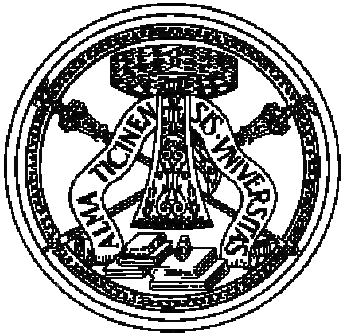
UNIVERSITÀ DEGLI STUDI DI PAVIA
FACOLTÀ DI INGEGNERIA



Algoritmi

Algoritmi classici

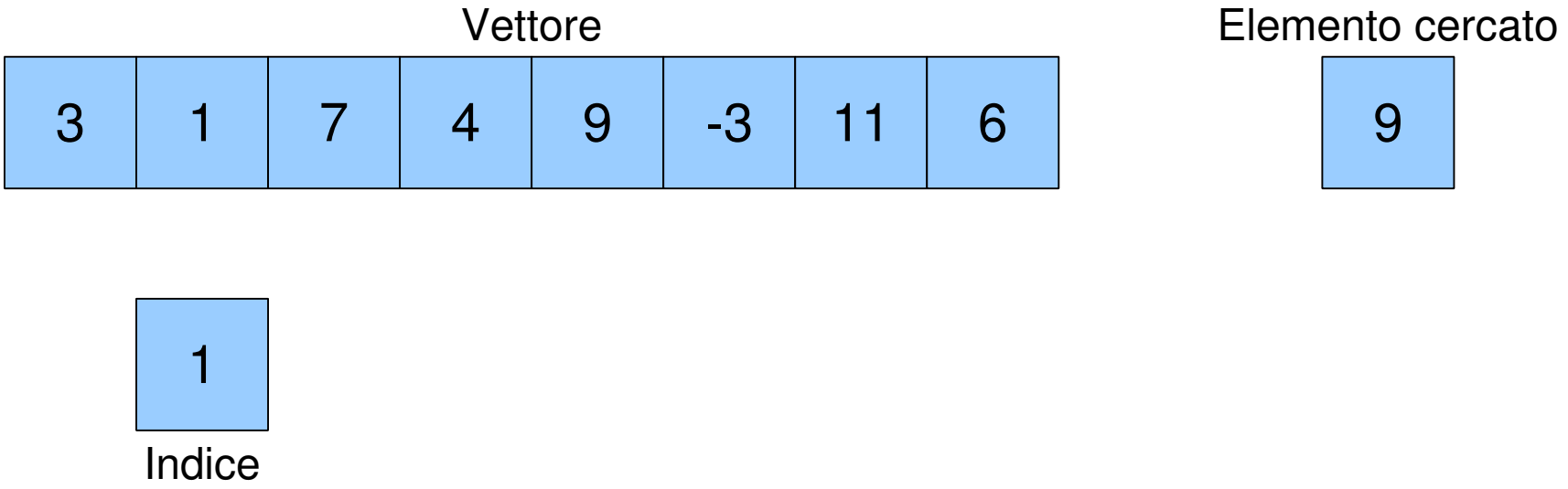
- Alcuni problemi si presentano con elevata frequenza e sono stati ampiamente studiati
  - Ricerca di un elemento in un vettore
  - Ricerca del minimo e del massimo
  - Ordinamento
- Gli algoritmi impiegabili in questi casi sono numerosi. I più noti, di seguito presentati, vengono spesso impiegati anche come termini di paragone per valutare le prestazioni di nuove soluzione proposte

Algoritmi di ricerca

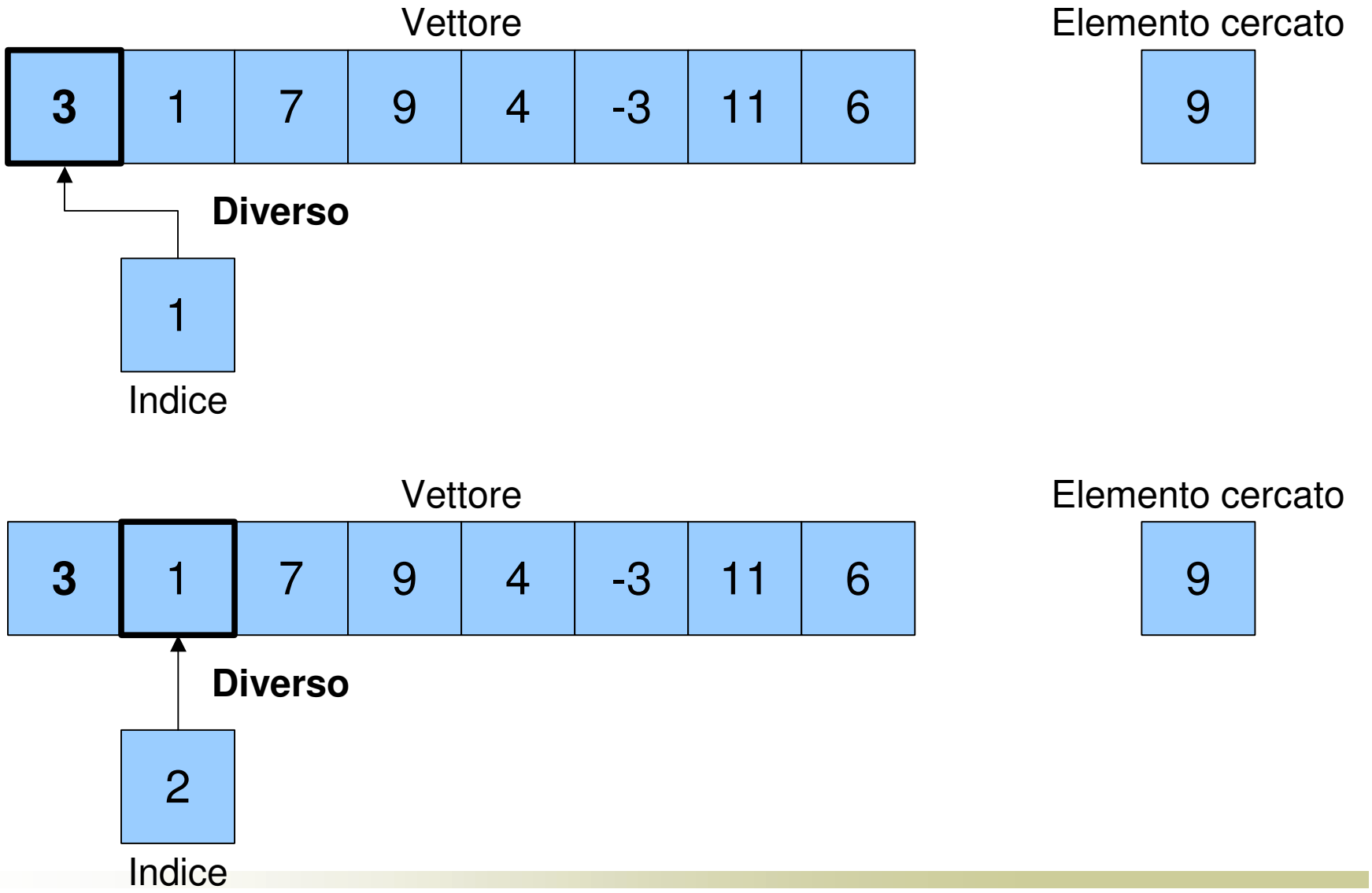
- Il problema della ricerca di un elemento in un vettore si presenta frequentemente:
  - Occorre verificare se l'elemento appartiene al vettore
  - Ad un elemento (o alla sua posizione) sono associate informazioni supplementari
- Esistono due algoritmi "standard" per la risoluzione di questo problema

Ricerca sequenziale (1)

