# Corso di Laurea Ingegneria Informatica

## Fondamenti di Informatica 2

Dispensa 05

# Complessità

C. Limongelli

Febbraio 2008

# Contenuti

- Costo computazionale di un programma/algoritmo
- Misura del costo computazionale
- Studio del comportamento asintotico
- Notazione *O*-grande
- Stima dei tempi di calcolo associati alle funzioni di costo
- Operazione dominante
- Esercizi

# Costo di un programma: un esempio . . .

- *Problema:* dato un array *a* di interi ordinati in modo crescente, determinare se l'intero *k* è presente in *a*
- Esempio: array *a* = [1, 3, 9, 17, 34, 95, 96], intero *k* = 50
- Soluzione:

```
public static boolean
        ricercaSequenziale(int[] a, int k){
  boolean trovato;
  i = 0; trovato = false;
  while (i < a.length && !trovato) {
   if (a[i]==k) trovato = true;
   i++;  }
  return trovato; }
```

# . . . Costo di un programma: un esempio

- Il metodo ricercaSequenziale realizza un algoritmo corretto per risolvere il problema (non sfrutta l'ordinamento dell'array)
- Quanto costa la sua computazione?
- Esistono metodi/algoritmi più efficienti?

# Costo computazionale di un programma . . .

- Il costo computazionale di un programma, di un metodo o di un algoritmo è un costo definito in termini di risorse di calcolo
- Le risorse di calcolo fondamentali sono due:
  - **quantità di tempo** necessario alla computazione (**tempo**)
  - **quantità di memoria** utilizzata (**spazio**)

# . . . Costo computazionale di un programma

- Altre risorse di calcolo che possono essere rilevanti in casi particolari (non trattate in questo corso)
  - quantità di dati trasferiti da e su disco
  - quantità di traffico generata su una rete di calcolatori.
- Stabilire il costo computazionale di un programma serve a confrontare diversi programmi che risolvono lo stesso problema, in modo da scegliere il più efficiente.

# Misura del costo computazionale (tempo)…

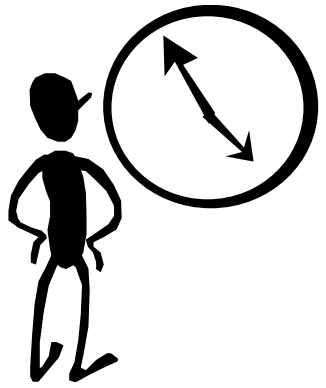
- Scegliamo una particolare configurazione dei dati in ingresso ed eseguiamo il programma misurando il tempo di calcolo. Ad esempio:
  - Configurazione dei dati in ingresso (*input*):
    *a* = [1, 3, 9, 17, 34, 95, 96], *k* = 50
  - Risultato calcolato (*output*): false
  - Tempo cronometrato: 10 $\mu$*sec*
- Cosa sappiamo sul costo del programma?
  Molto poco !
- Non abbiamo alcuna indicazione su come e quanto questo tempo calcolato sia influenzato da diversi fattori . . .

# … Misura del costo computazionale (tempo)

- **I fattori che possono influenzare il tempo calcolato:**
  - ***La macchina utilizzata*: hardware usato, sistema operativo, carico sul sistema, qualità del compilatore, linguaggio di programmazione utilizzato, ecc.**
  - ***La configurazione dei dati in ingresso*: ad esempio, se nel nostro esempio il valore cercato *k* fosse stato 1, il tempo sarebbe stato molto minore**
  - ***La dimensione dell'input*: nel nostro caso corrisponde alla dimensione del vettore; maggiore è tale dimensione, maggiore potrebbe essere il tempo necessario alla ricerca**

