequenza di valori viene inserita progressivamente
    - Va chiesto quanti elementi compongono la sequenza

Inversione

- Risistemare gli elementi di un array di dimensione *n*, in modo tale che appaiano al contrario

  | a | k | h | e | x | b | h | → | h | b | x | e | h | k | a |

  – Inserirli in ordine inverso in un array di appoggio, quindi ricopiarlo in quello originale?
    - Spreco di memoria
    - Spreco di tempo

Inversione

- Effetto dell'inversione
  - Il primo diventa l'ultimo
  - Il secondo diventa il penultimo
  - …
  - Il penultimo diventa il secondo
  - L'ultimo diventa il primo
- Soluzione: scambiare il primo con l'ultimo, il secondo col penultimo, ecc.

Inversione

- 2 indici
  - Partono dagli estremi dell'array
  - Procedono verso l'interno
    - Ogni volta che il primo viene incrementato, il secondo viene decrementato
  - Si scambiano i valori via via incontrati
- Basta un solo indice
  - Conta la distanza dagli estremi (sia destro che sinistro)
    - (i, n – i)

Inversione

- Proviamo
  - $i = 1 \rightarrow (1, n - 1)$
    - *n* è stato saltato NON FUNZIONA!!!
    - Per considerarlo si può aggiungere 1
- Riproviamo
  - $i = 1 \rightarrow (1, n - 1 + 1) = (1, n)$
  - $i = 2 \rightarrow (2, n - 2 + 1) = (2, n - 1)$
  - …
- OK!

Inversione

- Quando fermarsi?
  - Ogni elemento si scambia col simmetrico
    - Se sono pari, gli scambi sono la metà degli elementi
    - Se sono dispari, quello centrale resterà al centro
      - E' già sistemato, non si scambia
  - $\lfloor n/2 \rfloor$ scambi

Inversione
Algoritmo e Programma Pascal like

- Stabilire l'array di *n* elementi da invertire
- Calcolare il numero di scambi *r* = *n* DIV 2
- Mentre il numero di scambi *i* è minore di *r*
  - Scambiare l'*i*-mo elemento con l'(*n*–*i*+1)-mo
- Restituire l'array invertito

```
begin
    r := n DIV 2;
    for i := 1 to r do
     begin
          t := a(i);
          a(i) := a(n–i+1);
          a(n–i+1) := t
     end
end.
```

- Aggiungere dichiarazioni, acquisizione dell'array e stampa del risultato

Rimozione dei Duplicati

- Dato un array ordinato, compattarlo in modo che i valori duplicati siano rimossi

  | 3 | 3 | 5 | 8 | 10 | 10 | 10 | 14 | 20 | 20 |

  → | 3 | 5 | 8 | 10 | 14 | 20 |

  – I doppi sono adiacenti
    - Saltarli
      – Confrontare elementi a coppie
    - Spostare il successivo valore distinto nella posizione seguente al più recente valore distinto
      – Contatore degli elementi distinti

Rimozione dei Duplicati
Algoritmo

Definire l'array di *n* elementi

Impostare a 2 l'indice di scansione

Finché si trovano valori distinti

    Confrontare coppie di elementi adiacenti

Impostare il numero di valori distinti

Finché ci sono coppie da esaminare

    Se la prossima coppia non ha duplicati

        Incrementa il contatore dei valori distinti

        Sposta l'elemento di destra della coppia in tale posizione

Rimozione dei Duplicati
Note Implementative

- Inizializzazione dell'indice di scansione
  - Prima posizione dell'array
    - Confronti col valore successivo
  - Seconda posizione dell'array
    - Confronti col valore precedente
- Seconda opzione più intuitiva
  - Consente di terminare il ciclo ad *n*

Ricerca

- Determinare se (e dove) un certo elemento *x* compare in un certo insieme di *n* dati
  - Supponiamo gli elementi indicizzati 1 … *n*
    - Il fatto che l'elemento non sia stato trovato è rappresentabile tramite il valore di posizione 0
    - Deve funzionare anche per 0 elementi
  - Possibili esiti:
    - Elemento trovato nell'insieme
      - Restituirne la posizione
    - Elemento non presente nell'insieme

Ricerca Sequenziale

- Scorrimento di tutti gli elementi della sequenza, memorizzando eventualmente la posizione in cui l'elemento è stato trovato
  - Nessuna ipotesi di ordinamento
  - Utilizzabile quando si può accedere in sequenza agli elementi della lista

Ricerca Lineare Esaustiva
Algoritmo

- Scandisce tutti gli elementi dell'elenco
  - Restituisce l'ultima (posizione di) occorrenza
  - Utile quando si vogliono ritrovare tutte le occorrenze del valore

```
j ← 0
posizione ← 0
fintantoché j < n
    j ← j + 1
    se lista(j) = x allora posizione ← j
```

