ALGORITMI E STRUTTURE DATI

Prof. Manuela Montangero

A.A. 2022/23

BINARY SEARCH
(RICERCA BINARIA o DICOTOMICA)

UNIMORE
UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

BINARY SEARCH

PROBLEMA

INPUT:

- sequenza ordinata di n elementi memorizzata in un array L

$$L[0] \leq L[1] \leq L[2] \leq \ldots \leq L[n-2] \leq L[n-1]$$

- Un elemento key dello stesso tipo degli elementi in L

OUTPUT:

- Posizione di key in L, se key sta in L, -1 altrimenti

ESEMPI

INPUT

$L = < 1,3,4,6,7,8,23,56,78,89,125,136 >$
$key = 8$

OUTPUT

5

INPUT

$L = < 1,3,4,6,7,8,23,56,78,89,125,136 >$
$key = 9$

OUTPUT

-1

BINARY SEARCH

PROBLEMA

INPUT:

- sequenza ordinata di n elementi memorizzata in un array L

$$L[0] \leq L[1] \leq L[2] \leq \ldots \leq L[n-2] \leq L[n-1]$$

- Un elemento key dello stesso tipo degli elementi in L

OUTPUT:

- Posizione di key in L, se key sta in L, -1 altrimenti

SOLUZIONE Naive

Partendo da $i = 0$, confrontare $L[i]$ con key. Se sono uguali, restituire i.
Se $key < L[i]$, restituire -1, altrimenti incrementare i. Fermarsi quando $i = n$, restituendo -1.

QUANTO COSTA? $T(n) \in O(n)$ POSSIAMO FARE DI MEGLIO?

BINARY SEARCH

IDEA PIU' INTELLIGENTE

- Confrontiamo key con l'elemento in posizione centrale k di L
- Se sono uguali, abbiamo trovato k
- Se $key > L[k]$, continuo a cercare solo a DESTRA di k
- Altrimenti, continuo a cercare solo a SINISTRA

COME "continuo a cercare" a DESTRA o SINISTRA?

Esattamente come ho fatto prima

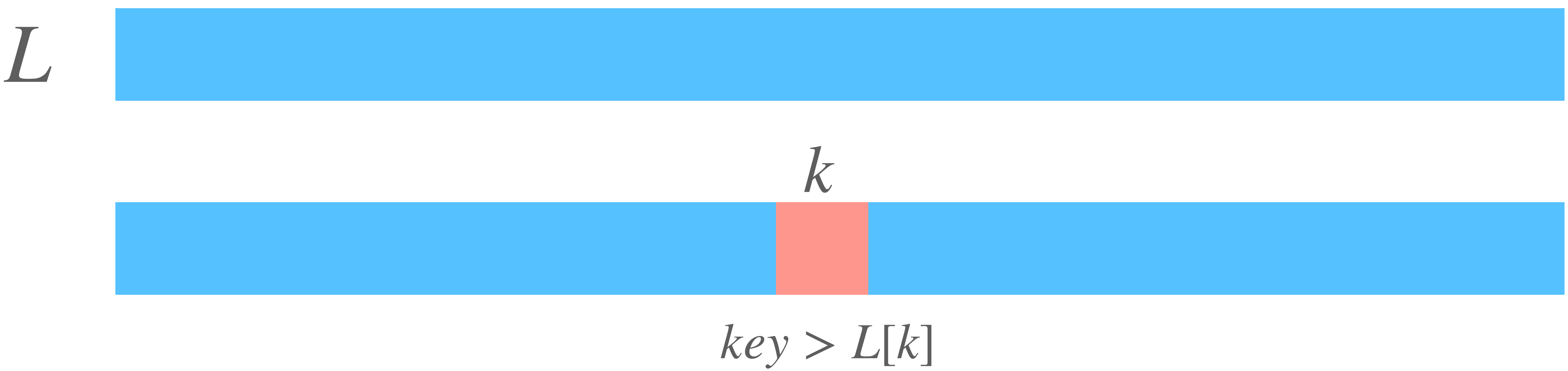
QUANDO smetto di cercare?

