# Fondamenti di Informatica T-1
# Modulo 2

## Contenuti

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
(ordinamento)

**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
(ordinamento)

**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
(ordinamento)

- 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 Sort

  e stampi a video il numero di confronti e scambi effettuato da ogni algoritmo

- Qual'è il caso "peggiore" per bubble sort?



# Esercizio 3
(ordinamento)

- 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 operazione di confronto siano ancora effettivametne valide…
- Sarà necessario controllare che le operazioni di assegnamento siano *compatibili* col nuovo tipo…

# Esercizio 3
(ordinamento)

Per comodità:

```
void insOrd(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++)
        insOrd(v,k);
}
```



# Esercizio 4
(ordinamento)

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 rispettivametne 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
(ordinamento)

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
(ordinamento)

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
(ordinamento)

- 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

22

## Esercizio 6
(riepilogo su ordinamento)

**"Astrazione" degli algoritmi di ordinamento**

- Implementare i diversi algoritmi di ordinamento, facendo in modo di ***astrarre completamente dal tipo*** degli elementi del vettore
- Fare anche in modo che vengano stampate delle ***statistiche sul numero di confronti e di scambi*** effettuati
- Validare la soluzione su un vettore di interi, un vettore di caratteri, un vettore di stringhe

26

## Esercizio 6
(riepilogo su ordinamento)

- Quali sono le istruzioni utilizzate in fase di ordinamento che dipendono dal TIPO dell'elemento?
  - Confronto tra due elementi
  - Assegnamento di un elemento a un altro elemento
  - Swap?
    - dipende dal tipo a causa degli assegnamenti effettuati
    - quindi ci riconduciamo al caso precedente



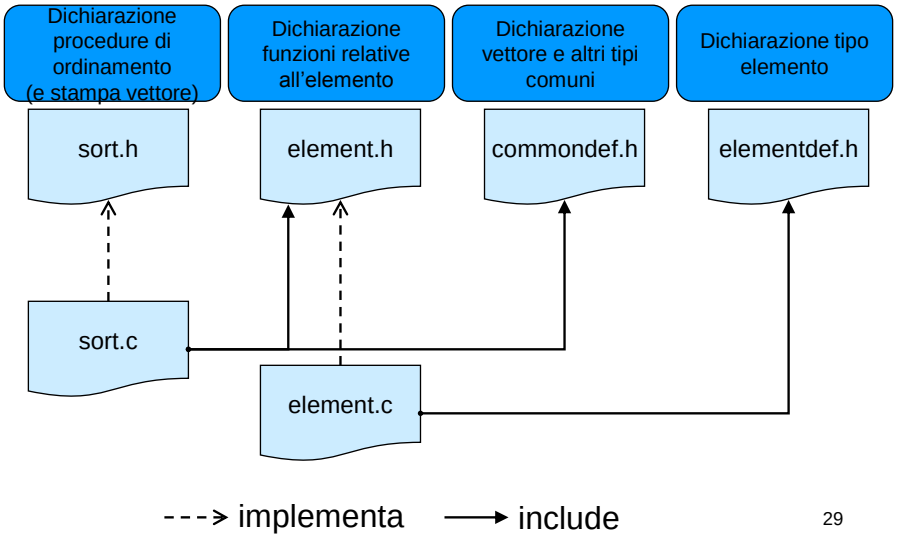
## Esercizio 6
(riepilogo su ordinamento)

- Quindi dobbiamo sostituire
  - Confronti
  - Assegnamenti
- …con delle funzioni capaci di eseguire il confronto e l'assegnamento

```
int compare(Element e1, Element e2);
void assign(Element *lvalue, Element rvalue);
```

# Esercizio 6

(riepilogo su ordinamento)






# Esercizio 6

(riepilogo su ordinamento)

- **`elementdef.h`**
  - Contiene la dichiarazione typedef … Element;
- **`element.h`**
  - Contiene le dichiarazioni delle funzioni per manipolare un elemento
- Quindi se cambio tipo devo aggiornare unicamente
  - **`elementdef.h`**
  - **`element.c`** (l'header rimane uguale, cambia l'implementazione in base al tipo)

# Esercizio 6

(riepilogo su ordinamento)

## Contenuto di element.h

- `int compare(Element e1, Element e2);`
  - Restituisce un numero negativo se `e1 < e2`, 0 se `e1 == e2`, un numero positivo se `e1 > e2`
- `void swap(Element *e1, Element *e2);`
  - Scambio elementi (utilizzando assign!!!)
- `void assign(Element *lvalue, Element rvalue);`
  - Assegna il contenuto di `rvalue` a `lvalue`
- `void printElement(Element e);`
  - Stampa l'elemento a video
- `void printStatistics();`
  - Stampa le statistiche relative a confronti e scambi
  - Suggerimento: utilizzare due variabili contatore globali