

Ricorsione

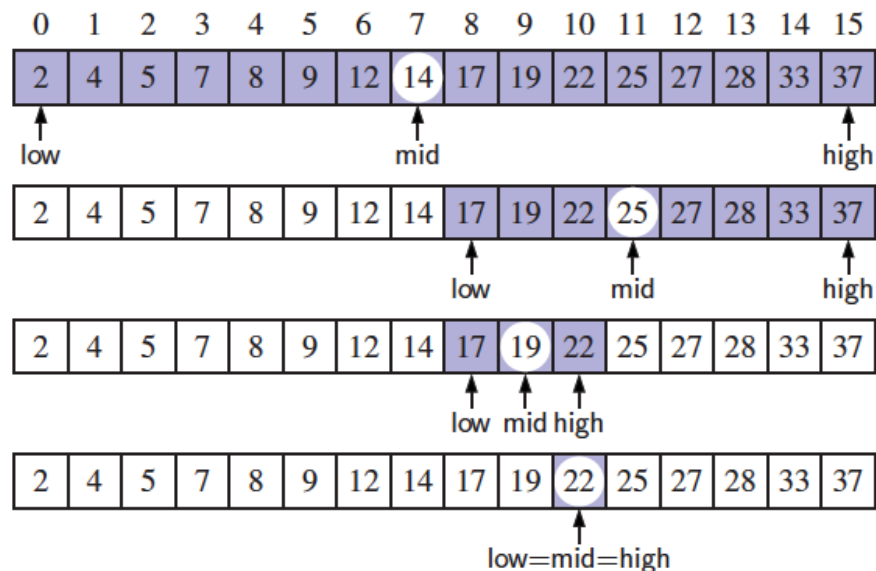
Luca Becchetti

Presentazione tratta dalle slide che accompagnano il testo Data Structures and Algorithms in Java, 6th edition, by M. T. Goodrich, R. Tamassia, and M. H. Goldwasser, Wiley, 2014



Ricerca binaria

- Se $\text{target} = \text{data}[\text{mid}] \rightarrow$ trovato
- Se $\text{target} < \text{data}[\text{mid}] \rightarrow$ cerchiamo in metà inferiore
- Se $\text{target} > \text{data}[\text{mid}] \rightarrow$ cerchiamo in metà superiore



Ricerca binaria

```
1  /**
2   * Returns true if the target value is found in the indicated portion of the data array.
3   * This search only considers the array portion from data[low] to data[high] inclusive.
4   */
5  public static boolean binarySearch(int[ ] data, int target, int low, int high) {
6      if (low > high)
7          return false;                                // interval empty; no match
8      else {
9          int mid = (low + high) / 2;
10         if (target == data[mid])
11             return true;                                // found a match
12         else if (target < data[mid])
13             return binarySearch(data, target, low, mid - 1); // recur left of the middle
14         else
15             return binarySearch(data, target, mid + 1, high); // recur right of the middle
16     }
17 }
```



Analisi

- **Dopo ogni tentativo fallito la dimensione della porzione di array da esaminare è pari a uno dei seguenti valori:**

$$(mid - 1) - low + 1 = \left\lfloor \frac{low + high}{2} \right\rfloor - low \leq \frac{high - low + 1}{2}$$

$$high - (mid + 1) + 1 = high - \left\lfloor \frac{low + high}{2} \right\rfloor \leq \frac{high - low + 1}{2}.$$