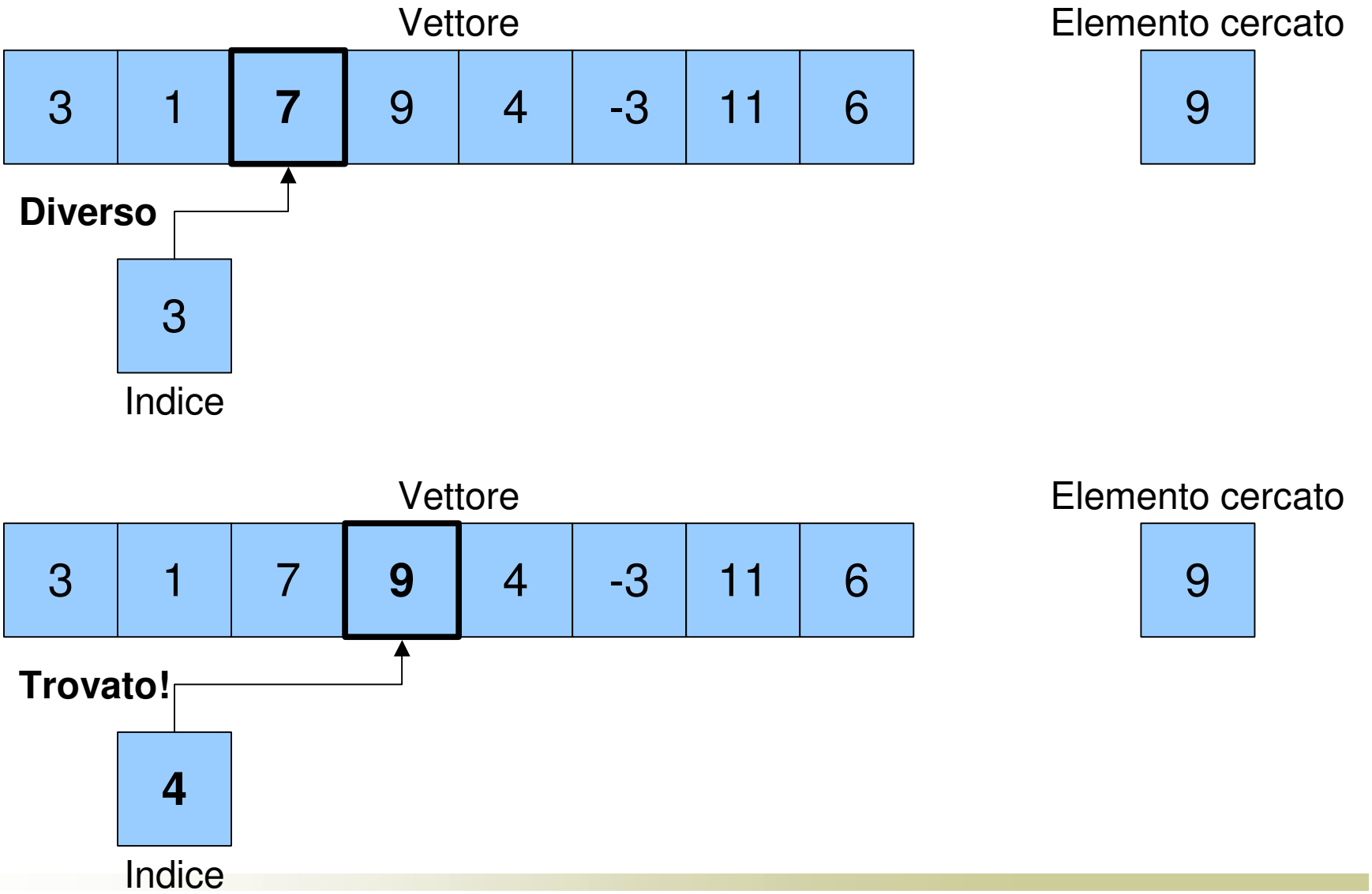
- L'idea di fondo è semplice: l'intero vettore viene scorso alla ricerca dell'elemento che interessa



Ricerca sequenziale (2)



Ricerca sequenziale (3)



Ricerca sequenziale (4)

- L'algoritmo di ricerca sequenziale funziona senza richiedere particolari ipotesi sull'ordinamento dei dati
- Mediamente occorre scandire metà vettore per trovare l'elemento cercato (se c'è); se non c'è, occorre scorrere tutto il vettore
- Diciamo che il tempo richiesto dall'algoritmo è proporzionale alla lunghezza N del vettore
  - Si dice che il tempo di esecuzione cresce linearmente al variare di N

Ricerca binaria (1)

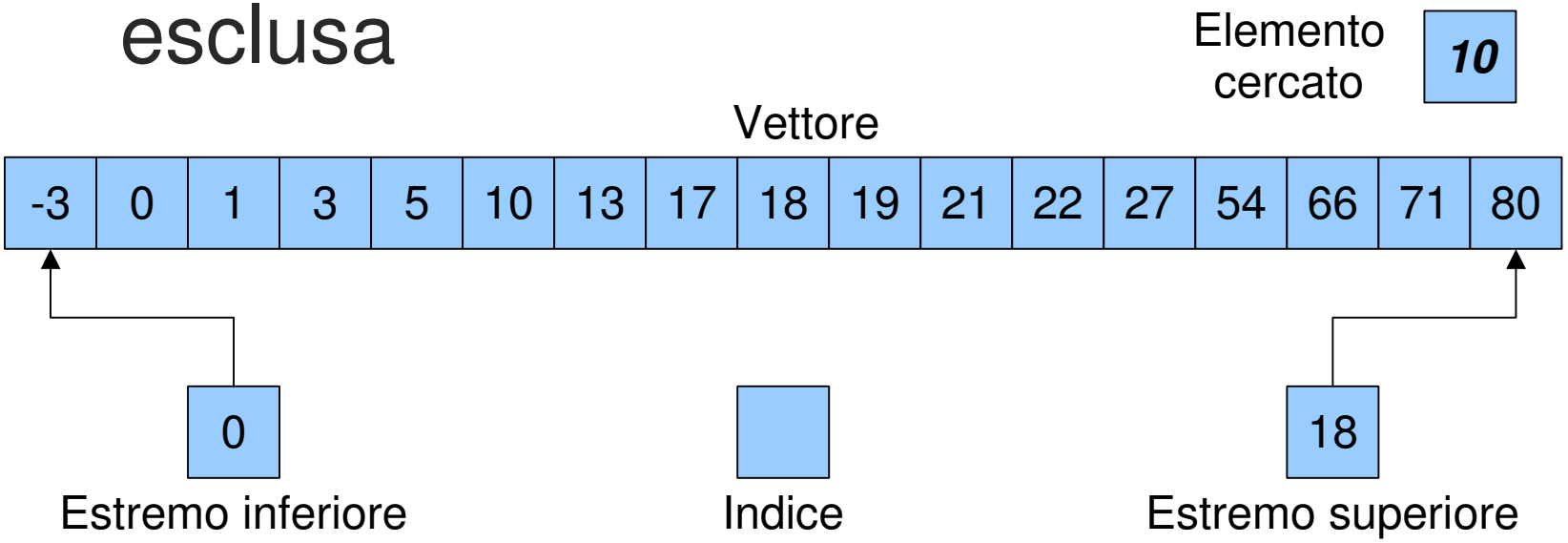
- **Se è soddisfatta l'ipotesi di ordinamento dei dati all'interno del vettore**, allora è possibile utilizzare approcci più efficienti rispetto alla ricerca sequenziale
- La ricerca binaria prevede l'osservazione dell'elemento al centro del vettore considerato.
  - Se è quello cercato termina, altrimenti scarta tutta una metà del vettore a seconda che l'elemento analizzato sia maggiore o minore di quello cercato
  - Il processo si ripete fino a trovare l'elemento cercato o a scartarli tutti

Ricerca binaria (2)

- L'approccio è lo stesso adottato per cercare una parola nel dizionario o un nome nella rubrica telefonica
  - Es. cerco il numero di Rossi Mario
    - Apro circa a metà, sulla lettera N
    - Vado avanti di diverse pagine
    - Arrivo alla lettera S
    - Torno indietro a Ra…
    - Vado avanti di poco, arrivando a Rov..
    - Ci sono quasi, torno indietro di una pagina
    - etc…