Quando trovo key
OPPURE
Quando non ci sono più elementi tra cui cercare,
e vuol dire che key non c'è

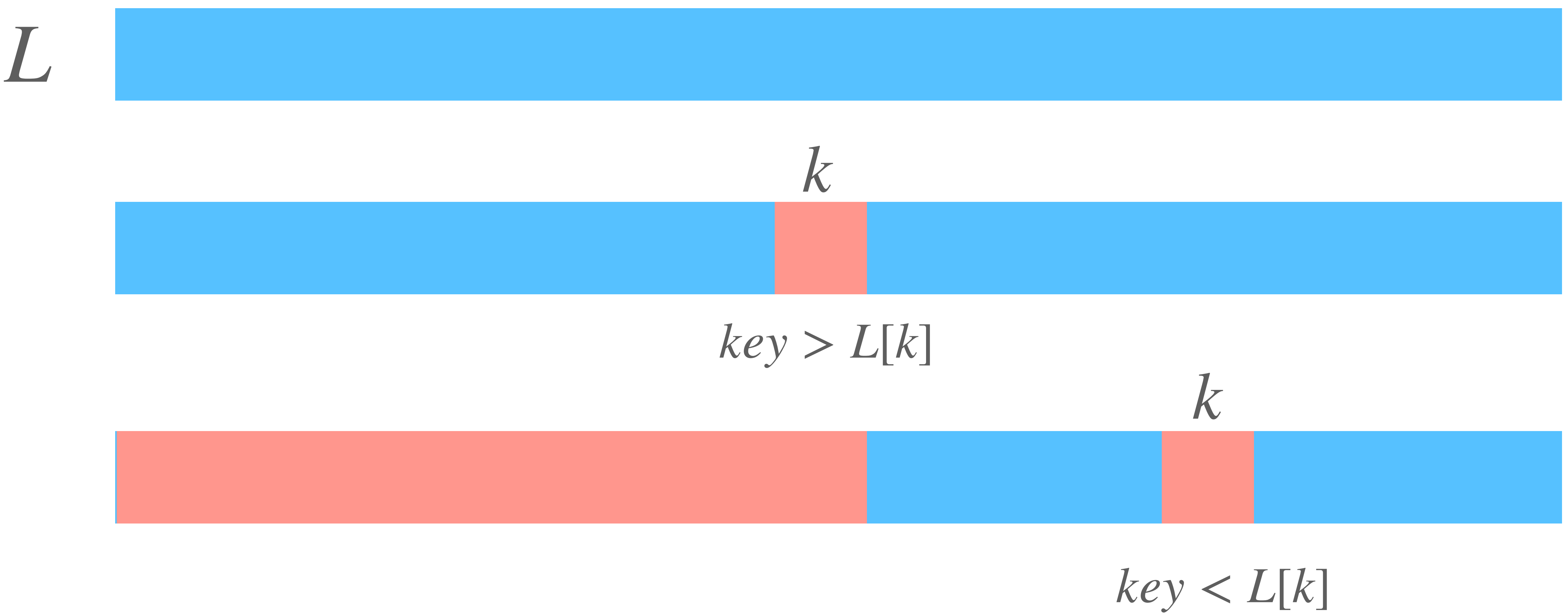
BINARY SEARCH

L

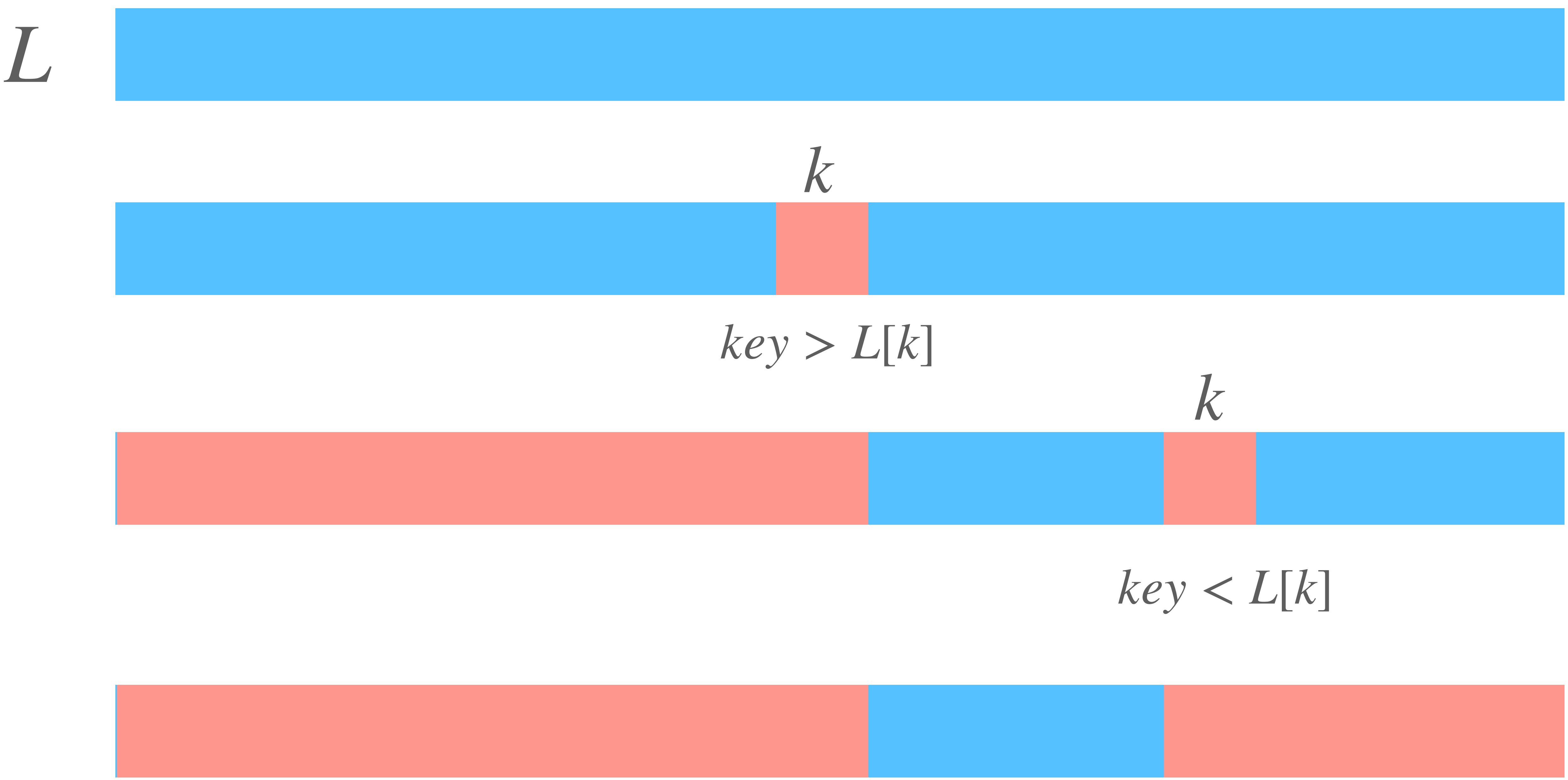
BINARY SEARCH



BINARY SEARCH



BINARY SEARCH

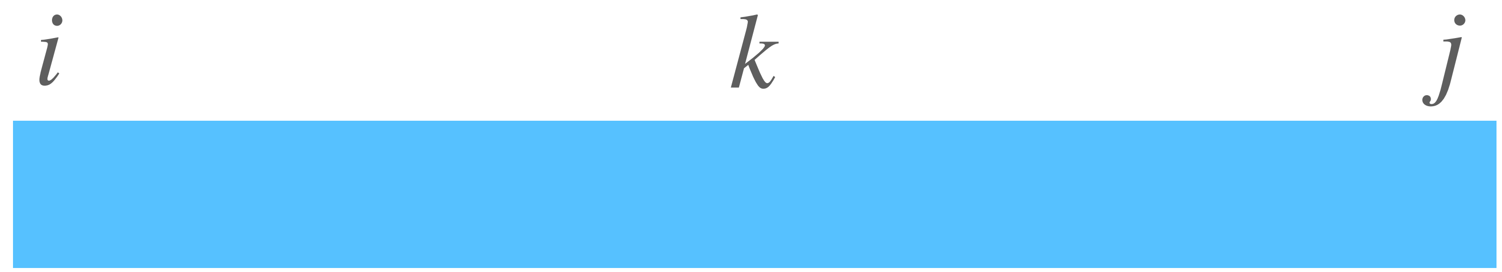


… e cosi via…

BINARY SEARCH

In un generico passo dell'algoritmo dovremo cercare key all'interno di una porzione dell'array L, che va dall'indice i all'indice j

Come si calcola l'indice dell'elemento che sta a meta' di questa porzione?

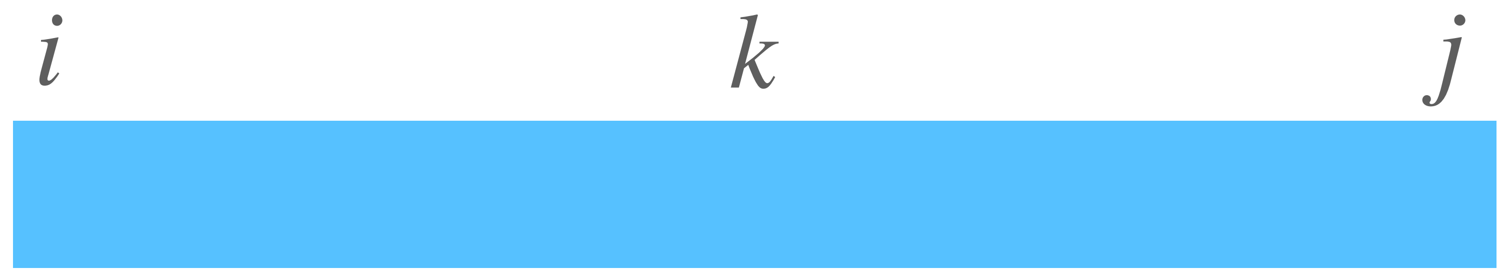


