UNIVERSITÀ
DI TORINO

Laboratorio di Programmazione I

Lezione n. 10: Ricorsione (parte 2)

Alessandro Mazzei

Slides: prof. Elvio Amparore

Regole per gli esercizi d'esame

Funzioni **iterative**:

- Una sola istruzione **return**. Usare variabili sentinella per combinare le condizioni logiche calcolate dalle funzioni.
- Non si possono usare: **case**, **switch**, **break**.

Funzioni **ricorsive**:

- Non si possono usare cicli **for** o **while**.
- Non ci sono limiti sul numero di istruzioni **return** usate.
- Prestate attenzione al tipo di funzione ricorsiva richiesta (se dicotomica o no, ecc...)

Preparazione all'esame: svolgere tutti gli esercizi forniti, inclusi gli esercizi **iter-**** e **recur-** (domani ...)**

Outline

- Ripasso sui pattern di ricorsione
- ricorsioni.c
- Ricorsione su coppie di array
- coppieR.c
- Lab10-Es1 Ricerca binaria
- Lab10-Es2 Somma di due
- sqrtR.c
- Ricorsione su matrici
- matriciR.c
- Ricorsione non lineare
- fibonacci_vis.c
- permutazioni.c

Outline

- Ripasso sui pattern di ricorsione
- ricorsioni.c
- Ricorsione su coppie di array
- coppieR.c
- Lab10-Es1 Ricerca binaria
- Lab10-Es2 Somma di due
- sqrtR.c
- Ricorsione su matrici
- matriciR.c
- Ricorsione non lineare
- fibonacci_vis.c
- permutazioni.c

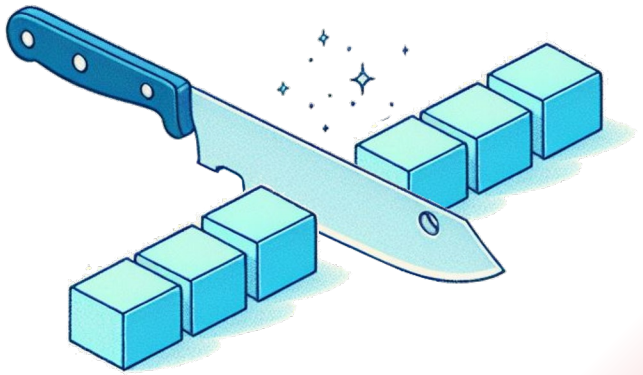
Ricorsione su array

Consideriamo questi approcci:

1. Ricorsione **con indici** su intervalli semiaperti [Lab09]:
   a. Intervallo **[0, len)** fisso
   b. Cambia l'indice della ricorsione: ***i*** ∈ **[0, len)**

2. Ricorsione **con intervalli** semiaperti generali:
   a. Intervallo **[left, right)** variabile
   b. La ricorsione restringe l'intervallo (da sinistra o da destra)