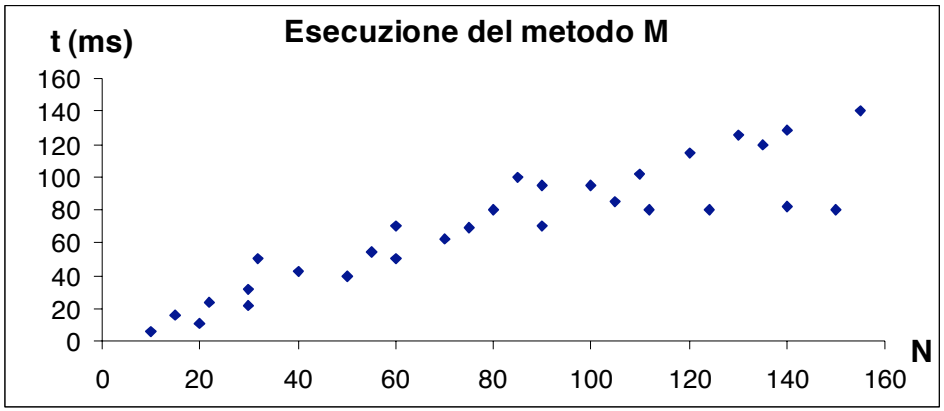
# Determinazione sperimentale della complessità …

- La complessità temporale di un metodo può essere determinata in modo sperimentale eseguendo più volte il metodo, usando ogni volta un diverso insieme di dati di ingresso, e misurando con un cronometro il tempo (macchina) impiegato per ciascuna esecuzione del metodo



# … Determinazione sperimentale della complessità

- **Il risultato della determinazione sperimentale della complessità di un metodo può essere visualizzato disegnando un grafico in cui ciascun punto descrive una diversa esecuzione del metodo**
  - **ascissa: dimensione N dell'insieme dei dati di ingresso (un numero naturale)**
  - **ordinata: tempo di esecuzione (ad esempio, in millisecondi)**




  - **nel grafico, un punto di coordinate (N,T) indica che il tempo dell'esecuzione del metodo per un insieme di dati di ingresso di dimensione N è stato T**

# Problemi legati alla determinazione sperimentale della complessità

- La determinazione sperimentale è poco oggettiva, perché oltre che dall'insieme dei dati in ingresso dipende anche:
  - dal calcolatore utilizzato,
  - dall'insieme di dati di ingresso,
  - dalla configurazione hardware/software del calcolatore utilizzato,
  - dal carico del calcolatore al momento della sperimentazione
- La misurazione sperimentale della complessità di un metodo è possibile solo se
  - il metodo è stato completamente implementato
  - si ha un calcolatore a disposizione per eseguire il metodo

# Modello di costo approssimato

- La determinazione sperimentale della complessità non è soddisfacente per capire il costo computazionale insito nell'algoritmo che il nostro metodo realizza
- Noi vogliamo conoscere questo costo:
  - indipendentemente dalla particolare macchina utilizzata;
  - indipendentemente dallo specifico linguaggio di programmazione utilizzato per codificare il problema;
  - indipendentemente dalla configurazione di ingresso;
  - al variare della dimensione dell'input.
- Questo si può ottenere con un modello di costo approssimato che permette di astrarre dai dettagli implementativi

# Misura del costo computazionale soddisfacente

- **Una misura del costo computazionale soddisfacente deve:**
  - **basarsi su un *modello di calcolo astratto*, indipendente dalla particolare macchina;**
  - **svincolarsi dalla configurazione dei dati in ingresso, ad esempio basandosi sulle configurazioni più sfavorevoli (*caso peggiore*), così da garantire che le prestazioni nei casi reali saranno al più costose quanto il caso analizzato;**
  - **essere una *funzione della dimensione dell'input*, e non un semplice numero;**
  - **essere *asintotica*, cioè fornire un'idea dell'andamento del costo all'aumentare della dimensione dell'input**

