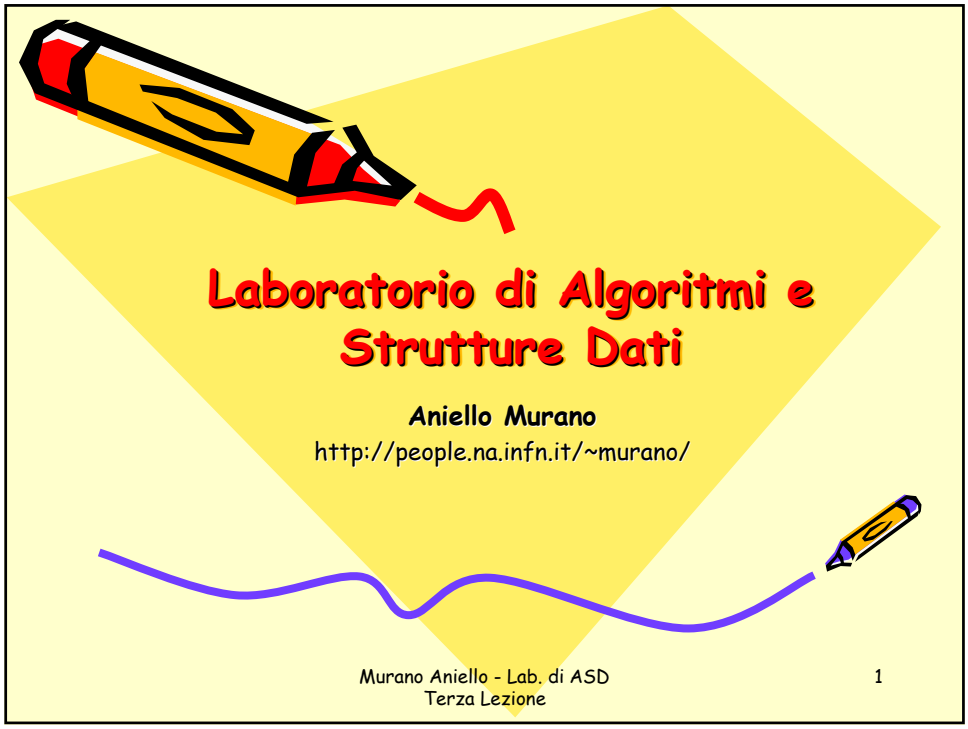
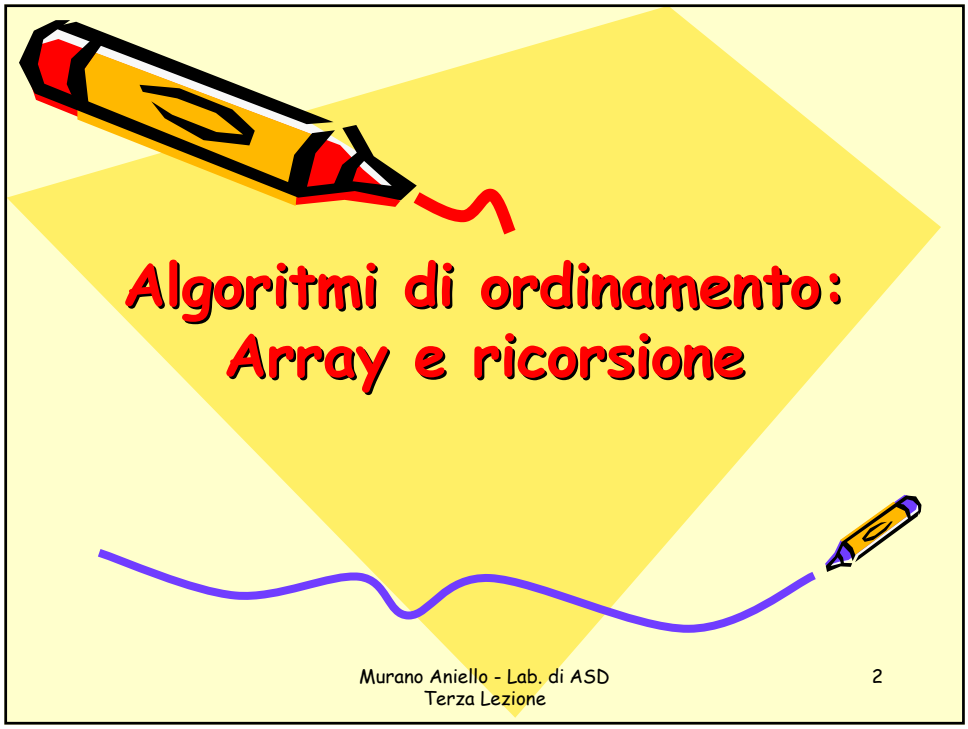