3. Ricorsione **dicotomica**:
   a. Intervallo **[left, right)** variabile
   b. La ricorsione dimezza l'intervallo.



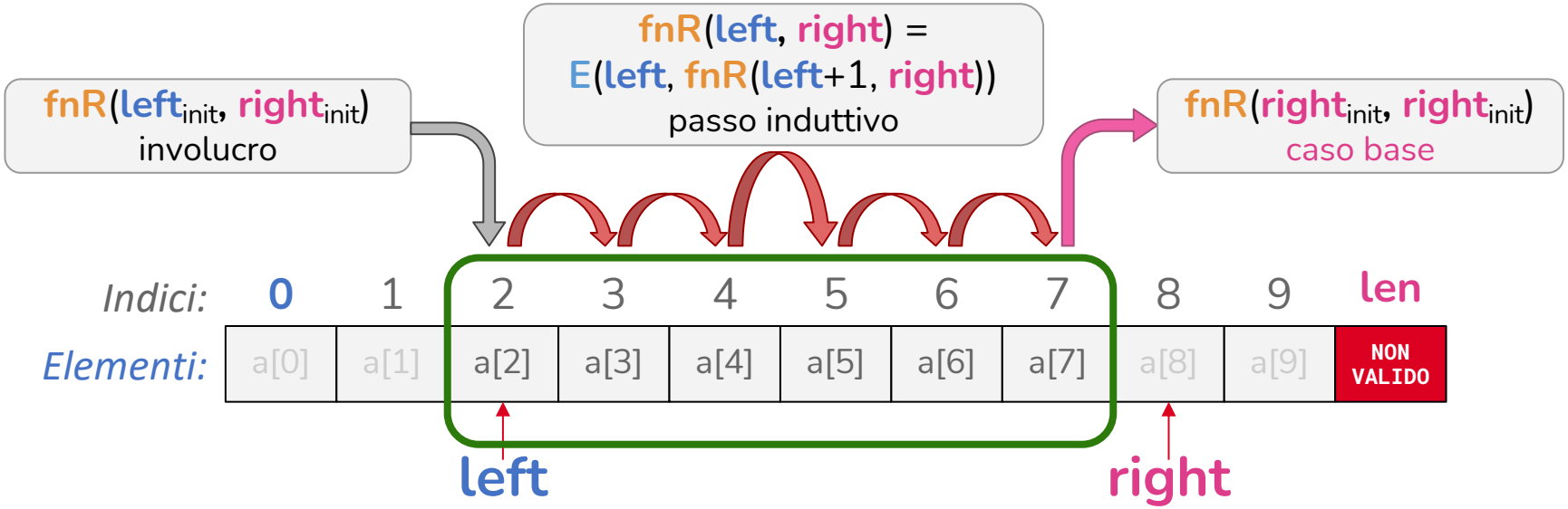
Ricorsione **controvariante** con intervalli semiaperti **generali**

Consideriamo un intervallo semiaperto di indici

$i \in$ **[left, right)** con: $0 \le$ **left** $\le$ **right** $\le$ **len**

Ricorsione controvariante: estremo **left** crescente, **right** fisso

- Caso base: **left** ≥ **right** ⇒ valore base o su array vuoto
- Passo induttivo: uso **a**[**left**] ⇒ Ricorsione su **[left**+1, **right)**
- Involucro: non serve



controvariante = da sinistra a destra, o *left-to-right*.

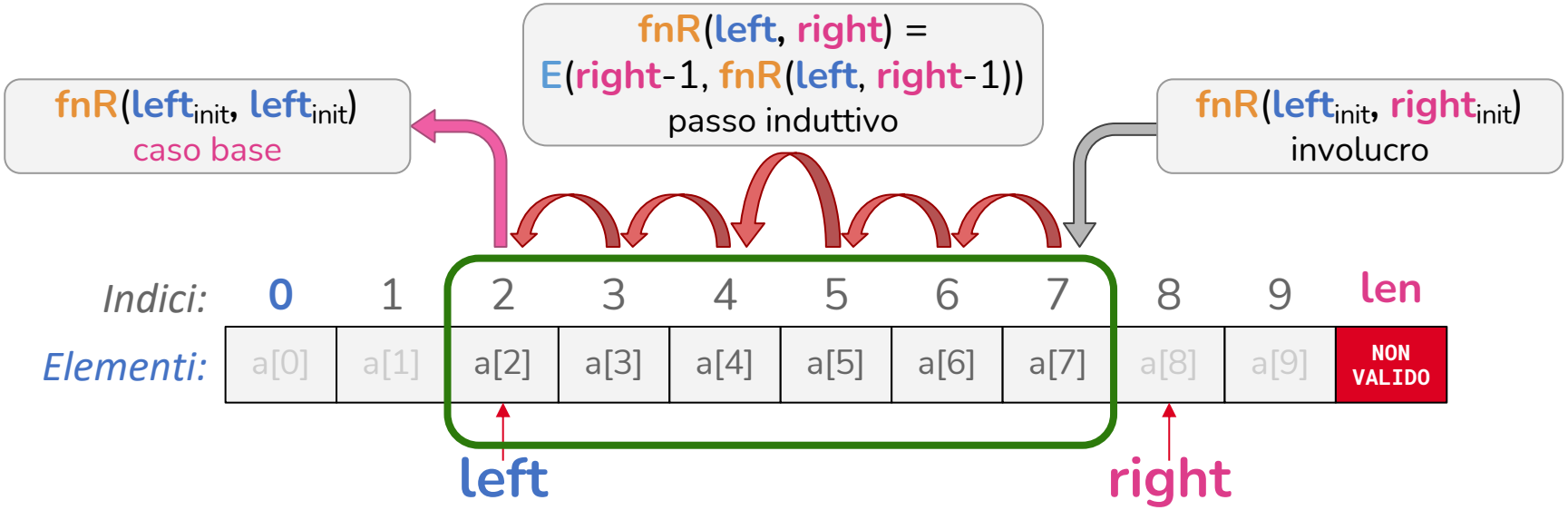
Ricorsione **covariante** con intervalli semiaperti **generali**