- **Si dimezza spazio di ricerca $\rightarrow O(\log n)$ chiamate ricorsive**



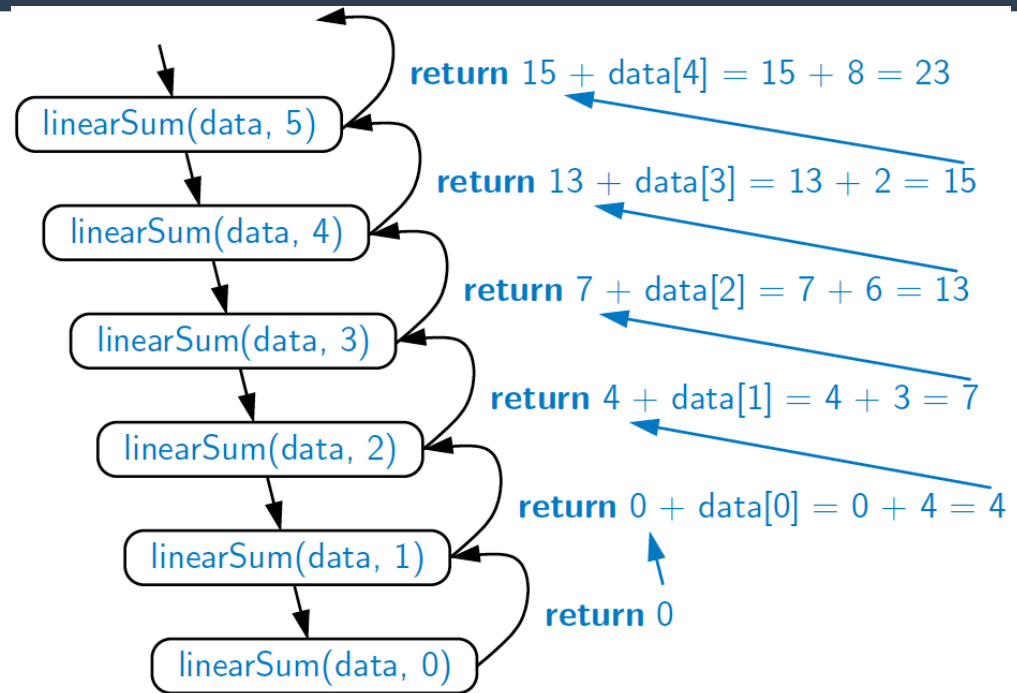
Ricorsione lineare

- **Si ha quando il corpo del metodo contiene al massimo un'invocazione ricorsiva**
- **Il metodo `binarySearch` implementa una ricorsione lineare**
- **Scrivere un algoritmo/metodo che ricorsivamente calcoli la somma degli elementi di un array**



Somma degli elementi in un array

```
linearSum(A, n)
if n = 0 then
    return 0
else
    return
linearSum(A, n - 1) + A[n - 1]
```



```
34  /** Returns the sum of the first n integers of the given array. */
35  public static int linearSum(int[] data, int n) {
36      if (n == 0)
37          return 0;
38  else
39      return linearSum(data, n-1) + data[n-1];
40  }
```

Inversione di un array

Algorithm **reverseArray**(A, i, j):

Input: array A e indici non-negativi i e j

Output: The array contenente gli elementi di A compresi tra gli indici i e j, in ordine invertito

if $i < j$ then

 Swap A[i] and A[j]

 reverseArray(A, i + 1, j - 1)

return



Inversione di un array (cont.)

- **Definizione dei metodi che faciliti la ricorsione**
- **Per esempio, definiamo il metodo `reverseArray(A, i, j)` e non `reverseArray(A)`**

```
1  /** Reverses the contents of subarray data[low] through data[high] inclusive. */
2  public static void reverseArray(int[ ] data, int low, int high) {
3      if (low < high) {                                // if at least two elements in subarray
4          int temp = data[low];                        // swap data[low] and data[high]
5          data[low] = data[high];
6          data[high] = temp;
7          reverseArray(data, low + 1, high - 1);      // recur on the rest
8      }
9  }
```



Attenzione ai costi ...

- **Calcolo della funzione $p(x,n)=x^n$**

$$p(x,n) = \begin{cases} 1 & \text{if } n=0 \\ x \times p(x,n-1) & \text{else} \end{cases}$$

- **Scrivere algoritmo corrispondente**
 - Qual è il costo computazionale?
 - Il costo è polinomiale?
 - Possiamo fare meglio?