Laboratorio di Algoritmi e Strutture Dati

Aniello Murano
http://people.na.infn.it/~murano/



Algoritmi di ordinamento: Array e ricorsione

Indice

- Algoritmi di ordinamento:
  - Insertion Sort
    - ✓ Codice
    - ✓ Complessità
    - ✓ Documentazione
- Mergesort
  - Ordinamento ricorsivo

Insertion Sort

- L'insertion Sort è uno algoritmo di ordinamento molto efficiente per ordinare un piccolo numero di elementi
- L'idea di ordinamento è simile al modo che un giocatore di bridge potrebbe usare per ordinare le carte nella propria mano.
- Si inizia con la mano vuota e le carte capovolte sul tavolo
- Poi si prende una carta alla volta dal tavolo e si inserisce nella giusta posizione
- Per trovare la giusta posizione per una carta, la confrontiamo con le altre carte nella mano, da destra verso sinistra. Ogni carta più grande verrà spostata verso destra in modo da fare posto alla carta da inserire.

Esempio di Insertion Sort

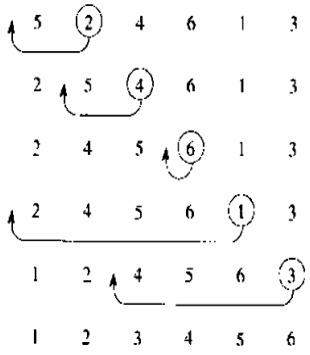
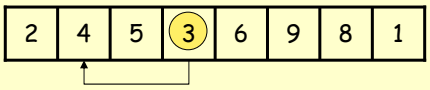


Fig. 1.2 L'elaborazione di Insertion-Sort sull'array A=⟨5,2,4,6,1,3⟩. La posizione dell'indice j è indicata con un circolo.

Funzione Insertion Sort

```
void insertion(int interi[20],int tot)
{
 int  temp;       /* Indice temporaneo per scambiare elementi */
 int  prossimo;
 int  attuale;

 for (prossimo=1;prossimo<tot;prossimo++)
    {
     temp=interi[prossimo];
     attuale=prossimo-1;
     while ((attuale>=0) && (interi[attuale]>temp))
         {
       interi[attuale+1]=interi[attuale];
       attuale=attuale-1;
      }
     interi[attuale+1]=temp;
    }
}
```



Programma per Insertion Sort

```
# include <stdio.h>
# define MAX 20
int  i,j;              /* Indici di scorrimento dell'array */
void insertion(int interi[MAX],int tot);
main()
{
int  interi[MAX]   /* Array che contiene i valori da ordinare */
int  tot;          /* Numero totale di elementi nell'array */
 printf("\nQuanti elementi deve contenere l'array: ");
 scanf("%d",&tot);
 while (tot>20)
   {printf("\n max 20 elementi: ");
   scanf("%d",&tot);}
 for (i=0;i<tot;i++)
    { printf("\nInserire il %d° elemento: ",i+1);
     scanf("%d",&interi[i]);}
   insertion(interi,tot);
   printf("\nArray Ordinato:");
   for (i=0;i<tot;i++)
     printf(" %d",interi[i]);
}
```


Ingegneria del software...

- **Autore**: Nome, Cognome, matricola …
- **Titolo**: nome della funzione (o modulo) implementata.
- **Scopo:** obiettivi dell'algoritmo implementato(sintetico)
- **Specifiche:** Nomi di funzioni, array, variabili importanti
- **Descrizione:** Informazioni sull'algoritmo implementato.
- **Lista dei Parametri:** Parametri input e output
- **Complessità di Tempo e Di Spazio**
- **Altri parametri eventualmente vuoti:**
  - **Indicatori di Errore:**
  - **Routine Ausiliarie**
  - **Indicazioni sull'utilizzo**
- **Implementazione**

Documentazione per Insertion Sort

- **Scopo** : Ordinamento di un array di numeri interi dato in ingresso
- **Specifiche:**
  - ✓ array di interi "interi[MAX]"
  - ✓ void insertion(int interi[MAX], int tot);
- **Descrizione** : L'insertion sort è un algoritmo molto efficiente per ordinare pochi numeri. Questo algoritmo è simile al modo che si potrebbe usare per ordinare un mazzo di carte…
- **Lista dei Parametri**
- *Input:*
  - ✓ interi[ ]: vettore contenente gli elementi da ordinare
  - ✓ tot numero degli elementi contenuti nel vettore
- *Output:* array interi[ ] ordinato in ordine crescente.

