ORDINAMENTO E RICERCA

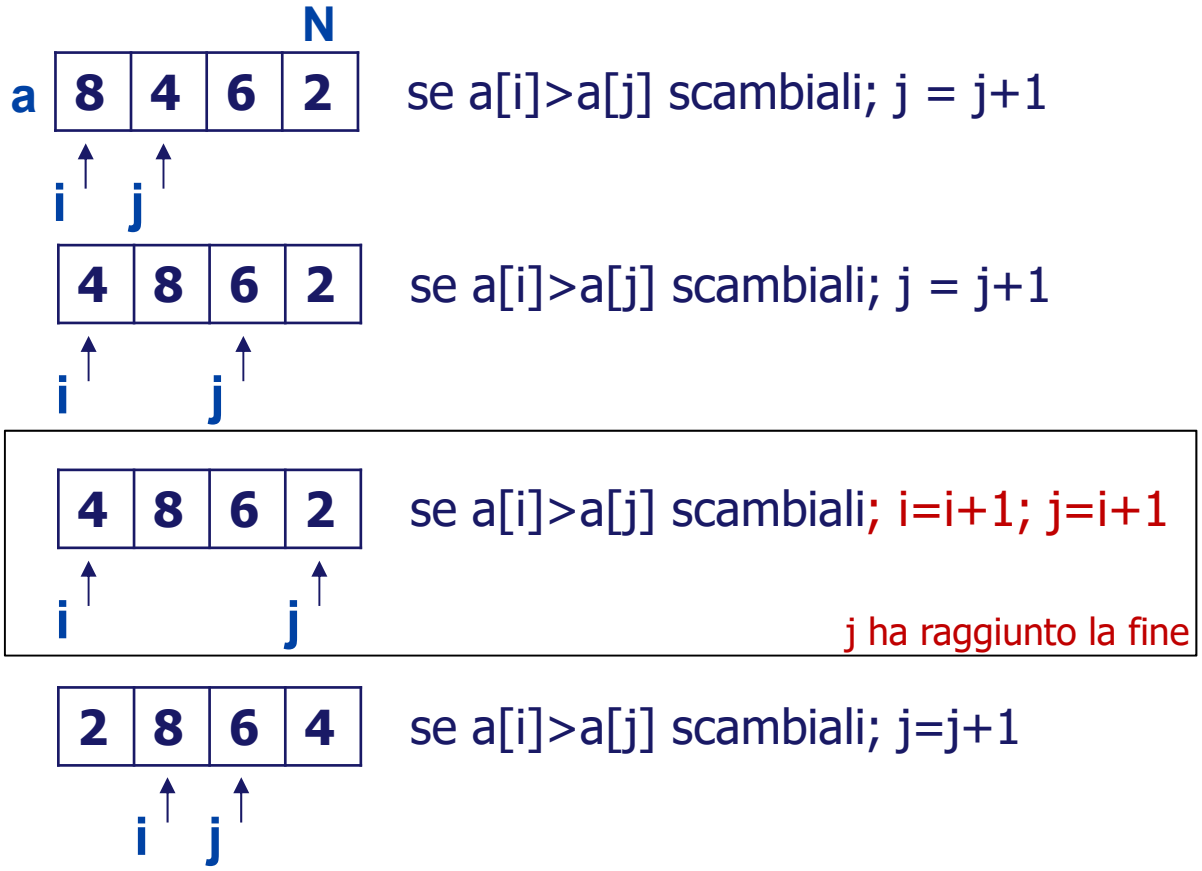
Roberto Nardone, Luigi Romano

ROADMAP

- Ordinamento per Scambio
- Ordinamento per Selezione
- Ordinamento a bolle
- Ricerca Sequenziale
- Ricerca Binaria

ORDINAMENTO PER SCAMBIO

- Idea di base: confrontare ogni elemento con tutti i successivi e scambiare tra di loro gli elementi non posizionati correttamente.



