

*Fondamenti di Informatica e Laboratorio T-AB e Fondamenti di Informatica T1  
Ingegneria Elettronica e Telecomunicazioni e  
Ingegneria dell'Automazione  
a.a. 2010/2011*

---

# Lab 14

## Algoritmi di ordinamento

# Algoritmi di ordinamento

---

Scopo di questa esercitazione:

- Comprendere la complessità del problema “ordinamento”... cerchiamo di valutare il “costo” di una soluzione (confrontandola con un'altra...)
- Modifiche al codice degli algoritmi di ordinamento, per supportare tipi di dato “complessi”

# Esercizio 1

---

## Naive Sort con conteggio degli scambi e dei confronti

- Come facciamo a valutare la “bontà” di un algoritmo?
  - Idea: contiamo quante volte eseguiamo le operazioni “costose” di un algoritmo
  - Con Naïve Sort, le operazioni costose possono essere i *confronti* e gli *scambi*.

# Esercizio 1

---

## Naive Sort con conteggio degli scambi e dei confronti

- In un apposito modulo `ordinamento.h` / `ordinamento.c`, realizzare l'algoritmo Naive Sort contando quanti *confronti* e quanti *scambi* vengono effettuati
- Per comodità, definiamo i contatori come variabili *globali statiche*, in modo da potervi accedere da più funzioni

# Esercizio 2

---

- Implementare e modificare l'algoritmo Bubble Sort visto a lezione al fine di contare i confronti e gli scambi eseguiti
- Realizzare un programma che legga un vettore di MAXDIM elementi e ne esegua
  - L'ordinamento con Naive Sort
  - L'ordinamento con Bubble Sorte stampi a video il numero di confronti e scambi effettuato da ogni algoritmo
- Qual'è il caso "peggiore" per bubble sort?

# Esercizio 2 – Soluzione

---

Per comodità:

```
void bubbleSort(int v[], int n) {
    int i;
    int ordinato = 0;

    while (n>1 && ordinato==0) {
        ordinato = 1;
        for (i=0; i<n-1; i++) {
            if ( v[i] > v[i+1] ) {
                scambia( &v[i], &v[i+1]);
                ordinato = 0;
            }
        }
        n--;
    }
}
```

# Esercizio 3

---

- Implementare e modificare l'algoritmo Insert Sort visto a lezione al fine di ordinare un array di float
- Realizzare un programma che legga un vettore di MAXDIM elementi di tipo float, lo ordini usando l'algoritmo InsertSort e stampi a video l'array ordinato.

Suggerimenti:

- Sarà necessario modificare i prototipi delle funzioni usate...
- Sarà necessario controllare che le operazioni di confronto siano ancora effettivamente valide...
- Sarà necessario controllare che le operazioni di assegnamento siano *compatibili* col nuovo tipo...

# Esercizio 3 – Soluzione

---

Per comodità:

```
void insMinore(int v[], int pos) {
    int i = pos-1, x = v[pos];

    while (i>=0 && x<v[i]) {
        v[i+1]= v[i]; /* crea lo spazio */
        i--;
    }
    v[i+1]=x; /* inserisce l'elemento */
}

void insertSort(int v[], int n) {
    int k;
    for (k=1; k<n; k++)
        insMinore(v,k);
}
```



# Esercizio 4

---

Un sito web del turismo trentino tiene un elenco aggiornato delle stazioni sciistiche e del manto nevoso (in cm, un intero). Si deve realizzare un programma che chieda in ingresso, per MAXDIM località, il nome di una località (al più 20 caratteri senza spazi), e l'altezza del manto nevoso (un intero).

- A tal fine si definisca una apposita struttura dati **stazione**
- Si definisca un array di MAXDIM elementi di tipo **stazione**, e si chiedano all'utente i dati relativi a MAXDIM località (nome e neve), memorizzandoli nell'array
- Si realizzi una funzione **compare(stazione s1, stazione s2)** che restituisce **-1**, **0** o **1** a seconda che il manto nevoso in **s1** sia rispettivamente minore, uguale o maggiore al manto nevoso in **s2**
- Si modifichi l'algoritmo Merge Sort visto a lezione, e lo si utilizzi per ordinare le località in base alla neve presente (suggerimento: si usi la funzione **compare (...)** )
- Si stampi a video l'elenco ordinato delle località

# Esercizio 4 – Soluzione

---

Per comodità:

```
void merge(int v[], int i1, int i2, int fine, int vout[]) {
    int i=i1, j=i2, k=i1;

    while ( i <= i2-1 && j <= fine ) {
        if (v[i] < v[j]) {
            vout[k] = v[i];
            i++;
        }
        else {
            vout[k] = v[j];
            j++;
        }
        k++;
    }
    while (i<=i2-1) {
        vout[k]=v[i];
        i++; k++;
    }
    while (j<=fine) {
        vout[k]=v[j];
        j++; k++;
    }
    for (i=i1; i<=fine; i++) v[i] = vout[i];
}
```

# Esercizio 4 – Soluzione

---

Per comodità:

```
void mergeSort(int v[], int iniz, int fine, int vout[]) {
    int mid;

    if ( iniz < fine ) {
        mid = (fine + iniz) / 2;
        mergeSort(v, iniz, mid, vout);
        mergeSort(v, mid+1, fine, vout);
        merge(v, iniz, mid+1, fine, vout);
    }
}
```

# Esercizio 5

---

- Implementare e modificare gli algoritmi Insert Sort, Merge Sort e Quick Sort visti a lezione al fine di contare i confronti e gli scambi eseguiti
- Realizzare un programma che legga un vettore di MAXDIM elementi e ne esegua l'ordinamento con gli algoritmi di cui al punto precedente, e stampi a video il numero di confronti e scambi effettuato da ogni algoritmo