

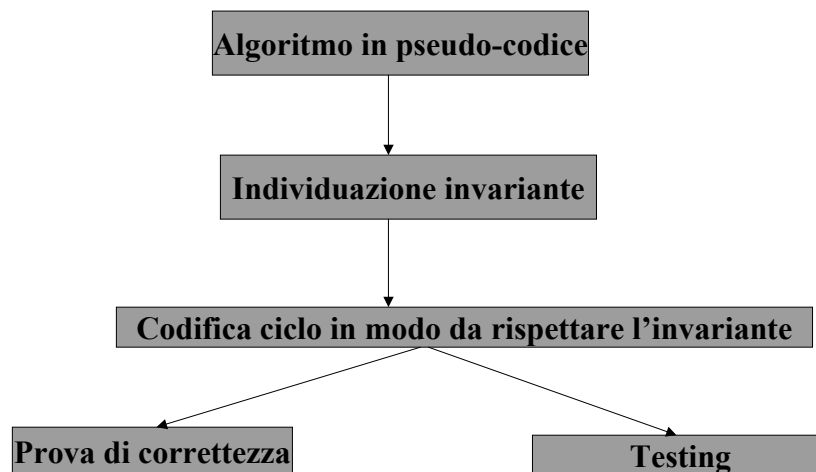
# Scrivere programmi corretti

L'esempio della ricerca binaria o  
dicotomica

**J. Bentley, Programming Pearls, Addison Welsey.**

1

Schema processo produzione funzione iterativa



2

## Il problema

B è un vettore di n interi, ordinato in modo crescente, cioè tale che  $B[i] \leq B[i+1]$ , per  $0 \leq i < n$ .

Se  $n = 0$ , il vettore è vuoto.

Vogliamo sapere se un intero t è presente in B e, se lo è, qual'è la sua posizione.

La risposta al problema è quindi un valore p (per posizione) tra 0 e n-1, se l'elemento è presente, e -1 altrimenti.

3

## L'algoritmo

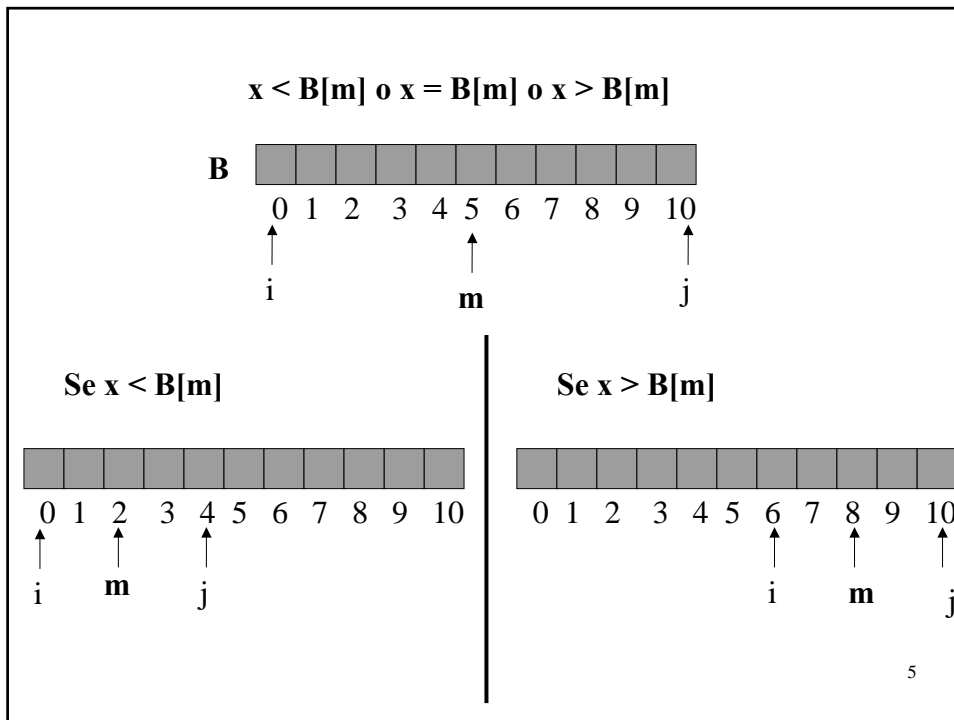
Inizialmente l'intervallo di ricerca

dell'elemento è l'intero vettore (array).

