Algoritmo

- Sequenza di operazioni che a partire da un input permettono di arrivare ad un output.
- Strumento per risolvere problemi di tipo computazionale.
- La descrizione del problema che vogliamo risolvere specifica la relazione input/output da realizzare
- Un algoritmo descrive una precisa procedura computazionale per realizzare tale relazione.
- Un buon punto di partenza sono gli algoritmi di ordinamento elementari.

Scheduling della CPU

- I moderni sistemi operativi consentono l'esecuzione di più programmi allo stesso tempo.
- La CPU divide il suo tempo tra più programmi.
- La decisione di eseguire i programmi in una determinata sequenza si chiama *scheduling*.
  - *First come first serve* (FCFS). I programmi vengono eseguiti nell'ordine di arrivo
  - *Shortest Job First* (SJF). Vengono prima eseguiti i programmi con tempo di esecuzione stimato più breve.
- In quest'ultimo caso dobbiamo poter ordinare i tempi in questione.

Scheduling della CPU

- Consideriamo ad esempio i seguenti programmi (ed i corrispondenti tempi previsti)

P_0	P_1	P_2	P_3
21 ms	3 ms	1 ms	2 ms

- Usando una strategia FCFS i tempi di attesa sarebbero

P_0	P_1	P_2	P_3
0 ms	21 ms	24 ms	25 ms

Tempo medio di attesa: 17,5 ms

- Con una strategia STJ invece

P_2	P_3	P_1	P_0
1 ms	2 ms	3 ms	21 ms

Tempo medio di attesa: 2,5 ms

algoritmi di ordinamento elementari

- Esistono vari algoritmi di ordinamento
- Per adesso studieremo alcuni tra i più semplici

- Ordinamento per scambio.
- Ordinamento per selezione.
- Ordinamento per inserimento.
- Ordinamento a bolle.

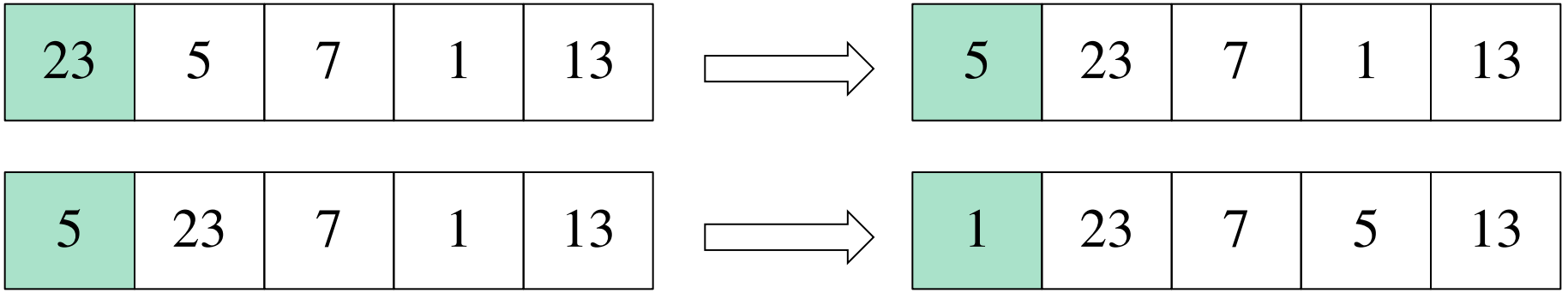
ordinamento per scambio

input: a[] vettore di interi non ordinati
output: a[] vettore di interi ordinati (in senso crescente)

Idea: Confrontiamo il primo elemento con i successivi e facciamo uno scambio se l'ordine non è corretto.

Dopo la prima iterazione ripetiamo il ragionamento per tutte le posizioni successive alla prima.

