Approfondimento: operazioni su array

introduzione alla programmazione

introduzione

non unico!

- gli array unidimensionale sono **uno** strumento efficace per memorizzare liste di valori omogenei
- le tipiche operazioni che possiamo svolgere su di essi includono
  - ordinamento
  - ricerca di un elemento
  - inserimento di un elemento
  - eliminazione di un elemento

ordinamento o *sorting*

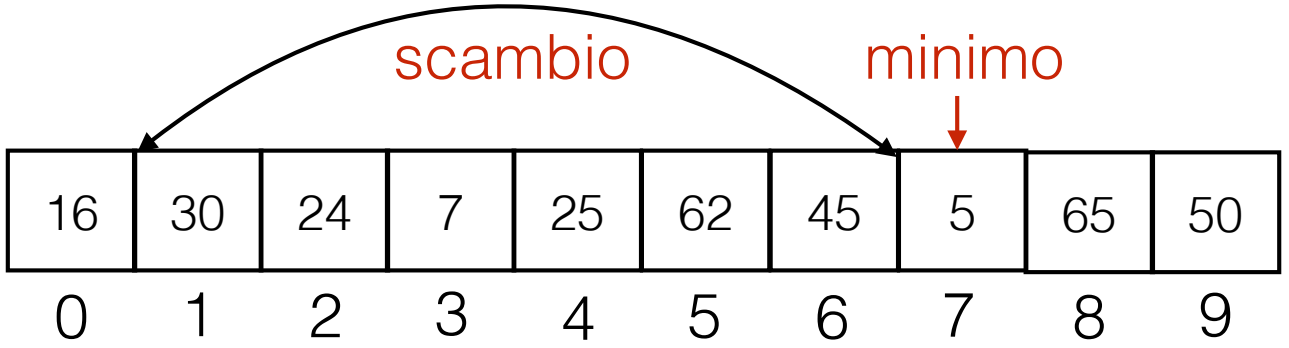
- obiettivo: ordinare gli elementi in modo crescente (o decrescente)
- consideriamo algoritmi classici
  - selection sort
  - insertion sort
  - (+bubble sort)
- sono algoritmi *in place* (ordinano senza necessità di fare una copia - riduco la necessità di spazio)

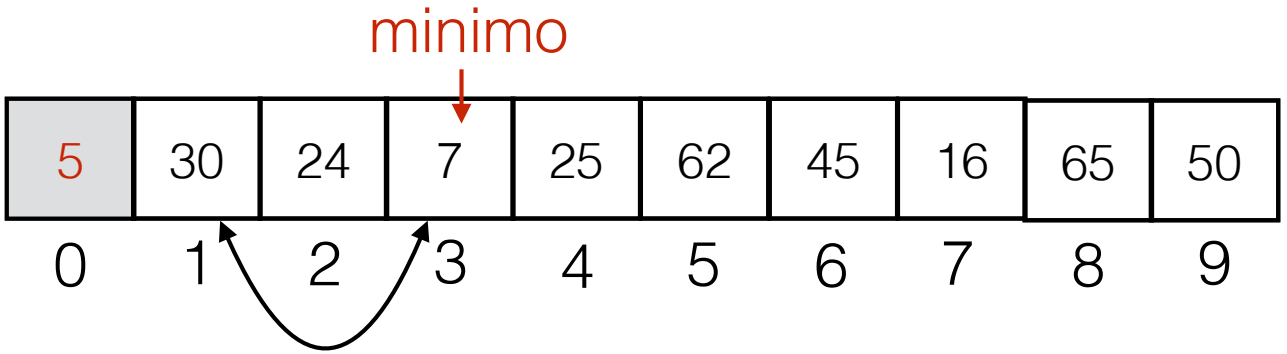
date un'occhiata a: www.sorting-algorithms.com/

