Tecniche della Programmazione, lez.13

Problema dell'ordinamento di una sequenza di elementi

… perché serve?

perché lavorare sulla sequenza ordinata può essere utile in parecchi casi

- ricerca di un elemento
- unicità di un elemento
- numero di occorrenze
- i due elementi più vicini
- elemento più frequente

What's "Sequenza Ordinata"?

Assumiamo che la sequenza sia di **N elementi**, ciascuno di tipo **TipoElemento**

$$a_0,\ a_1,\ a_2,\ \ldots\ a_{N-1}$$

... la possiamo definire "ordinata" se

- esiste una **relazione d'ordine** "<" (oppure "≤")
  applicabile agli elementi $a_0,\ a_1,\ \ldots\ a_{N-1}$

&&

- per ogni coppia $a_i,\ a_{i+1}$ vale $a_i < a_{i+1}$ (oppure $a_i \le a_{i+1}$)

a

4	9	16	18	21	30

a[0] ≤ a[1] ≤ a[2] ≤ a[3] ≤ a[4] ≤ a[5]

ovviamente viene naturale rappresentare la sequenza come un array ... ma non è l'unico modo ... come vedremo ...

What's "Ordinamento"? (anche "Sorting")

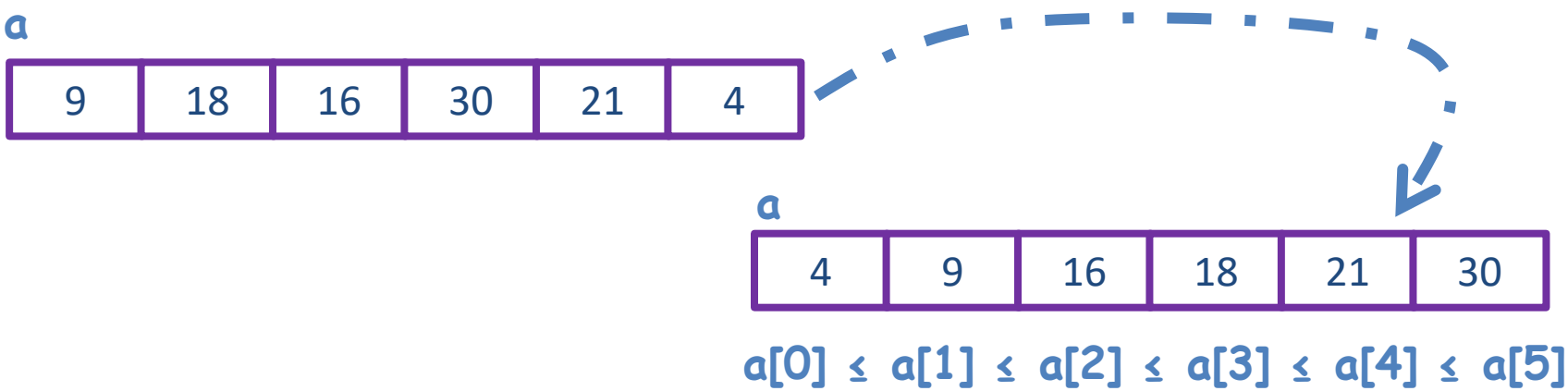
Problema di ORDINAMENTO

Data una sequenza $a_0, a_1, a_2, a_3, \ldots a_{N-1}$ modificarla, spostando gli elementi, in modo che sia ordinata.

NB

Si può modificare la sequenza, oppure ottenere una nuova sequenza, $b_0, b_1, b_2, \ldots b_{N-1}$, fatta con i medesimi elementi ma ORDINATA)

Assumeremo che gli elementi siano INTERI e contenuti in un array, a
ALLORA, il PROBLEMA DELL'ORDINAMENTO è il problema di spostare opportunamente gli elementi dell'array, se e quando serve, in modo che a diventi ordinato.



Un algoritmo di ordinamento: sort per scambio, sort a bolle

A Sorting algorithm: Bubble Sort

si procede scorrendo l'array e confrontando gli elementi vicini (a[j],a[j+1]): se i due non sono in ordine tra loro, li si scambia ... E si scorre l'array più volte ...

ordinamento ... crescente

```
per ogni coppia a[j], a[j+1]
  se a[j]>a[j+1]
     scambia a[j] con a[j+1]
```

codice per la parte di algoritmo qui sopra?

☺

N=6 int a[N]

30 16 31 9 18 4

30 16 31 9 18 4 j,j+1=0,1

stato attuale	*16 30 31 9 18 4*

16 30 31 9 18 4 j,j+1=1,2

stato attuale	*16 30 31 9 18 4*

Un algoritmo di ordinamento: sort per scambio, sort a bolle

A Sorting algorithm: Bubble Sort ... PARZIALE

```
per ogni coppia a[j], a[j+1]
  se a[j]>a[j+1]
     scambia a[j] con a[j+1]
```

```
for (j=0; j<N-1; j++){
  if (a[j]>a[j+1])
     scambia a[j] con a[j+1]
}
```

N=6

int a[N]

swap

30 16 31 9 18 4 j,j+1=0,1

16 30 31 9 18 4 1,2 no swap

stato attuale	*16 30 31 9 18 4*

16 30 31 9 18 4 2,3 swap

stato attuale	*16 30 9 31 18 4*

16 30 9 31 18 4 3,4 swap

stato attuale	*16 30 9 18 31 4*

Un algoritmo di ordinamento: sort per scambio, sort a bolle

A Sorting algorithm: Bubble Sort ... PARZIALE

```
per ogni coppia a[j], a[j+1]
  se a[j]>a[j+1]
     scambia a[j] con a[j+1]
```

```
for (j=0; j<N-1; j++){
  if (a[j]>a[j+1])
     scambia a[j] con a[j+1]
}
```

N=6

int a[N]