# Esempi

- Vedremo che l'algoritmo di ricerca sequenziale ha un costo lineare, cioè il costo è una funzione che varia come $a * n + b$, dove $n$ è la dimensione dell'array (dimensione dell'input), $a$ e $b$ sono due fattori costanti indipendenti dalla dimensione dell'input.
- Invece l'algoritmo di ricerca binaria ha un costo *logaritmico*, cioè il costo è una funzione che varia come $a * log(n) + b$, dove $n$ è la dimensione dell'array (dimensione dell'input) e $a$ e $b$ sono due fattori costanti indipendenti dalla dimensione dell'input.
- Quindi la ricerca binaria è un algoritmo migliore (*più efficiente*) della ricerca sequenziale.
- Ma come calcolare il costo di un algoritmo?

# Calcolo del costo per un modello astratto di macchina...

- Costo in termini di tempo
- Come stabilire il costo di ogni istruzione:
  - *istruzione semplice* : ha costo unitario;
    - x=10 e v[i]=10*x*y/4+250*z/3-12*v[i+1] e (a<b && i!=0)

    hanno entrambe costo unitario
  - *istruzione condizionale* :

    if (*<condizione>*)
    *<parte-if>*
    else
    *<parte-else>*

    ha costo pari al costo della *condizione* + il costo della *parte-if* se la condizione è vera e

    al costo della *condizione* + il costo della *parte-else* se la condizione è falsa

# ... Calcolo del costo per un modello astratto di macchina

- *istruzione iterativa* (ad esempio l'istruzione while)

  *while (<condizione>)*
  *<istruzione>*

  - ha costo pari al costo della *condizione* + il costo dell'istruzione alla *i*-esima iterazione, moltiplicato per il numero di volte che viene eseguita

- *blocco*

  *{ <istruzione>*
  *...*
  *<istruzione> }*

  - ha costo pari alla somma dei costi delle istruzioni che lo compongono

- *invocazione di un metodo*

  - ha costo pari al costo dell'esecuzione del corpo del metodo

# Esempio

- Calcolare il costo del seguente metodo sull'input
  *a* = [1, 3, 9, 17, 34, 95, 96], *k* = 9

```
public static boolean ricercaSequenziale(int[] a, int k){
    boolean trovato;
    int i;
    i = 0;                                          1
    trovato = false;                                1
    while (i < a.length && !trovato) {              4
      if (a[i]==k)                                  3
            trovato = true;                         1
      i++;                                          3
    }
    return trovato;                                 1
}
```

**Totale = 14**

# Esempio

- Calcolare il costo del seguente metodo sull'input
  *a* = [1, 3, 9, 17, 13, 95, 96], *k* = 50

```
public static boolean ricercaSequenziale(int[] a, int k){
    boolean trovato;
    int i;
    i = 0;                                          1
    trovato = false;                                1
    while (i < a.length && !trovato) {              8
      if (a[i]==k)                                  7
            trovato = true;                         0
      i++;                                          7
    }
    return trovato;                                 1
}
```

**Totale = 25**

# Astrarre dalla configurazione dei dati in ingresso: il caso peggiore

- I due esempi mostrano costi differenti
- L'analisi del caso peggiore offre una garanzia del comportamento dell'algoritmo in tutte le situazioni
  - Se l'algoritmo è sufficientemente efficiente nel caso peggiore, lo sarà anche in tutti gli altri casi.
- Caso peggiore per ricercaSequenziale: l'elemento cercato *k* non occorre nell'array *a*.