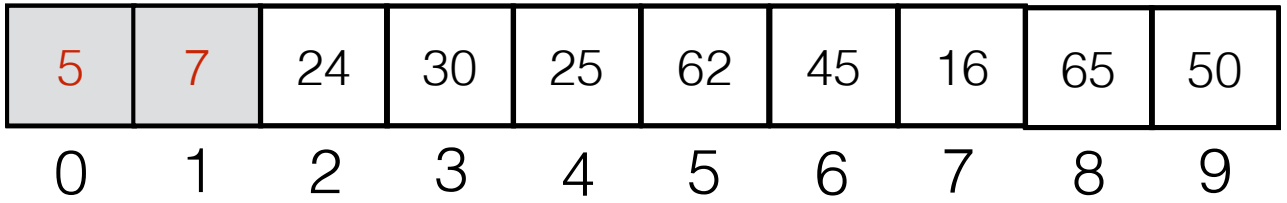
selection sort

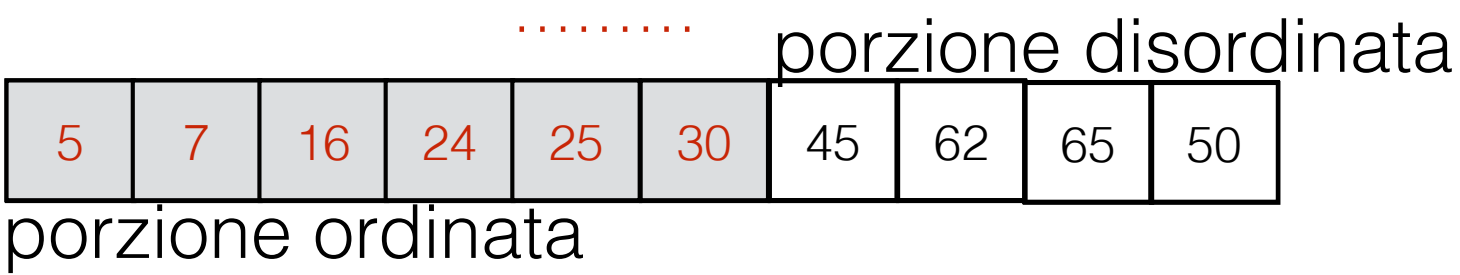
- seleziono un elemento della lista e lo porto nella posizione corretta
- se voglio ordinare in modo crescente
  - seleziono l'elemento più piccolo della lista e lo porto all'inizio
  - poi seleziono il secondo elemento piò piccolo e lo porto nella seconda posizione
  - …

Selection sort - Esempio









Analisi del problema

- passi principali

  A. trova la posizione dell'elemento minimo

  B. sposta l'elemento minimo all'inizio della porzione non ordinata

Analisi del problema

- for (int i=0; i<length-1;i++)
  A. cerca lo smallestIndex nella porzione list[i]…list[length-1]
  B. scambia list[i] con list[smallestIndex]

Verso un'implementazione

- Passo A: *cerca lo smallestIndex nella porzione list[i]… list[length-1]*

```
smallestIndex=i;
for (int mini=i+1; mini<length; mini++)
      if (list[mini]<list[smallestIndex])
            smallestIndex=mini;
```

Verso un'implementazione

- Passo B: *scambia list[i] con list[smallestIndex]*

```
temp=list[smallestIndex];
list[smallestIndex]=list[i];
list[i]=temp;
```

temp - variabile ausiliaria

Numero di operazioni

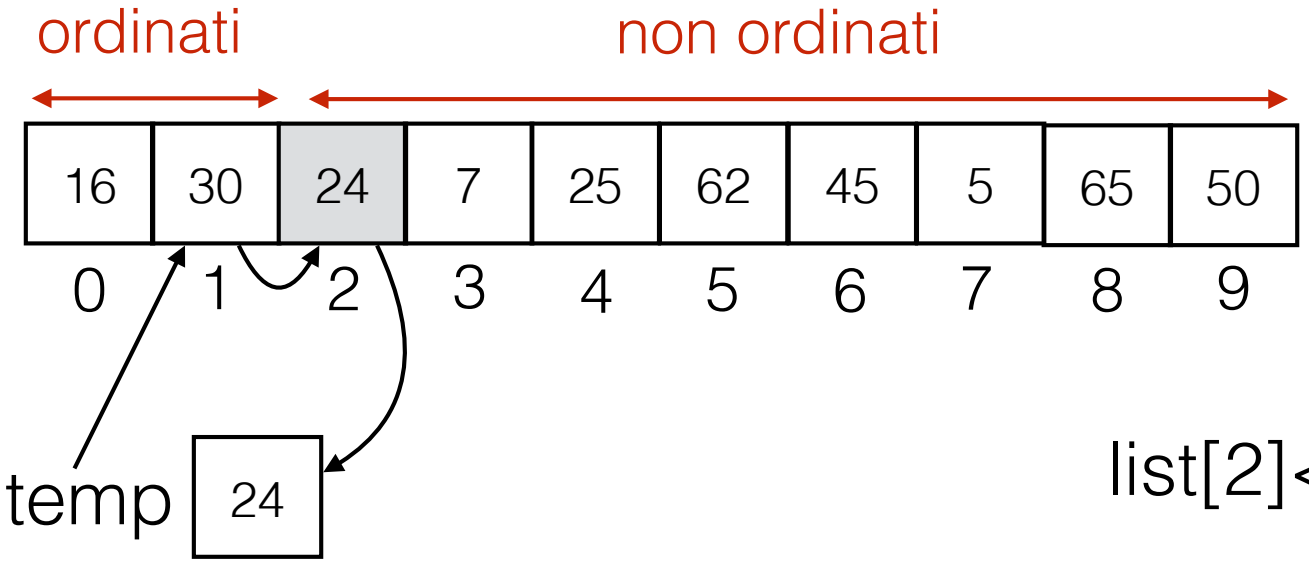
- l'algoritmo selection sort richiede, **in media**
  - n(n-1)/2 confronti
  - 3(n-1) assegnazioni
- http://www.sorting-algorithms.com/selection-sort

se n=1000?

