# Laboratorio di algoritmi e strutture dati

Docente: Violetta Lonati

Gli esercizi contenuti in questa scheda consistono nell'implementare due algoritmi di ordinamento. Abbiamo già implementato l'*ordinamento per inserimento*, utile per riempire un array mantenendolo ordinato ad ogni passo. In questo caso consideriamo invece un array già riempito ma in disordine e vediamo alcuni algoritmi per ordinarlo: *selection sort* (in italiano ordinamento per selezione) e *mergesort* (in italiano ordinamento per immersione). Per semplicità faremo riferimento soltanto ad array di interi.

Tutti e due gli algoritmi sono basati sui *confronti* tra elementi e sono *ricorsivi*: la base della ricorsione è data dagli array di lunghezza 0 o 1, che sono sempre ordinati.

Innanzitutto scrivete una funzione che stampi il contenuto dell'array `a` di lunghezza `lung`:

```
void stampa( int a[], int lung );
```

Quindi scrivete la funzione `main` che legga da standard input una sequenza di interi, la memorizzi in un array e stampi l'array chiamando la funzione `stampa` di cui sopra. Una volta completato, questo main servirà per testare i due algoritmi di ordinamento.

Poi scrivete una funzione che scambi di posto due elementi di un array:

```
void scambia( int a[], int i, int j );
```

Per ciascun algoritmo di ordinamento, dovrete scrivere una funzione seguendo le indicazioni che seguono. I prototipi dovranno essere i seguenti:

```
void selectionsort( int a[], int n );
void mergesort( int a[], int sx, int dx );
```

Il programma risultante dovrà quindi avere questa struttura:

```
#include ...
#define N ...

void stampa( int a[], int lung );
void scambia( int a[], int i, int j );
void selectionsort( int a[], int n );
void mergesort( int a[], int sx, int dx );

int main( void ) {
        int a[N];

        ... /* lettura di un array di interi da standard input */

        /* scegliere uno dei due algoritmi:
        selectionsort( a, N);
        mergesort( a, 0, N);
        */
```

```
        stampa( a, N );
        return 0;
}

/* definizione delle altre funzioni */
...
```

## 1 Selection sort

Ricordo che la funzione `selectionsort( int a[], int n )` deve funzionare come segue:

1. innanzitutto cerca nel vettore l'elemento massimo e lo sposta nell'ultima posizione dell'array;
2. poi richiama se stessa ricorsivamente per ordinare i primi $n-1$ elementi dell'array.

La base della ricorsione è data dagli array di lunghezza 0 o 1, che sono sempre ordinati.

## 2 Mergesort

Ricordo che la funzione `mergesort` deve funzionare come segue:

1. divide l'array in due sotto-array di dimensione circa uguale;
2. ordina il sotto-array di sinistra richiamando se stessa;
3. ordina il sotto-array di destra richiamando se stessa;
4. integra (*merge*) i due array in un array ordinata.

La base della ricorsione è, anche qui, data dagli array di lunghezza 0 o 1, che sono sempre ordinati.

La parte di integrazione (merge) di due array ordinati `a1` e `a2` funziona con un vettore di supporto: si scorrono entrambi gli array da sinistra a destra usando due indicatori `i1` e `i2` rispettivamente; ad ogni passo si confronta `a1[i1]` con `a2[i2]` e si sceglie l'elemento più piccolo, lo si copia nell'array di supporto (nella prima posizione libera) e si incrementa l'indicatore relativo ad esso. Quando `i1` esce da `a1` oppure `i2` esce da `a2`, la parte rimanente dell'altro array viene copiata nell'array di supporto. Alla fine si copia il contenuto del file di supporto nel file originale.