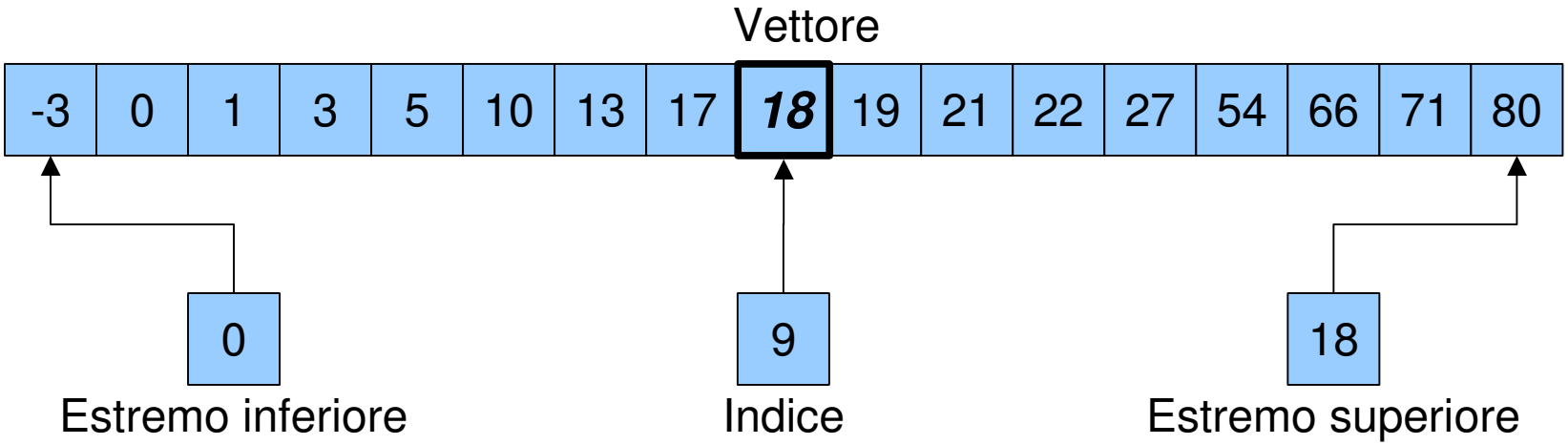
Ricerca binaria (3)

- Rispetto all'algoritmo sequenziale, la ricerca binaria usa due ulteriori indici per individuare gli estremi della porzione del vettore non ancora esclusa



Ricerca binaria (4)

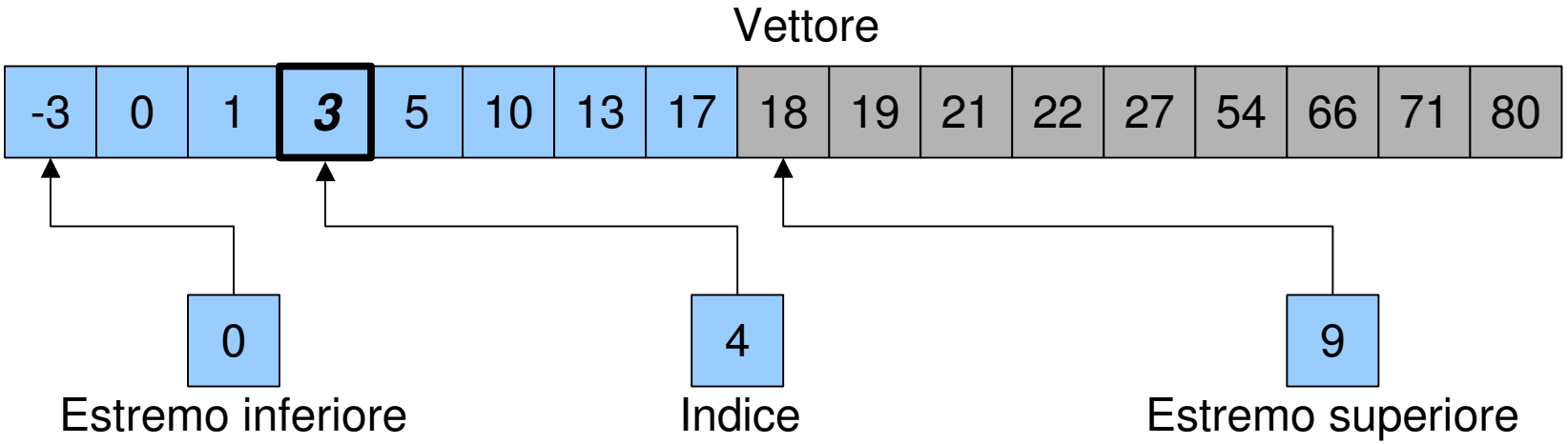
- L'indice iniziale è sempre a metà tra gli indici degli estremi



- L'elemento cercato (10) è minore di 18, per cui si esclude la seconda metà del vettore

Ricerca binaria (5)

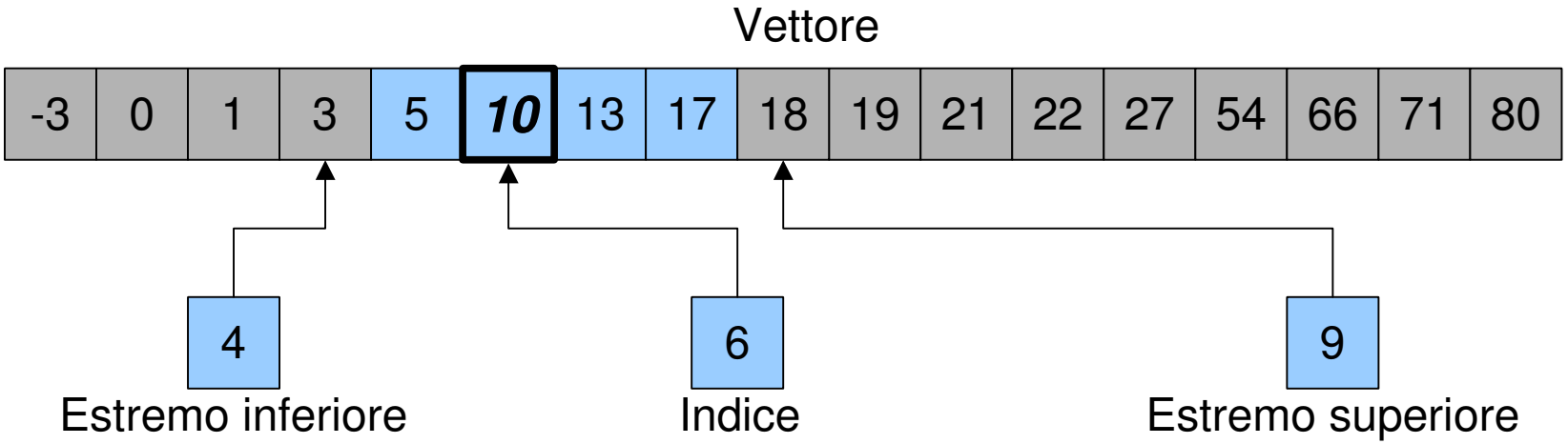
- La metà tra 0 e 9 è 4,5: poiché gli indici devono essere interi, si sceglie tra 4 e 5



- Questa volta l'elemento è precedente quello cercato, per cui si prosegue nella ricerca escludendo la metà di sinistra

Ricerca binaria (6)