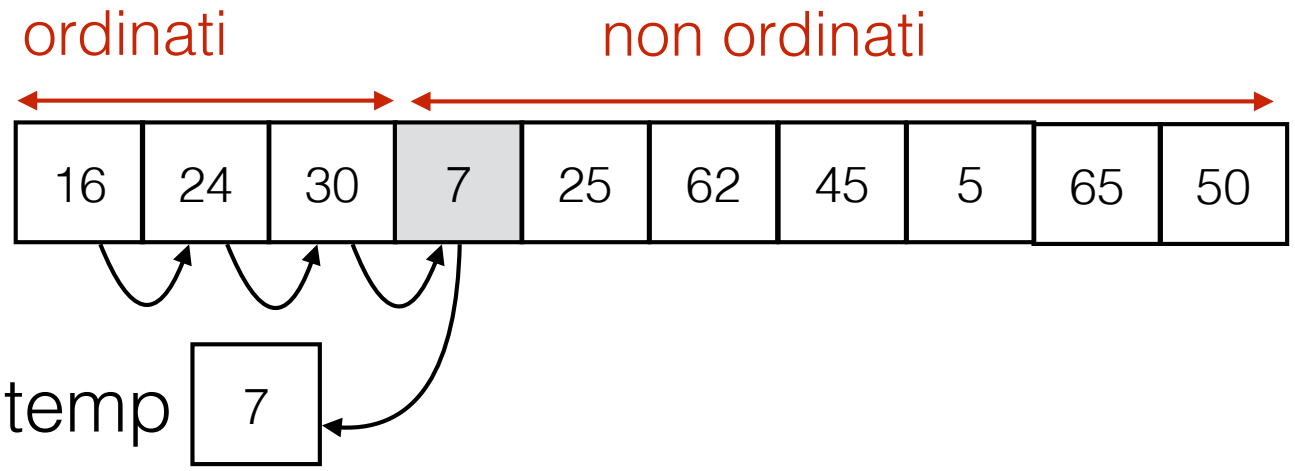
Insertion sort

- ordina una lista spostando ogni elemento fino a che non arriva nella posizione giusta
    - rispetto a selection sort riduce il numero di confronti (ma aumenta il numero di scambi)

Insertion sort - Esempio



list[2]<list[1]



Analisi del problema

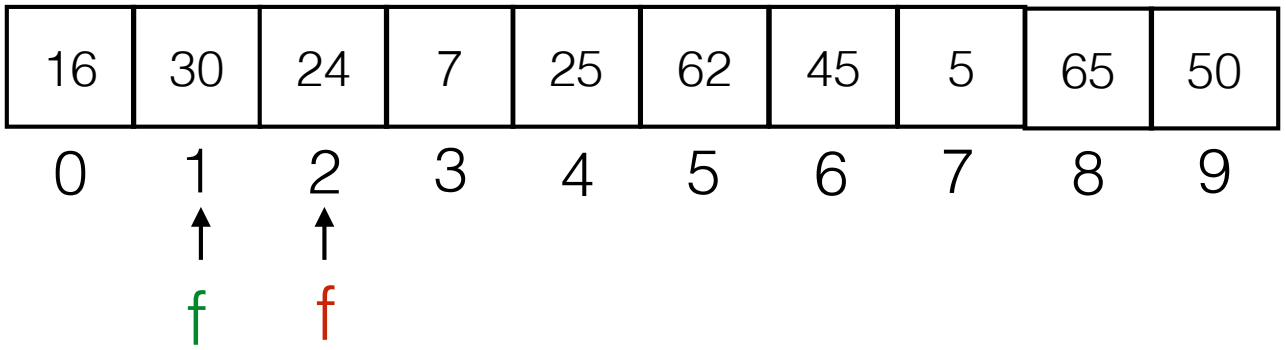
- in ogni passo la lista è divisa in due parti, una ordinata e l'altra no
- il primo elemento non in ordine viene scambiato con tutti gli elementi che lo precedono (della parte ordinata) fino a che non cade nella posizione giusta

Verso l'implementazione

- for (int f=1; f<length; f++) {
  if (list[f] < list[f-1])
  {
  temp←list[f]
  location ← f
  do {
  list[location]←list[location-1]
  location - -
  }
  while (location>0 && list[location-1]>temp)
  list[location]←temp
  }