Primo tentativo:

BINARY SEARCH

In un generico passo dell'algoritmo dovremo cercare key all'interno di una porzione dell'array L, che va dall'indice i all'indice j

Come si calcola l'indice dell'elemento che sta a meta' di questa porzione?



Primo tentativo: $k = \frac{j - i}{2}$

Se i = 100 e j = 109 —> k = 4,5

- k non e' nell'intervallo tra i e j
- k non e' intero

BINARY SEARCH

In un generico passo dell'algoritmo dovremo cercare key all'interno di una porzione dell'array L, che va dall'indice i all'indice j

Come si calcola l'indice dell'elemento che sta a meta' di questa porzione?

Secondo tentativo: per arrivare a k, devo aggiungere ad i la meta' della distanza tra i e j

$$k = \lfloor i + \frac{j-i}{2} \rfloor = \lfloor \frac{j+i}{2} \rfloor$$

ESEMPI

3, 4, 5, 6, 7, 8, 9, 10, 11

3, 4, 5, 6, 7, 8, 9, 10, 11, 12

BINARY SEARCH

PSEUDO-CODICE

```
BinarySearch(L,n,key)
 i := 0
 j := n -1
 while i ≤ j
   k := ⌊(i+j)/2⌋
   if key = L[k]
    then
     return k
   if key < L[k]
    then
     j := k-1
    else
     i := k+1
 return -1
```

COSTO COMPUTAZIONALE

$O(1)$

$O(1)$

Quante volte
viene ripetuto
il ciclo while
NEL CASO
PEGGIORE?

