```
list mergeNonDistruttiva(list l,list m) {
    if(l==NULL)
        return m;
    else if(m==NULL)
        return l;
    else if(l->info <= m->info){
        return Cons(l->info, merge(l->next, m));
    }else{    // l->info > m->info
        return Cons(m->info, merge(l, m->next));
    }
}
```

```
list merge(list l,list m) {
    if(l==NULL)
        return m;
    else if(m==NULL)
        return l;
    else if(l->info <= m->info){
        l->next = merge(l->next, m);
        return l;
    }else{    // l->info > m->info
        m->next = merge(l, m->next);
        return m;
    }
}
```

```
list split (list l) {
    list slow = l;
    list fast = l->next;
    while (fast && fast->next) {
        slow = slow->next;
        fast = fast->next->next;
    }
    list secondHalf = slow->next;
    slow->next = NULL;
    return secondHalf;
}
```

```
list mergeSort(list l) {
    if (l == NULL || l->next == NULL)
        return l;
    else {
        list m = split(l); // divide d
                           // ad una m
        l = mergeSort(l);
        m = mergeSort(m);
        return merge(l, m);
    }
}
```

```
list intersezione(list l, list m) {
    if(l == NULL) return m;
    if(m==NULL) return l;
    else if(l->info == m->info)
        return Cons(l->info, intersezione(l->next, m->next));
    else if(l->info < m->info)
        return intersezione(l->next, m);
    else return intersezione(l, m->next);
}
```

```
list unione(list l, list m) {
    if(l == NULL) return m;
    if(m==NULL) return l;
    else if(l->info == m->info)
        return Cons(l->info, unione(l->next, m->next));
    else if(l->info < m->info)
        return Cons(l->info, unione(l->next, m));
    else return Cons(m->info, unione(l, m->next));
}
```

```
list differenza(list l, list m) {
    if(l == NULL) return m;
    if(m==NULL) return l;
    else if(l->info == m->info)
        return differenza(l->next, m->next);
    else if(l->info < m->info)
        return Cons(l->info, differenza(l->next, m));
    else return differenza(l, m->next);
}
list symmDiff(list l, list m) {
    if(l == NULL) return m;
    if(m==NULL) return l;
    else if(l->info == m->info)
        return symmDiff(l->next, m->next);
    else if(l->info < m->info)
        return Cons(l->info, symmDiff(l->next, m));
    else return Cons(m->info, symmDiff(l, m->next));
}
```

```
int rank(list l) {
    if(l==NULL) return 0;
    else{
        l->info += rank(l->next);
        return l->info;
    }
}
```

```
list reverse(list l) {
    list reversed = NULL;
    while(l != NULL){
        reversed = Cons(l->info, reversed);
        l=l->next;
    }
    return reversed;
}
```

```
list fast_reverse_aux(list l, list m){
    if (l==NULL) return m;
    else return fast_reverse_aux(l->next, Cons(l->info, m));
}
list fast_reverse(list l){return fast_reverse_aux(l, NULL);}
```

```
int corank_aux (list l, int cumulativeCoRank) {
    if(l==NULL) return cumulativeCoRank;
    else{
        int tmp = l->info;
        l->info += cumulativeCoRank;
        return corank_aux(l->next, cumulativeCoRank + tmp);
    }
}
int corank(list l){return corank_aux(l, 0);}
```

```
bool equal(list l, list r){
    if(l==NULL) return r==NULL;
    else if(r==NULL) return l==NULL;
    else return l->info == r->info && equal(l->next, r->next);
}
bool palindrome(list l) {
    if(l==NULL) return true;
    else{
        list reversered= reverse(l);
        return equal(l, reversered);
    }
}
```

```
list insert(list as, list bs, int n) {
    if(n==0) return concat(as,bs);
    else{
        bs->next = insert(as, bs->next, n-1);
        return bs;
    }
}
```

```
list deleteAll(int n, list as) {
    if(as == NULL) return NULL;
    else if(as->info == n){
        list t= as->next;
        free(as);
        return deleteAll(n, t);
    }else{
        as->next= deleteAll(n, as->next);
        return as;
    }
}
```

```
// DISPARI
list odd_aux(list l){
    if(l==NULL) return l;
    else if(l->info % 2 == 1)
        return Cons(l->info, odd_aux(l->next));
    else return odd_aux(l->next);
}
// PARI
list even_aux(list l){
    if(l==NULL) return l;
    else if(l->info % 2 == 0)
        return Cons(l->info, even_aux(l->next));
    else return even_aux(l->next);
}
list oddFisrt(list l) {return concat(odd_aux(l), even_aux(l));}
```