# Astrarre dalla configurazione dei dati in ingresso: il caso peggiore

- Costo di ricercaSequenziale per un array di dimensione n (dimensione dell'input) :

```
public static boolean ricercaSequenziale(int[] a, int k){
    boolean trovato;
    int i;
    i = 0;                                          1
    trovato = false;                                1
    while (i < a.length && !trovato) {              n+1
      if (a[i]==k)                                  n
            trovato = true;                         0
      i++;                                          n
    }
    return trovato;                                 1
}
```

Totale = 3*n+4

# Altre possibilità

- **Caso peggiore: configurazione dell'input più *sfavorevole* per l'algoritmo.**
- **Caso migliore: configurazione dell'input più *favorevole* per l'algoritmo.**
- **Caso medio: *media* tra le configurazioni dell'input possibili.**
  - **Si fissa una distribuzione per le configurazioni dell'input,**
  - **Si calcola il costo medio (non lo vedremo)**

# Dimensione dell'input…

- Il costo di un programma/metodo/algoritmo è una funzione dell'input.
- *Esempio*:
  - Il costo (nel caso peggiore) di ricercaSequenziale è $3 * n + 4$
  - $n$, la dimensione dell'array (numero dei suoi elementi), è il parametro che caratterizza la dimensione dell'input.

# … Dimensione dell'input…

- La scelta del parametro che caratterizza la dimensione dell'input è spesso semplice:
- ricerca in un insieme:
  - la dimensione dell'input è il numero degli elementi dell'insieme;
- Valutazione di un polinomio in un punto:
  - la dimensione dell'input è il grado del polinomio;
- Risoluzione di un sistema di equazioni lineari in *n* incognite:
  - la dimensione dell'input è il numero dei coefficienti $n^2$

# … Dimensione dell'input…

- Casi più complessi: esempio

```
public static int fattoriale(int n) {
   int i;
   fatt = 1;                          1
   int i = 1;                         1
   while (i <= n){                    n+1
         fatt = fatt*i;               n
         i++;                         n
   }
return fatt;                          1
}
```

Totale = 3*n+4

Il costo del fattoriale è lineare!!

# … Dimensione dell'input

- Dobbiamo fare attenzione a quale è il parametro che caratterizza la dimensione dell'input:
- se consideriamo il valore *n*: allora il costo rispetto alla dimensione dell'input *n* è $3n + 4$, cioè *costo lineare*.
- se consideriamo il numero di cifre *d* utilizzato per rappresentare *n*: *d* è circa uguale a $log_{10}(n)$, allora $10^d = n$ (per la def. di logaritmo) e il costo rispetto alla dimensione *d* dell'input è $3 * 10^d + 4$, cioè *costo esponenziale*!

- Nota: secondo il modello utilizzato `fatt = fatt*i` costa sempre 1. Ciò non è sempre vero.

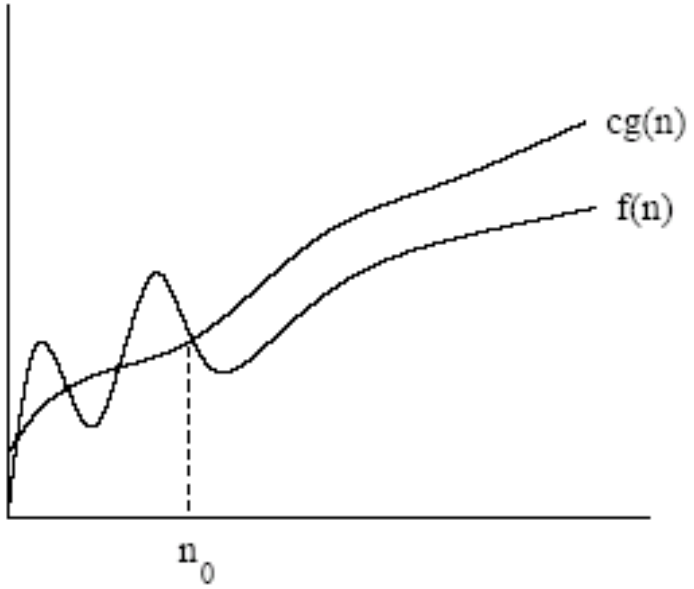
# Studio del comportamento asintotico

- Nello studio del costo dei programmi è importante valutare la funzione di costo al crescere della dimensione dell'input.
- In questo caso si possono trascurare costanti moltiplicative e termini di ordine inferiore.
- Lo studio asintotico del costo:
  - fornisce una idea dell'andamento del costo all'aumentare della dimensione dell'input,
  - generalizza (ed approssima) ulteriormente il modello di costo adottato,
  - consente di semplificare i calcoli.
- *Esempi*:
  - $a*n + b$ lineare
  - $a * n^2 + b *n + c$ quadratico
  - $a * log_b(n) + c$ logaritmico