30 16 31 9 18 4	j,j+1=0,1	swap
16 30 31 9 18 4	1,2	no swap
16 30 31 9 18 4	2,3	swap
16 30 9 31 18 4	3,4	swap
16 30 9 18 31 4	4,5	swap

stato attuale *16 30 9 18 4 31*

16 30 9 18 4 **31** **j==5 stop**

Un algoritmo di ordinamento: sort per scambio, sort a bolle

A Sorting algorithm: Bubble Sort ... PARZIALE

Abbiamo fatto una "passata"

```
per ogni coppia a[j], a[j+1]
  se a[j]>a[j+1]
     scambia a[j] con a[j+1]
```

```
for (j=0; j<N-1; j++){
  if (a[j]>a[j+1])
     scambia a[j] con a[j+1]
}
```

Prima passata! (n-1 controlli): l'array non è ordinato, ma il più grande è "salito" al suo posto

N=6

int a[N]

30 16 31 9 18 4	j,j+1=0,1	swap
16 30 31 9 18 4	1,2	no swap
16 30 31 9 18 4	2,3	swap
16 30 9 31 18 4	3,4	swap
16 30 9 18 31 4	4,5	swap

stato attuale *16 30 9 18 4 31*

16 30 9 18 4 **31** **j==5 stop**

Un algoritmo di ordinamento: sort per scambio, sort a bolle

A Sorting algorithm: Bubble Sort ... PARZIALE

```
per ogni coppia a[j], a[j+1]
  se a[j]>a[j+1] scambia a[j] con a[j+1]
```

```
for (j=0; j<N-1; j++){
  if (a[j]>a[j+1])
    scambia a[j] con a[j+1]
}
```

NB

N=6

int a[N]

30	16	31	9	18	4	j,j+1=0,1
16	30	31	9	18	4	1,2
16	30	31	9	18	4	2,3
16	30	9	31	18	4	3,4
16	30	9	18	31	4	4,5
16	30	9	18	4	**31**	**j==5 stop**

NB

che succede se mettiamo `j<=N-1`?

Prova a rispondere.
Vedi Esercizi e poi prosegui

A Sorting algorithm: Bubble Sort

si procede scorrendo l'array e confrontando gli elementi vicini (a[j],a[j+1]): se i due non sono in ordine tra loro, li si scambia ... **E si scorre l'array più volte ...**

ordinamento ... crescente

Algoritmo = sequenza di passate

(dopo n-1 passate l'array è ordinato)

16 30 9 18 4 **31**	j=0
16 30 9 18 4 **31**	j=1
16 9 30 18 4 **31**	j=2
16 9 18 30 4 **31**	j=3
16 9 18 4 **30 31**	~~j=4~~
	~~j=5~~

"passata"

```
for (j=0; j<N-2; j++)
  if (a[j]>a[j+1])
    scambia a[j] con a[j+1]
```

Seconda passata (n-2 controlli) il penultimo a posto

A Sorting algorithm: Bubble Sort

TERZA passata (n-3 controlli) il terz'ultimo a posto

16 9 18 4 **30 31**	j=0
9 16 18 4 **30 31**	j=1
9 16 18 4 **30 31**	j=2
9 16 4 **18 30 31**	

terza "passata"

```
for (j=0; j<N-3; j++)
  if (a[j]>a[j+1])
    scambia a[j] con a[j+1]
```

(dopo N-3, cioe` 3, passate, i tre elementi piu` grandi sono al loro posto!

A Sorting algorithm: Bubble Sort

QUARTA passata (N-4 controlli)

prima 9 16 4 **18 30 31**

dopo 9 4 **16 18 30 31**

"passata"

```
for (j=0; j<N-4; j++)
  if (a[j]>a[j+1])
    scambia a[j] con a[j+1]
```

QUINTA passata cioè (N-1)-esima passata (N-5 = N-N+1 = 1 controllo)

dopo 4 **9 16 18 30 31**

quinta "passata"

```
for (j=0; j<N-5; j++)
  if (a[j]>a[j+1])
    scambia a[j] con a[j+1]
```

(dopo N-1 passate l'array è ordinato) 4 9 16 18 30 31

A Sorting algorithm: Bubble Sort, funzione bubbleSort()

i-esima passata

```
for (j=0; j<N-i; j++)
  if (a[j]>a[j+1])
    scambia a[j] con a[j+1]
```

algoritmo

Con i che conta da 1 a N-1 

esegui la i-esima passata

```
/* ordina l'array arr di N elementi */
void bubbleSort (int a[N]) {
 int i, j, aux;
 for (i=1; i<=N-1; i++)
   for (j=0; j<N-i; j++)
     if (a[j]>a[j+1]) {
       aux = a[j];
       a[j] = a[j+1];
       a[j+1] = aux;
     }
return;
}
```

A Sorting algorithm: Bubble Sort

Applicazione sull'array **16 8 2 15 4 9**

16 8 15 2 4 9 j=0

8 16 15 2 4 9 j=1

...

8 15 2 4 9 **16** j=5

Prima passata
5 confronti e
5 scambi

---

8 2 4 9 **15 16**

Seconda passata n-2 confronti
e 3 scambi

---

2 4 8 9 **15 16**

Terza passata n-3 confronti
e 2 scambi

---

2 4 8 9 15 16

Quarta passata, n-4 conf. e 0 scambi

---

2 4 8 9 15 16

Quinta passata, n-5 conf. e 0 scambi

---

2 4 8 9 15 16 !!

A Sorting algorithm: Bubble Sort

Applicazione sull'array **16 18 20 25 34 39**

16 18 20 25 34 39

Prima passata	**n-1 confronti**	**e**	**0 scambi**
seconda	**n-2**	**e**	**0**
terza	**n-3**	**e**	**0**
quarta	**n-4**	**e**	**0**
quinta	**n-5**	**e**	**0**

L'array potrebbe essere già ordinato prima del termine delle n-1 passate: dipende dalla configurazione della sequenza da ordinare

L'algoritmo però esegue sempre N-1 passate

e l'i-esima passata esegue sempre N-i confronti (a[i] > a[i+1]) con eventuale scambio

... si può migliorare, cercando di evitare le passate inutili, il più possibile ...