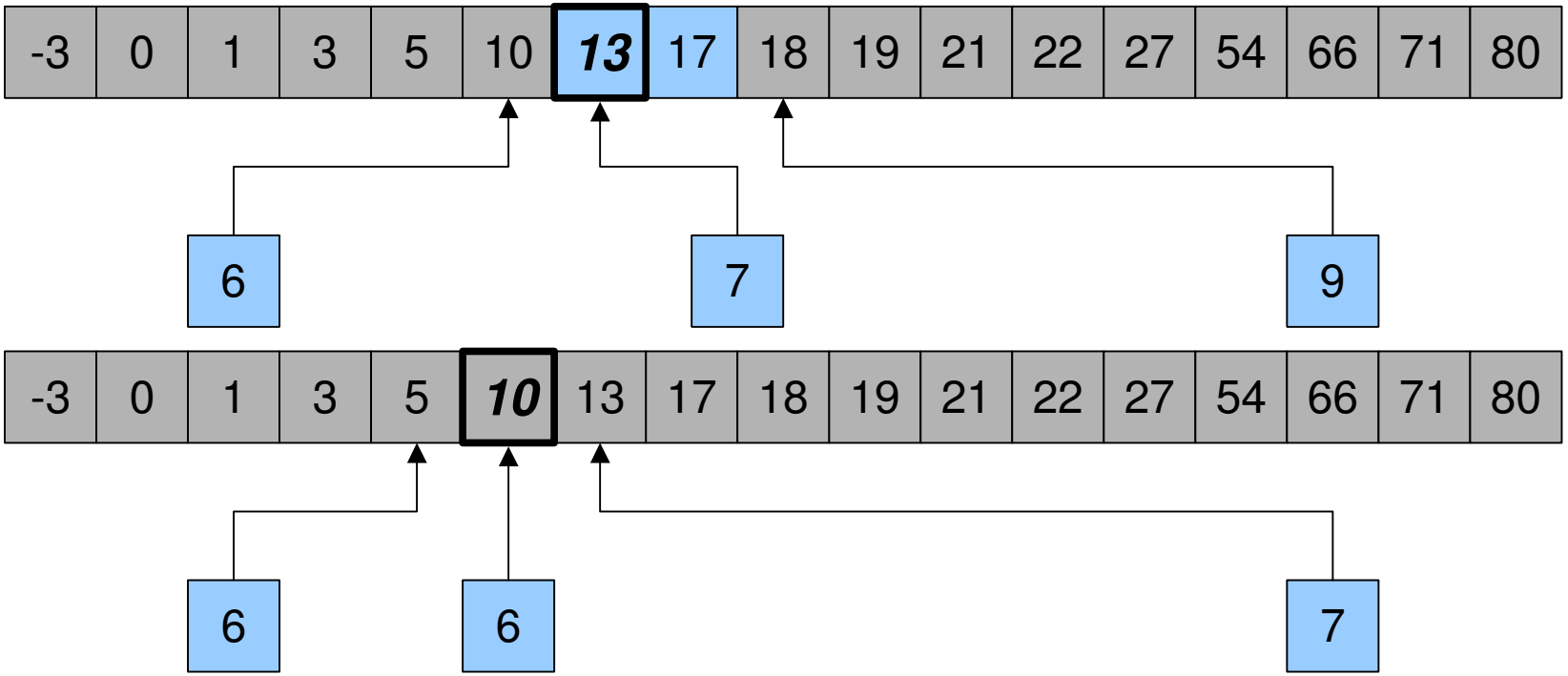
- Come nel caso precedente, si tronca il valore dell'indice all'intero inferiore



- L'elemento cercato, presente nell'algoritmo viene quindi trovato.
  - La risposta fornita dall'algoritmo è **6**, cioè la posizione corrispondente al valore cercato

Ricerca binaria (7)

- Se avessimo cercato 12, i passi fino a qui sarebbero stati gli stessi. Poi:



- In questo caso, l'algoritmo terminerebbe senza successo

Ricerca binaria (8)

- L'algoritmo di ricerca binaria dimezza la dimensione dello *spazio di ricerca* ad ogni passo
  - Il tempo necessario all'esecuzione dell'algoritmo è dunque proporzionale al logaritmo di N
- Visto che *log*N cresce più lentamente di N, la ricerca binaria è più efficiente di quella sequenziale (ma richiede l'ipotesi supplementare di ordinamento dei dati)
- Nel caso pessimo l'algoritmo termina quando la dimensione dello spazio di ricerca diventa 1

Ricerca del minimo (1)

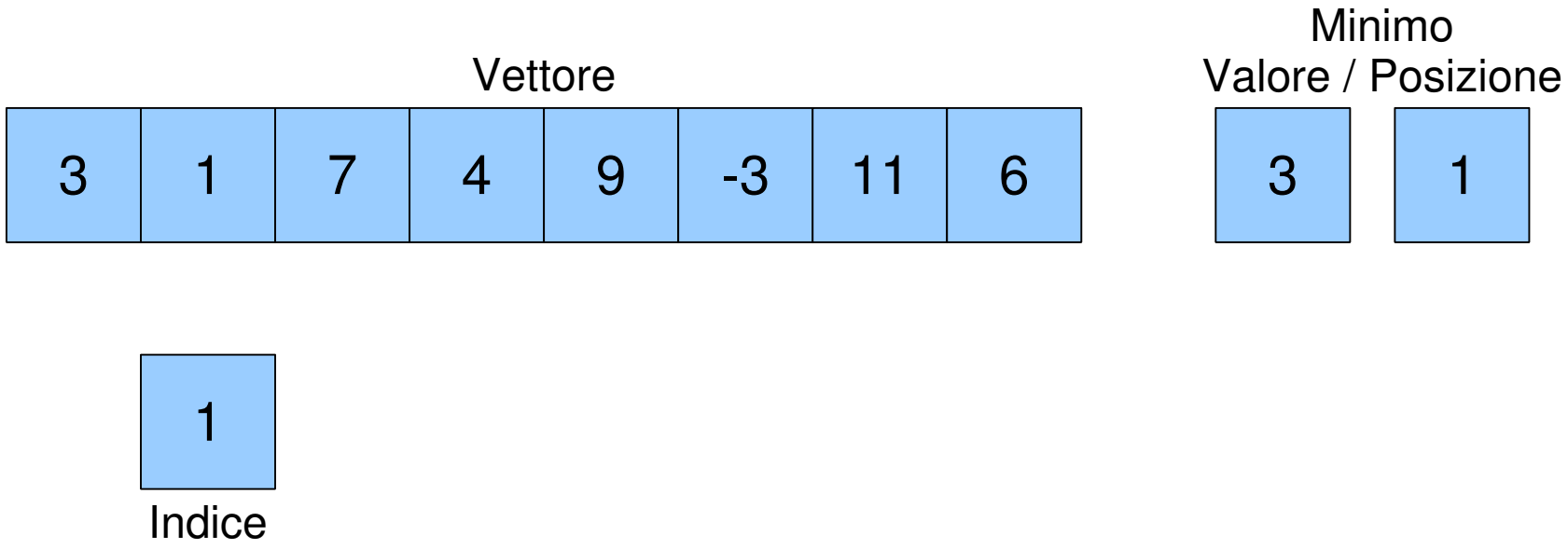
- Un secondo problema riguarda la ricerca del valore minimo (o massimo) all'interno di un vettore non ordinato
  - Naturalmente il problema è banale se invece il vettore è ordinato!
- Dato che i problemi di ricerca del minimo e di ricerca del massimo sono del tutto analoghi, di seguito si farà riferimento esclusivamente alla ricerca del minimo

Ricerca del minimo (2)

- Per risolvere il problema vengono utilizzate due variabili di supporto, contenenti:
  - il valore minimo trovato sinora
  - la posizione (indice) di tale valore
- L'algoritmo scorre l'intero vettore e confronta ciascun elemento col minimo contenuto nella variabile di supporto
  - Se l'elemento nel vettore è inferiore a quello nella variabile di appoggio, allora sostituisce la variabile di supporto con l'elemento considerato