Test

length=10

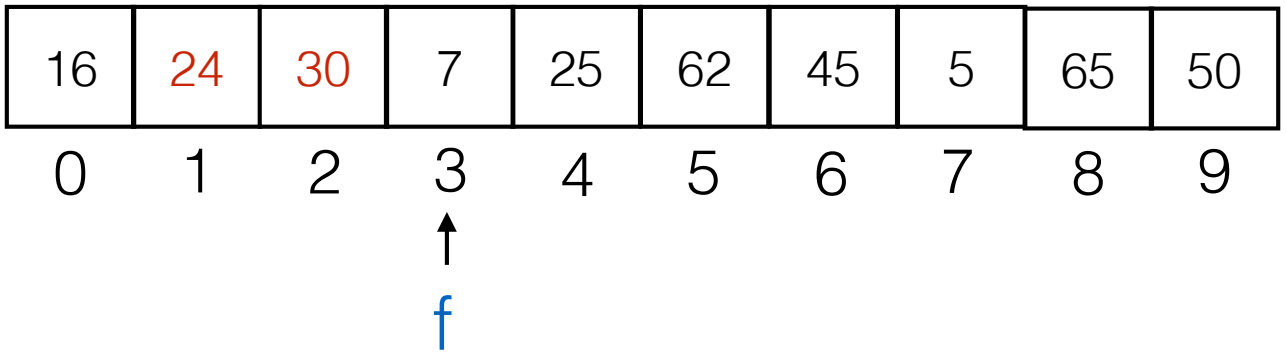


- f=1
  list[f]>list[f-1] //if

- f=2
  list[f]<list[f-1] //if
  temp=24
  location=2
  list[2]=30
  location=1
  location>0 e list[location-1]<temp //while (esco)
  list[1]=24

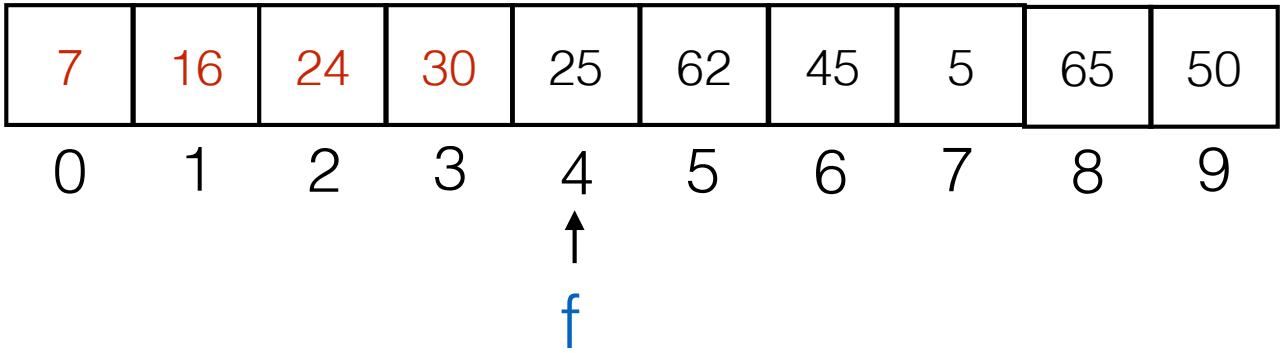
Test

length=10



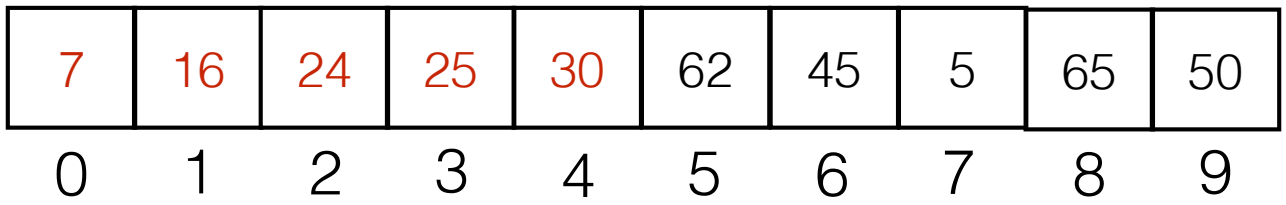
- f=3
  list[f]<list[f-1] //if
  temp=7
  location=3
  list[3]=list[2]=30
  location=2
  location>0 e list[2-1]>temp // (true) do while
  list[2]=list[1]=24
  location=1
  location>0 e list[1-1]>temp // (true) do while
  list[1]=list[0]=16
  location=0
  location=0 e list[-1]??? // esco da do while
  list[0]=temp

Test



length=10

- f=4
  list[f]<list[f-1] //if
  temp=25
  location=4
  list[4]=list[3]=30
  location=3
  location>0 e list[3-1]<temp // (esco) do while
  list[3]=temp



Codifica

```
void insertionSort(int list[], int length)
{
    int location;
    int temp;

    for (int f=1;f<length;f++) {
        if (list[f]<list[f-1])
        {
            temp=list[f];
            location=f;
            do
            {
                list[location]=list[location-1];
                location--;
            }
            while (location>0 && list[location-1] > temp);
            list[location]=temp;
        }
        stampaLista(list,length);
    }
}
```

Numero di operazioni

- l'algoritmo selection sort richiede, **in media**
  - **(**n*n+3n-4)/4 confronti
  - n(n-1)/4 assegnazioni
- http://www.sorting-algorithms.com/insertion-sort

se n=1000?