A Sorting algorithm: Bubble Sort

miglioramento

Mentre non abbiamo finito {

esegui una passata;

~~**se l'array risulta ordinato abbiamo finito**~~

se durante la passata non abbiamo eseguito scambi, abbiamo finito

}

```
while (finito==0) {      /* mentre non ... finito */
  fattoscambio=0;
  for (j=0; ... )
    if (a[j]>a[j+1]) {
      scambia a[j] con a[j+1]
      fattoscambio = 1;
    }
  if (fattoscambio==0)
      finito=1;
```

MA
- init finito
- manca i ... in parte sostituita da finito, ok, ma ci servirebbe ancora, come contatore per guidare la passata (j va da zero a ... N-i ... se non c'è i come si fa?)

A Sorting algorithm: Bubble Sort

```
finito=0;
i=0;       /* fatte 0 passate */
while (finito==0) {
  i=i+1;
  fattoscambio=0;
  for (j=0; j<N-i; j++ )
    if (a[j]>a[j+1]) {
      scambia a[j] con a[j+1]
      fattoscambio = 1;
    }
  if ((fattoscambio==0)||(i==N-1))
     finito=1;
}
```

A Sorting algorithm: Bubble Sort, funzione migliorata: bSort()

```
void bsort (double a[N]) {
 int i, j, finito, fattoscambio;
 finito=0;
 i=0;    /* fatte 0 passate */
 while (finito==0) {
   i=i+1;
   fattoscambio=0;
   for (j=0; j<N-i; j++ )
     if (a[j]>a[j+1]) {
       scambia a[j] con a[j+1]
       fattoscambio = 1;
     }
   if ((fattoscambio==0)||(i==N-1))
     finito=1;
 }
return;
}
```

A Sorting algorithm: Bubble Sort: ANALISI

Per Analisi, qui intendiamo in sostanza, cercare di capire **quanto "costa"** ordinare un array con il bubble sort, usando la funzione bubbleSort() o quella bSort().

Di solito il "costo" di un algoritmo viene misurato considerando **quanto tempo** ci mette l'algoritmo a fornire un risultato. **Anche lo spazio di memoria** usato dall'algoritmo entra in gioco, ma nel nostro caso attuale (ordinamento di un array) risulta secondario.

E` ragionevole aspettarsi che ordinare un array di 100 elementi costi meno che ordinare un array di 1000000 di elementi. **Per l'ordinamento di un array si dice che il numero di elementi dell'array e` la "dimensione"** del problema.

Il calcolo esatta del costo e` complicato, quindi si tende a ragionare per **"ordini di grandezza"** rispetto alla dimensione del problema.

- O(n): il tempo di esecuzione dell'algoritmo cresce linearmente al crescere della dimensione (stampa array, ricerca in array)
- $O(n^2)$: il tempo di esecuzione cresce quadraticamente al crescere della dimensione (cioè al crescere di n). E` abbastanza il nostro bubble caso ☺.
- O(log n): il tempo di esecuzione è logaritmico rispetto ad n, cresce molto lentamente al crescere di n.
- $O(2^n)$: il tempo di esecuzione cresce esponenzialmente rispetto alla dimensione del problema.

L'ordine di grandezza da` un'idea del costo (ad esempio se possiamo aspettarci di vedere il risultato in secondi, minuti, ... secoli, ... ere geologiche ...

Per calcolare il costo, secondo questi termini, si tede a considerare il punto focale dell'algoritmo, quello che determina la parte di ordine maggiore ... si chiama **Istruzione Dominante** (magari non e` l'unica ... ma ne basta una ...).

L'analisi procede anche considerando i **principali casi** in cui il problema si puo` presentare (casi in cui l'algoritmo ha performance molto migliori, casi in cui il comportamento e` nella media, casi in cui si raggiungono le performance peggiori ... e spiace dirlo, ma il costo certificato dell'algoritmo e` usualmente quello del caso peggiore ...

A Sorting algorithm: Bubble Sort: ANALISI

N e` la dimensione del problema.

L'istruzione dominante e` quella che viene ripetuta piu` volte, cioe` queIla all'interno del ciclo su j ...

Istruzione dominante

```
if (a[j]>a[j+1]) {
        scambia a[j] con a[j+1]
        fattoscambio = 1;
      }
```

16 18 20 25 34 39

Caso migliore: dopo la prima passata (N-1 confronti e 0 scambi) finiamo

(lineare in N)

30 16 31 9 18 4

Caso peggiore: numero di confronti massimo

39 34 25 20 18 16

Caso #!%@!!! : massimo numero di confronti, ognuno con scambio e assegnazione

Il miglioramento permette di mettere a frutto eventuali configurazioni favorevoli (ad es. array parzialmente ordinato): nei casi sfavorevoli si comporta come la versione iniziale; nei casi favorevoli guadagna un po'.

A Sorting algorithm: Bubble Sort: ANALISI

Quanti confronti?

Usiamo n invece di N nelle formule

prima passata	n-1 confronti
seconda passata	n-2 confronti
...	
(i-esima passata)	n-i confronti
...	
(n-2)-esima passata	2 confronti
(n-1)-esima passata	1 confronto

Caso peggiore: n-1 passate

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

costo proporzionale a n^2

quadratico in n: $O(n^2)$

(bSort() e bubbleSort() hanno la stessa complessità)

Caso migliore: 1 passata **unica passata: n-1 confronti**

Si presenta solo per la bSort() (miglioramento)

costo proporzionale a n

lineare in n: $O(n)$

Un altro algoritmo di ordinamento

Another Sorting algorithm: *Selection Sort*

si seleziona il minimo della porzione non ancora sistemata e lo si mette al primo posto di tale porzione … … porzione??

0) l'elemento più piccolo dell'array viene messo in a[0]
(NB è il minimo in a[0]-a[N-1])

(ora "a[0] è a posto!")

1) l'elemento più piccolo della porzione a[1]-a[N-1] viene messo in a[1]

(ora a[0] e a[1] sono "a posto!")