# Notazione *O*-grande…

- ***Definizione***: **Una funzione *f* (*n*) è *O*(*g*(*n*)), se e solo se esistono due costanti positive *c* e $n_0$ tali che, per tutti i valori $n >= n_0$, si ha:**

$$f(n) <= c * g(n)$$




- ***O*(*g*(*n*)) rappresenta l'insieme di tutte le funzioni *f(n)* che sono limitate superiormente da *g(n)* a meno di una costante positiva *c*.**

# … Notazione *O*-grande

- ***f(n) = O( g(n) )*** **vuole dire che la funzione *f* cresce (in modo asintotico) al più quanto la funzione *g***
  - **a meno di fattori costanti e termini di ordine inferiore, e per *n* sufficientemente grande**
- **Intuitivamente, questo significa che, da un certo valore $n_0$ in poi, *il valore di f (n) non supera quello di g(n)*, a meno di un certo fattore di scala *c*.**
- **Se *f* (*n*) è *O*(*g*(*n*)), allora *f* (*n*) è ad esempio anche *O*(10**g*(*n*)) e anche *O*(5 + *g*(*n*)).**

# Esempi

## Esempio 1:

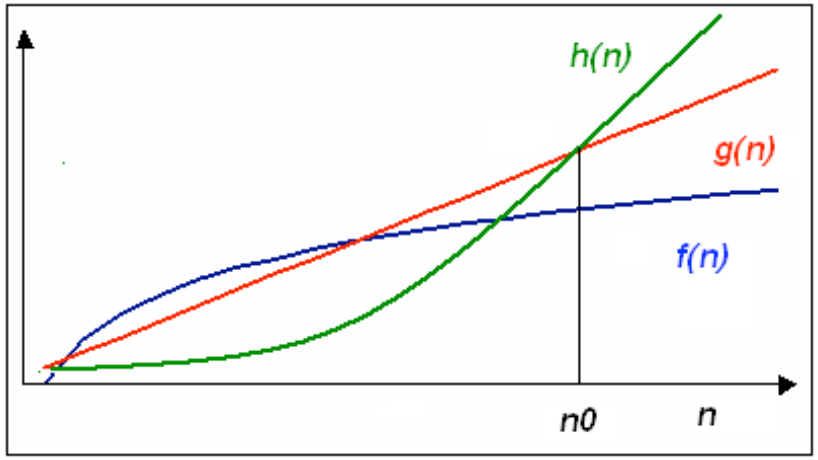
- Consideriamo la funzione $f(n) = (n + 1)^2$.
  $(n + 1)^2 <= 2n^2$, per ogni $n >= 3$.
- Applicando la definizione di *O*-grande con
  $c = 2$ e $n_0 = 3$, abbiamo che
  $f(n)$ è $O(n^2)$.

## Esempio 2:

- Consideriamo la funzione $f(n) = 2n^4 + 5n^3 - 4n^2 + 4$.
  $f(n) = 2n^4 + 5n^3 - 4n^2 + 4 < 2n^4 + 5n^4 + 4n^4 = 11n^4$.
- Applicando la definizione di *O*-grande con
  $c = 11$, ed $n_0 = 1$, abbiamo che
  $f(n)$ è $O(n^4)$.

# Osservazioni sulla notazione *O*…

- Per determinare *O* di un polinomio basta considerare il coefficiente di grado maggiore: $a_k n^k + a_{k-1} n^{k-1} + \ldots + a_0$ è $O(n^k)$
- *Esempio*: $4n^2 + 16n + 18$ è $O(n^2)$
- $3n$ è $O(n)$ ma – ovviamente - anche $O(n^2)$.
- Si sceglie però sempre l'approssimazione migliore possibile. In questo caso, $O(n)$, piuttosto che $O(n^2)$.