Il caso peggiore si ha
quando *key*
non e' nell'array

BINARY SEARCH

PREPOSIZIONE

Sia $\#it(n)$ il numero di iterazioni necessarie nel caso peggiore a "Binary Search" per arrivare ad avere $i > j$, allora abbiamo che

$$\#it(n) \leq \log n + 1$$

DIMOSTRAZIONE PER INDUZIONE:

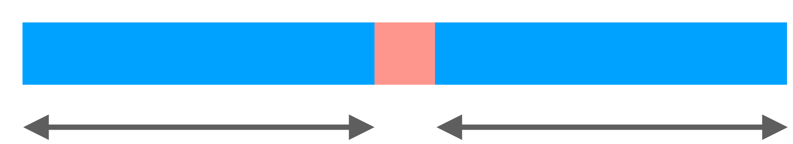
CASO BASE: $n = 1 \rightarrow \#it(1) = 1 \leq 1 = \log 1 + 1$

IPOTESI INDUTTIVA: $\forall k = 1,\ldots,n-1 : \#it(k) \leq \log k + 1$

PASSO INDUTTIVO: dimostriamo la tesi per n

$$\#it(n) \leq 1 + \#it(\lfloor \frac{n}{2} \rfloor)$$

n DISPARI



$$\frac{n-1}{2} = \frac{n}{2} - \frac{1}{2} = \lfloor \frac{n}{2} \rfloor$$

BINARY SEARCH

PREPOSIZIONE

Sia $\#it(n)$ il numero di iterazioni necessarie nel caso peggiore a "Binary Search" per arrivare ad avere $i > j$, allora abbiamo che

$$\#it(n) \leq \log n + 1$$

DIMOSTRAZIONE PER INDUZIONE:

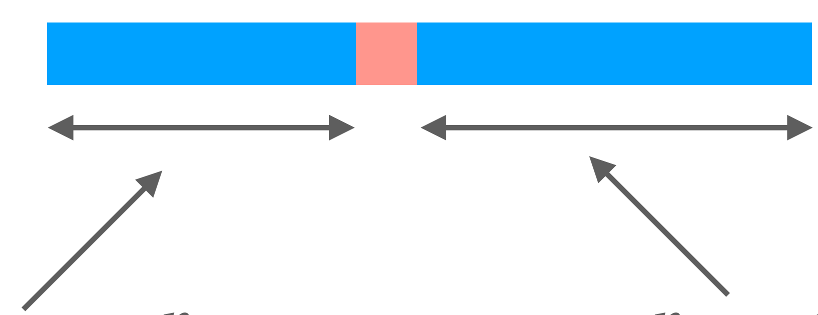
CASO BASE: $n = 1 \rightarrow \#it(1) = 1 \leq 1 = \log 1 + 1$

IPOTESI INDUTTIVA: $\forall k = 1,\ldots,n-1 : \#it(k) \leq \log k + 1$

PASSO INDUTTIVO: dimostriamo la tesi per n

$$\#it(n) \leq 1 + \#it(\lfloor \frac{n}{2} \rfloor)$$

n PARI



$\frac{n}{2} - 1 \leq \lfloor \frac{n}{2} \rfloor$ $\quad\quad$ $\frac{n}{2} = \lfloor \frac{n}{2} \rfloor$

BINARY SEARCH

PREPOSIZIONE

Sia $\#it(n)$ il numero di iterazioni necessarie nel caso peggiore a "Binary Search" per arrivare ad avere $i > j$, allora abbiamo che

$$\#it(n) \leq \log n + 1$$

DIMOSTRAZIONE PER INDUZIONE:

CASO BASE: $n = 1 \rightarrow \#it(1) = 1 \leq 1 = \log 1 + 1$

IPOTESI INDUTTIVA: $\forall k = 1, \ldots, n-1 : \#it(k) \leq \log k + 1$

PASSO INDUTTIVO: dimostriamo la tesi per n

$$\#it(n) \leq 1 + \#it(\lfloor \frac{n}{2} \rfloor) \underset{\text{Per ipotesi induttiva}}{\leq} 1 + \log(\lfloor \frac{n}{2} \rfloor) + 1 \underset{\text{Perche' } \lfloor n/2 \rfloor \leq n/2 \text{ e logn e' crescente}}{\leq} 1 + \log(\frac{n}{2}) + 1$$