2) l'elemento più piccolo della porzione a[2]-a[N-1] viene messo in a[2]

…

(ora a[0] a[1] e a[2] sono "a posto!")

N-2) l'elemento minimo della porzione a[N-2]-a[N-1] viene messo in a[N-2]

(remember, a[N-1] è l'ultimo elemento di a)

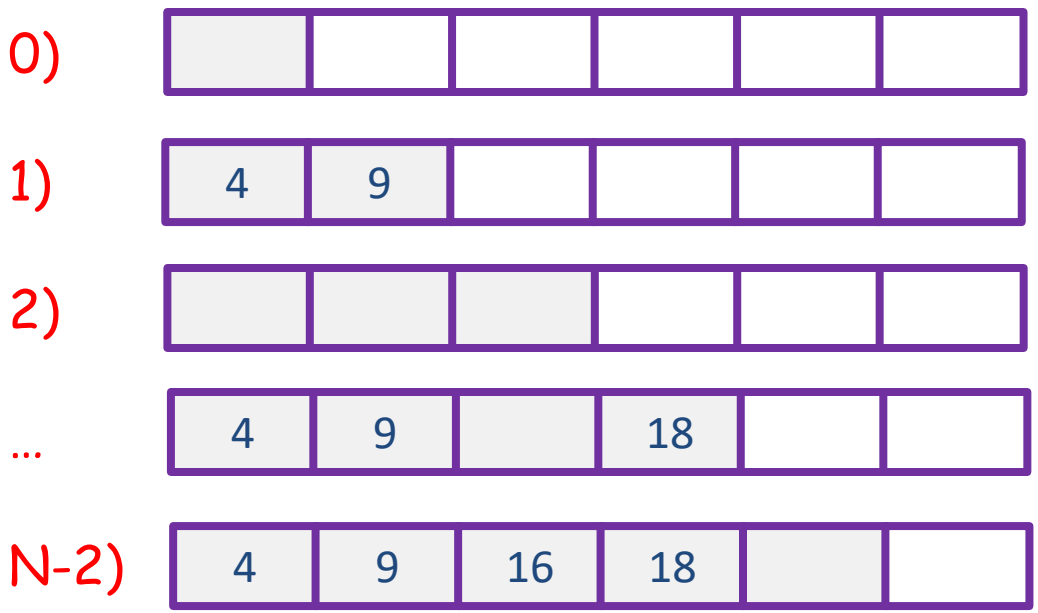
--- stop, dopo N-1 passi sono tutti a posto

Another Sorting algorithm: Selection Sort

applicazione intuitiva: ☺ disegnare i 5 stati successivi dell'array

`int a[6]` N=6

9	18	16	4	30	21



0) l'elemento più piccolo dell'array viene messo in a[0]

1) l'elemento più piccolo della porzione a[1]-a[N-1] viene messo in a[1]

2) l'elemento più piccolo della porzione a[2]-a[N-1] viene messo in a[2]

...

N-2) l'elemento più piccolo della porzione a[N-2]-a[N-1] viene messo in a[N-2]

--- stop,

NB
quando mettiamo il minimo della porzione a[i]...a[N-1] in a[i], l'elemento che era in a[i] non deve andare perso ... quindi lo mettiamo dove stava il minimo ... (scambio)

Another Sorting algorithm: Selection Sort

applicazione intuitiva
disegnare i 5 stati successivi dell'array

`int a[6]` N=6

9	18	16	4	30	21

0)	4	18	16	9	30	21	scambio tra 4 e 9 (a[0] e a[3])
1)	4	9	16	18	30	21	scambio tra a[1] e a[3]
2)	4	9	16	18	30	21	16 rimane a[2]
3)	4	9	16	18	30	21	18 rimane in a[3]
N-2)	4	9	16	18	21	30	scambio tra a[N-2] e a[N-1]

ora anche a[N-1] è a posto

NB
quando mettiamo il minimo della porzione a[i]...a[N-1] in a[i], l'elemento che era in a[i] non deve andare perso ... quindi scambiamo le posizioni del minimo e di a[i] ...

Another Sorting algorithm: Selection Sort

applicazione intuitiva
disegnare i 5 stati successivi dell'array

`int a[6]` N=6

9	18	16	4	30	21

0)	4	18	16	9	30	21
1)	4	9	16	18	30	21
2)	4	9	16	18	30	21
3)	4	9	16	18	30	21
N-2)	4	9	16	18	21	30

scambio tra 4 e 9 (a[0] e a[3])

scambio tra a[1] e a[3]

16 rimane a[2]

18 rimane in a[3]

scambio tra a[N-2] e a[N-1]
ora anche a[N-1] è a posto

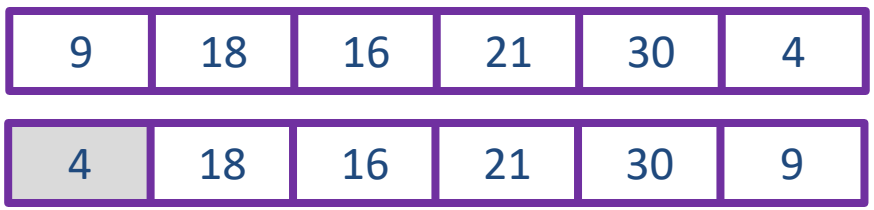
chiamare "0" il primo passo è una buona tattica quando si lavora con gli array.
Notare la corrispondenza tra ... la numerazione di un passo
... e l'indice dell'elemento sistemato dal passo ...