Esempio



- Andiamo a vedere come implementare l'algoritmo

ordinamento per selezione

- Volendo ordinare il vettore in senso crescente, si parte selezionando l'elemento più piccolo e ponendolo al primo posto;
- si procede applicando lo stesso processo al sottovettore non ordinato, fino a che quest'ultimo si riduce ad un solo elemento (il maggiore del vettore)
- anche l'ordinamento per selezione compie n-1 iterazioni

```
Ordinamento per selezione del vettore di interi 11, 9, 17, 5, 14:
Iterazione 0:  9<11 => selezione: 9
           17>9 => niente
           5<9 => selezione: 5
           14>5 => niente
           scambio: 5,9,17,11,14
Iterazione 1:  17>9 => niente
           11>9 => niente
           14>9 => niente
Iterazione 2:  11<17 => selezione: 11
           14>11 => niente
           scambio: 5,9,11,17,14
Iterazione 3:  14<17 => selezione: 14
           scambio: 5,9,11,14,17 (Vettore ordinato)
```


ordinamento per selezione

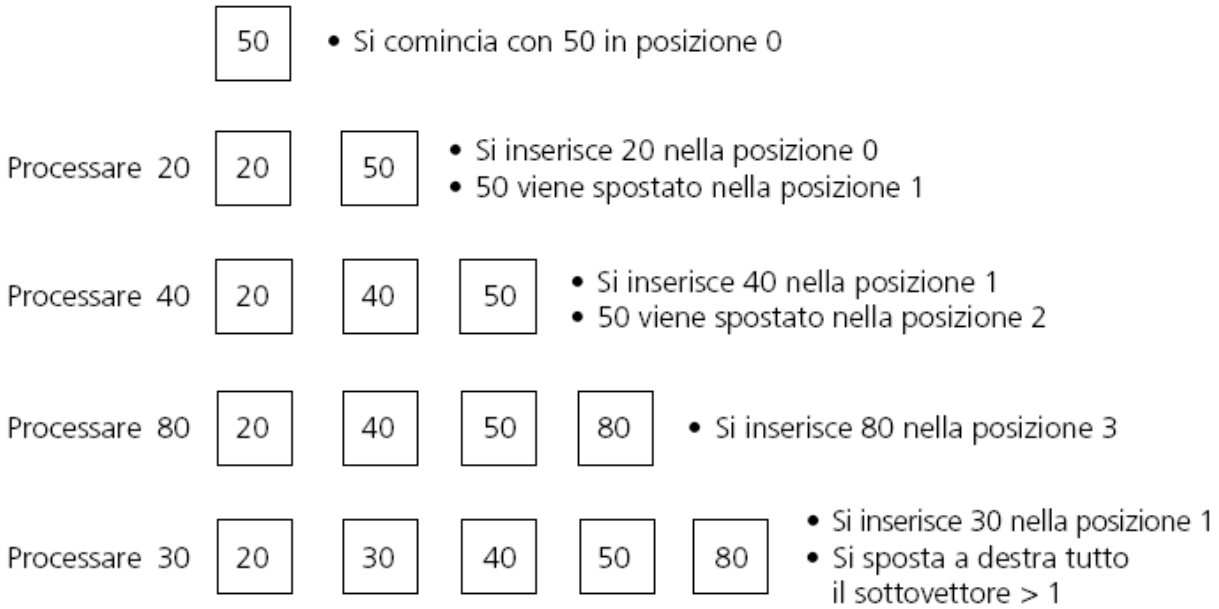

```
void Selezione (int a[], int n)
{
   int i,j, indice, aux;

   for (i=0;i<n-1;i++)
   {  indice=i;
      for (j=i+1; j<n; j++)
         if (a[indice] > a[j])
            indice=j;
      aux=indice;
      a[indice]=a[i];
      a[i]=aux;
}
```

Qual è la “complessità” di questo algoritmo?

ordinamento per inserimento

Si inserisce un elemento alla volta nella posizione che gli spetta in un vettore già ordinato, partendo dal vettore vuoto. Ad esempio, nel caso del vettore di interi `A = 50, 20, 40, 80, 30`:



Ordinamento a bolle (Bubble sort)

Idea: per ogni iterazione si confrontano elementi adiacenti e si scambiano i loro valori quando il primo elemento è maggiore del secondo;
Alla fine di ogni iterazione, l'elemento maggiore è risalitofino alla cima del vettore attuale;