BINARY SEARCH

PREPOSIZIONE

Sia $\#it(n)$ il numero di iterazioni necessarie nel caso peggiore a "Binary Search" per arrivare ad avere $i > j$, allora abbiamo che

$$\#it(n) \leq \log n + 1$$

DIMOSTRAZIONE PER INDUZIONE:

CASO BASE: $n = 1 \rightarrow \#it(1) = 1 \leq 1 = \log 1 + 1$

IPOTESI INDUTTIVA: $\forall k = 1,\ldots,n-1 : \#it(k) \leq \log k + 1$

PASSO INDUTTIVO: dimostriamo la tesi per n

$$\#it(n) \leq 1 + \#it(\lfloor \frac{n}{2} \rfloor) \leq 1 + \log(\lfloor \frac{n}{2} \rfloor) + 1 \leq 1 + \log(\frac{n}{2}) + 1$$

$$\leq 1 + \log n - \log 2 + 1 = 1 + \log n$$

BINARY SEARCH

PSEUDO-CODICE

```
BinarySearch(L,n,key)
 i := 0
 j := n -1
 while i ≤ j
   k := ⌊(i+j)/2⌋
   if key = L[k]
    then
     return k
   if key < L[k]
    then
     j := k-1
    else
     i := k+1
 return -1
```

COSTO COMPUTAZIONALE

$O(1)$

$O(1)$

$O(\log n)$

$\xrightarrow{+}$ $O(\log n)$

Decisamente meglio dell'algoritmo naive!

BINARY SEARCH ricorsiva

Un altro modo di vedere le cose

Scriviamo una FUNZIONE che operi su una porzione qualsiasi dell'array delimitata dagli indici *i* (a sinistra) e *j* (a destra)

```
BinarySearch(L,i,j,key)
 if j-i < 0
   then
    return -1
   else
    k := ⌊(i+j)/2⌋
    if key = L[k]
      then
       return k
    if key < L[k]
      then
       return BinarySearch(L,i,k-1,key)
      else
       return BinarySearch(L,k+1,j,key)
```

BINARY SEARCH ricorsiva

Un altro modo di vedere le cose

Scriviamo una FUNZIONE che operi su una porzione qualsiasi dell'array delimitata dagli indici *i* (a sinistra) e *j* (a destra)

```
BinarySearch(L,i,j,key)
 if j-i < 0
  then
   return -1
  else
   k := ⌊(i+j)/2⌋
   if key = L[k]
    then
     return k
   if key < L[k]
    then
     return BinarySearch(L,i,k-1,key)
    else
     return BinarySearch(L,k+1,j,key)
```

i e j NON sono INPUT del problema

È necessario specificare con quali valori questi parametri devono essere istanziati nella prima chiamata, quella PRINCIPALE

Chiamata principale

`BinarySearch(L,0,n-1,key)`

BINARY SEARCH ricorsiva

ESEMPIO

```
BinarySearch(L,i,j,key)
  if j-i < 0
    then
      return -1
    else
      k := ⌊(i+j)/2⌋
      if key = L[k]
        then
          return k
      if key < L[k]
        then
          return BinarySearch(L,i,k-1,key)
        else
          return BinarySearch(L,k+1,j,key)
```

BinarySearch(L,0,n-1,key)

INPUT

$$\begin{array}{l} \quad\quad 0,\ 1,\ 2,\ \ 3,\ \ \ 4,\ \ \ 5,\ \ 6,\ \ 7,\ \ 8,\ \ 9 \\ L = <1,\ 2,\ 5,\ 10,\ 11,\ 14,\ 17,\ 23,\ 24,\ 30> \\ key = 23 \end{array}$$

PRIMA CHIAMATA —> chiamata principale

BinarySearch(L,0,9,23)

$$i = 0 \quad j = 9$$

$$j - i = 9 > 0$$

$$k = \lfloor \frac{i+j}{2} \rfloor = \lfloor \frac{9}{2} \rfloor = 4$$

$$L[4] = 11 < 23 = key$$

La prima chiamata viene
SOSPESA
e il controllo passa alla
SECONDA CHIAMATA

BINARY SEARCH ricorsiva

ESEMPIO

```
BinarySearch(L,i,j,key)
  if j-i < 0
    then
      return -1
    else
      k := ⌊(i+j)/2⌋
      if key = L[k]
        then
          return k
      if key < L[k]
        then
          return BinarySearch(L,i,k-1,key)
        else
          return BinarySearch(L,k+1,j,key)
```