Consideriamo un intervallo semiaperto di indici

$i \in$ **[left, right)** con: $0 \le$ **left** $\le$ **right** $\le$ **len**

Ricorsione covariante: **left** fisso, estremo **right** decrescente

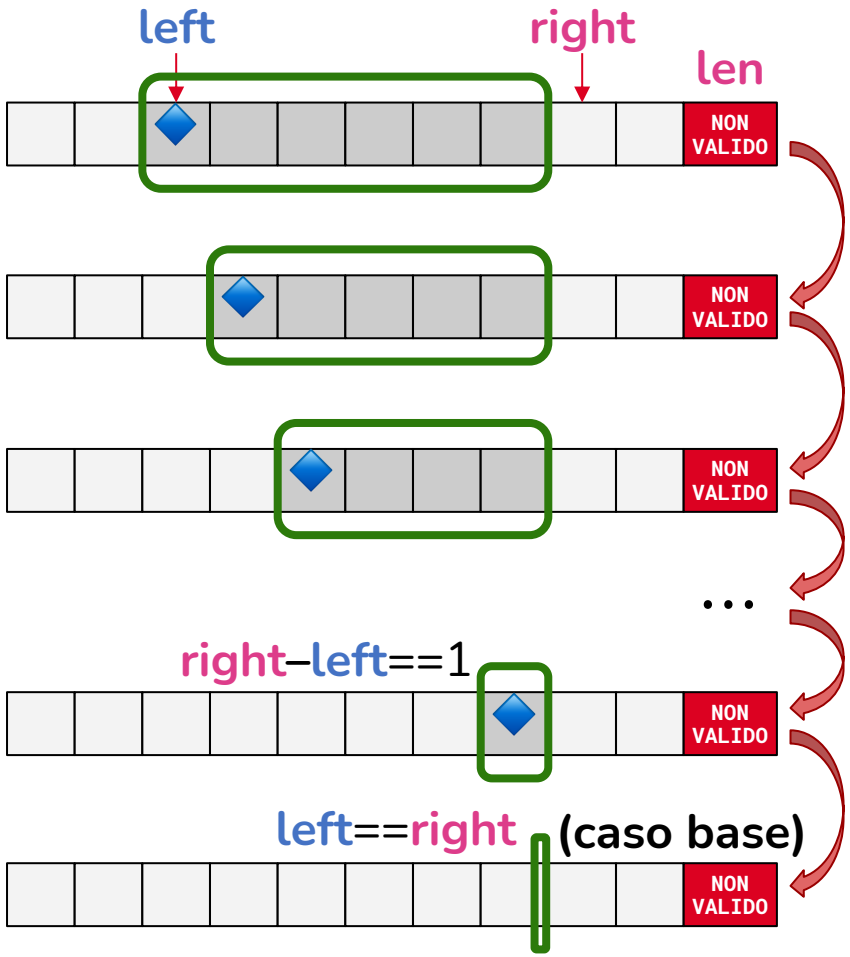
- Caso base: **left** ≥ **right** ⇒ valore base o su array vuoto
- Passo induttivo: uso **a**[**right**–1] ⇒ Ricorsione su **[left, right**–1**)**
- Involucro: non serve