# …Osservazioni sulla notazione *O*

- Per determinare *O* di un logaritmo, la base non conta.

$$log_a(n) = log_a(b) * log_b(n)$$

- Nota che $n + log(n) < n + n$ da un certo valore di $n$ in poi, quindi

$$n + log(n) \text{ è } O(n).$$

# Costo di un programma/metodo/algoritmo . . .

- ***Definizione*: Un programma/metodo/algoritmo *A* ha costo (o complessità) $O(g(n))$ se la quantità di tempo sufficiente per eseguire *A* su un input di dimensione *n* nel caso peggiore è $O(g(n))$.**

# . . . Costo di un programma/metodo/algoritmo

- **Le principali funzioni di costo sono:**
  - $O(1)$ funzione di costo *costante* (che non dipende dai dati in ingresso)
  - $O(n)$ funzione di costo *lineare*
  - $O(log(n))$ funzione di costo *logaritmica*
  - $O(n * log(n))$ funzione di costo *quasi-lineare*
  - $O(n^k)$ funzione di costo *polinomiale*, cioè limitata da un polinomio di grado $k$, dove $k$ è una costante
  - $O(k^n)$ funzione di costo *esponenziale*, cioè limitata da una funzione esponenziale $k^n$, con costante $k > 1$

# Stima dei tempi di calcolo associati alle funzioni di costo

- Sia *n* il numero di operazioni effettuate
- Sia *1$\mu$-sec (1 micro secondo = $10^{-6}$ secondi)* il tempo necessario per effettuare una singola operazione

| *O*-grande | n = 10 | n = 100 | n = 1000 |
|---|---|---|---|
| $O(1)$ | 1 $\mu sec$ | 1 $\mu sec$ | 1 $\mu sec$ |
| $O(log(n))$ | 2.3 $\mu sec$ | 4.6 $\mu sec$ | 6.9 $\mu sec$ |
| $O(n)$ | 10 $\mu sec$ | 100 $\mu sec$ | 1 $msec$ |
| $O(n\ log(n))$ | 23 $\mu sec$ | 460 $\mu sec$ | 6.9 $msec$ |
| $O(n^2)$ | 100 $\mu sec$ | 10 $msec$ | 1 $sec$ |
| $O(n^3)$ | 1 $msec$ | 1 $sec$ | $16min$ |
| $O(2^n)$ | 1 $msec$ | $10^{16} anni$ | ??? |

# Un modello di costo asintotico per Java

| Operazione s | Costo $T_s$ di una esecuzione di s |
|---|---|
| operazione o istruzione semplice (senza invocazioni di metodo) – assegnazione, valutazione di condizione, incremento, istruzione return e new | 1 |
| blocco – { $s_1$ $s_2$ ... $s_K$ } | $\max(T_{s1}, T_{s2}, ..., T_{sK})$ |
| istruzione condizionale – if (cond) $s_1$ else $s_2$ | $\max(T_{cond}, T_{s1})$ – se cond è true<br>$\max(T_{cond}, T_{s2})$ – se cond è false |
| istruzione ripetitiva – while (cond) s – nell'ipotesi che il corpo del while venga eseguito N volte | $T_{cond}^{(1)} + T_s^{(1)} + T_{cond}^{(2)} + T_s^{(2)} + ... + T_{cond}^{(N)} + T_s^{(N)} + T_{cond}^{(N+1)}$ |
| invocazione di metodo o costruttore o.m($p_1$, ..., $p_K$) | $\max(T_o, T_{p1}, ..., T_{pK}, T_m)$ |