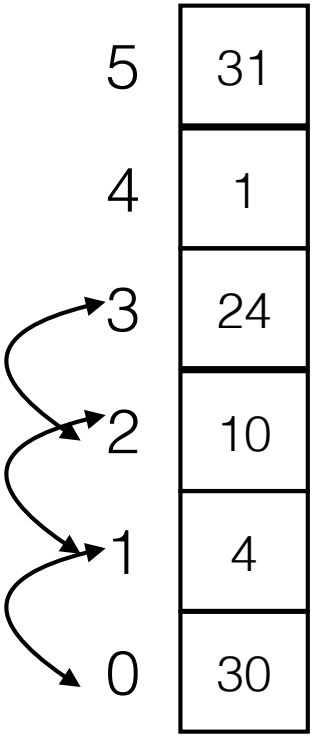
e se la lista è quasi tutta ordinata?

Insertion sort è un algoritmo adattivo!

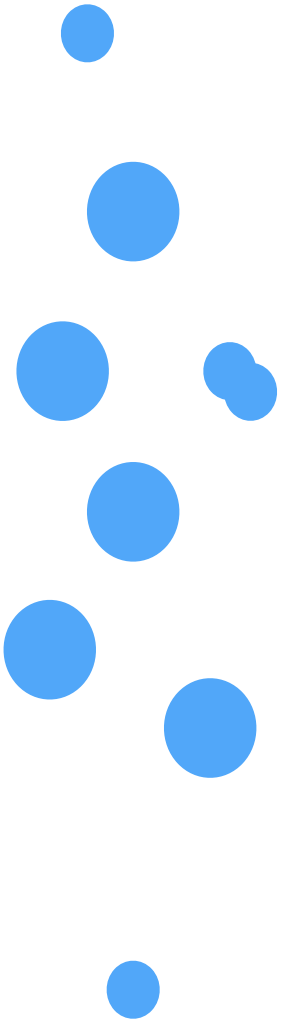
Bubble sort

- come selection e insertion sort, anche bubblesort non è un algoritmo ottimale
- l'aggiornamento avviene in modo da far "emergere" gli elementi piu' alti (per l'ordinamento crescente) —- da qui il nome (la procedura ricorda le bollicine)