BinarySearch(L,0,n-1,key)

INPUT

0, 1, 2, 3, 4, 5, 6, 7, 8, 9

$L = < 1,\ 2,\ 5,\ 10,\ 11,\ 14,\ 17,\ 23,\ 24,\ 30 >$

$key = 23$

PRIMA CHIAMATA —> chiamata principale

BinarySearch(L,0,9,23)

$i = 0 \quad j = 9$

$j - i = 9 > 0$

$$k = \lfloor \frac{i+j}{2} \rfloor = \lfloor \frac{9}{2} \rfloor = 4$$

$L[4] = 11 < 23 = key$

SECONDA CHIAMATA

BinarySearch(L,5,9,23)

$i = 5 \quad j = 9$

$j - i = 9 - 5 = 4 > 0$

$$k = \lfloor \frac{i+j}{2} \rfloor = \lfloor \frac{14}{2} \rfloor = 7$$

$L[7] = 23 = key$

return 7

BINARY SEARCH ricorsiva

ESEMPIO

```
BinarySearch(L,i,j,key)
  if j-i < 0
   then
     return -1
   else
     k := ⌊(i+j)/2⌋
     if key = L[k]
       then
         return k
     if key < L[k]
       then
         return BinarySearch(L,i,k-1,key)
       else
         return BinarySearch(L,k+1,j,key)
```

BinarySearch(L,0,n-1,key)

INPUT

$$\begin{array}{lcccccccccc} & 0, & 1, & 2, & 3, & 4, & 5, & 6, & 7, & 8, & 9 \\ L = < & 1, & 2, & 5, & 10, & 11, & 14, & 17, & 23, & 24, & 30 > \end{array}$$

$key = 23$

PRIMA CHIAMATA —> chiamata principale

BinarySearch(L,0,9,23)

$i = 0 \quad j = 9$

$j - i = 9 > 0$

$k = \lfloor \frac{i+j}{2} \rfloor = \lfloor \frac{9}{2} \rfloor = 4$

$L[4] = 11 < 23 = key$

7

SECONDA CHIAMATA

BinarySearch(L,5,9,23)

7

$i = 5 \quad j = 9$

$j - i = 9 - 5 = 4 > 0$

$k = \lfloor \frac{i+j}{2} \rfloor = \lfloor \frac{14}{2} \rfloor = 7$

$L[7] = 23 = key$

return 7

BINARY SEARCH ricorsiva

Un altro modo di vedere le cose

Scriviamo una FUNZIONE che operi su una porzione qualsiasi dell'array delimitata dagli indici *i* (a sinistra) e *j* (a destra)

```
BinarySearch(L,i,j,key)
  if j-i < 0
    then
      return -1
    else
      k := ⌊(i+j)/2⌋
      if key = L[k]
        then
         return k
      if key < L[k]
        then
         return BinarySearch(L,i,k-1,key)
        else
         return BinarySearch(L,k+1,j,key)
```

in alcuni casi la funzione restituisce un risultato direttamente

in alcuni casi la funzione restituisce un risultato, che è frutto di una chiamata a funzione che è la stessa funzione che siamo definendo!

RICORSIONE

BINARY SEARCH ricorsiva

Un altro modo di vedere le cose

Scriviamo una FUNZIONE che operi su una porzione qualsiasi dell'array delimitata dagli indici *i* (a sinistra) e *j* (a destra)

```
BinarySearch(L,i,j,key)
 if j-i < 0
   then
    return -1
   else
    k := ⌊(i+j)/2⌋
    if key = L[k]
      then
       return k
    if key < L[k]
      then
       return BinarySearch(L,i,k-1,key)
      else
       return BinarySearch(L,k+1,j,key)
```

TUTTI i possibili flussi di esecuzione TERMINANO con un'istruzione `return`