# Esempio - Costo asintotico del metodo somma

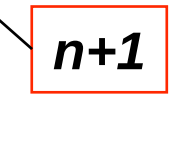
- Determinare la funzione di costo asintotico del metodo **sommaArray** (rispetto alla lunghezza *n* di **a**)

```
/* Calcola la somma degli elementi dell'array a. */
public static int sommaArray(int[] a) {
    // pre: a!=null
    int i;          // indice per la scansione di a
    int somma;      // somma degli elementi di a

    /* scandisce e somma gli elementi di a */
    somma = 0;
    for (i=0; i<a.length; i++)
        somma += a[i];
    return somma;
}
```

è un blocco, la cui istruzione più costosa è l'istruzione ripetitiva

# . . . Costo asintotico del metodo somma

```
/* Calcola la somma degli elementi dell'array a. */
public static int sommaArray(int[] a) {
    // pre: a!=null
    int i;          // indice per la scansione di a
    int somma;      // somma degli elementi di a

    /* scandisce e somma gli elementi di a */
    somma = 0;
    for (i=0; i<a.length; i++)
        somma += a[i];
    return somma;
}
```




- il costo complessivo è *2*n+1*

$$T_{sommaArray}(n) = O(n)$$

# Operazione dominante

- Il modello di costo asintotico suggerisce di
  - identificare i blocchi di operazioni che vengono eseguiti in sequenza
  - determinare, per ciascun blocco, il costo asintotico della sola operazione più costosa
    - si trascurano le operazioni il cui costo è dominato dal costo di altre operazioni
- Un modo semplificato per calcolare la funzione di costo asintotico di un metodo
  - considera il costo della sola operazione dominante del metodo
    - una operazione il cui costo di esecuzione è asintoticamente uguale al costo di esecuzione dell'intero metodo

# Valutazione semplificata del costo: operazione dominante . . .

- *Definizione*: Sia *A* un programma con costo $O(f_A(n))$. Una operazione si dice dominante (per *A*) se, nel caso peggiore, viene ripetuta un numero di volte che è $O(f_A(n))$.
  - Il costo asintotico del programma corrisponde al costo asintotico dell'operazione dominante

# . . . Valutazione semplificata del costo: operazione dominante

- Per individuare le operazioni dominanti, basta esaminare le condizioni e le istruzioni eseguite nei cicli più interni (annidati) dei programmi, oppure eseguiti nell'ambito delle attivazioni ricorsive.
- *Esempio:* Per valutare il costo di ricercaSequenziale tramite l'individuazione dell'operazione dominante, si consideri che ci sono diverse operazioni dominanti:
  - `i < a.length && !trovato`
  - `i++`
  - `a[i]==k`
- Ciascuna di esse viene eseguita nel caso peggiore $n$ volte. Quindi il costo è $O(n)$.

# Ricerca binaria

- Inizialmente lo spazio di ricerca è composto da tutti gli elementi dell'array
- Finché non è stato trovato un elemento dell'array uguale alla chiave (e comunque lo spazio di ricerca è non vuoto)
  - confronta la chiave con l'elemento centrale dello spazio di ricerca
  - se la chiave è uguale all'elemento centrale dello spazio di ricerca, allora la ricerca termina con esito positivo
  - se invece la chiave è maggiore dell'elemento centrale dello spazio di ricerca, rimuovi dallo spazio di ricerca l'elemento centrale e tutti quelli alla sua sinistra
  - se invece la chiave è minore dell'elemento centrale dello spazio di ricerca, rimuovi dallo spazio di ricerca l'elemento centrale e tutti quelli alla sua destra