Ricerca Lineare Esaustiva
Programma Pascal like

- Completare con le dichiarazioni

```
begin
    j := 0;
    posizione := 0;
    while j < n do
    begin
        j := j + 1;
        if a(j) = x then posizione := j
     end
end
```

Ricerca Lineare Esaustiva
Considerazioni

- Complessità
  - Basata sul numero di confronti
    - In ogni caso: *n*
      - Si devono controllare comunque tutti gli elementi fino all'ultimo
- Soluzione più naturale
  - A volte non interessa scandire tutto l'elenco
    - Ci si può fermare appena l'elemento viene trovato

Ricerca Lineare con Sentinella
Algoritmo

- Si ferma alla prima occorrenza del valore
  - Restituisce la prima (posizione di) occorrenza
  - Utile quando
    - Si è interessati solo all'esistenza, oppure
    - Il valore, se esiste, è unico

j ← 0
posizione ← 0
fintantoché (j < n) **e** (posizione = 0)
    j ← j + 1
    se lista(j) = x **allora** posizione ← j

Ricerca Lineare con Sentinella
Programma Pascal like

- Completare con le dichiarazioni

```
begin
    j := 0;
    posizione := 0;
    while j < n and posizione = 0 do
     begin
          j := j + 1;
          if a[j] = x then posizione := j
     end
end
```

Ricerca Binaria
o Dicotomica

- Analizzare un valore interno all'elenco, e se non è quello cercato basarsi sul confronto per escludere la parte superflua e concentrarsi sull'altra parte
  - Applicabile a insiemi di dati ordinati
    - Guadagno in efficienza
    - Incentivo all'ordinamento
  - Richiede l'accesso diretto

Ricerca Binaria

- Scelta della posizione da analizzare
  - Più vicina ad uno dei due estremi
    - Caso migliore: restringe più velocemente il campo
    - Caso peggiore: elimina sempre meno elementi
  - Centrale
    - Compromesso che bilancia al meglio i casi possibili
- Necessità di ricordare la porzione valida
  - Prima posizione
  - Ultima posizione

Ricerca Binaria
Algoritmo

Fintantoché la parte di elenco da analizzare contiene più di un elemento e quello cercato non è stato trovato

Se l'elemento centrale è quello cercato

allora è stato trovato in quella posizione

altrimenti se è minore di quello cercato

allora analizzare la metà elenco successiva

altrimenti analizzare la metà elenco precedente

Ricerca Binaria
Esempio

x = 29 | 2 | 4 | 7 | 11 | 24 | 25 | 29 | 32 | 38 | 44 | 53 | 61 |

12 elementi

I tentativo ~~| 2 | 4 | 7 | 11 | 24 | 25 |~~ 29 | 32 | 38 | 44 | 53 | 61 |

↑ Eliminati 6

II tentativo | 2 | 4 | 7 | 11 | 24 | 25 | 29 | 32 | ~~38 | 44 | 53 | 61 |~~

↑ Eliminati 4

III tentativo | 2 | 4 | 7 | 11 | 24 | 25 | 29 | 32 | 38 | 44 | 53 | 61 |

↑ Trovato!

- Se fosse stato 31: non trovato in 4 tentativi

Ricerca Binaria
Programma Pascal like

```
begin
    posizione := 0;
    first := 1; last := n;
    while (first <= last) and (posizione = 0) do
     begin
          j := (first + last) DIV 2;      (* per difetto *)
          if lista(j) = x then
                posizione := j
          else
                if lista(j) < x then first := j + 1 else last := j – 1
     end
end
```

Ordinamento

- Disporre gli elementi di un insieme secondo una prefissata relazione d'ordine
  - Dipendente dal tipo di informazione
    - Numerica
      - Ordinamento numerico
    - Alfanumerica
      - Ordinamento lessicografico
  - 2 possibilità
    - Crescente
    - Decrescente
  - Altri criteri personali

Ordinamento

- Possibilità di avere elementi costituiti da più componenti
  - Associazione a ciascuno di una *chiave*
    - Lo identifica univocamente
    - Stabilisce la sua posizione
    - Unica componente rilevante per l'ordinamento
  - Esempio: Record costituito da più campi
- Supponiamo, nel seguito, di richiedere un ordinamento numerico crescente
  - Indicazione della sola chiave

Ordinamento

- Gran varietà di algoritmi
  - Basati su confronti e scambi fra gli elementi
    - Relazione d'ordine, criterio
  - Non esiste uno migliore in assoluto
    - La bontà dipende da fattori connessi ai dati su cui deve essere applicato
      - Dimensione dell'insieme di dati
        - » Numerosità
      - Grado di pre-ordinamento dell'insieme di dati
        - » Già ordinato, parzialmente, ordine opposto, casuale