Selection Sort: algoritmo

0) lavoriamo su int a[N], ci serve 'min', ... i ...
1) per i da 0 a N-2
   1.1) trovare 'min' tale che a[min] è il minimo elemento in a[i]-a[N-1]
   1.2) scambiare a[i] con a[min]
2) fine

`int a[6]` N=6

i=0

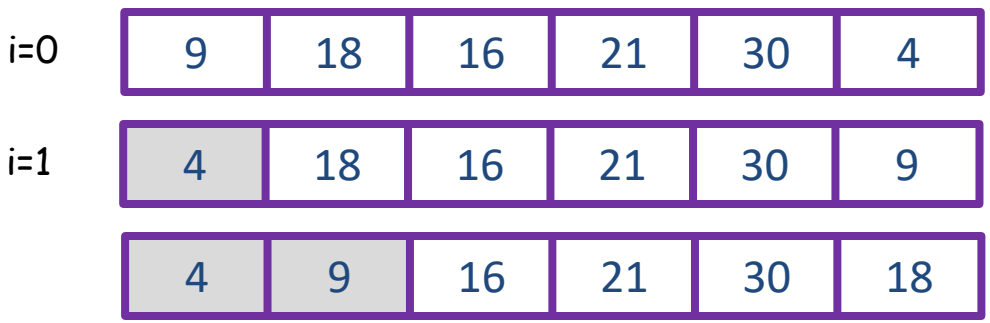


il minimo in a[0]-a[5] è in a[5], cioè min==5: scambiare a[0] con a[5]

Selection Sort: algoritmo

0) lavoriamo su int a[N], ci serve 'min', ... i ...
1) per i da 0 a N-2
   1.1) trovare 'min' tale che a[min] è il minimo elemento in a[i]-a[N-1]
   1.2) scambiare a[i] con a[min]
2) fine

`int a[6]` N=6



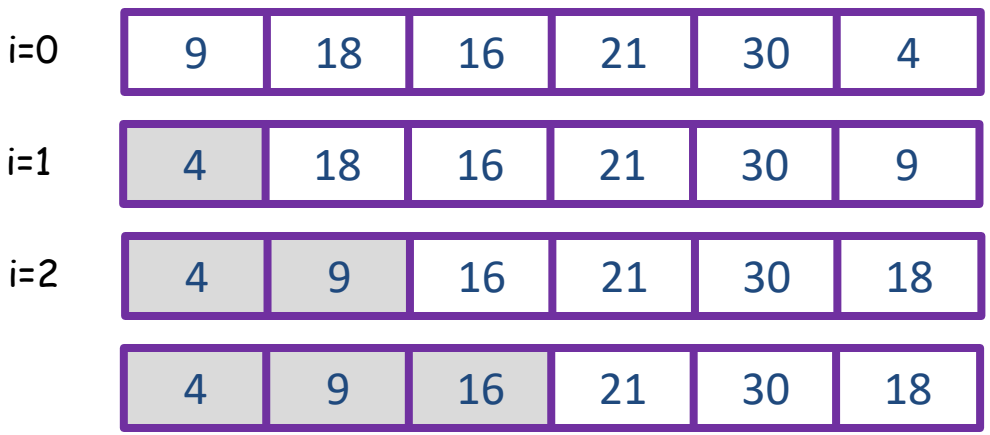
il minimo in a[0]-a[5] è in a[5], cioè min==5:
scambiare a[0] con a[5]

minimo in a[1]-a[5] e` in a[5]: min==5
scambiare a[1] con a[min]

Selection Sort: algoritmo

0) lavoriamo su int a[N], ci serve 'min', ... i ...
1) per i da 0 a N-2
   1.1) trovare 'min' tale che a[min] è il minimo elemento in a[i]-a[N-1]
   1.2) scambiare a[i] con a[min]
2) fine

`int a[6]` N=6



il minimo in a[0]-a[5] è in a[5], cioè min==5:
scambiare a[0] con a[5]

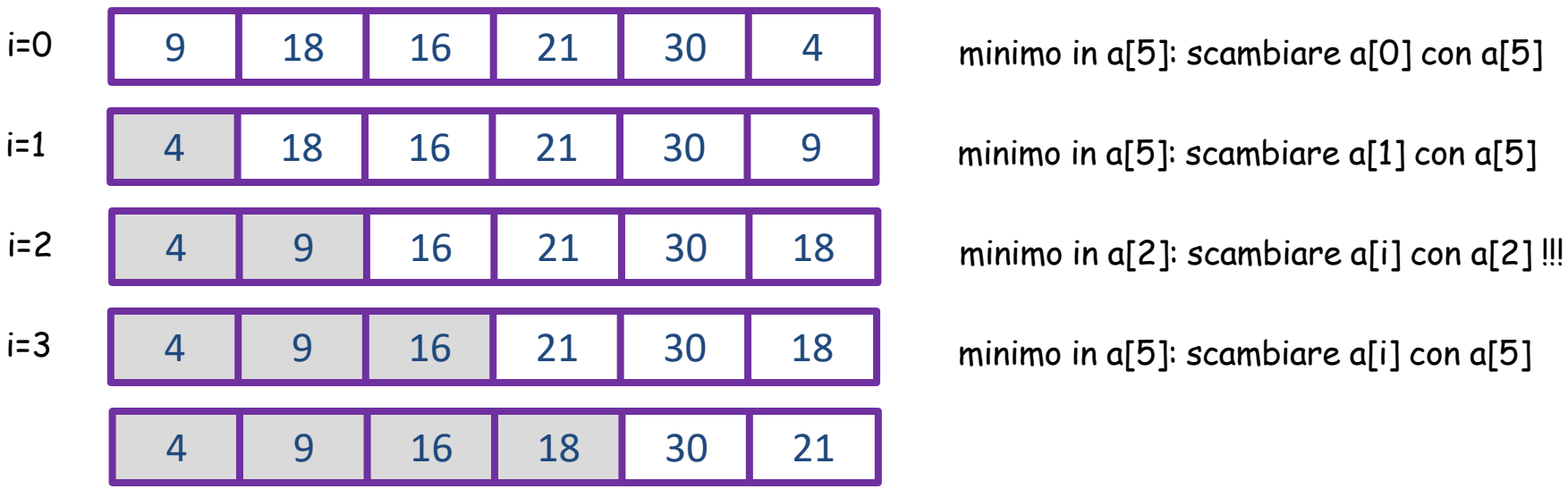
minimo in a[1]-a[5] e` in a[5]: min==5
scambiare a[1] con a[min]

porzione a[2]-a[5] : min==2;
scambio a[2] a[min]