# Ricerca binaria: codifica…

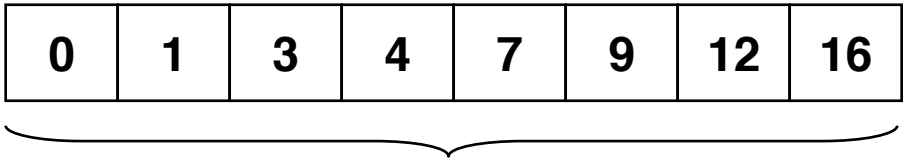
```
/* Ricerca chiave nell'array a ordinato in modo
 * non descrescente, restituendo l'indice di un
 * elemento di a uguale a chiave (oppure -1). */
public static int ricercaBinaria(int[] a, int chiave) {
    // pre: a!=null && a è ordinato in modo non decrescente
    int posizione;   // posizione di un elemento di a
                     // uguale a chiave, se esiste
    int sinistra;    // indice del primo elemento dello
                     // spazio di ricerca
    int destra;      // indice del primo elemento oltre
                     // lo spazio di ricerca
    int centro;      // indice dell'elemento centrale
                     // dello spazio di ricerca
    boolean trovato; // vero se l'elemento viene trovato
```

# ...Ricerca binaria: codifica

```
   /* inizialmente lo spazio di ricerca comprende
     * tutti gli elementi di a */
    sinistra = 0;
    destra = a.length;
    trovato = false;
    /* cerca l'indice di un elemento di a uguale
     * a chiave, se esiste, mediante la ricerca binaria */
    posizione = -1;
    while (sinistra<destra && !trovato) {
        centro = (sinistra+destra)/2;
        if (a[centro]==chiave){     // trovato
            trovato = true;
             posizione = centro; }
        else if (a[centro]>chiave)  // continua a sinistra
            destra = centro;
        else                        // continua a destra
            sinistra = centro+1;
    }
    return posizione;
}
```

# Analisi dei costi della ricerca binaria

*Ipotesi: n è una potenza di due $n = 2^h$*




*Ad ogni iterazione lo spazio di ricerca si dimezza*

| Confronto | spazio di ricerca |
|---|---|
| 1° | $n$ |
| 2° | $n / 2$ |
| 3° | $n / 2^2$ |
| …. | …. |
| *h+1-esimo* | $n / 2^h = 1$ |

***Istruzione dominante:***

*confronto di una componente dell'array con la chiave.*

*Quante volte viene eseguita l'istruzione dominante?*

*h+1*, ma $h = \log_2 n$, quindi $\log_2 n + 1$ volte

La complessità asintotica di ricerca binaria è *O(log n)*

# Esercizi

- Determinare il costo rispetto al tempo dell'algoritmo di fusione di due array ordinati.

- Determinare il costo rispetto al tempo del seguente metodo che visualizza una matrice A di dimensione *n* x *m*, utilizzando la notazione *O*.

```
public static void stampaRigheMatrice(short[][] A) {
   for (int i = 0; i < A.length; i++) {
       for (int j = 0; j < A[0].length; j++)
           System.out.print(A[i][j] + " ");
           System.out.println(); // fine riga
       }
   }
```

# … Esercizio

- Determinare il costo in tempo dei seguenti metodi che calcolano la somma degli elementi della diagonale principale di una matrice quadrata di interi, utilizzando la notazione *O*.

```
public static int sommaDiagonale1(int[][] A) {
  int n = A.length;
  int somma = 0;
  for (int i = 0; i < n; i++)
    for (int j = 0; j < n; j++)
      if (i == j) somma += A[i][j];

  return somma;
}
```

# Esercizio…

```
public static int sommaDiagonale2(int[][] A) {
  int n = A.length;
  int[] v = new int[n];
  int somma = 0;
  for (int i = 0; i < n; i++)
    v[i] = A[i][i];
  for (int i = 0; i < n; i++)
    somma += v[i];

  return somma;
}
public static int sommaDiagonale3(int[][] A) {
  int n = A.length;
  int somma = 0;
  for (int i = 0; i < n; i++)
     somma += A[i][i];

  return somma;
}
```

# Riferimenti al libro di testo

- Per lo studio di questi argomenti si fa riferimento al libro di testo, e in particolare al Capitolo 23