Una definizione alternativa

$$p(x, n) = \begin{cases} 1, & n = 0 \\ x \cdot p(x, (n-1)/2)^2, & n > 0 \text{ dispari} \\ p(x, n/2)^2, & n > 0 \text{ pari} \end{cases}$$

- **Esempio**

$$2^4 = 2^{(4 \div 2)^2} = (2^{4 \div 2})^2 = (2^2)^2 = 4^2 = 16$$

$$2^5 = 2^{1+(4 \div 2)^2} = 2(2^{4 \div 2})^2 = 2(2^2)^2 = 2(4^2) = 32$$

$$2^6 = 2^{(6 \div 2)^2} = (2^{6 \div 2})^2 = (2^3)^2 = 8^2 = 64$$

$$2^7 = 2^{1+(6 \div 2)^2} = 2(2^{6 \div 2})^2 = 2(2^3)^2 = 2(8^2) = 128$$



Metodo alternativo: algoritmo

Algorithm **Power**(x, n):

Input: A number x and integer $n \geq 0$

Output: The value x^n

if $n = 0$ **then**

return 1

if n is odd **then**

$y = \text{Power}(x, (n - 1) / 2)$

return $x \cdot y \cdot y$

else

$y = \text{Power}(x, n / 2)$

return $y \cdot y$



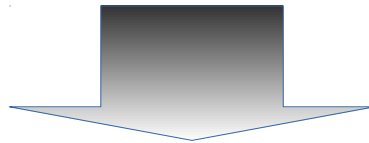
Complessità computazionale

- **Siamo in presenza di ricorsione lineare**
- **Ad ogni invocazione ricorsiva:**
 - $n \rightarrow (n - 1)/2$ oppure
 - $n \rightarrow n/2$
- **In ogni caso: secondo argomento dell'algoritmo dimezza (almeno)**
- **Conseguenza $\rightarrow O(\log n)$ invocazioni ricorsive**



Ricorsione in coda

```
1  /** Reverses the contents of subarray data[low] through data[high] inclusive. */
2  public static void reverseArray(int[ ] data, int low, int high) {
3      if (low < high) {                                // if at least two elements in subarray
4          int temp = data[low];                        // swap data[low] and data[high]
5          data[low] = data[high];
6          data[high] = temp;
7          reverseArray(data, low + 1, high - 1);      // recur on the rest
8      }
9  }
```



Algorithm *IterativeReverseArray*(A, i, j):

```
    while i < j do
        Swap A[i ] and A[ j ]
        i = i + 1
        j = j - 1
    return
```

Ricorsione in coda → facile conversione in algoritmo iterativo



Ricorsione doppia (ed errori comuni)

- **Numeri di Fibonacci**

- $F_0 = 0$
- $F_1 = 1$
- $F_i = F_{i-1} + F_{i-2}, i > 1$

```
/** Returns the nth Fibonacci number (inefficiently). */  
public static long fibonacciBad(int n) {  
    if (n <= 1)  
        return n;  
    else  
        return fibonacciBad(n-2) + fibonacciBad(n-1);  
}
```

- **Complessità computazionale?**



Versione efficiente (ricorsione lineare)

```
/** Restituisce un array contenente la coppia F(n) e F(n-1). */  
public static long[] fibonacciGood(int n) {  
    if (n <= 1) {  
        long[] answer = {n, 0};  
        return answer;  
    } else {  
        long[] temp = fibonacciGood(n - 1); // restituisce {F(n-1), F(n-2)}  
        long[] answer = {temp[0] + temp[1], temp[0]}; // vogliamo {F(n), F(n-1)}  
        return answer;  
    }  
}
```

- **Costo $O(n)$**
- **Prova**
 - Ricorsione lineare
 - Argomento diminuisce di 1 a ogni invocazione



Versione iterativa (Python-like)

```
def f(n):  
    a = 0  
    b = 1  
    for i = 0 to n:  
        temp = a  
        a = b  
        b = b + temp  
    return a
```