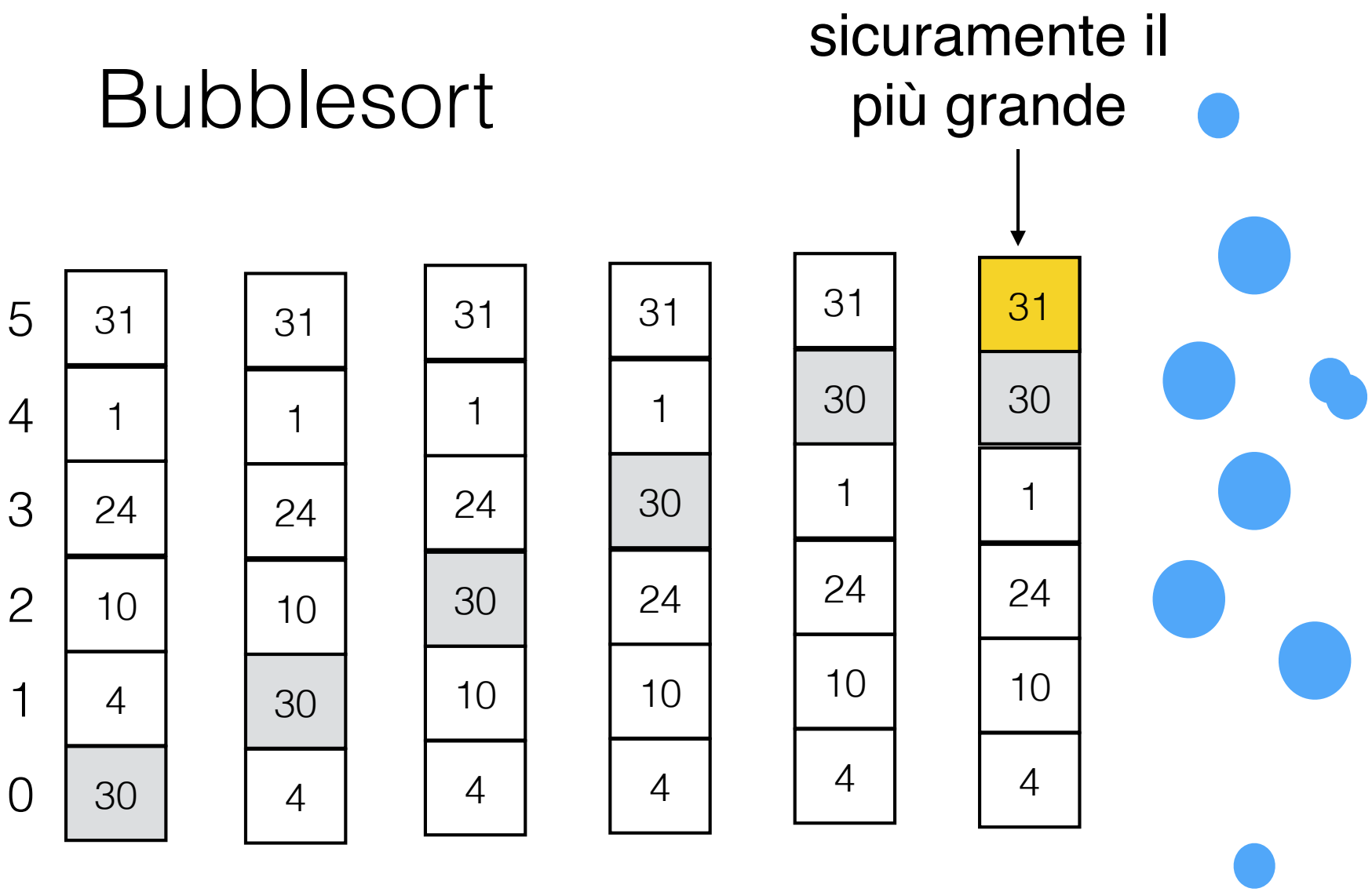
Bubblesort



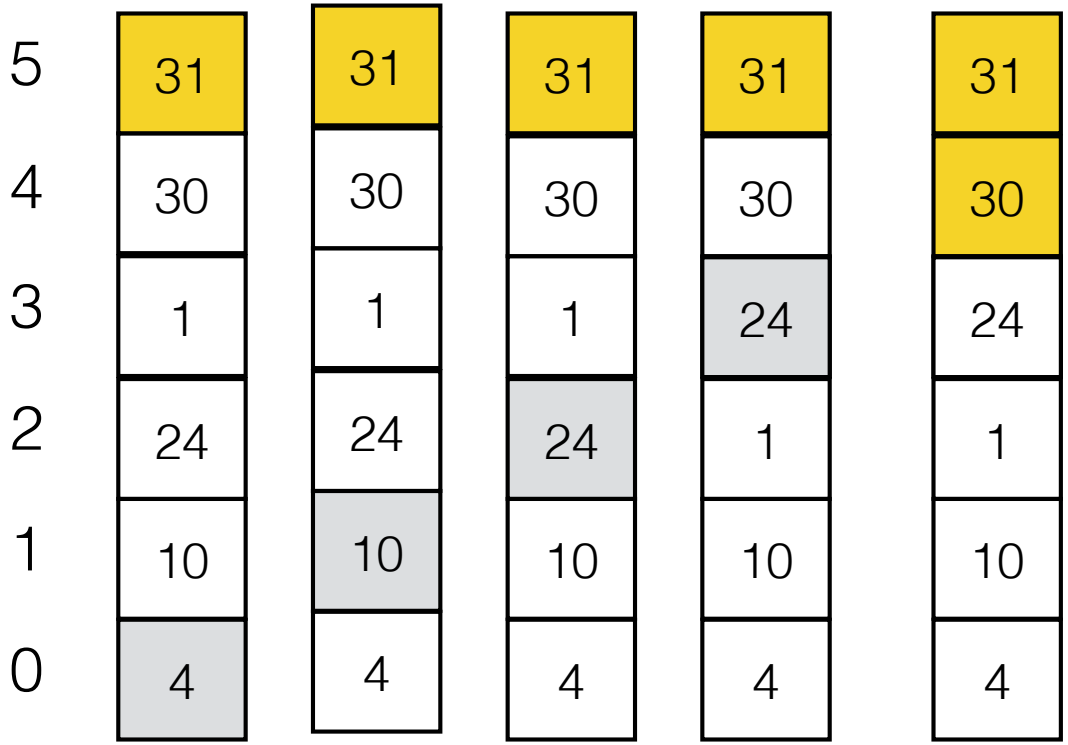
confronto a due a due
e scambio se necessario