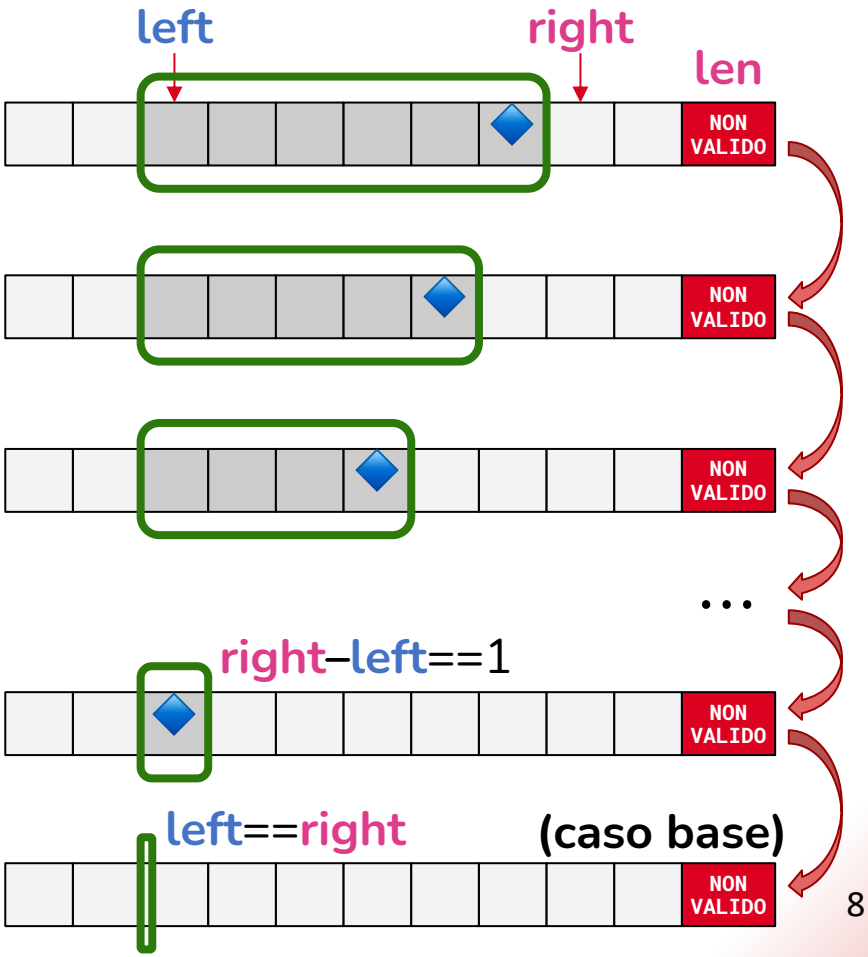
covariante = da destra a sinistra, o *right-to-left*.

Ricorsione con intervalli semiaperti generali

Ricorsione controvariante:
estremo **left** crescente, **right** fisso
uso **a**[**left**]



Ricorsione covariante:
estremo **left** fisso, **right** decrescente
uso **a**[**right** – 1]



Modelli di ricorsione su array con intervalli semiaperti **generali**

Ricorsione controvariante: estremo **left** crescente, **right** fisso

```
retType fnR(const int a[], const size_t left, const size_t right) {
        if (left >= right)
                return val_base; // caso base
        else
                return E(a[left], fnR(a, left+1, right));
}
retType fn(const size_t aLen, const int a[]) { // involucro
        return fnR(a, 0, aLen)
}
```

Ricorsione covariante: **left** fisso, estremo **right** decrescente

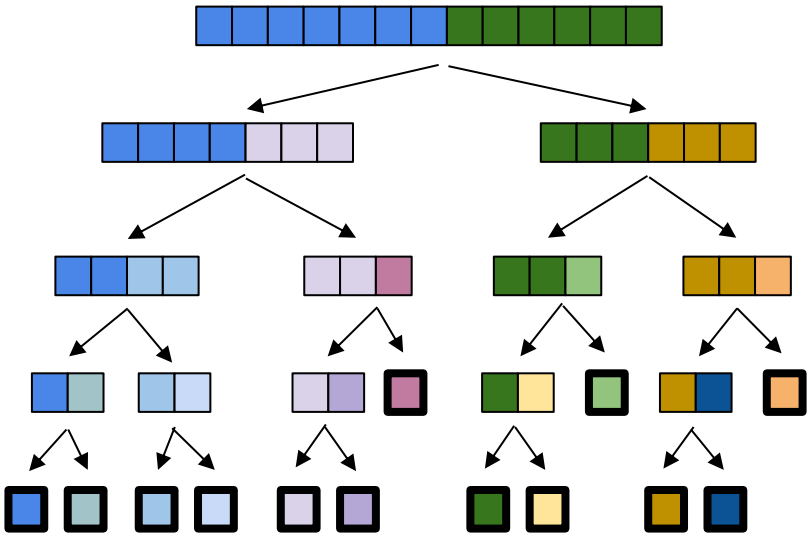
```
retType fnR(const int a[], const size_t left, const size_t right) {
        if (left >= right)
                return val_base; // caso base
        else
                return E(a[right-1], fnR(a, left, right-1));
}
retType fn(const size_t aLen, const int a[]) { // involucro
        return fnR(a, 0, aLen)
}
```