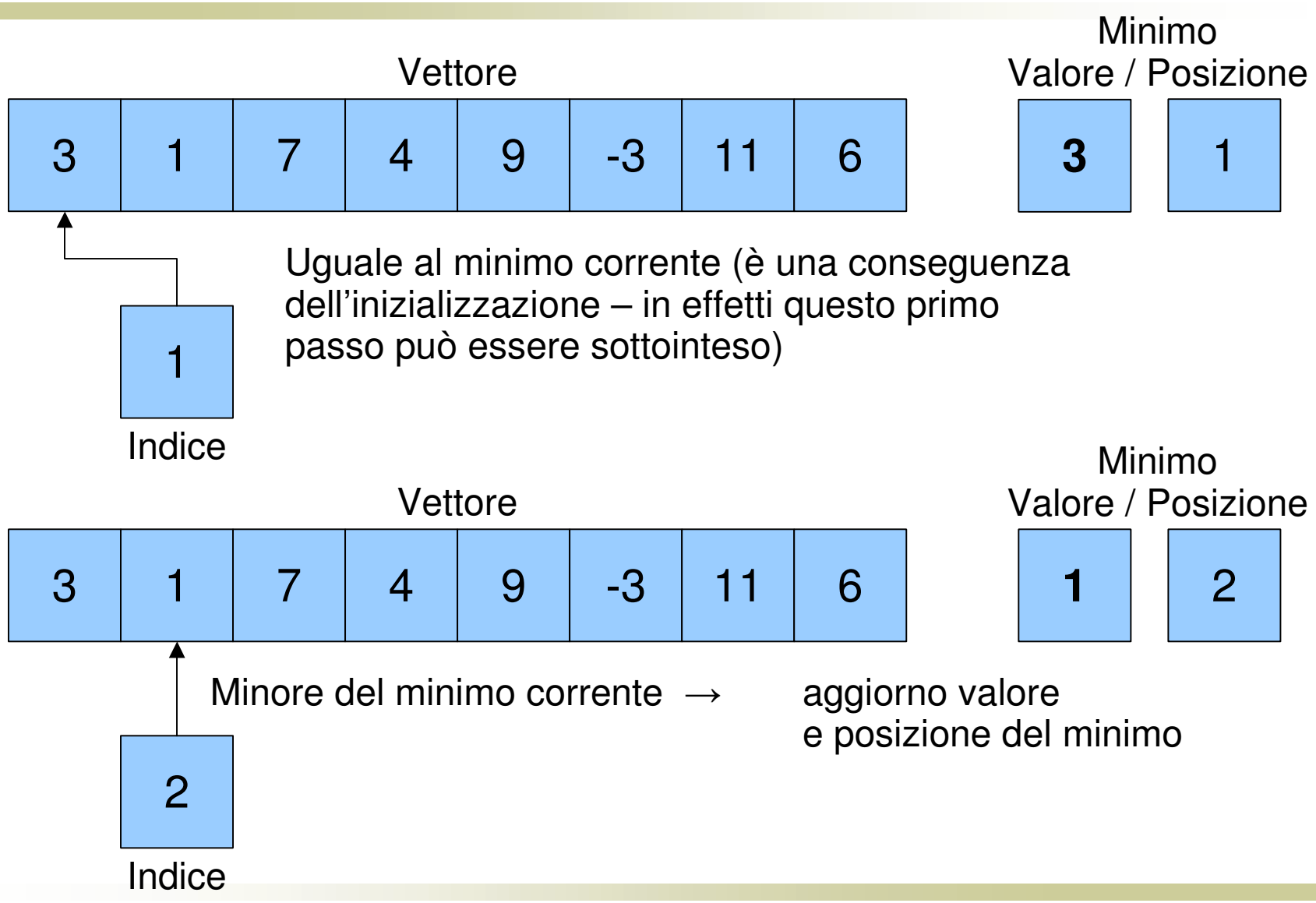
Ricerca del minimo (3)

- Si inizializzano le variabili di supporto con la posizione e il valore del primo elemento