Bubblesort

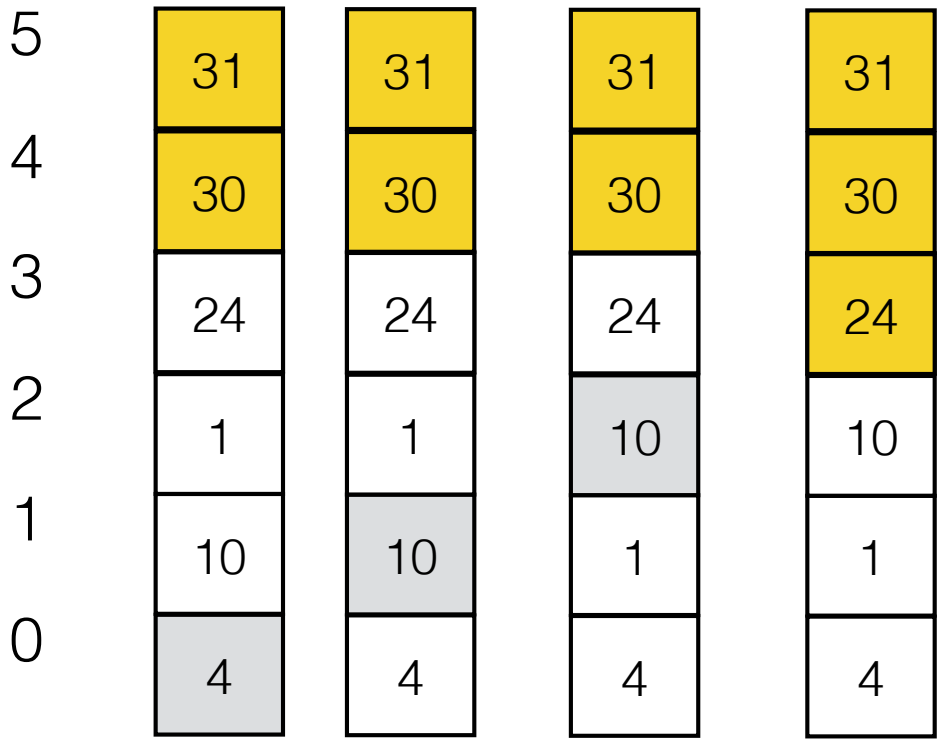

sicuramente il più grande
5
4
3
2
1
0
31
1
24
10
4
30
31
1
24
10
30
4
31
1
24
30
10
4
31
1
30
24
10
4
31
30
1
24
10
4
31
30
1
24
10
4

Bubblesort



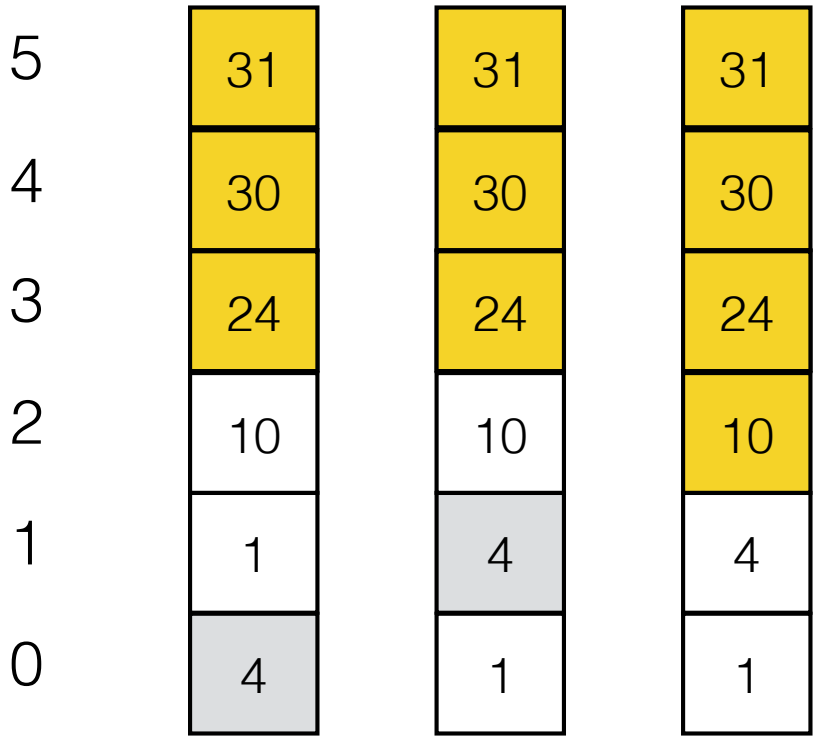
sicuramente il più grande

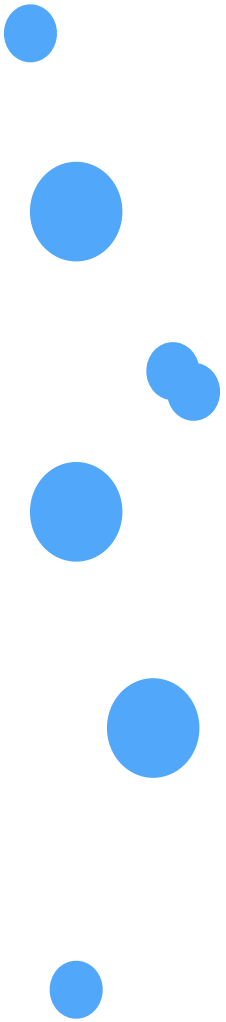
Bubblesort



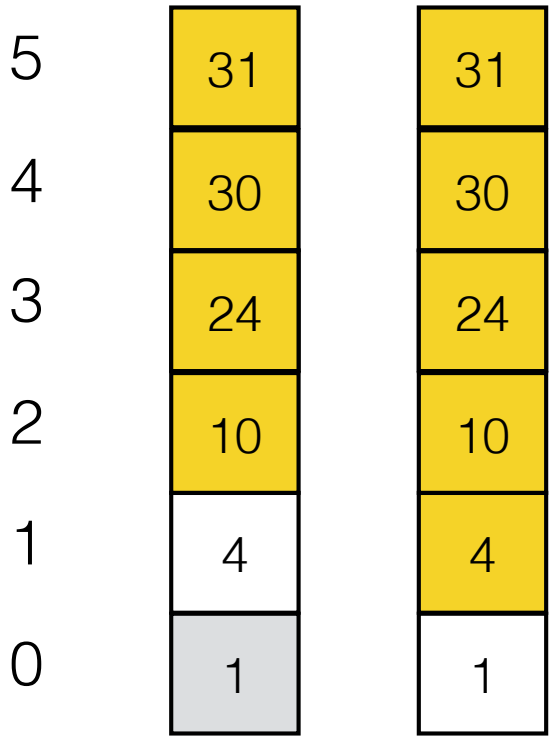
Bubblesort

sicuramente il più grande





Bubblesort



e’ già a posto ma
dobbiamo verificarlo
(nella parte bianca)

