LISTAS

A)
```
List *list_merge_In_Order(const List *pla, const List *plb){
    List *resultado = NULL;
    Node *pa = NULL, *pb = NULL;
    if(!pla || !plb) return NULL;

    resultado = lista_crear();
    if(!resultado) return NULL;
    pa = pla->first;
    pb = plb->first;
    while(pa != NULL && pb != NULL){
        if(elemento_cmp(pa->info, pb->info) > 0){   // si el elemento b es mayor al a o mismo al elemento a. se asume que ningún pa->info es NULL.
            lista_insertar_Fin(resultado, pa->info);
            pa = pa->next;
        }
        else{
            lista_insertar_Fin(resultado, pb->info);
            pb = pb->next;
        }
    }
    while(pa != NULL){
        lista_insertar_Fin(resultado, pa->info);
        pa = pa->next;
    }
    while(pb != NULL){
        lista_insertar_Fin(resultado, pb->info);
        pb = pb->next;
    }
    return res;
}
```

B) La complejidad depende de cómo esté implementada lista_insertar_Fin. Se asume, por la estructura, que cuesta de tiempo $O(n)$ (recorre todos los elementos hasta llegar al final), así que el coste de tiempo es, al tener el bucle while, $O(2n \cdot n) = O(n^2)$.

↓ recorre 1 vez la lista a y otra la b

↓ el coste de insertar_Fin

En memoria la complejidad es de tres variables auxiliares más la lista resultante, así que es $O(m+n)$, donde m elementos tiene la lista a y n la b.

```
c) List *list_mergeInOrder(const List *pla, const List *plb){
    List *res = NULL, *pla2 = NULL, *plb2 = NULL;
    Node *pa = NULL, *pb = NULL;
    if(!pla || !plb) return NULL;

    pla2 = list_copy(pla);
    plb2 = list_copy(plb);
    while(list_isEmpty(pla2) == FALSE && list_isEmpty(plb2) == FALSE){
        pa = list_extractFront(pla2);
        pb = list_extractFront(plb2);
        if(node_cmp(pa, pb) > 0){
            list_insertRear(res, pa);
            list_insertFront(plb2, pb);
        }
        else{ list_insertRear(res, pb);
              list_insertFront(pla2, pa);
        }
        node_free(pa);
        node_free(pb);
    }
    while(list_isEmpty(pla2) == FALSE){
        pa = list_extractFront(pla2);
        list_insertRear(res, pa);
        node_free(pa);
    }
    while(list_isEmpty(plb2) == FALSE){
        pb = list_extractFront(plb2);
        list_insertRear(res, pb);
        node_free(pb);
    }
    list_free(pla2);
    list_free(plb2);
    return res;
}
```

d) Similar a b), si el coste de insertar al final es O(n) entonces la función es $O(2n^2 + 2n^2) = O(n^2)$, y si es O(1) entonces $O(2n + 2n) = O(n)$. El de la memoria es $O(3n) = O(n + m)$, el tamaño de m + el de l.