Esercizio 1

Scrivere un programma che legga da input una sequenza di punti a coordinate intere, dove per ciascun punto l'utente immette le due coordinate. Quindi il programma stampa la sequenza di punti ordinata per valori crescenti dell'ascissa. L'input è formattato nel solito modo: il primo numero che l'utente inserisce è N e identifica la lunghezza della lista di punti. Seguono poi $2N$ interi in input che rappresentano le coordinate di N punti (l'ascissa precede l'ordinata).
Suggerimento: utilizzare la libreria `qsort` per l'ordinamento dei punti.

Esercizio 2

Scrivere un programma che legga da tastiera una sequenza di n interi distinti e li inserisca in una lista mono-direzionale (nell'ordine dato). Il programma entra poi in un ciclo infinito nel quale legge un intero i da tastiera e lo cerca nella lista. Se i si trova nella lista stampa la sua posizione (contando da 0) e porta l'elemento in testa alla lista (`MTF`, Move To Front), altrimenti stampa `NO`. L'input è formattato nel seguente modo: nella prima riga si trova la lunghezza della sequenza e nelle successive si trovano gli interi che compongono la sequenza. **Gli interi sono tutti distinti.**
Suggerimento: per forzare la terminazione di un programma (perché per esempio è fermo su un ciclo infinito) premere la combinazione di tasti `Ctrl+C` nel terminale.

Esercizio 3

Scrivere un programma che legga da tastiera una sequenza di N interi e li inserisca in una lista bidirezionale. Il programma entra in un ciclo infinito nel quale legge un intero i da tastiera e lo cerca nella lista. Se i si trova nella lista, stampa la sua posizione (contando da 0), altrimenti stampa `NO`. Ogni elemento della lista mantiene anche un contatore che dice quante volte è stata cercata la corrispondente chiave. Tutti i contatori sono inizialmente impostati a 0. Dopo ogni ricerca si deve garantire che gli elementi della lista siano ordinati in ordine non-crescente di contatore.
L'input è formattato al solito modo: nella prima riga si trova la lunghezza della sequenza e nelle successive si trovano gli interi che compongono la sequenza.
NOTA: non si devono utilizzare algoritmi di ordinamento ma osservare che inizialmente la lista è ordinata e che dopo ogni ricerca solo un contatore viene incrementato.

Esercizio 4

Definire una `struct` in questo modo:

```
typedef struct _Nodo{
   int key; // chiave intera
   struct _Nodo* next; // puntatore al prossimo nodo
} Nodo;

typedef Nodo* Lista;
```

Una lista è rappresentata da un puntatore al suo nodo di testa (o `NULL` se vuota). Implementare le seguenti funzioni per liste:

- `void print(Lista l);`
  stampa le chiavi di tutti i nodi di `l`.
- `Lista insertBack(Lista l, Nodo *n);`
  inserisce `n` in fondo ad `l` e restituisce la nuova lista
- `Lista insertFront(Lista l, Nodo *n);`
  inserisce `n` in testa ad `l` e restituisce la nuova lista
- `void deallocate(Lista l);`
  libera lo spazio occupato da tutti i nodi di `l`.