Ricorsione e ricerca dicotomica

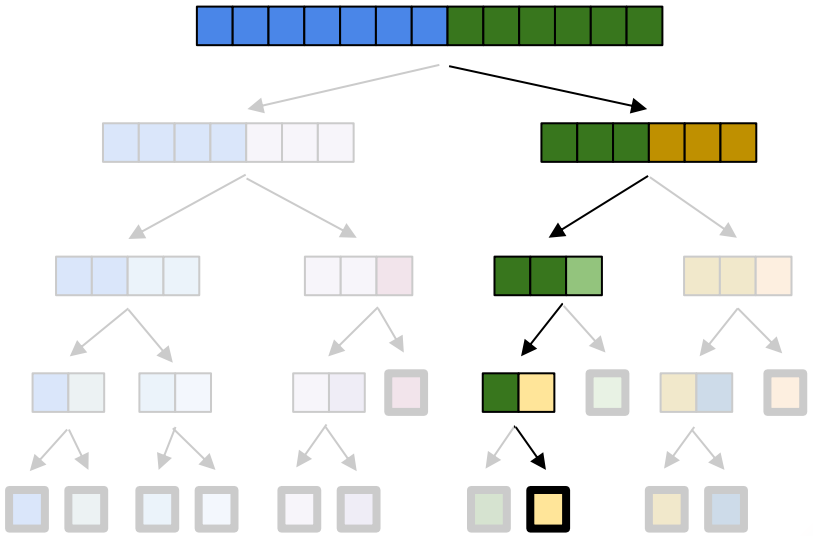
Distinguiamo due approcci dicotomici diversi:

- **Ricorsione**: tutti gli intervalli vengono "visitati" in una chiamata ricorsiva. La ricorsione forma un albero di intervalli.
- **Ricerca**: ad ogni dimezzamento, si considera solo uno dei due intervalli, in funzione di un criterio.
- **Caso base**: l'intervallo ha un solo valore.

Ricorsione dicotomica:



Ricerca dicotomica:



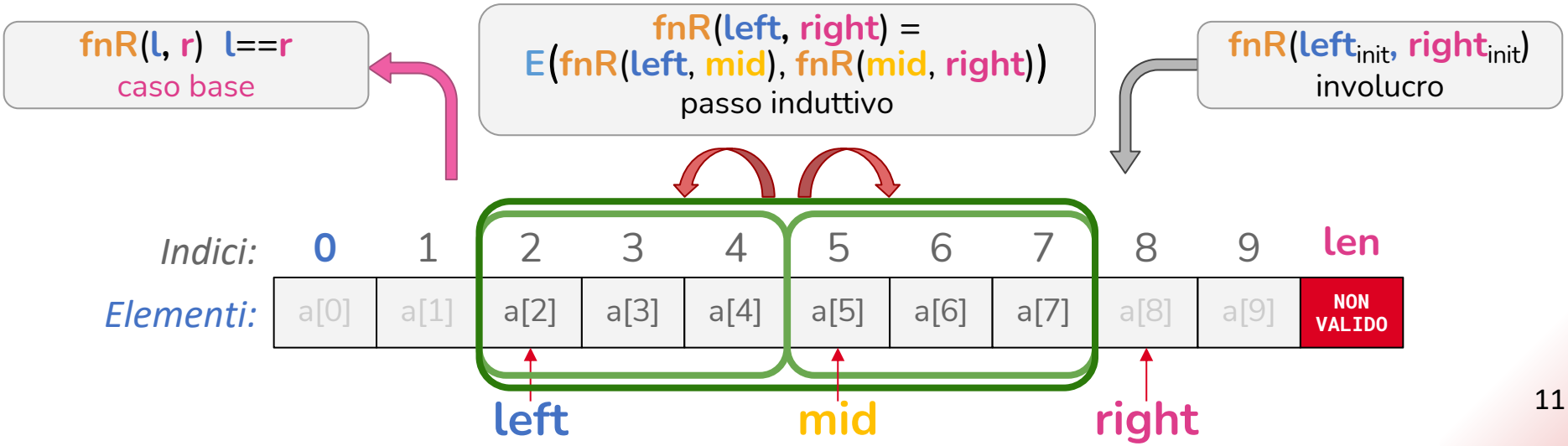
Ricorsione **dicotomica** con intervalli semiaperti **generali**

