

Problema 1 (15 puntos)

Considere la siguiente definición de listas de enteros de memoria dinámica:

```
typedef nodoLista * Lista
struct nodoLista{ int dato; Lista sig; }
```

a) Implemente una función iterativa **ordenar** que dada una lista l de tipo *Lista* que puede contener valores exclusivamente en el rango $[0 : m]$ (entre 0 y m inclusive, con $m > 0$), retorne una nueva lista (sin compartir memoria) ordenada de mayor a menor que contenga los elementos de l pero sin incluir las repeticiones. Si l es vacía (NULL), el resultado debe ser NULL. Se pueden usar estructuras de datos auxiliares, manejando adecuadamente la memoria (pedido y liberación, si corresponde). **La función *ordenar* debe ser $O(\max(n,m))$ en el peor caso**, siendo n el largo de l .

PRE: Cada elemento x de la lista l cumple: $0 \leq x \leq m$, con $m > 0$
Lista ordenar(Lista l, int m)

Por ejemplo, si l es $[2,4,9,4,1,4,2,9,2,99,2,5]$ y m es 99, el resultado debe ser $[99,9,5,4,2,1]$.

b) Justifique muy brevemente el cumplimiento del orden exigido en la parte a) para su implementación de **ordenar**.

Problema 2 (15 puntos)

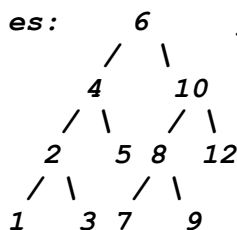
Considere la siguiente definición del tipo *ABB* de los árboles binarios de búsqueda de enteros, en memoria dinámica:

```
typedef nodoABB * ABB
struct nodoABB { int dato; ABB izq, der; }
```

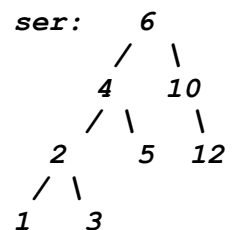
a) Implemente un procedimiento recursivo **elimSubArbol** que dado un *ABB* t y dado un entero x , elimine de t el subárbol que tenga a x como raíz, liberando toda la memoria que corresponda. Si x no está en t (en particular si t es NULL), el **elimSubArbol** no tendrá efecto. El procedimiento deberá evitar recorrer elementos de t que no sean estrictamente necesarios. Se sugiere implementar un procedimiento auxiliar.

void elimSubArbol(ABB & t, int x)

Ejemplo: Si t es:



y $x=8$ entonces el resultado debe ser:



b) Indique el orden de tiempo de ejecución para el peor caso de **elimSubArbol**. Explique muy brevemente el peor caso.

SOLUCIONES**1-a)**

PRE: Cada elemento x de la lista l cumple: $0 \leq x \leq m$, con $m > 0$

```
Lista ordenar(Lista l, int m){
    Lista ret = NULL;
    bool * elementos = new bool[m+1];
    for (int i=0; i<=m; i++){
        elementos[i] = false;
    }
    while (l!=NULL){
        elementos[l->dato] = true;
        l = l->sig;
    }
    for (int i=0; i<=m; i++){
        if(elementos[i])
            insComienzo(i,ret); // inserción al comienzo de ret
    }
    delete [] elementos;
    return ret;
}
```

// inserta un elemento al comienzo de una lista

```
void insComienzo(int e, Lista & l){
    Lista nuevo = new nodoLista;
    nuevo -> dato = e;
    nuevo -> sig = l;
    l = nuevo;
}
```

1-b)

Lista ordenar(Lista l, int m){ *n es la cantidad de elementos de l*

<i>Lista ret = NULL;</i>	<i>O(1)</i>			
<i>bool * elementos = new bool[m+1];</i>				
<i>for (int i=0; i<=m; i++)</i>			<i>O(m)</i>	
<i>elementos[i] = false;</i>				
<i>while (l!=NULL){</i>				
<i>elementos[l->dato] = true;</i>			<i>O(n)</i>	<i>O(max(n,m))</i>
<i>l = l->sig;</i>				regla de la suma
<i>}</i>				
<i>for (int i=0; i<=m; i++){</i>				
<i>if(elementos[i])</i>				
<i>insComienzo(i,ret);</i>	<i>O(1)</i>		<i>O(m)</i>	
<i>}</i>				
<i>delete [] elementos;</i>				
<i>return ret;</i>	<i>O(1)</i>			
<i>}</i>				

2-a)

```
void elimSubArbol(ABB & t, int x){
    if (t != NULL){
        if (x == t->dato)
            elimArbol(t);
        else if (x < t->dato)
            elimSubArbol(t->izq, x);
        else elimSubArbol(t->der, x);
    }
}

// Elimina todos los elementos del árbol.
void elimArbol(ABB & t){
    if (t != NULL){
        elimArbol(t->izq);
        elimArbol(t->der);
        delete t;
        t = NULL;
    }
}
```

2-b)

elimSubArbol es $O(n)$ peor caso, siendo n la cantidad de elementos/nodos del árbol. El peor caso se da si se quiere eliminar la raíz del árbol (se debe recorrer todo el árbol para borrar sus nodos).