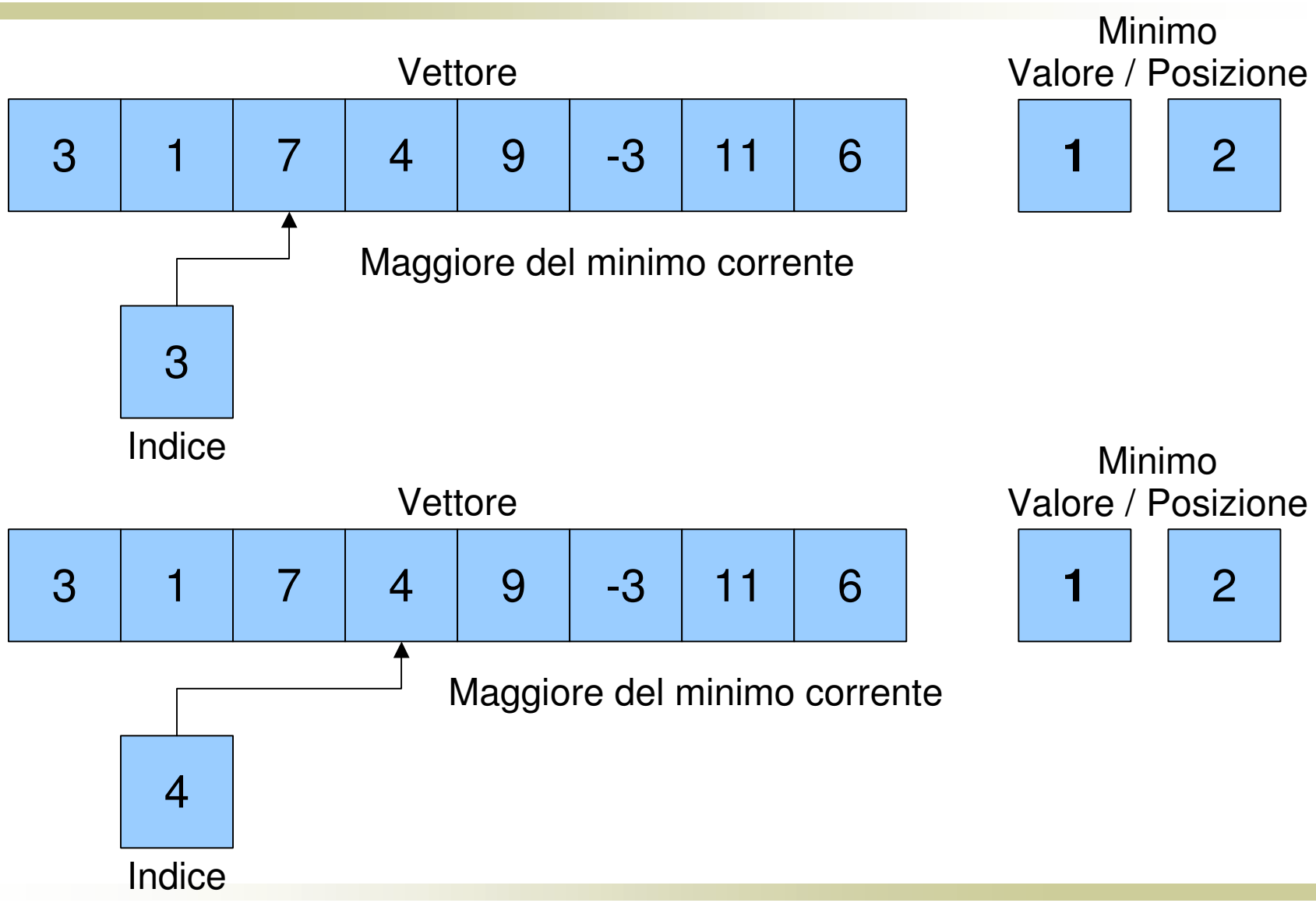


- In questo modo il minimo temporaneo è non inferiore al minimo del vettore

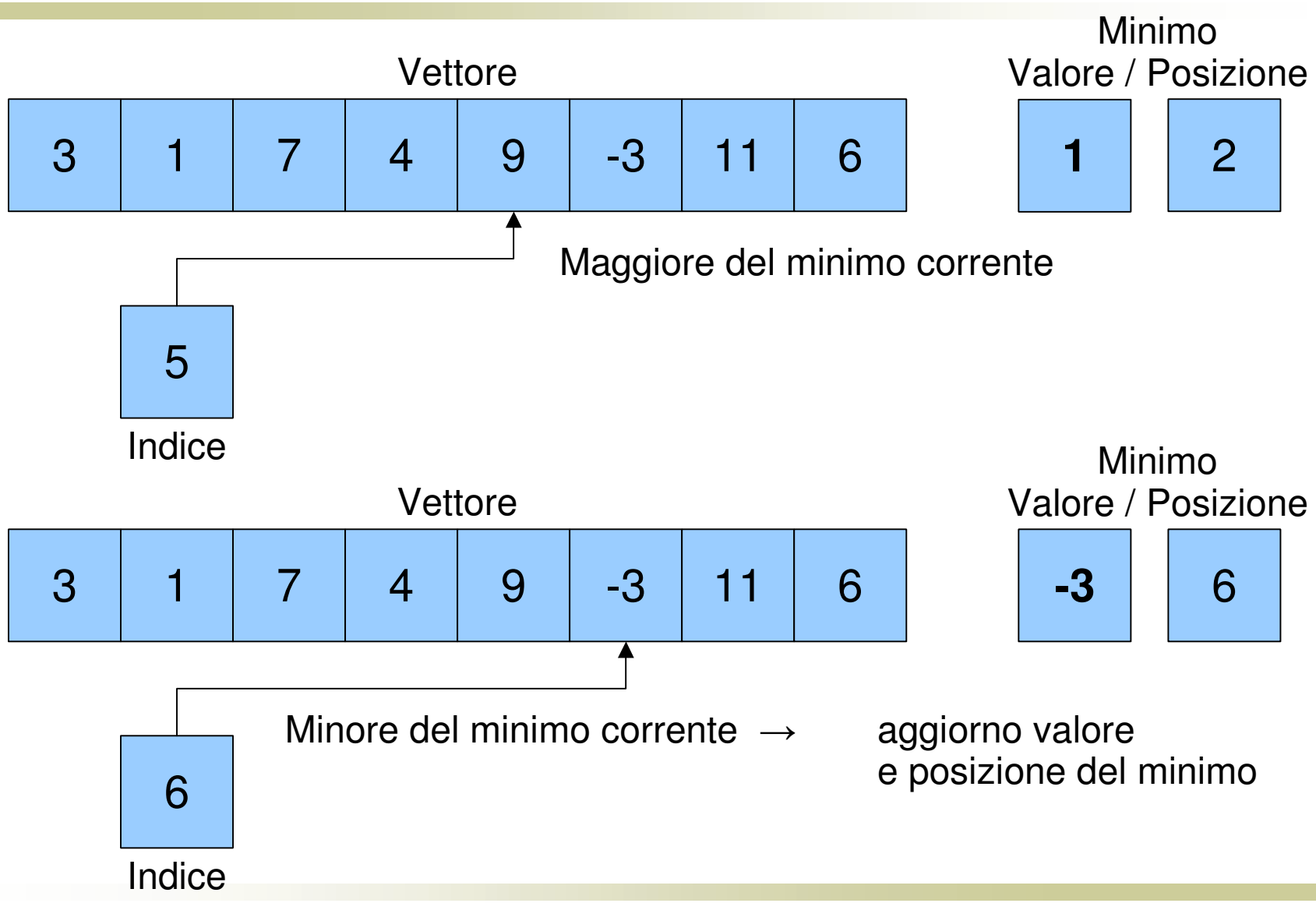
Ricerca del minimo (4)



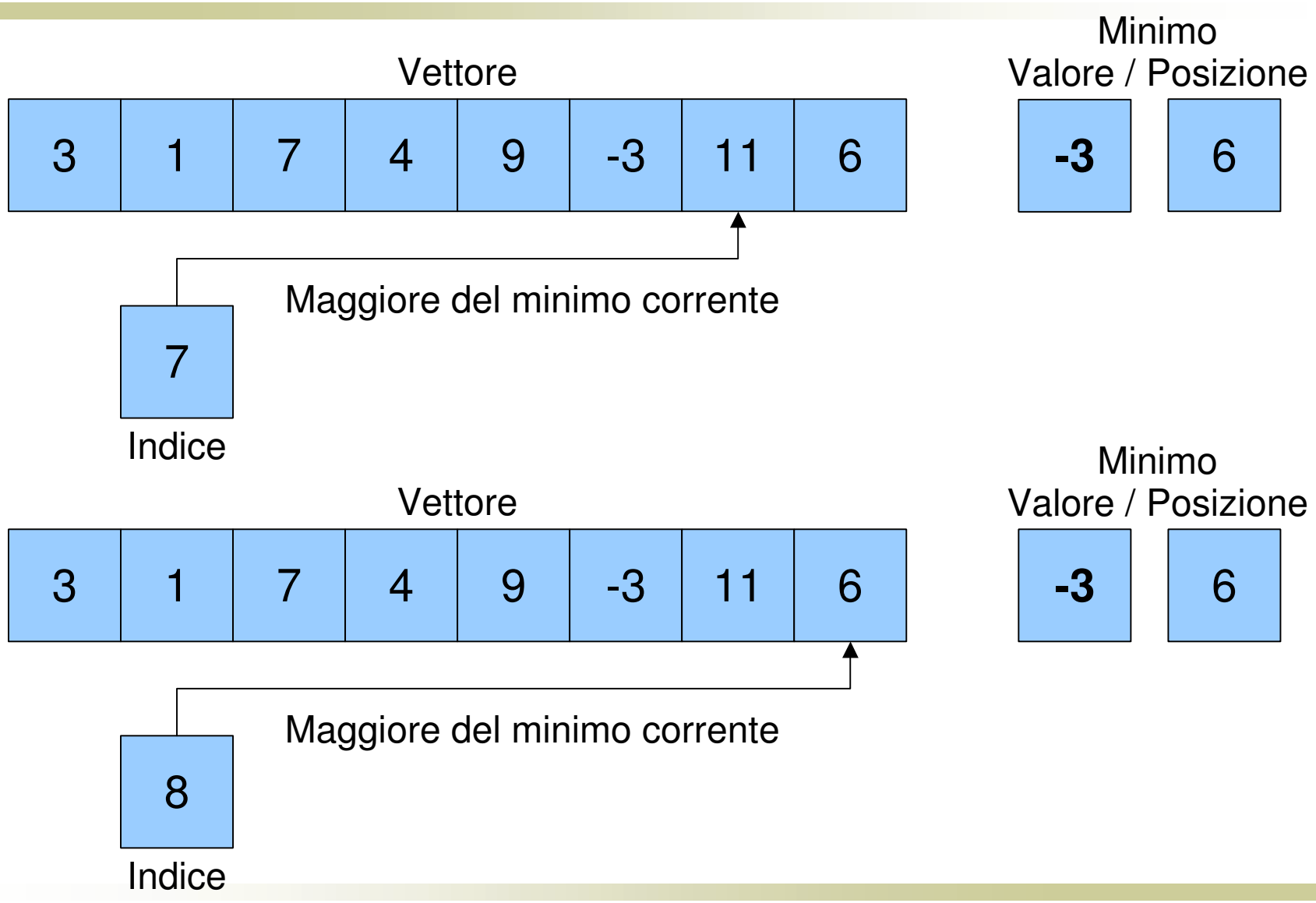
Ricerca del minimo (5)



Ricerca del minimo (6)



Ricerca del minimo (7)



Ricerca del minimo (8)

- Al termine dell'algoritmo le variabili di supporto contengono il valore e la posizione del minimo
  - Se fossero presenti più minimi uguali sarebbe possibile decidere quale tenere in considerazione
- Data la lunghezza N del vettore, è necessario effettuare N confronti
  - Il tempo necessario al completamento dell'esecuzione è proporzionale alla dimensione del vettore