Documentazione per Insertion Sort

Complessità di Tempo

- L'algoritmo inserisce il componente **interi[i]** nel vettore già ordinato di componenti **interi[0]...interi[i-1]** spostando di una posizione tutti i componenti che seguono quello da inserire. Ad ogni passo la procedura compie nel caso peggiore (vettore ordinato al contrario), N-1 confronti, essendo N la lunghezza del vettore corrente e i-1 spostamenti. Le operazioni nel caso peggiore sono dunque (1+2+...+N-1), cioè, nel caso peggiore la complessità asintotica di tempo è $O(n^2)$. Nel caso migliore (vettore ordinato) bastano N-1 confronti.

Documentazione per Insertion Sort

Complessità di Spazio:

- La struttura dati utilizzata per implementare l'algoritmo è un ARRAY monodimensionale, contenente i valori da ordinare, di conseguenza la complessità di spazio è O(n).

- Esempi di esecuzione:
  - Dati in ingresso i numeri: 20 11 45, si ottiene in uscita: 11 20 45

Si veda il documento "Documentazione InsertionSort.pdf"

Merge Sort

- L'algoritmo di Merge Sort per mette di ordinare una sequenza di numeri memorizzata in un array utilizzando il seguente ragionamento:
- Si suddivide l'array in due parti di ugual lunghezza con l'obiettivo di applicare alle due parti contigue la procedura di fusione per ottenere il vettore finale ordinato. Poiché i due vettori di partenza devono essere ordinati, riapplica ricorsivamente il metodo a questi vettori

Esempio di Esecuzione

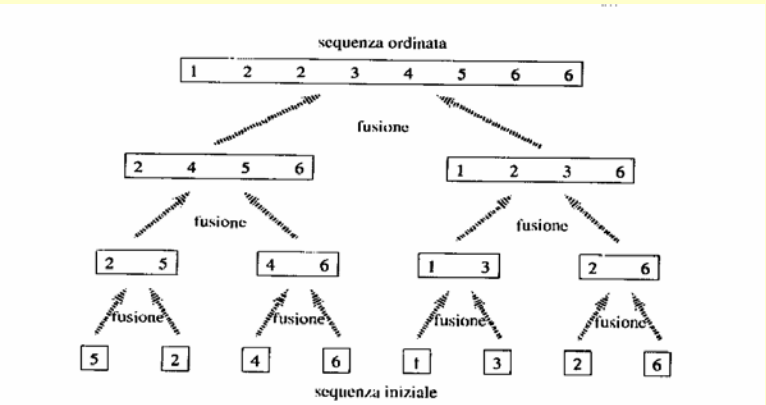


Fig. 1.3 *L'elaborazione di merge sort sull'array A=⟨5,2,4,6,1,3,2,6⟩. La lunghezza delle sequenze ordinate su cui fare la fusione aumenta via via che l'algoritmo progredisce dal basso all'alto.*

Funzione Merge Sort

```
void merge_sort(int interi[], int indice_basso, int indice_alto)
{
      int indice_di_mezzo;
        /* se "ndice_alto==indice_basso" la lista ha un solo elemento */
      if(indice_alto==indice_basso) return;
                                /* calcolo dell'indice a meta' dell'array */
      indice_di_mezzo=(indice_alto-indice_basso)/2+indice_basso;
                 /* potrei ottenere anche semiliste di un solo elemento */
      merge_sort(interi,indice_basso,indice_di_mezzo);
      merge_sort(interi, indice_di_mezzo+1, indice_alto);
                /* le due semiliste sono ordinate e basta fonderle */
      fuse(interi, indice_basso, indice_di_mezzo, indice_alto);
}
```

Complessità di Merge Sort

- Detto k = $\log_2 N$, le chiamate alla procedura mergesort sono in numero di $1+2+4+ \ldots + 2^k = 2^0+2^1+2^2+ \ldots + 2^k = 2^{k+1}$.
- Ad ogni livello di ricorsione, si fa la fusione di M vettori di lunghezza L/M
- Poiché il numero di confronti necessari alla funzione di fusione nel caso peggiore è 2*L, ogni livello compie:
- 2*L*M/2 = L*M confronti
- Pertanto, il numero globale di confronti e':
- **$1*N+2*N/2 + 4*N/4 + \ldots + N*1 = N * \log_2 N$**

Esercizio di laboratorio

- Implementare una funzione per la fusione
- Scrivere un main che a scelta multipla permetta di ordinare una sequenza di numeri utilizzando l'insertion Sort oppure il Merge Sort.

Nella prossima lezione

- In occasione della festività del 1/11, anticipiamo la lezione sugli Stack programmata per l'1/11 il giorno 28/10.