L'intervallo di ricerca è ristretto in funzione del confronto tra l'elemento cercato e quello mediano: se l'elemento cercato è più piccolo di quello confrontato la ricerca si restringe alla metà sinistra del vettore, altrimenti a quella destra.

Il processo si ripete fino a che si trova l'elemento, oppure l'intervallo di ricerca diventa vuoto.

4



**Volendo implementare l'algoritmo con un programma iterativo, conviene utilizzare un invariante per progettare correttamente il ciclo.**

**Invariante:  $deveStare(i,j)$  come abbreviazione dell'affermazione "se  $t$  è presente nel vettore dovrà essere nell'intervallo  $i,j$ , cioè tra gli elementi  $B[i], \dots, B[j]$ ".**

**pseudocodice:**

1. inizializza intervallo a  $0, n-1$ .
2. loop  
{invariante  $deveStare(intervallo)$ }
3. if (intervallo è vuoto) esci dal ciclo e restituisci -1
4. calcola il punto mediano dell'intervallo in  $m$
5. usa  $m$  per restringere l'intervallo di ricerca
6. se  $t$  è trovato esci dal ciclo e restituisci la sua posizione.

6

## Raffinamento

Vediamo di implementare riga per riga lo pseudocodice:

1. **inizializza intervallo a 0, n-1**

Qui decidiamo di rappresentare l'intervallo con due indici  $i$  e  $j$ .  
L'inizializzazione che rispetta l'invariante è

1.  $i = 0, j = n-1$

quindi  $deveStare(0, n-1)$  è vero all'ingresso nel ciclo.

il prossimo passo è raffinare la terza e la quarta riga:

3. **if (intervallo è vuoto) esci dal ciclo e restituisci -1**

4. **calcola il punto mediano dell'intervallo in  $m$**

Un intervallo così individuato è vuoto se  $i > j$  (quando  $i = j$  c'è un elemento)

Quindi la riga 3 diventa:

3. **if ( $i > j$ ) return -1;**

E la quarta:

4.  **$m = (i+j)/2$**

7

## Raffinamento

1.  $i=0, j= n-1$

{ $deveStare(i,j)$ }

loop

3. **if ( $i > j$ ) return -1;**

4.  **$m = (i+j)/2$**

5. **usa  $m$  per restringere l'intervallo di ricerca**

6. **se  $t$  è trovato esci dal ciclo e restituisci la sua posizione.**

Per raffinare queste ultime due righe si deve confrontare  $t$  con  $B[m]$  e compiere delle azioni nel rispetto dell'invariante:

**if ( $B[m] == t$ ) return  $m$ ;**

**else if ( $B[m] < t$ ) [implica  $deveStare(m+1,j)$ , quindi]  $i=m+1$  ;**

**else [qui  $B[m] > t$  implica  $deveStare(i,m-1)$ , quindi]  $j=m-1$ ;**

8

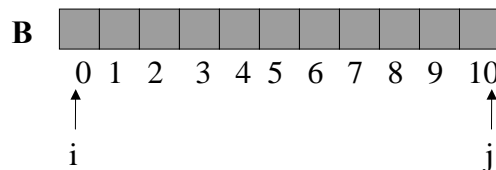
## Raffinamento: fine

```
int binarySearch(int *v, int t, unsigned int n)
/* restituisce la posizione di t in un vettore ordinato v se presente,
-1 altrimenti
prec v!=NULL && v[i] ≤ v[i+1], 0 ≤ i < n.*/
{ int i,m,j;
  i = 0;
  j = n-1;
  while (i <= j)
  {invariante :deveStare(i,j)}
    { m= (i+j)/2;
      if ( v[m] < t) i = m+1;
        else if (v[m] == t) return m;
          else /* v[m] > t*/
            j = m-1;
        }
  return -1;}