Consideriamo un intervallo semiaperto di indici

$i \in$ **[left, right)** con: $0 \le$ **left** $\le$ **right** $\le$ **len**

Ricorsione dicotomica:

- Caso base: **left** ≥ **right** ⇒ valore base o su array vuoto
- Caso base: **right**–**left**==1 ⇒ uso **a**[**left**]
- Passo induttivo:
  - definisco il perno (pivot): **mid** = **left** + (right–**left**)/2
  - ricorsione sui due intervalli: **[left, mid)** e **[mid, right)**
- Involucro: inizializza intervallo semiaperto



Modelli di ricorsione su array con intervalli semiaperti **generali**

Ricorsione dicotomica: dimezzamento intervallo

```
retType fnR(const int a[], const size_t left, const size_t right) {
        if (left >= right) {
                return val_base; // caso base: intervallo vuoto
        }
        else if (right-left == 1) {
                return V(a[left]); // caso base: intervallo con 1 elemento
        }
        else { // passo induttivo: dimezzamento intervallo
                int mid = left + (right-left) / 2;
                return E(fnR(a, left, mid), fnR(a, mid, right));
        }
}
retType fn(const size_t aLen, const int a[]) { // involucro
        return fnR(a, 0, aLen)
}
```

NOTA: Si può gestire il caso base dell'intervallo vuoto spostandolo nell'involucro. Questo caso base si verifica soltanto nel caso in cui **a**[] sia un array vuoto.

Modelli di ricorsione su array con intervalli semiaperti **generali**

Ricerca dicotomica: dimezzamento intervallo

```
retType fnR(const int a[], const size_t left, const size_t right) {
        if (left >= right) {
                return val_base; // caso base: intervallo vuoto
        }
        else if (right-left == 1) {
                return V(a[left]); // caso base: intervallo con 1 elemento
        }
        else { // passo induttivo: dimezzamento intervallo
                int mid = left + (right-left) / 2;
                if (Crit(a[mid]))
                        return fnR(a, left, mid);   // discesa a sinistra
                else
                        return fnR(a, mid, right); // discesa a destra
        }
}
retType fn(const size_t aLen, const int a[]) { // involucro
        return fnR(a, 0, aLen)
}
```

Esercizi sulla ricorsione con intervalli

Aprire il file **ricorsioni.c** fornito nel codice iniziale, leggere il contenuto ed infine implementare le funzioni ricorsive dichiarate, seguendo la specifica.

Outline

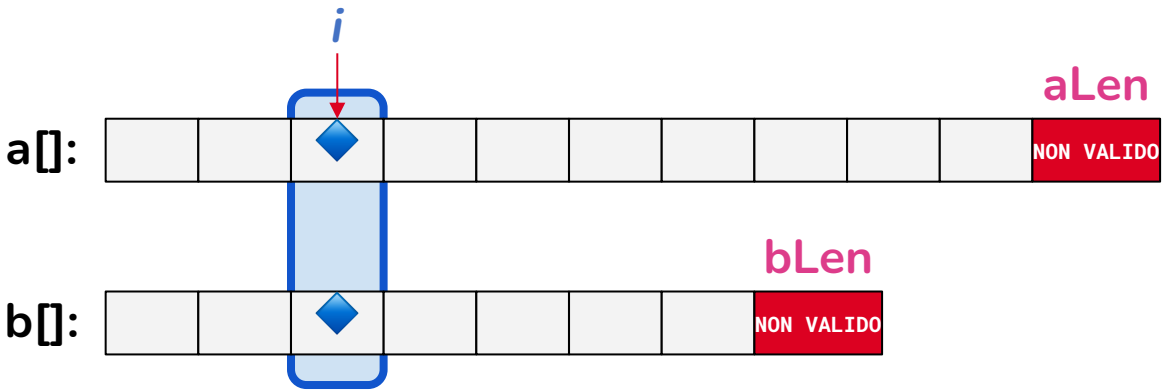
- Ripasso sui pattern di ricorsione
- ricorsioni.c
- Ricorsione su coppie di array
- coppieR.c
- Lab10-Es1 Ricerca binaria
- Lab10-Es2 Somma di due
- sqrtR.c
- Ricorsione su matrici
- matriciR.c
- Ricorsione non lineare
- fibonacci_vis.c
- permutazioni.c

Ricorsione su coppie di array

Siano **a**[] e **b**[] due array di lunghezza indipendente.

