```
/*
Versione 3 - vedi SA.
Una versione C del QSort definito utilizzando la Metodologia della Programmazione Struttura
Vedere il relativo materiale discusso nelle lezioni di ... 2015 a Matematica.
Vedere il progetto e le motivazioni di questa versione C, prima di studiarla.
Versione 3 - avanzameno:
L'operazione di divisione e' unica e calcola la coppia delle sottosequenze della copertura,
come nella versione 2, ma le sottosequenze non sono due nuove sequenze bensi le stesse su
cui è stata effettuata una ridistribuzioni dei valori degli elementi.
(Vedere la versione 1 e versione 2 per confronto).
Algoritmo di divisione.
Si utilizza una sequenza modificabile e a doppia scansione.
Siano inf e sup gli accessi agli estremi inferiore e superiore della sottosequenza [inf,sup
elementi, della sequenza data, che vogliamo dividere rispetto ad un separatore a.
L'algoritmo scambia i contenuti degli elementi in [inf,sup] e restistuisce due coppie (inf1
(inf2,sup2) tali che: indicato con oldseq la sequenza in [inf,sup] prima della divisione,
 - inf1=inf , sup2=sup
 - [inf1,sup1] contiene tutti gli elementi in oldseq minori a o equivalenti;
 - [inf2,sup2] contiene tutti gli elementi in oldseq maggiori di a o equivalenti.

OSSERVARE.
La struttura del C impone di dichiarare la struttura della coppia nella intestazione del pr
invece che nelle dichiarazioni locali di QSort (come dovrebbe essere, essendo questa defini
introdotta ad uso esclusivo del procedimento seguito da QSort).
*/

#include <stdio.h>
#include <stdlib.h>
#include "mutSubS.h"

struct Coppia {//coppia di elementi -- privata di QSort
    intMutS left;
    intMutS right;
};

typedef struct Coppia Pair; //privata di QSort
typedef Pair * elemPair;    //privata di QSort

elemPair mkPair(intMutS left, intMutS right){//privata di QSort
    elemPair pair = (elemPair)malloc(sizeof(Pair));
    pair->left = left;
    pair->right = right;
    return pair;
}


int ord(int t1, int t2){//parametro di QSort
        //un'ordinamento sugli interi tra gli infiniti possibili
    return (t1<t2);
}
int separatore(intMutS u){//privata di QSort
        // richiede: !isEmpty(u)
        /* calcola: un elemento a caso */
    return val(u);
}

int eqOrd(int x, int y){//parametro di QSort
        //operazione di equivalenza su valori ordinati
    return ((x==y)||(ord(x,y)&&ord(y,x)));
}
```

```
elemPair divisione(intMutS inf, intMutS sup, int a){//privata di QSort
        // manca il parametro ord.
        /* Calcola coppia (sup1,inf2) delle due sequenze della divisione.
         */
    int met = 0;
    intMutS inf2 = inf;
    intMutS sup1 = sup;
    met = met || (sup1 == inf2);
    while (!met) {
        int x = val(inf2);
        while ((inf2!=sup) && (ord(x,a)||eqOrd(x,a))) {
            inf2 = moveR(inf2);
            x = val(inf2);
            met = met || (sup1==inf2);
            }
        int y = val(sup1);
        while ((sup1!=inf) && ord(a,y) && (!eqOrd(x,a))) {
            sup1 = moveL(sup1);
            y = val(sup1);
            met = met || (sup1==inf2);
            }
        if (!met) swap(sup1,inf2);
        }
    if (inf2 == sup1) sup1 = moveL(sup1);
    return mkPair(sup1,inf2);
}


void QSort(intMutSubS cc){
    /* Osservazioni Versione1 e Versione2.
       Questa QSort ordina gli elementi della sottosequenza [cInf,cSup],
       lasciando invariati tutti gli altri elementi, se presenti.
    */
    intMutS cInf = inf(cc);
    intMutS cSup = sup(cc);
    if (cInf == cSup) return;
    {
        int a = separatore(cSup);
        elemPair sup1inf2 = divisione(cInf,cSup,a);
        QSort(mkSubS(cInf,sup1inf2->left));
        QSort(mkSubS(sup1inf2->right,cSup));
    }
}
```