Ultimo ciclo per:

- i=N-1; j=N

#Confronti: (N-1)+(N-2)+(N-3)+…+1=N(N-1)/2

#Scambi:

Migliore:0

Pegg.: (N-1)+(N-2)…

Complessità asintotica media $O(N^2)$

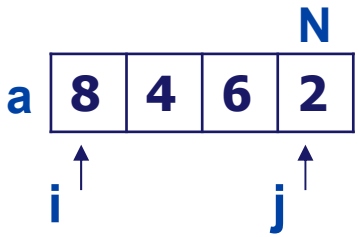
ORDINAMENTO PER SCAMBIO: CODIFICA


```
int main(){
   const int N = 10; // Dimensione vettore
   int a[N] = {7,8,3,5,2,9,10,4,6,1};
   int i, j;


   // Ordinamento per scambio
   for (i=0; i < N-1; i++) {
      for (j=i+1; j < N; j++) {
         if (a[i]>a[j]){
            scambia(a,i,j);
         }
      }
   }
}
```

ORDINAMENTO PER SELEZIONE

- Idea di base: trovare il più piccolo elemento del vettore [i,N] e scambiarlo con l'elemento i-esimo poi ripetere con il sotto-vettore [i+1,N] degli elementi rimanenti



Scambia (a[i], a[j] con j:a[j]==min{a[i] – a[N]});
i = i+1

Ultimo ciclo per:

- i=N-1

$O(N^2)$

ORDINAMENTO PER SELEZIONE: CODIFICA

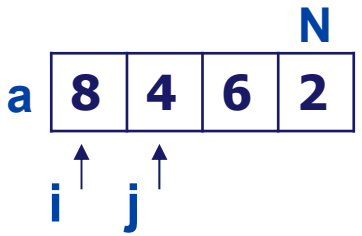

```
int main(){
    const int N = 10;
    int a[N] = {7,8,3,5,2,9,10,4,6,1};
    int i, j;
    // Ordinamento per selezione
    for (i=0; i < N-1; i++) {
        j=minimo(a,i,N);
        scambia(a,i,j);
    }
}
int minimo(int a[], int i, const int N){
    int min = i;
    for (i=i+1; i < N; i++) {
        if (a[min]>a[i]) min = i;
    }
    return min;
}
```

ORDINAMENTO A BOLLE (BUBBLE SORT)

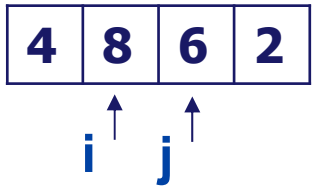
- E’ un algoritmo poco efficiente ma molto semplice
- Si confrontano a due a due gli elementi adiacenti del vettore (partendo dal primo) e si scambiano se non ordinati. Si passa ai successivi due ripetendo fino alla fine del vettore con l’elemento più grande che avrà dunque raggiunto l’ultima posizione.

  A questo punto si ricomincia dall’inizio e si ripete finché non si sono analizzate tutte le coppie senza nessuno scambio.

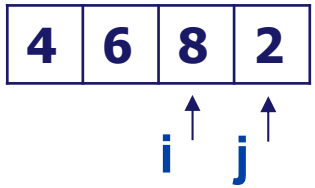
BUBBLE SORT



scambi = 0;
se a[i]>a[j] scambiali e scambi++; i++; j++

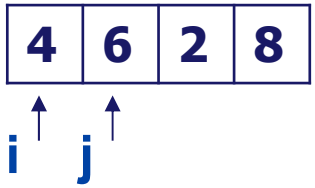


se a[i]>a[j] scambiali e scambi++; i++; j++



se a[i]>a[j] scambiali e scambi++; i=1; j=2; scambi=0

j ha raggiunto la fine



se a[i]>a[j] scambiali e scambi++; i++; j++

...

