Elementi Di Informatica E Programmazione

Prof. Andrea Loreggia

UNIVERSITÀ
DEGLI STUDI
DI BRESCIA

Ordinamento

- Il problema dell'ordinamento consiste nel sistemare gli elementi dello array in un preciso ordine. Ad esempio:
  - ordinamento crescente;
  - ordinamento decrescente.

Ordinamento

- Ci sono molti algoritmi di ordinamento:
  - **Bubblesort**
  - **SelectionSort**
  - **InsertionSort**
  - QuickSort
  - MergeSort
  - HeapSort
  - ShellSort

BubbleSort

L’array viene scansionato, ogni coppia di elementi adiacenti viene comparata ed i due elementi vengono invertiti di posizione se sono nell'ordine sbagliato.

Procedura per l’ordinamento crescente:

- si effettua un certo numero di visite dell’intero array.
- Ad ogni visita si confrontano coppie di elementi adiacenti:
  - se il primo valore è maggiore del secondo, la loro posizione sarà invertita;
  - Se durante una visita non avviene alcuno scambio, ciò significa che l’array è ordinato, dunque l’algoritmo può terminare.

BubbleSort

- **Esempio di output** (le parentesi quadre indicano le coppie di elementi in fase di confronto ed eventualmente di scambio/swap).

  - 1: [8 4] 6 1 2 7 5 3
  - 1: 4 [8 6] 1 2 7 5 3
  - 1: 4 6 [8 1] 2 7 5 3
  - 1: 4 6 1 [8 2] 7 5 3
  - 1: 4 6 1 2 [8 7] 5 3
  - 1: 4 6 1 2 7 [8 5] 3
  - 1: 4 6 1 2 7 5 [8 3]
  - 2: [4 6] 1 2 7 5 3 8

BubbleSort

- In pratica, nel BubbleSort, **dopo la k-esima visita dello array**, il k-esimo elemento più grande trova il giusto posto nello ordinamento finale.
- 1: [**8** 4] 6 1 2 7 5 3
- 1: 4 [**8** 6] 1 2 7 5 3
- 1: 4 6 [**8** 1] 2 7 5 3
- . . .
- 2: [4 6] 1 2 7 5 3 **8**

- **Worst case**: $(n - 1) \cdot (n - 1)$ iterazioni.

SelectionSort

L'algoritmo *seleziona* di volta in volta il numero minore/maggiore nella sequenza di partenza e lo sposta nella sequenza ordinata.

Procedura per l'ordinamento crescente:

- ricerca il minimo dell'intero array (A[0 . . . n-1])
- scambia il minimo con l'elemento di posto zero;
- ricerca il minimo nel sottoarray A[1. . . n-1];
- scambia il minimo con l'elemento di posto 1;
- …

SelectionSort

- Esempio di output (le parentesi quadre indicano le coppie di elementi che si scmabieranno la posizione).

- [8] 4 6 [1] 2 7 5 3
- 1 [4] 6 8 [2] 7 5 3
- 1 2 [6] 8 4 7 5 [3]
- 1 2 3 [8] [4] 7 5 6
- 1 2 3 4 [8] 7 [5] 6
- 1 2 3 4 5 [7] 8 [6]

- 1 2 3 4 5 6 [8] [7]

InsertionSort

- Si basa sull'inserimento di ogni elemento in un **sottoarray ordinato.** Descrizione (informale) dello algoritmo:
  - si consideri il **primo elemento** dell'array. Esso rappresenta un sottoarray di lunghezza 1;
  - si **inserisca** il **secondo elemento** al posto giusto nel sottoarray ordinato di lunghezza 1:
    - se questo è minore del primo e unico elemento, quest'ultimo si sposterà a destra;
  - si **inserisca** al posto giusto il **terzo elemento**, spostando li elementi del sottoarray, se necessario, per mantenere l'ordinamento;
  - …

InsertionSort

Esempio di output.