- Iterazione `0`: si confrontano gli elementi adiacenti
  `(A[0],A[1]),(A[1],A[2]),(A[2],A[3]),...(A[n-2],A[n-1])`
  - si effettuano *n*-1 confronti;
  - per ogni coppia (`A[i],A[i+1]`) si scambiano i valori se `A[i+1]<A[i]`;
  - alla fine dell'iterazione, il massimo sarà situato in `A[n-1]`
- Iterazione 1: si effettuano gli stessi confronti e scambi, terminando con l'elemento di secondo maggiore valore in `A[n-2]`
- il processo termina con l'iterazione *n-1*, nella quale l'elemento più piccolo si colloca in `A[0]`

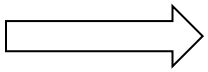
Esempio

25	60	45	35	12

a[0] con a[1] (25 con 60) non c'è scambio

a[1] con a[2] (60 con 45) scambio

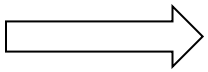
25	60	45	35	12



25	45	60	35	12

a[2] con a[3] (60 con 35) scambio

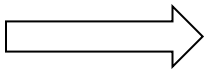
25	45	60	35	12



25	45	35	60	12

a[3] con a[4] (60 con 12) scambio

25	45	35	60	12



25	45	35	12	60

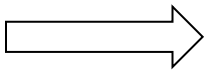
II Iterazione

25	45	35	12	60

a[0] con a[1] (25 con 45) non c'è scambio

a[1] con a[2] (45 con 35) scambio

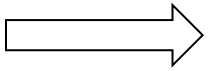
25	45	35	12	60



25	35	45	12	60

a[2] con a[3] (45 con 12) scambio

25	35	45	12	60



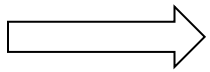
25	35	12	45	60

III Iterazione

25	35	12	45	60

a[0] con a[1] (25 con 35) non c'è scambio
a[1] con a[2] (35 con 12) scambio

25	35	12	45	60



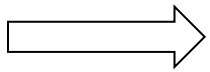
25	12	35	45	60

IV Iterazione

25	12	35	45	60

a[0] con a[1] (25 con 12) scambio

25	12	35	45	60



12	25	35	45	60

Shell Sort

- Ordinamento per inserimento con incrementi decrescenti.
- Si confrontano elementi che possono non essere contigui

Idea:

- Si divide il vettore iniziale (n elementi) in n/2 gruppi di 2 elementi
- Gli elementi di ogni gruppo sono a distanza n/2.
- Si ordina ciascun gruppo separatamente

Quindi

- Si divide il vettore iniziale in n/4 gruppi di 4 elementi (con una distanza di n/4)
- Si ordina ciascun gruppo separatamente

Si itera il procedimento fino a rimanere con un solo gruppo di n elementi.

Complessità del sorting

- Gli algoritmi di ordinamento visti hanno complessità $O(n^2)$
- Ogni algoritmo basato su confronti deve fare almeno n passi (deve poter leggere l'input)
- In realtà ogni algoritmo di ordinamento basato su confronti richiede Ω (nlog n) passi

Ordinamento per Fusione (Merge Sort)

- Richiede O(n log n) passi (e O(n) spazio aggiuntivo)

Decomposizione: Se la sequenza contiene almeno due elementi dividila in 2 sottosequenze di uguale lunghezza.

Ricorsione: ordina ricorsivamente le 2 sottosequenze

Ricombinazione: fondi le 2 sottosequenze ordinate in un'unica sequenza ordinata.

Basato sul paradigma divide et impera

Divide et Impera

- Paradigma molto diffuso
- Il problema è diviso in 2 o più sottoproblemi
  - Da risolvere ri applicando il principio
  - Si arriva a problemi elementari che possono essere risolti in modo diretto
- I sottoproblemi sono istanze dello stesso problema di dimensioni ridotte.