```

9

## Testing per la ricerca binaria: scelta dei dati a scatola nera



Dalle post condizioni ricaviamo che abbiamo  $n+1$  possibili output:  $i$  valori tra 0 e  $n-1$  in caso di successo e  $-1$  in caso di insuccesso. Poiché gli elementi sono ordinati, notiamo che il caso di insuccesso può verificarsi in  $n+1$  modi diversi (*perchè?*). Per ogni vettore, dividiamo l'insieme dei possibili valori cercati in sottoinsiemi equivalenti rispetto al test: gli insiemi  $\{B[0]\}, \dots, \{B[n-1]\}$ , ciascuno dei quali corrisponde a una ricerca con successo e gli  $n+1$  intervalli  $(-\infty, B[0]), (B[0], B[1]), \dots, (B[n-2], B[n-1]), (B[n-1], \infty)$  per le ricerche con insuccesso.

10

## Testing per la ricerca binaria: scelta dati a scatola trasparente

Nessuna, una o due esecuzioni del ciclo, con le possibili diverse uscite, sono state già considerate, aggiungiamo il caso in cui cerchiamo un elemento presente nel vettore ma fuori dal range della ricerca.

## Testing per la ricerca binaria: scelta dati per l'organizzazione del test

Non ha molta importanza quali dati contiene il vettore. Possiamo organizzare il testing nel modo più semplice possibile considerando vettori di lunghezza  $n$  (che parte da 0 fino a un valore massimo), e inizializzando il vettore con multipli di 10. Considereremo inoltre il caso in cui il vettore contiene elementi tutti diversi e quello in cui il vettore contiene elementi tutti uguali

11

## Organizzazione del testing 1

```
#include ...
#define MAXL 100
...

int main(void)
{int n,i, x[MAXL];
for (n=0;n<MAXL;n++)
  {for (i=0;i<=n;i++) /* inizializzazione del vettore,
notare che x contiene n+1 elementi */
  x[i] = 10*i;
/* ricerca sul vettore vuoto */
if (n==0) assert(binSearch(x,10,n) == -1);
else
  {for (i=0;i<n;i++) /*ricerca sui primi n elementi */
    {/*ricerca su tutti i presenti*/
    assert( binSearch(x,10*i,n)==i );
/* ricerca sui primi n intervalli di assenza:
(-∞,0), (0,10), ..., ((n-2)*10, (n-1)*10)*/
    assert(binSearch(x,10*i-5,n)==-1);
    }
  }
}
```

12

## Organizzazione del testing 2

```
/* ricerca nell'ultimo intervallo: ((n-1)*10,∞)*/
assert(binSearch(x,10*n-5,n)==-1);
/*ci assicuriamo che non localizzi un elemento pur presente nel
vettore ma al di fuori dell'ambito della ricerca */
assert(binSearch(x,10*n,n)==-1);
    } /*chiude l'else */
/* facciamo le stesse verifiche su elementi tutti uguali */
for (i=0;i<n;i++)
    x[i] = 10;
if (n==0) assert(binSearch(x,10,n) == -1);
else /*ricerca di un presente */
{assert((0 <= binSearch(x,10,n))&&(binSearch(x,10,n) < n));
/*next ricerca nell'intervallo di assenza (-∞,10)*/
assert(binSearch(x,5,n)==-1);
/*oppure ricerca nell'intervallo di assenza (10,∞)*/
assert(binSearch(x,15,n)==-1);
}
} /*chiude il ciclo for */
return 0;}
```

13