Bubblesort - codifica

```
void bubble (int a[], int size) {

    for (int i=0;i<size; i++) {

        for (int j=0; j<size-1-i; j++)
            if (a[j]>a[j+1]) {
                swap(a[j],a[j+1]);

            }

    }
}

void swap (int &x, int &y) {
    int temp;
    temp=x;
    x=y;
    y=temp;
}
```

Bubblesort - codifica

```
void bubble (int a[], int size) {
    bool swapped=false;
    for (int i=0;i<size; i++) {
        swapped=false;
        for (int j=0; j<size-1-i; j++)
            if (a[j]>a[j+1]) {
                swap(a[j],a[j+1]);
                swapped=true;
            }
        if (!swapped) return;
    }
}
```

possiamo aggiungere una variabile booleana che tiene traccia degli swap (ed esca dalla funzione appena in un giro non ci sono stati swap..)

Ricerca di un elemento

- input - lista, lunghezza, elemento da cercare

- output
  - successo se è stato trovato (e sua posizione)
  - fallimento

Ricerca sequenziale

```
int seqSearch(const int list[], int length, int item)
{
    int loc;
    bool found = false;

    for (loc=0;loc<length; loc++)
        if (list[loc]==item) {
            found=true;
            break;
        }
    if (found)
        return loc;
    else
        return -1;
}
```

Ricerca sequenziale - analisi

- in media la ricerca sequenziale richiede n/2 confronti
  - se l'elemento cercato è in cima pochi confronti
  - se è in fondo o non c'è un numero di confronti pari a n (lunghezza lista)
- E' un algoritmo che non fa ipotesi sull'ordinamento della lista
  - se la lista è ordinata possiamo adottare algoritmi più efficienti

Ricerca binaria

- lavora su liste ordinate seguendo il paradigma del *divide et impera*

- *idea:*
  confronto l'elemento da cercare (item) con il centro della lista list[centro].

  - se item è uguale a list[centro] fine ricerca

  - se item è minore di list[centro] continuo a cercare nella prima metà della lista

  - se item è maggiore di list[centro] continuo a cercare nella seconda metà della lista

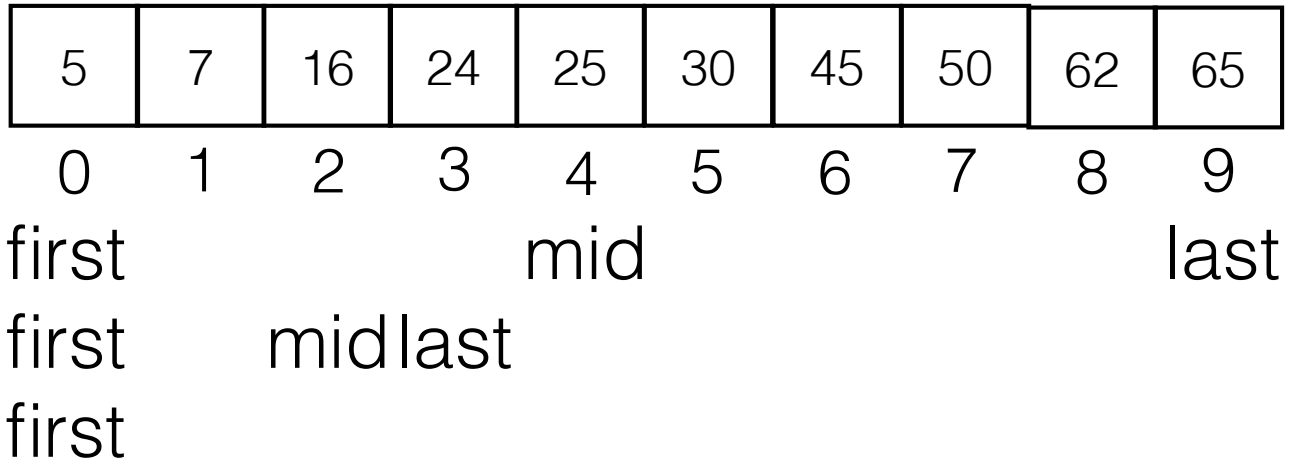
Ricerca binaria

```
int binarySearch(const int list[], int length, int item)
{
    int first=0;
    int last=length-1;
    int mid;
    bool found=false;

    while(first<=last && !found)
    {
        mid=(first+last)/2;
        if (list[mid]==item)
            found=true;
        else
            if (list[mid]>item)
                last=mid-1;
            else first=mid+1;
    }
    if (found)
        return mid;
    else
        return -1;
}
```

Ricerca binaria

item=7



Ricerca binaria

item=44