Diamo una formulazione ricorsiva per risolvere un problema che opera sulle coppie di elementi di **a**[] e **b**[] nelle medesime posizioni, con due criteri:

- **fino alla lunghezza minima** (tutte le coppie sono definite)
- **fino alla lunghezza massima** (le coppie sono definite sino a **min**(aLen, bLen), dopodichè si usano gli elementi singoli rimasti.



Modelli di ricorsione su coppie di array di lunghezza indipendente

Ricorsione controvariante fino alla lunghezza minima.

- Tutte le coppie (a[i], b[i]) sono definite

```
retType fnR(const size_t minLen, const int a[],
            const int b[], const size_t i)
{
        if (i >= minLen)
                return val_base; // caso base
        else
                return E(a[i], b[i], fnR(minLen, a, b, i+1));
}
retType fn(const size_t aLen, const int a[],
           const size_t bLen, const int b[]) { // involucro
        return fnR(aLen<bLen ? aLen : bLen, a, b, 0)
}
```

Modelli di ricorsione su coppie di array di lunghezza indipendente

Ricorsione controvariante fino alla lunghezza massima.

- Le coppie (a[i], b[i]) sono definite solo fino alla lunghezza minima dei due array a[] e b[]
- Oltrepassata la lunghezza minima, solo uno dei due array è definito.

```
retType fnR(const size_t aLen, const int a[],
            const size_t bLen, const int b[], const size_t i)
{
        if (i >= aLen && i >= bLen)
                return val_base; // caso base
        else {
                if (i >= bLen)            // a[] disponibile, b[] terminato
                        return E(a[i], fnR(aLen, a, bLen, b, i+1));
                else if (i >= aLen)       // b[] disponibile, a[] terminato
                        return E(b[i], fnR(aLen, a, bLen, b, i+1));
                else                      // coppia a[] e b[] disponibile
                        return E(a[i], b[i], fnR(aLen, a, bLen, b, i+1));
    }
}
retType fn(const size_t aLen, const int a[],
           const size_t bLen, const int b[]) { // involucro
        return fnR(aLen, a, bLen, b, 0)
}
```

Esercizi di ricorsione con coppie

Aprire il file **coppieR.c** fornito nel codice iniziale, leggere il contenuto ed infine implementare le funzioni ricorsive dichiarate, seguendo la specifica.

Outline

- Ripasso sui pattern di ricorsione
- ricorsioni.c
- Ricorsione su coppie di array
- coppieR.c
- Lab10-Es1 Ricerca binaria
- Lab10-Es2 Somma di due
- sqrtR.c
- Ricorsione su matrici
- matriciR.c
- Ricorsione non lineare
- fibonacci_vis.c
- permutazioni.c

Ricerca binaria

Sulla pagina Moodle trovate un esercizio con nome

Lab10-Es1 Ricerca binaria

Completate il programma.

NOTA: l'input (l'array **a**[]) non viene visualizzato perché può essere molto grande…

Somma di due

Sulla pagina Moodle trovate un esercizio con nome

Lab10-Es2 Somma di due

Completate il programma.