Selection Sort: algoritmo

0) lavoriamo su int a[N], ci serve 'min', ... i ...
1) per i da 0 a N-2
   1.1) trovare 'min' tale che a[min] è il minimo elemento in a[i]-a[N-1]
   1.2) scambiare a[i] con a[min]
2) fine

`int a[6]` N=6



Selection Sort: algoritmo

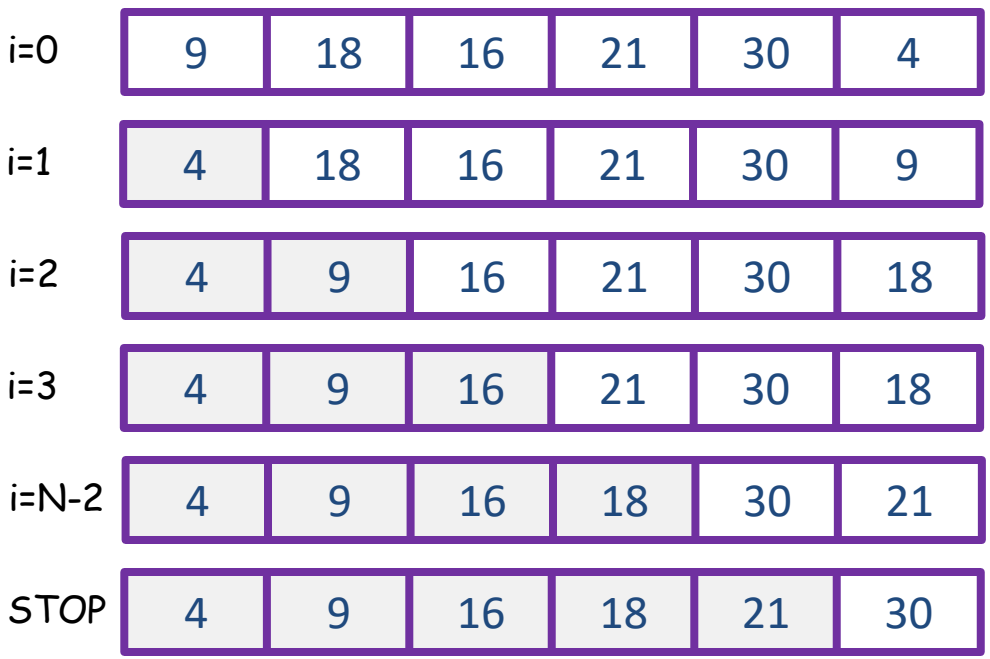
0) lavoriamo su int a[N], ci serve 'min', ... i ...
1) per i da 0 a N-2
   1.1) trovare 'min' tale che a[min] è il minimo elemento in a[i]-a[N-1]
   1.2) scambiare a[i] con a[min]
2) fine

`int a[6]` N=6

Selection Sort: algoritmo

0) lavoriamo su int a[N], ci serve 'min', ... i ...
1) per i da 0 a N-2
   1.1) trovare 'min' tale che a[min] è il minimo elemento in a[i]-a[N-1]
   1.2) scambiare a[i] con a[min]
2) fine

`int a[6]` n=6



minimo in a[5]: scambiare a[0] con a[5]

minimo in a[5]: scambiare a[1] con a[5]

minimo in a[2]: scambiare a[i] con a[2] !!!

minimo in a[5]: scambiare a[i] con a[5]

minimo in a[5]: scambiare a[4] con a[5]

a[5] può solo essere al posto giusto - STOP

Selection Sort: funzione che implementa l'algoritmo

funzione selectionSort(?)

0) void; riceve int a[N]; usa i, min … j

1) per i da 0 a N-2
   1.1) trovare 'min' tale che a[min] è il minimo elemento in a[i]-a[N-1]

      1.1.1) ricerca del minimo in a[i]-a[N-1]
         - min inizializzato con i
         - ciclo con tecnica del "minimo parziale", scorrendo gli elementi a[i+1]-a[N-1]

```
per j da  i+1  a  N-1
    se a[j] < a[min]    min cambia
```

   1.2) scambiare a[i] con a[min]

2) fine

funzione selectionSort()

```
void selectionSort (int arr[N]) {
  int i, j, min;

 for (i=0; i<N-1; i++) {          /* min è l'indice del minimo parz */
    min=i;                        /* min inizializzato con i */
    for (j=i+1; j<=N-1; j++)
       if (arr[j] < arr[min])
          min=j;

    …                  /*scambio tra arr[i] e arr[min] */   … come si fa? ☺
 }
return;
}
```

funzione selectionSort()

```
void selectionSort (int arr[N]) {
  int i, j, min, aux;              /* aux usato per lo scambio */

 for (i=0; i<N-1; i++) {
                                   /* min sarà l'indice del minimo */
    min=i;                         /* init minimo parziale */

    for (j=i+1; j<=N-1; j++)       /* calcolo del minimo */
       if (arr[j] < arr[min])
          min=j;

                                   /*scambio tra arr[i] e arr[min] */
    aux = arr[i];
    arr[i] = arr[min];
    arr[min] = aux;
 }
return;
}
```

funzione selectionSort() --- analisi

```
void selectionSort (int arr[N]) {
  int i, j, min, aux;
 for (i=0; i<N-1; i++) {
    min=i;
    for (j=i+1; j<=N-1; j++)
       if (arr[j] < arr[min])
          min=j;

    ... /* scambio tra arr[i] e arr[min] */
 }
return;
}
```

Quanto ci vuole ad eseguire il sort?

quante volte viene eseguita min=i ?

quante volte viene eseguito il confronto? Più di min=i? ☺

quante volte viene eseguita min=j ?

funzione selectionSort() --- analisi

```
void selectionSort (int arr[N]) {
  int i, j, min, aux;
 for (i=0; i<N-1; i++) {
    min=i;
    for (j=i+1; j<=N-1; j++)
       if (arr[j] < arr[min])
          min=j;

    ... /* scambio tra arr[i] e arr[min] */
 }
return;
}
```