Divide et Impera: Tre fasi

- Decomposizione.
  - Identifica pochi sottoproblemi
    - Su insiemi di dati di dimensioni inferiori
- Ricorsione
  - Si risolve ricorsivamente ogni sottoproblema
- Ricombinazione.
  - Le soluzioni ai sottoproblemi vengono ricombinate per fornire una soluzione al problema iniziale.

Merge Sort

```
MergeSort (a , sin, des) // 0 <= sin, des <= n-1
    If (sin < des) {
        centro = (sin + des)/2;
        MergeSort (a, sin, centro);
        MergeSort (a, centro, des);
        Fusione (a, sin, centro, des);
    }
```


Merge Sort

```
Fusione (a , sx, cx, dx) // 0 <= sx, cx, dx  <= n-1
    i=sx; j=cx+1; k=0;
    WHILE ((i <= cx) && (j<=dx)) {
        IF (a[i] <= a[j]) {
            b[k]=a[i]; i=i+1;
        } ELSE {
            b[k]=a[j]; j=j+1;
        }
        k = k+1;
    }
    FOR ( ; i<= cx ; i = i+1, k=k+1) b[k]=a[i];
    FOR ( ; j<= dx ; j = j+1, k=k+1) b[k]=a[j];
    FOR (i=sx; i<=dx; i=i+1) a[i]=b[i-sx];
```


Quicksort

- Ideato da Hoare nel 1962, è il più efficiente fra gli algoritmi di ordinamento noti

Idea: Dividiamo gli *n* elementi in tre partizioni: la *sinistra*, la *centrale* (con un solo elemento denominato *pivot)* e la *destra*

Invariante: (ordinamento crescente)

- Gli elementi della partizione sinistra dovranno essere più piccoli del pivot
- Quelli della parte destra più grandi

- Applicando ricorsivamente l'algoritmo sia alla partizione sinistra che a quella destra si arriverà ad ordinare tutto il vettore
- Il pivot può essere scelto a caso all'interno del vettore

Esempio

- Vettore iniziale:

  2 96 18 38 12 45 10 55 81 43 39

- Pivot scelto: 38

  2 18 12 10 38 96 45 55 81 43 39

Commenti

- L'efficienza del quicksort dipende dalla scelta del pivot
- Scelto il pivot si generano le due partizioni
- Il primo passo di questa procedura consiste nello scambiar di posto il pivot e l'ultimo elemento dell'array

Quicksort: esempio

Vettore iniziale 8 1 4 9 6 3 5 2 7 0
Pivot (elemento centrale) 6
Scambio del pivot con l'ultimo elemento 0

Lo scambio produce il vettore

8 1 4 9 0 3 5 2 7 6

Per generare le due partizioni:

1. Si scorre il vettore da sinistra a destra con un contatore `i` che viene inizializzato all'indice minore (`inferiore`) cercando un elemento maggiore del pivot.
2. Si scorre il vettore da destra verso sinistra con un altro contatore `j` che viene inizializzato alla posizione più alta (`superiore-1`) cercando un elemento minore del pivot.

Il contatore `i` seleziona l'elemento 8 (maggiore del pivot) mentre il contatore `j` si ferma all'elemento 2 (minore del pivot).

Quicksort: esempio

A questo punto si scambiano 8 e 2 perché vadano a far parte delle corrette partizioni.

2	1	4	9	0	3	5	8	7	6

Continuando, `i` si ferma al 9, mentre `j` si ferma al 5

2	1	4	9	0	3	5	8	7	6
			↑			↑			
			i			j			

Le coppie continuano ad essere scambiate fintanto che `i` e `j` non s'incrociano. Nel nostro caso si scambiano 9 e 5.