Esempio. Dato l'array ordinato:

```
a[] = {4, 8, 15, 16, 23, 42}
```

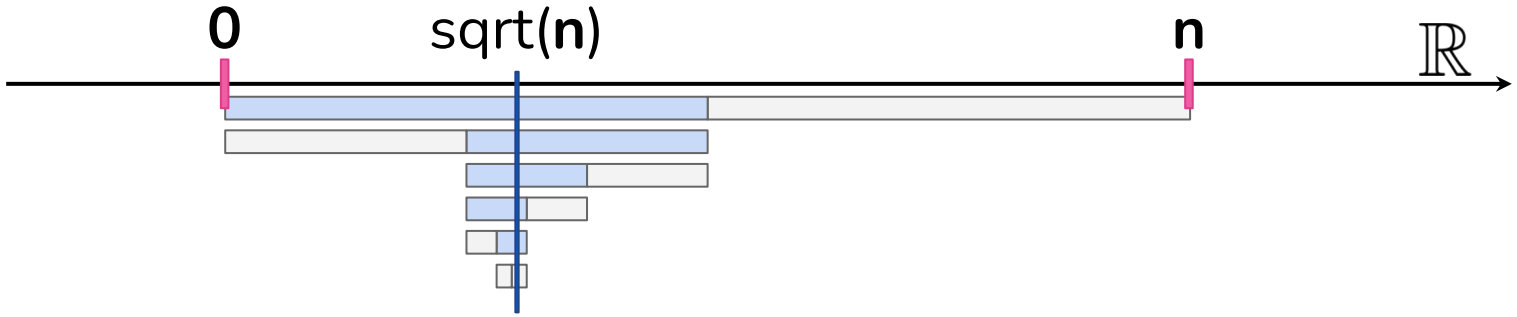
- esiste una coppia di elementi che somma a 31 = 8+23
  (trova_coppia ritorna true, assegnando *pIndex1=1 e *pIndex2=4)
- non esiste una coppia di elementi che somma a 40
  (trova_coppia ritorna false)

Ricorsione aritmetica dicotomica: approssimazione della radice quadrata

Si consideri un numero reale non negativo **n**, ed un intervallo semiaperto **[left, right)** con valore iniziale sufficientemente ampio.

Sia **mid** il punto medio dell'intervallo. Trattiamo **mid** come una approssimazione di sqrt(**n**). Si consideri la seguente strategia:

- se $\text{mid}^2 > \mathbf{n}$ allora sqrt(**n**) ∈ **[left, mid)**
- se $\text{mid}^2 \leq \mathbf{n}$ allora sqrt(**n**) ∈ **[mid, right)**



Procedere per dimezzamenti successivi dell'intervallo, finchè la dimensione non è minore di un valore **ε** piccolo. Usare quindi il valore **mid** finale come approssimazione di sqrt(**n**).

Implementare la ricerca della radice di n per dimezzamenti dicotomici partendo dal file **sqrtR.c** fornito nel codice iniziale.

Outline

- Ripasso sui pattern di ricorsione
- ricorsioni.c
- Ricorsione su coppie di array
- coppieR.c
- Lab10-Es1 Ricerca binaria
- Lab10-Es2 Somma di due
- sqrtR.c
- Ricorsione su matrici
- matriciR.c
- Ricorsione non lineare
- fibonacci_vis.c
- permutazioni.c

Ricorsione su matrici

Esercizi di ricorsione su matrici

Una matrice é un array bidimensionale.

Molti problemi su matrici si possono agevolmente risolvere in forma ricorsiva seguendo questo approccio:

- una **prima** funzione ricorsiva scorre tutte le righe *i*;
- una **seconda** funzione ricorsiva viene chiamata per ciascuna riga *i*, e ne scorre tutti gli elementi.

Aprire il file **matriciR.c** fornito nel codice iniziale, leggere il contenuto ed infine implementare le funzioni ricorsive dichiarate, seguendo la specifica.

Outline

- Ripasso sui pattern di ricorsione
- ricorsioni.c
- Ricorsione su coppie di array
- coppieR.c
- Lab10-Es1 Ricerca binaria
- Lab10-Es2 Somma di due
- sqrtR.c
- Ricorsione su matrici
- matriciR.c
- Ricorsione non lineare
- fibonacci_vis.c
- permutazioni.c

Ricorsione non lineare

Ricorsione non lineare

La maggior parte dei casi di ricorsione che abbiamo visto formano catene lineari, ma ci sono casi interessanti che non sono lineari.

- Aprire il file **fibonacci_vis.c** fornito nel codice iniziale, leggere il contenuto e provarlo (il programma è già completo).
- Aprire il file **permutazioni.c** fornito nel codice iniziale, leggere il contenuto e provarlo (il programma è già completo).
  DOMANDE:
  - perché funziona? → capire il principio.
  - come si può riscrivere in forma non ricorsiva?
    (nota: è molto difficile)
  - data una stringa di lunghezza L, quante chiamate ricorsive vengono effettuate?