Ordinamento

- Una delle attività di elaborazione più importanti
  - Stima: 30% del tempo di calcolo di un elaboratore
- Obiettivo: efficienza
  - Sfruttare “al meglio” i confronti ed i conseguenti spostamenti degli elementi
    - Piazzare gli elementi prima possibile più vicino alla loro posizione finale nella sequenza ordinata

Ordinamento

- Algoritmi esterni
  - Usano un array di appoggio
    - Occupazione di memoria doppia
    - Copia del risultato nell'array originale
- Algoritmi interni
  - Eseguono l'ordinamento lavorando sullo stesso array da ordinare
    - Scambi di posizione degli elementi

Ordinamento Esterno

- Enumerativo
  - Ciascun elemento confrontato con tutti gli altri per determinare il numero degli elementi dell'insieme che sono più piccoli in modo da stabilire la sua posizione finale

Ordinamento Interno

- Per Selezione
  - Elemento più piccolo localizzato e separato dagli altri, quindi selezione del successivo elemento più piccolo, e così via
- A bolle
  - Coppie di elementi adiacenti fuori ordine scambiate, finché non è più necessario effettuare alcuno scambio
- Per Inserzione
  - Elementi considerati uno alla volta e inseriti al posto che gli compete all'interno degli altri già ordinati

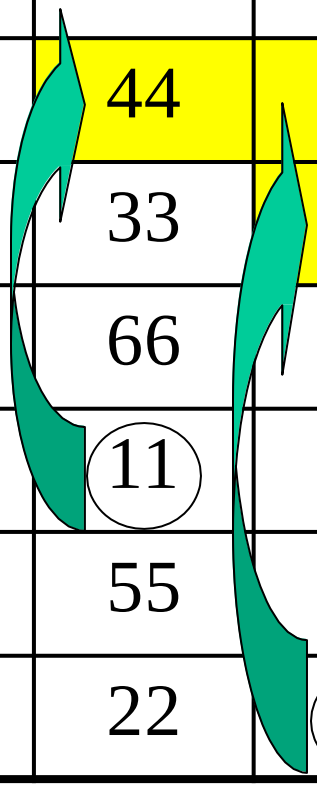
Ordinamento per Selezione

- Minimi successivi
  - Trovare il più piccolo elemento dell'insieme e porlo in prima posizione
    - Scambio con l'elemento in prima posizione
  - Trovare il più piccolo dei rimanenti (n – 1) elementi e sistemarlo in seconda posizione
  - …
  - Finché si trovi e collochi il penultimo elemento
    - Ultimo sistemato automaticamente

Ordinamento per Selezione

Esempio

	Inizio	**I**	**II**	**III**	**IV**	**V**
array(1)	44	44	11	11	11	11
array(2)	33	33	33	22	22	22
array(3)	66	66	66	66	33	33
array(4)	11	11	44	44	44	44
array(5)	55	55	55	55	55	55
array(6)	22	22	22	33	66	66



Ordinamento per Selezione
Algoritmo

$i \leftarrow 1$

mentre $i < n$ **esegui**

trova il minimo di lista (i … n)

scambia la posizione del minimo con lista(i)

$i \leftarrow i + 1$

- Utilizza algoritmi noti
  - Ricerca del minimo
  - Scambio

Ordinamento per Selezione
Programma Pascal like

```
begin
    i := 1;
    while i < n do
        min := a(i]) p := i;
        j := i + 1
        while j < = n do
            if a(j) < min then
                begin    min := a(j); p := j    end;
            j := j + 1
        endwhile
        a(p) := a(i);
        a(i) := min;
        i := i + 1
    endwhile
end.
```

Ordinamento per Selezione
Complessità

- Confronti
  - Sempre $(n - 1) * n / 2$
    - $n - 1$ al I passo di scansione
    - $n - 2$ al II passo di scansione
    - …
    - 1 all'$(n - 1)$-mo passo di scansione
- Scambi
  - Al più $(n - 1)$
    - 1 per ogni passo

Ordinamento per Selezione
Considerazioni

- Ogni ciclo scorre tutta la parte non ordinata
- Numero fisso di confronti
  - Non trae vantaggio dall'eventuale preordinamento
- Pochi scambi

Ordinamento a bolle (Bubble sort)
Programma Pascal like

```
Begin                                                    Var n, i, j, t: integer
      sort := false;                                           sort: boolean
       i:= 0                                                   min stesso tipo a(i)
    while (i < n) and (not sort) do
          sort := true;
          i := i + 1; j:= n
          while j > i do
                if a(j) < a(j-1) then
                      begin      t:= a(j); a(j):= a(j-1); a(j-1):= t; sort:= false end;
               j := j - 1
          endwhile
          i := i + 1
    endwhile
end.
```