Quanto ci vuole ad eseguire il sort?

quante volte viene eseguita min=i ? N-1

quante volte viene eseguito il confronto ? più di tutti

quante volte viene eseguita min=j ? mah! … tra 0 e #confronti

funzione selectionSort() --- analisi

Usiamo n invece di N nelle formule

```
void selectionSort (int arr[N]) {
  int i, j, min, aux;
 for (i=0; i<N-1; i++) {
    min=i;
    for (j=i+1; j<=N-1; j++)
       if (arr[j] < arr[min])
          min=j;

    ... /*
 }
return;
}
```

Quindi consideriamo il confronto arr[j]<arr[min] come dominante.
Questo viene eseguito

- **n-1 volte alla prima passata**
- **n-2 …**
- **… n-k volte alla k-esima**
- **1 volta alla n-1-esima passata**

cioè un numero di volte proporzionale a n^2

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

E questo vale sempre ...
Dal punto di vista del numero di confronti non c'è un caso peggiore.
Se proprio vogliamo identificare un caso #!%@!!!, questo potrebbe essere quando l'array è così messo male che min=j viene eseguito sempre, per ogni j … costa un po' di più, ma l'ordine di grandezza del costo è sempre **n^2**: $O(n^2)$

Ulteriore modularizzazione di selectionSort()

Ora che abbiamo analizzato l'algoritmo complessivo, potremmo scrivere una funzione selectionSort2() concepita in modo piu` modulare ...
Rivediamo l'algoritmo ...

0) void; selectionSort() riceve int a[N]; usa i, min ...

1) per i da 0 a N-2
   1.1) trovare 'min' tale che a[min] è il minimo elemento in a[i]-a[N-1]
   sottoproblema!
   Usiamo una funzione minimo() per assegnare min

   1.2) scambiare a[i] con a[min]

2) fine

Scrivi la funzione selectionSort2(), che ottiene lo stesso risultato di selectionSort, ma usa una funzione minimo() per assegnare min nel ciclo principale. Scrivi direttamente il codice, con la chiamata di minimo() che ritieni piu` giusta.
Vedi Approfondimenti

Tecniche della Programmazione, lez. 13

- Esercizi

Un algoritmo di ordinamento: sort per scambio, sort a bolle

A Sorting algorithm: Bubble Sort ... PARZIALE

```
per ogni coppia a[j], a[j+1]
  se a[j]>a[j+1]
     scambia a[j] con a[j+1]
```

```
for (j=0; j<N-1; j++){
  if (a[j]>a[j+1])
     scambia a[j] con a[j+1]
}
```

NB

NB

che succede qui?

verificare mediante l'esecuzione simulata e poi passare alla prossima slide

N=6

int a[N]

30	16	31	9	18	4	j,j+1=0,1
16	30	31	9	18	4	1,2
16	30	31	9	18	4	2,3
16	30	9	31	18	4	3,4
16	30	9	18	31	4	4,5
16	30	9	18	4	**31**	**j==5 stop**

Un algoritmo di ordinamento: sort per scambio, sort a bolle

A Sorting algorithm: Bubble Sort

```
per ogni coppia a[j], a[j+1]
  se a[j]>a[j+1]
     scambia a[j] con a[j+1]
```

```
for (j=0; j<=N-1; j++){
  if (a[j]>a[j+1])
     scambia a[j] con a[j+1]
}
```

N=6

int a[N]

30	16	31	9	18	4	j,j+1=0,1
16	30	31	9	18	4	1,2
16	30	31	9	18	4	2,3
16	30	9	31	18	4	3,4
16	30	9	18	31	4	4,5
16	30	9	18	4	**31**	

succede che con l'iterazione quinta abbiamo scambiato e messo a posto il 31 e il 4, quindi ora la situazione e questa

e ora, con **j=N-1** eseguiamo una sesta iterazione, con **j=5 e j+1=6**, per cui il confronto **a[j]>a[j+1] e` tra** a[5] e ...

ok, adesso trorna alla lezione

Tecniche della Programmazione, lez. 13

- Approfondimenti

Ulteriore modularizzazione di selectionSort()

Ora che abbiamo analizzato l'algoritmo complessivo, potremmo scrivere una funzione selectionSort2() concepita in modo piu` modulare ...
Rivediamo l'algoritmo ...

0) void; selectionSort() riceve int a[N]; usa i, min ...

1) per i da 0 a N-2
   1.1) trovare 'min' tale che a[min] è il minimo elemento in a[i]-a[N-1]
   sottoproblema!
   Usiamo una funzione minimo() per assegnare min

   1.2) scambiare a[i] con a[min]

2) fine

Scrivi la funzione selectionSort2(), che ottiene lo stesso risultato di selectionSort, ma usa una funzione minimo() per assegnare min nel ciclo principale.
Scrivi direttamente il codice, con la chiamata di minimo() che ritieni piu` giusta.
provaci e poi prosegui

funzione selectionSort2()

```
void selectionSort2 (int arr[N]) {
  int i, j, min, aux;            /* aux usato per lo scambio */

 for (i=0; i<N-1; i++) {
    min = minimo(arr, i);        /* assegnazione di min */

    aux = arr[i];                 /*scambio tra arr[i] e arr[min] */
    arr[i] = arr[min];
    arr[min] = aux;
 }
return;
}
```

E solo scrivendo la chiamata abbiamo progettato la funzione **minimo()**
Ora non resta che scriverla: l'algoritmo e` in sostanza il punto 1.1.1 dell'algoritmo selectionSort() originale.