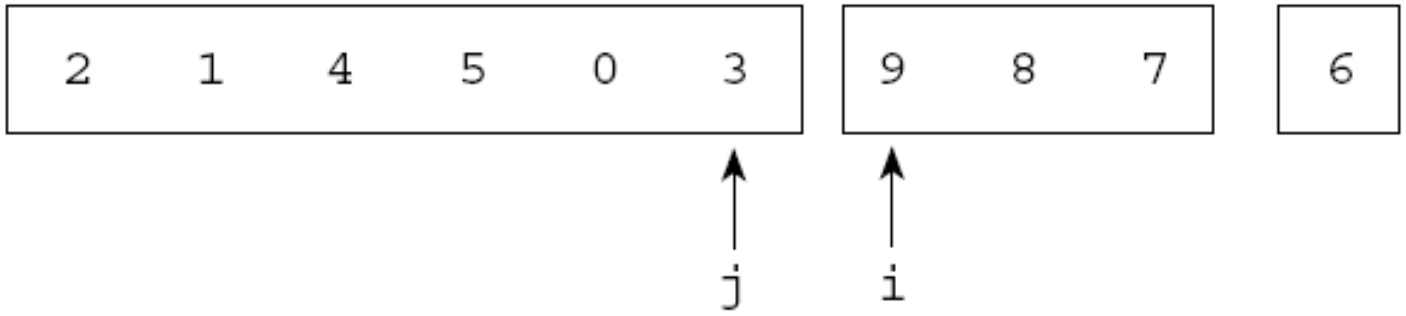
2	1	4	5	0	3	9	8	7	6

Continuando, `i` arriva all'elemento 9 e `j` scende fino al 3.

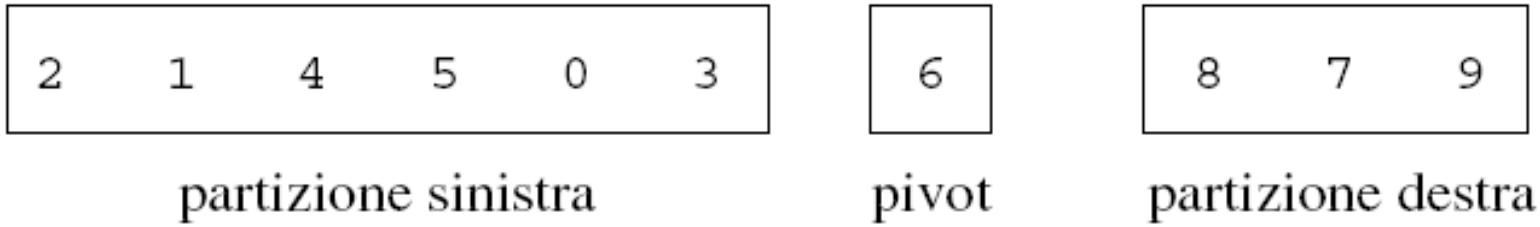
2	1	4	5	0	3	9	8	7	6
					↑	↑			
					j	i			

quicksort: esempio

Ora `i` e `j` si incrociano e non viene effettuato alcuno scambio perché i due elementi stanno già dalla parte corretta. Il vettore originale è stato diviso in due partizioni: la sinistra e la destra.