[8] 4 6 1 2 7 5 3

[4 8] 6 1 2 7 5 3

[4 6 8] 1 2 7 5 3

[1 4 6 8] 2 7 5 3

[1 2 4 6 8] 7 5 3

[1 2 4 6 7 8] 5 3

[1 2 4 5 6 7 8] 3

Efficienza algoritmi di sorting

- **L'efficienza di un algoritmo di sorting** può essere stabilita **analiticamente** contando il numero di confronti, **in funzione della dimensione n dello input**.
- Gli algoritmi di ordinamento {Bubble,Selection,Insertion}Sort operano in modo simile:
  - ciclo esterno con numero di iterazioni approssimativamente uguale alla lunghezza dello array;
  - ciclo interno che scandisce approssimativamente tutti gli elementi dello array;
  - di conseguenza, **circa n^2 confronti**

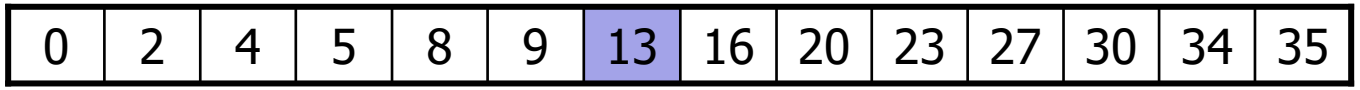
La ricerca dicotomica

- Per "cercare" un elemento in un vettore **ordinato** esiste un metodo detto **ricerca binaria** o **dicotomica**
  - Si confronta il valore **`val`** da ricercare con l'elemento centrale del vettore **`A[length/2]`**
  - Se **`val`** è minore dell'elemento mediano, si ripete la ricerca sulla metà sinistra del vettore, altrimenti si ricerca nella metà destra

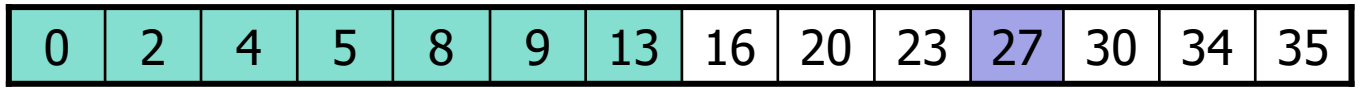
La ricerca dicotomica

- **Esempio:** ricerca del numero 23

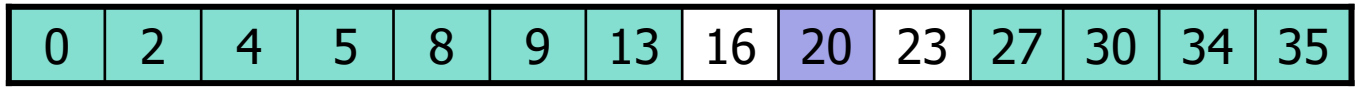
Si confronta 23 con 13



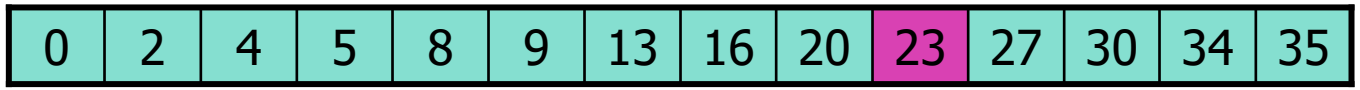
Ci si concentra sulla metà destra (da ind. 7 a ind. 13): si confronta 23 con 27



Ci si concentra sulla metà sinistra (da ind. 7 a ind. 9): si confronta 23 con 20



Ci si concentra sulla metà destra (da ind. 9 a ind. 9): trovato!!



La ricerca dicotomica


```
int search(int val, int A[], int from, int to)
{
   int center=(from+to)/2;

   if (from > to) return -1;
   if (from==to) {
       if (A[from]==val) {return from;}
           return -1;} // si esegue solo se A[from]!=val
   //si esegue solo se (from<to)
   if (val<A[center]){ return search(val,A,from,center-1); }
   if (val>A[center]){ return search(val,A,center+1,to); }
   return center;

}
```

Complessità della ricerca dicotomica

- La ricerca dicotomica divide il vettore in due ad ogni passo:
  - dopo p passi la dimensione del vettore è $n/2^p$
  - nel caso peggiore, la ricerca si ferma quando $n/2^p$ è 1, cioè quando $p=\log_2 n$
- Quindi la ricerca dicotomica è $\mathcal{O}(\log_2 n)$