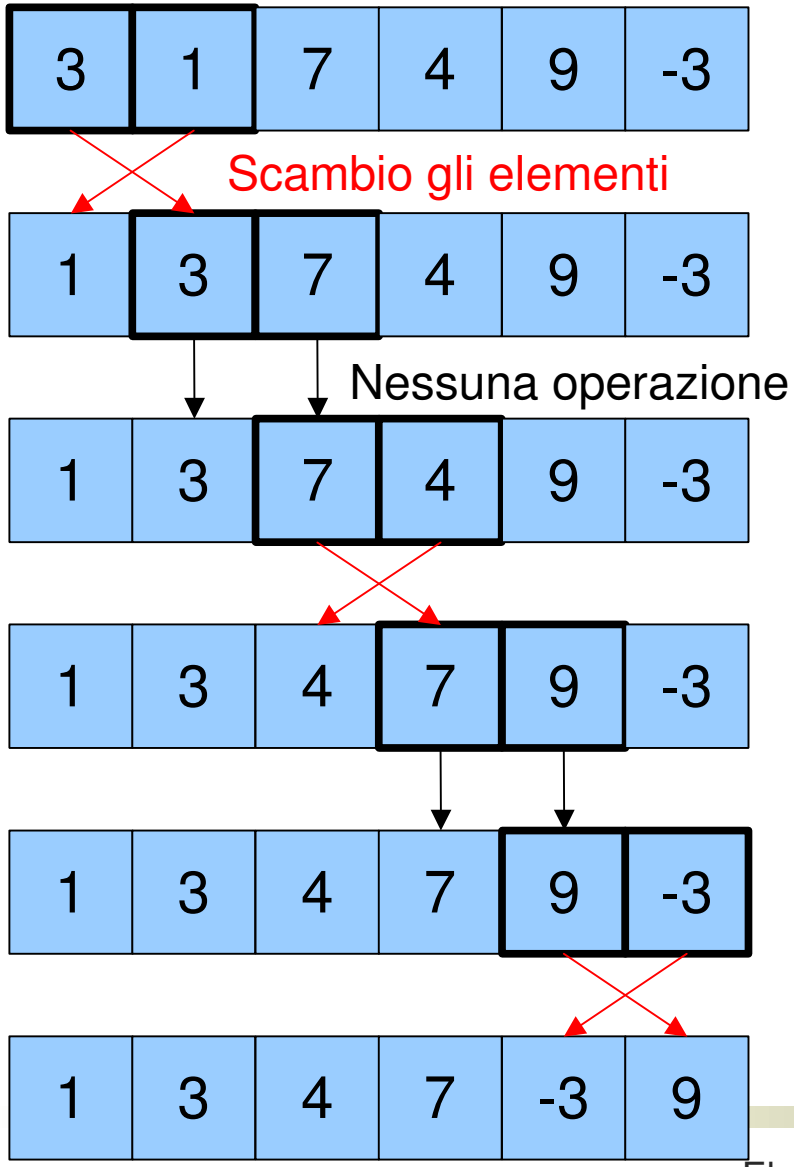
Ordinamento

- Un ultimo problema consiste nell'ordinare gli elementi di un vettore
- Esiste una grande varietà di algoritmi di ordinamento, con caratteristiche anche molto diverse in termini di velocità e memoria utilizzata

Ordinamento: Bubblesort (1)

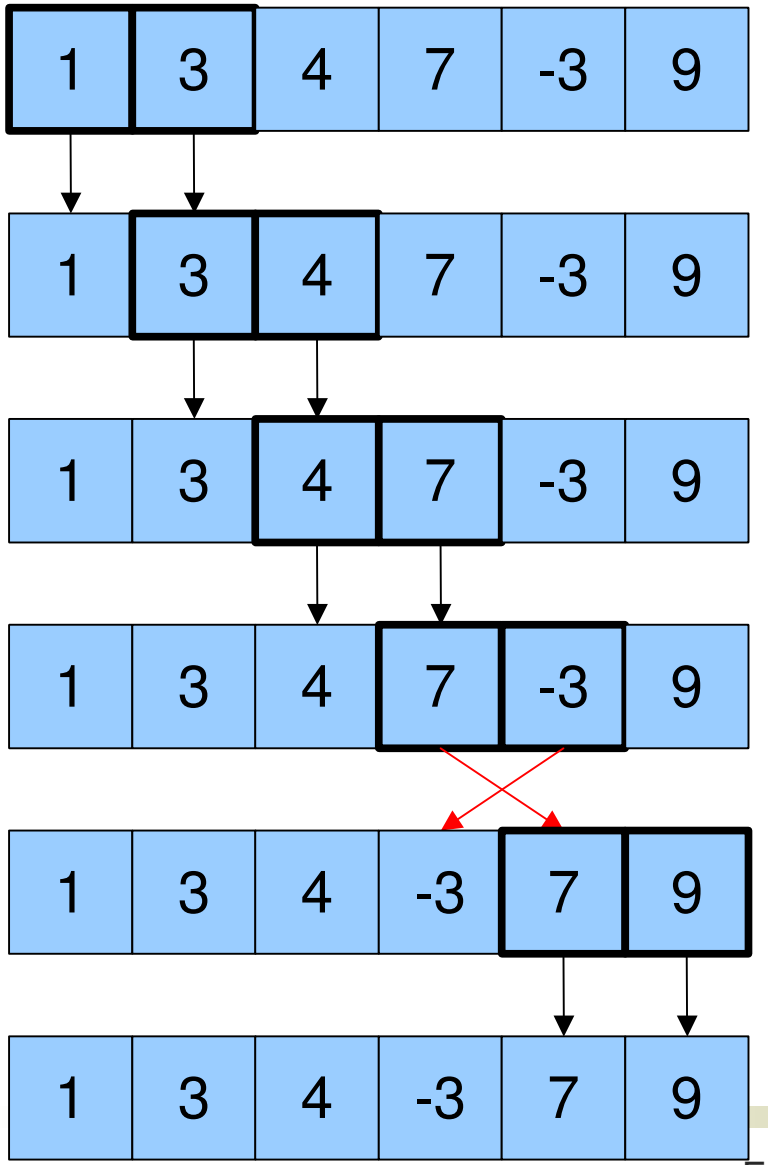
- Bubblesort confronta coppie di elementi adiacenti (primo-secondo, secondo-terzo, ...) e ne scambia il contenuto se gli elementi non sono nell'ordine corretto
- Dopo aver effettuato una prima iterazione dell'intero vettore, l'algoritmo inizia nuovamente dal primo elemento
- Il procedimento si ripete finché la lista di elementi non è ordinata, ovvero:
  - Dopo N passate
  - Oppure quando, durante un'intera passata, l'algoritmo non effettua scambi (il vettore è cioè già ordinato ad un passo intermedio)

Ordinamento: Bubblesort (2)



Al termine della prima iterazione, l'ultimo elemento del vettore è nella posizione corretta (il massimo è finito in ultima posizione)

Ordinamento: Bubblesort (3)



Dopo la seconda iterazione, anche il penultimo elemento del vettore è a posto.

Dopo altre 3 iterazioni il vettore risulterà ordinato

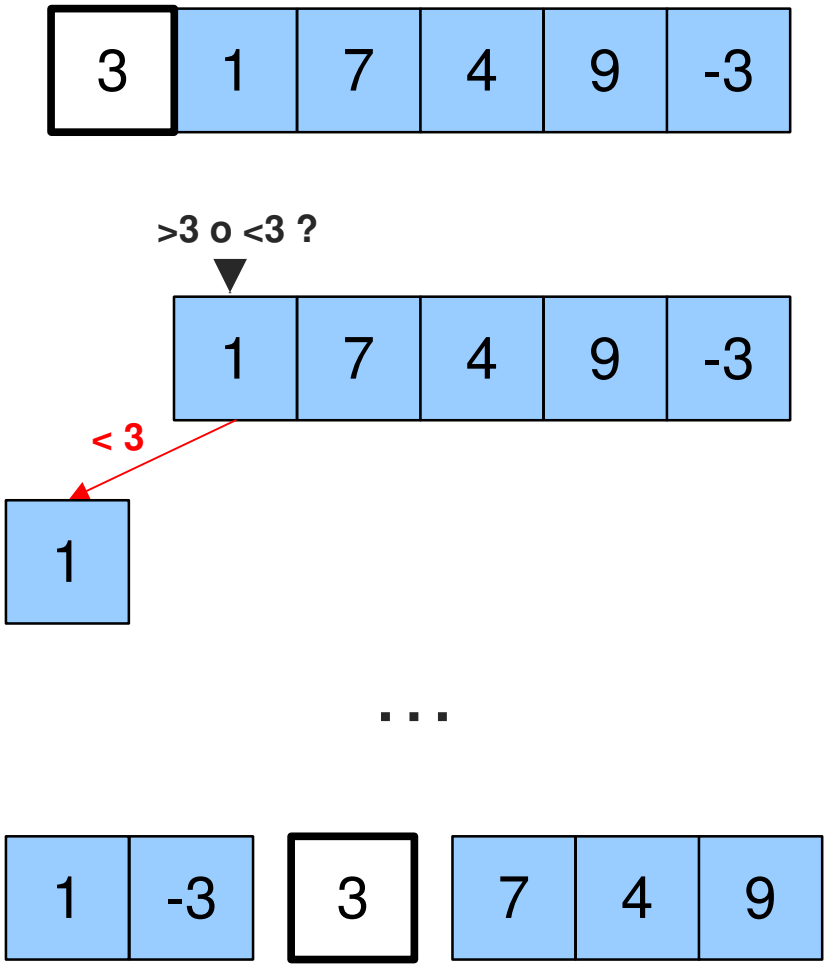
Ordinamento: Bubblesort (4)