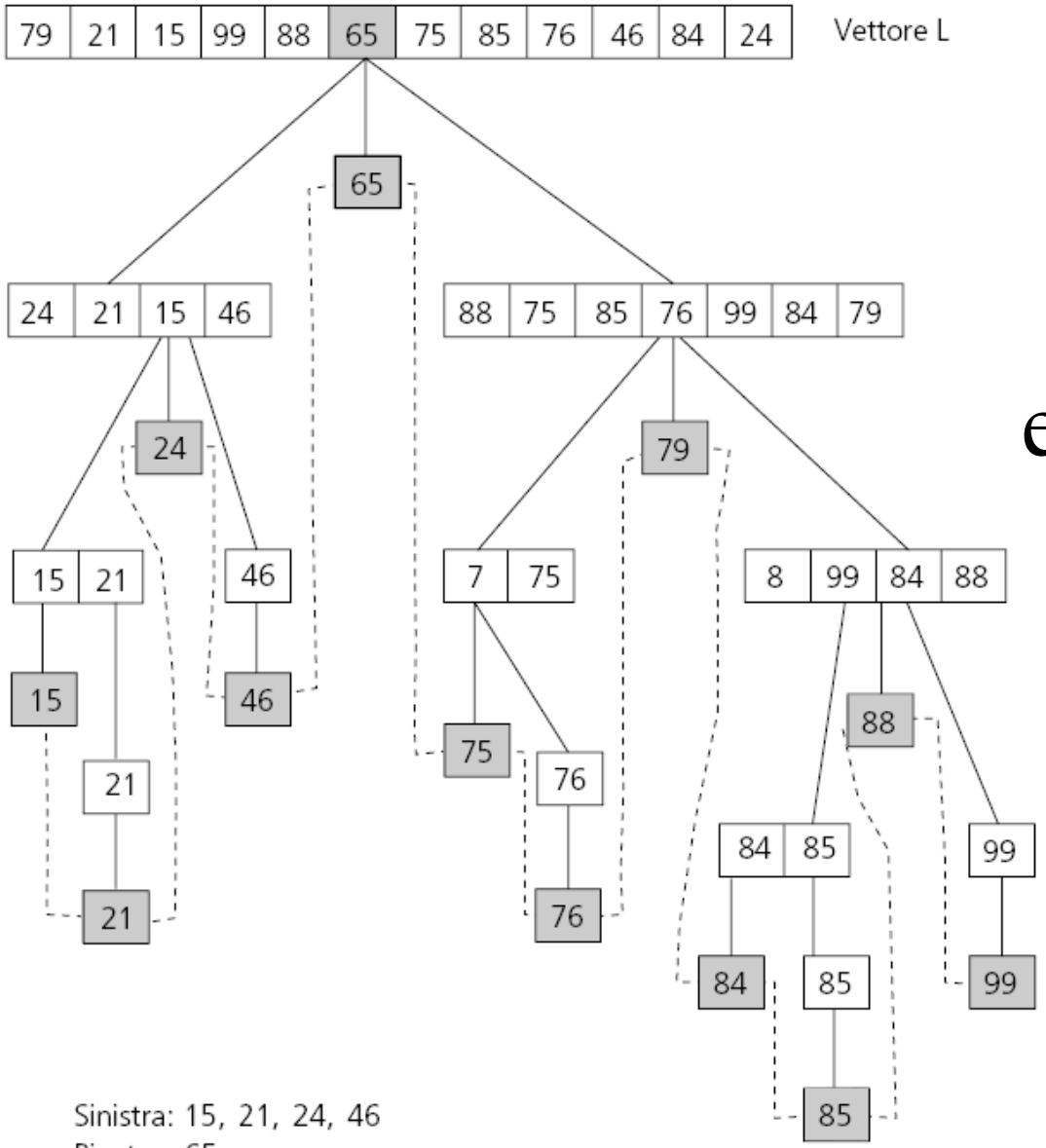


A questo punto si scambia l'elemento in posizione `i` con il pivot che sta all'ultimo posto, in modo da ritornare alla sequenza iniziale.



Riassumendo

- Si sceglie l'elemento centrale come pivot
- Gli elementi restanti si dividono nelle partizioni destra e sinistra
- Si ordina la partizione sinistra
  - Usando quicksort ricorsivamente
- Si ordina la partizione destra
  - Usando quicksort ricorsivamente
- La soluzione è ottenuta concatenando le due partizioni ed il pivot