1.1.1) ricerca del minimo in a[i]-a[N-1]
- min inizializzato con i
- ciclo con tecnica del "minimo parziale", scorrendo gli elementi a[i+1]-a[N-1]

per j da i+1 a N-1
se a[j] < a[min] min cambia

scrivi l'algoritmo per la funzione minimo (NOTA ... potresti aver scritto una chiamata diversa e ancora aver fatto bene ... dopo vediamo
provaci e poi prosegui

Ulteriore modularizzazione di selectionSort()

Algoritmo per la funzione minimo(), che
riceve un array 'a' di N interi, e un indice 'k'
e restituisce il minimo della porzione a[k]-a[N-1]

0) restituisce un intero; riceve array a[N] di int e indice k;
usa j, indiceMin …

1) indiceMin inizializzato con k

2) per j da k+1 a N-2
se a[j] < a[indiceMin] indiceMin = j (ora è questo l'indice del minimo finora analizzato)

3) fine, restituire indiceMin

definisci la funzione minimo
provaci e poi prosegui

funzione minimo()

```
void minimo (int a[N], int k) {
  int j, indiceMin;

 indiceMin = k;    /* init minimo parziale */
 for (j=k+1; j<=N-1; j++)
    if (arr[j] < arr[indiceMin])
          min=j;   /* (*) aggiornamento minimo parziale */

return indiceMin; /* quando arriviamo qui indiceMin contiene k, se nessun
altro indice, da k a N-1, corrispondeva ad un valore minore di a[k], oppure dopo
l'ultimo aggiornamento (*) contiene l'indice del minimo visto durante lo scorrimento
dell'array a  init minimo parziale */
}
```

prova selectionSort2 e minimo in un programma (puoi inizializzare in definizione l'array con i dati di prova)
provaci e poi prosegui

funzione minimo()

dunque, che succede se minimo() viene chiamata con parametri che la fanno uscire dall'array? 

ad esempio con k troppo grande, oppure negativo?

aggiungi a minimo qualche controllo, in modo che restituisca un dato corretto se possibile, oppure uno zero con annessa stampa diagnostica

```
void minimo (int a[N], int k) {
  int j, indiceMin  ;

 indiceMin = k;       /* init minimo parziale */
 for (j=k+1; j<=N-1; j++)
    if (arr[j] < arr[indiceMin])
          min=j;      /* (*) aggiornamento minimo parziale */

return indiceMin;     /* quando arriviamo qui indiceMin contiene k, se nessun altro
indice, da k a N-1, corrispondeva ad un valore minore di a[k], oppure dopo l'ultimo aggiornamento
(*) contiene l'indice del minimo visto durante lo scorrimento dell'array a  init minimo parziale
*/
}
```

funzione minimo(), modificata

```
void minimo (int a[N], int k) {
   int i, indiceMin;
/* se k<=N-2 va bene e proseguiamo normalmente: i casi da controllare sono altri
-  se k==N-1 il ciclo esce dall'array   quindi facciamo return prima del ciclo,
   restituendo a[N-1] che un minimo di senso lo ha;
-  se k e` >N-1 o addirittura negativo, non possiamo restituire nulla di sensato:
   restituiamo zero, facendo anche una stampa di avvertimento */

if ( (k>N-1) || (k<0) ){
    printf("\n\n\n eeeekkkkkk!!!\n\n\n");
    return 0;
}
if ( k==N-1 ) return k;


/* caso ragionevole */
 indiceMin = k;    /* init minimo parziale */
 for (i=k+1; i<=N-1; i++)
     if (arr[i] < arr[indiceMin])
           min=j;   /* (*) aggiornamento minimo parziale */

return indiceMin;
}
```

di seguito una soluzione diversa, per selectionSort2() tra quelle che potevano venire in mente

funzione selectionSort2()

```
void selectionSort2 (int arr[N]) {
  int i, j, min, aux;            /* aux usato per lo scambio */

 for (i=0; i<N-1; i++) {
    min = minimo2(arr, i, N-1);      /* assegnazione di min */

    aux = arr[i];                  /*scambio tra arr[i] e arr[min] */
    arr[i] = arr[min];
    arr[min] = aux;
 }
return;
}
```

In questo caso la funzione minimo2() e` progettata per avere due parametri indice: un indice iniziale della porzione ed un indice finale.
Ovviamente in questo caso l'indice finale nella chiamata in selectionSort2() e` sempre N-1.

Ma la funzione minimo2() e` piu` generale della minimo(), dato che potrebbe un giorno essere usata per trovare il minimo in una porzione qualsiasi di un array , delimitata dai due parametri indice.
Qui pero` va controllato di non sforare dall'array; il controllo addizionale, rispetto a quelli visti prima, e` su h, sempre assumendo di avere una costante N con la quale abbiamo definito l'array dati.

scrivi l'algoritmo per la funzione minimo2()
provaci e poi prosegui

Ulteriore modularizzazione di selectionSort()

Algoritmo per la funzione minimo2(), che riceve un array 'a' di N interi, e due indici 'k' e 'h'
e restituisce il minimo della porzione a[k]-a[h]

0) restituisce un intero; riceve array a[N] di int e indici k, h;
usa j, indiceMin …

1) indiceMin inizializzato con k

2) per j da k+1 a h
se a[j] < a[indiceMin] indiceMin = j (ora è questo l'indice del minimo finora analizzato)

3) fine, restituire indiceMin

definisci la funzione minimo2() solo in base all'algoritmo qui sopra – ricorda i controlli
provaci e poi prosegui

funzione minimo2()

```
void minimo2 (int a[N], int k, int h) {
  int j, indiceMin  ;

if ( (h>N-1) || (h<0) ){
   printf("\n\n\n eeeehhhhhhh!!!\n\n\n");
   return 0;
}

if ( (k>h) || (k<0) ){
   printf("\n\n\n eeeekkkkkk!!!\n\n\n");
   return 0;
}
if ( k==h ) return k;


/* caso ragionevole */
 indiceMin = k;       /* init minimo parziale */
 for (j=k+1; j<=h; j++)
    if (arr[j] < arr[indiceMin])
          min=j;     /* (*) aggiornamento minimo parziale */

return indiceMin;
}
```

prova selectionSort2 e minimo2 in un programma (puoi inizializzare in definizione l'array con i dati di prova)