- L'algoritmo Bubblesort è semplice da implementare ma è molto lento
  - Poiché ho N elementi, devo effettuare N-1 confronti ogni iterazione; inoltre l'algoritmo termina dopo N iterazioni. Il tempo di esecuzione dell'algoritmo è quindi proporzionale a $N*(N-1) \approx N^2$

Ordinamento: Quicksort (1)

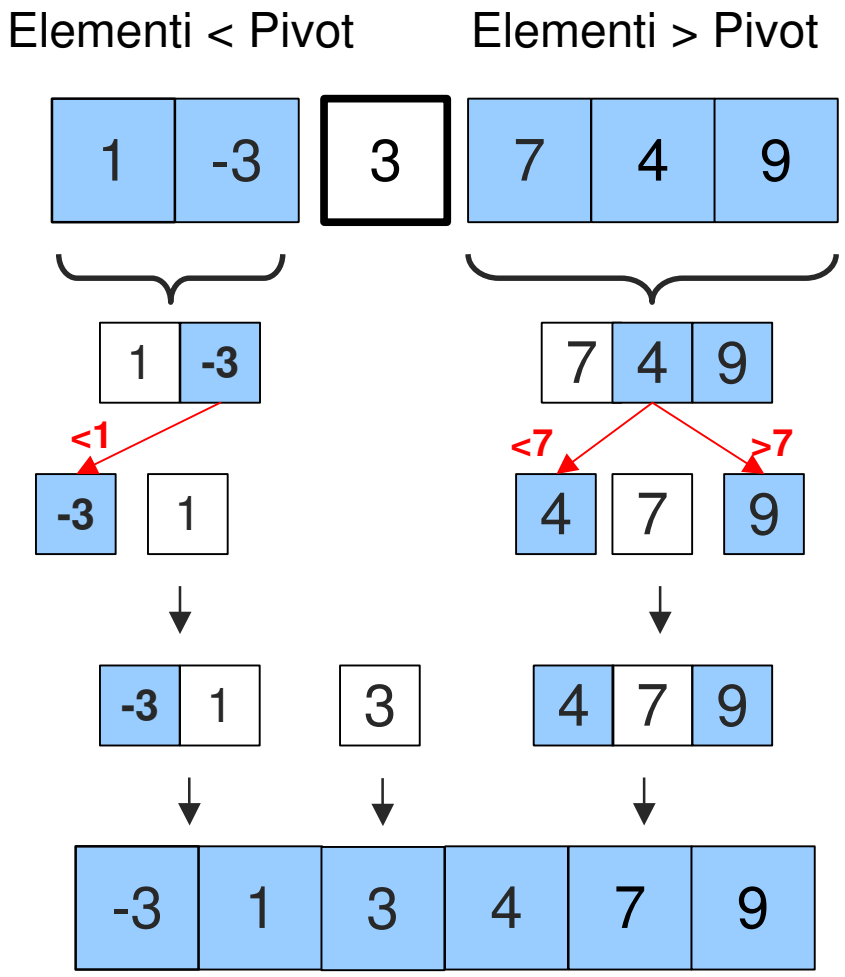
- Quicksort è un algoritmo ricorsivo
  - dato un problema lo scompone e risolve la versione semplificata utilizzando il medesimo algoritmo
- Se il vettore è di dimensione 1 allora è ordinato e termina, altrimenti:
  - sceglie un elemento del vettore (*pivot*)
  - separa gli elementi maggiori del *pivot* da quelli minori, cioè il vettore diventa {elementi <} pivot { elementi >}
  - esegue Quicksort separatamente sui vettori {elementi <} e {elementi >}

Ordinamento: Quicksort (2)



- Scelgo l'elemento pivot e lo escludo dal vettore
- Scorro quindi il resto della lista, separando in due vettori diversi gli elementi rispettivamente minori e maggiori del pivot
- Termino quando tutti gli elementi sono stati inseriti in uno dei due vettori

Ordinamento: Quicksort (3)



- Il procedimento viene ripetuto in maniera ricorsiva, applicato distintamente ai vettori a destra e sinistra dell'elemento pivot
- Quando l'algoritmo termina, il risultato viene "costruito" assemblando i diversi sotto-vettori ordinati, a loro volta formati da sotto-sotto vettori ordinati etc…

Ordinamento: Quicksort (4)

- Quicksort termina, nel caso medio, dopo circa N *log*N operazioni; nel caso pessimo il tempo è però ancora proporzionale a N^2
  - Una buona scelta del pivot è essenziale per evitare di ricadere nel caso pessimo
- Rispetto a Bubblesort, questo algoritmo è mediamente più veloce

Caratteristiche degli algoritmi

- In generale non vi è sempre un'unica soluzione. Al contrario, per uno stesso problema posso esistere vari algoritmi con caratteristiche anche molto differenti in termini di:
  - Complessità computazionale (numero di operazioni richieste per trovare la soluzione)
  - Quantità di memoria richiesta
  - Complessità implementativa

Complessità computazionale (1)

- L'analisi teorica della complessità computazionale ci fornisce uno strumento tecnico per conoscere a priori il numero di operazioni necessarie (e quindi i tempi di esecuzione) per ogni algoritmo
  - La conoscenza di questi parametri permette di confrontare in maniera oggettiva i diversi algoritmi tra loro

Complessità computazionale (2)

- Come è stato mostrato in precedenza, vi è spesso una relazione tra la dimensione dei dati in ingresso ed il numero di operazioni effettuate dall'algoritmo
- In simboli si usa spesso indicare che il tempo è un $O(f(\text{Dim}))$, dove Dim è la dimensione del problema e $f()$ è una funzione matematica
- E' importante osservare che più $f(\text{Dim})$ "cresce rapidamente" con Dim, più l'algoritmo diventa oneroso all'aumentare della dimensione dei dati in ingresso

Complessità computazionale (3)

- Ad esempio, la ricerca lineare ha complessità *O(*N*)* mentre l'algoritmo binaria è dell'ordine *O(log*N*)*
  - Ipotizzo che la ricerca di un numero di telefono nella rubrica (60 utenti circa) richieda mediamente:
    - 0,5 s con l'algoritmo lineare
    - 0,1 s con la ricerca binaria
  - Con questi tempi, la ricerca di un immatricolato tra gli studenti pavesi (60.000 circa) richiederebbe mediamente:
    - 480 s (8 minuti!) con il metodo lineare
    - 4 s con l'algoritmo di ricerca binaria

Caso medio e caso pessimo

- La scelta più consueta è quella di utilizzare il tempo medio, che risulta essere più rappresentativo del caso pessimo (auspicabilmente raro)
  - In alcune applicazioni è necessario però considerare il caso peggiore
- In generale, la scelta tra “prestazioni medie” e “prestazioni minime” riguarda tutti gli ambiti dell’Ingegneria, non solo l’Informatica

Riferimenti

- N. Wirth, “Algoritmi + Strutture dati = Programmi”, 1987
- D. Knuth, The Art of Computer Programming, 1970s
- G. Cupini et al., “Corso di Informatica”, 1987
- en.wikipedia.org/wiki/*(Big_O_notation | Sorting_algorithm | Search_algorithm | Selection_algorithm)*