quicksort: esempio sinottico

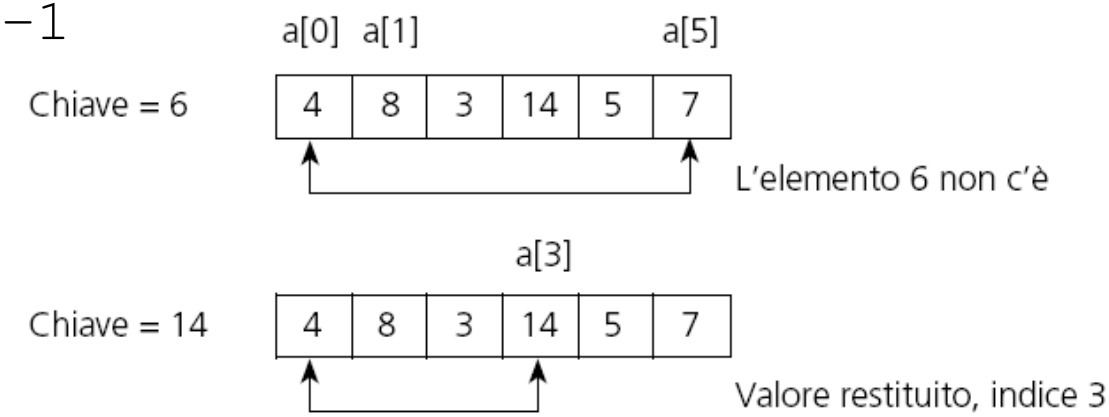
Complessità del quicksort

- Difficile da provare matematicamente
- Ne faremo un'analisi semplificata
  - $n=2^k$ e si sceglie sempre il corretto elemento mediano
- Nella prima scansione si fanno 2(n/2) confronti
- Nella seconda 4(n/4) e così via

$$2(n/2) + 4(n/4) + \ldots + n(n/n) = \underbrace{n+n+\ldots+n}_{\text{k volte}} = n*k = n*\log(n)$$

Ricerca sequenziale

- Confronta ogni elemento dell'array con una *chiave* di ricerca;
  - Si ferma se ne trova uno con il valore cercato o se termina il vettore;
  - Se l'elemento viene trovato se ne restituisce l'indice, altrimenti si restituisce il valore `-1`




```
int ricercaLineare (int vettore [], int n, int chiave)
{
   for (int i = 0; i < n; i++)
      if (vettore[i] == chiave)
         return i;
   return -1;
}
```


Ricerca Binaria

- Se il vettore è ordinato (in base al campo che contiene il valore)si può fare molto meglio

- Si parte da centro del vettore;
  - se l'elemento centrale non contiene la chiave cercata, allora si esamina l'elemento centrale del sottovettore destro (o di quello sinistro)

- Andiamo avanti ricorsivamente fino a trovare, se c'è, l'elemento cercato

Ricerca binaria

- Complessità dell'algoritmo: O(log n)
- Utile in contesti in cui valga una proprietà monotonica P(x) (al variare di x)
- Possiamo fare meglio di O(log n)?
  - No. Ogni algoritmo di ricerca basato su confronti richiede almeno Ω (log n) passi.
  - La ricerca binaria ha complessità Θ(log n)

Ricerca Binaria (iterativa)


```
RicercaBinarialterativa ( a,k )  // a contiene n elementi
    sin = 0;  des = n-1;          // a[sin] <= k <= a[des]
    trovato = 0;  indice = -1;
    WHILE ((sin <= des) && (!trovato)) {
        centro= (sin+des)/2;
        if (a[centro] > k) {
            des = centro - 1;
        } ELSE IF (a[centro] < k) {
            sin = centro;
        } ELSE {
        indice = centro; trovato = 1;}
    }
return indice;
```

ha complessità log perché ogni volta o scarto mezzo array o ho trovato il mio elemento

Ricerca binaria ed enigmi

- 128 bicchieri di vino, uno di essi è avvelenato.
- Abbiamo a disposizione un test (molto costoso) per determinare qual è il bicchiere avvelenato
- Qual è il minor numero di test possibile per arrivare alla risposta corretta?

Divido a metà e prendo un pò di vino da tutti e 64 bicchieri, se c'è soppiano che è nei primi 64 se no vedo: succe. Continuo fino a trovarlo.

Esercizio

Ad una festa partecipa una star del cinema. La star ha le seguenti caratteristiche

- Tutti la conoscono
- La star non conosce nessuno

L'obiettivo è di individuare la star chiedendo solo domande del tipo "Conosci lui (o lei)?"

1. Dimostrare che esiste una sola star.
2. Trovare una strategia per individuare la star.
3. Trovare una strategia che richieda un massimo di O(n) domande.