Ultimo ciclo per:

- j=N senza che ci siano stati scambi

$O(N^2)$

BUBBLE SORT: CODIFICA

- Esercizio (tutti gli algoritmi)

- Esercizio 2: Confrontare i tempi di esecuzione dei tre algoritmi per un vettore di 100000 elementi. Confrontare nei casi:
  - Vettore riempito con numeri casuali
  - Vettore ordinato in senso crescente (caso migliore)
  - Vettore ordinato in senso decrescente (caso peggiore)

ALTRI ALGORITMI DI ORDINAMENTO

- Shell Sort $O(n\log^2 n)$
- Quick Sort $O(n\log n)$

RICERCA SEQUENZIALE

- Quando usarlo: se il vettore non è ordinato
  - Altrimenti preferire ricerca binaria (vedere più avanti)
- Idea di base: si confronta ogni elemento con quello cercato finché non si trova l'elemento o è terminato il vettore in cui cercare, nel primo caso si ritorna l'indice associato alla posizione dell'elemento, nel secondo un valore di errore (es. -1)
- Numero di confronti
  - Caso peggiore N
  - Caso migliore 1
  - Caso medio: N/2
- O(N)

RICERCA SEQUENZIALE: CODIFICA


```
//ricerca l'elemento e nel vettore a di N
//elementi
int ricerca(int a[], const int N, int e){
   bool trovato = false;
   int i = 0;

   while(!trovato && i < N){
      if (a[i] == e) trovato = true;
      else i++;
   }
   if(!trovato) i = -1;
   return i;
}
```

ESERCIZIO: CONTEGGIO OCCORRENZE

- Modificare il programma precedente per contare il numero di volte che l'elemento e occorre all'interno del vettore a

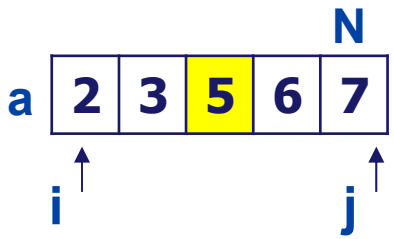
CONTEGGIO OCCORRENZE: SOLUZIONE


```
// conta il numero di occorrenze di e in a
int conta(int a[], const int N, int e){
   int n = 0;
   for (int i = 0; i < N; i++){
      if (a[i] == e)n++;
   }
   return n;
}
```

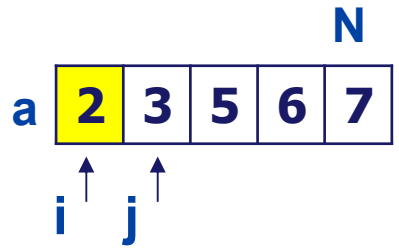
RICERCA BINARIA

- Quando usarlo: se il vettore è ordinato
  - Altrimenti usare ricerca sequenziale
- Idea: confrontare l'elemento da cercare con l'elemento centrale del vettore, se sono uguali, termina l'algoritmo; altrimenti se l'elemento da cercare è maggiore, ripetere con il semivettore di destra, se è minore ripetere con il semivettore di sinistra. Termina se il semivettore in cui cercare è vuoto.
- $O(\log_2 N)$

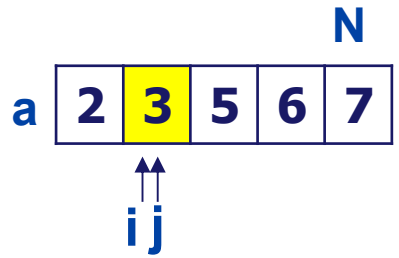
ESEMPIO



Ricerca e=3
centrale = (int) (i+j)/2=2
a[centrale]>e => **j**=centrale**-**1=1



centrale=0
a[centrale]<e => **i**=centrale**+**1=1



centrale=1
a[centrale]==e => **indice=i=j**

Nel caso `centrale=i=j` e `a[centrale] !=e` …

ELEMENTO NON PRESENTE

RICERCA BINARIA: CODIFICA


```
// ricerca e in a ordinato
int cercaBin(int a[], const int N, int e){
    int i = 0;
    int j = N-1;
    int indice = -1;
    bool trovato = false;

    while (i != j && indice==-1){
        c=(i+j)/2;
        if (a[c]==e) {
            indice = c;
        } else if(a[c] < e) {
            i=c+1;
        } else {
            j=c-1;
        }
    }
    return indice;
}
```