EXAMEN Estructura de Datos y Algoritmos.
Julio 2011
El examen suma 100 puntos. Para aprobar se necesitan 60.

Ejercicio 1) (15 puntos)

Calcule el orden del tiempo de ejecución de los siguientes programas:

a)
```
int sum=0;
for (int i=0;i<n;i++)
        for (int j=0;j<n-1;j++)
                sum++;
```

$$T(n) = 1 + \sum_{i=0}^{i=n-1} \sum_{j=0}^{j=n-1} 1 = 1 + \sum_{i=0}^{i=n-1} n = 1 + n.n = 1 + n^2 \leq cn^2 \forall n_0 \geq 1, c = 2 \Rightarrow T(n) \text{ es } O(n^2)$$

b)
```
int sum=0;
for (int i=1;i<n;i++)
        for (int j=0;j<i;j++)
                sum++;
```

$$T(n) = 1 + \sum_{i=0}^{i=n-1} \sum_{j=0}^{j=i-1} 1 = 1 + \sum_{i=0}^{i=n-1} i = 1 + \sum_{i=1}^{i=n} i = 1 + n.\frac{(n+1)}{2} = 1 + \frac{n^2 + n}{2} \leq cn^2 \forall n_0 \geq 1, c = 2 \Rightarrow T(n) \text{ es } O(n^2)$$

Ejercicio 2) (15 puntos)

Implemente el siguiente procedimiento que calcula el máximo y el mínimo de un arreglo de enteros.

void max_min(int arreglo [], int & max, int & min)

Se corrige cabezal de la función ya que falta la cantidad de elementos del arreglo.

```
void max_min(int arreglo [], int cantElem,  int & max, int & min){

     max = arreglo[0];
     min = arreglo[0];

     for(int i=1; i<cantElem;i++){
           if (arreglo[i]<min){
               min=arreglo[i];
           }
           if (arreglo[i]>max){
               max=arreglo[i];
           }
   }
}
```

Ejercicio 3 (30 puntos)

Considere la siguiente definición del tipo LISTA, de listas de enteros en memoria dinámica:

```
struct nodoLista {
      int info;
      nodoLista *sig;
};
typedef nodoLista *LISTA;
```

Se pide, sin usar funciones o procedimientos auxiliares, implementar la siguiente función iterativa:
IncUno: Dada una lista *L* de enteros de tipo LISTA, retorna una nueva lista (que no comparte registros de memoria con la lista parámetro) que contiene a todos los elementos de *L* incrementados en 1. Si *L* es una lista vacía, la función debe retornar la lista vacía.

Ejemplos:

Entradas	**Resultado de *IncUno***
L = [2,7,8,3]	[3,8,9,4]
L = [2]	[3]
L = []	[]

```
LISTA IncUno(LISTA l){
      LISTA laux = l;//lista auxiliar para recorrer l
      LISTA lres = NULL; //lres va a ser la nueva lista a construir y retornar
      LISTA lres_ult = NULL;//auxiliar para construir lres, apunta al último nodo de
                                                                        //lres
      laux = l;
      while(laux != NULL){
           //creo nuevo nodo en la lista a retornar y lo agrego al final
           If (lres_ult == NULL){
                 //estoy construyendo el primer nodo
                 lres = new struct nodoLista;
                //copio información de l en lres
                lres->info=laux->info+1;
                lres->sig=NULL;
                 lres_ult = lres;
           }
           else {
                 //no es el primer nodo
                 lres_ult->sig = new struct nodoLista;
                //copio información de l en lres
                 lres_ult = lres_ult->sig;
                lres_ult->info=laux->info+1;
                lres_ult->sig=NULL;
           }

          //avanzo l
           laux = laux->sig;
      }//while

      return lres;
}
```

Ejercicio 4 (40 puntos)

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

```
struct nodoABB {
        int dato;
        nodoABB * izq; // menores que dato
        nodoABB * der; // mayores que dato
};
typedef nodoABB *ABB;
```

a) Defina una función recursiva *BorrarMax* que dado un árbol binario de búsqueda de enteros *A* de tipo ABB, no vacío (precondición), elimine y retorne de *A* su máximo elemento. No se pueden usar funciones o procedimientos auxiliares. *BorrarMax* debe evitar recorrer nodos innecesarios de *A*. Deberá liberarse la memoria de la celda cuyo elemento sea eliminado. *A* luego de la función debe ser un árbol binario de búsqueda.

int BorrarMax (ABB & A)

b) Defina un procedimiento *BorrarRaiz* que dado un árbol binario de búsqueda de enteros *A* de tipo ABB, no vacío (precondición), elimine el entero que se encuentra en la raíz de *A*. No se pueden usar funciones o procedimientos auxiliares, con excepción de la función definida en la parte (a). *BorrarRaiz* no debe recorrer *A*, aunque si puede hacerlo eventualmente *BorrarMax*. *A* luego del procedimiento debe ser un árbol binario de búsqueda.

void BorrarRaiz (ABB & A)

a)

```
//Precondicion: A es no vacio.
int BorrarMax(ABB &A){
   ABB auxABB;
   int valMax;
   if (A->der != NULL){
      return BorrarMax(A->der);
   }
   else {
      auxABB = A;
      valMax = auxABB->dato;
      A = A -> izq;
      delete auxABB;
      auxABB = NULL;
      return valMax;
   }
}
```

b)

```
void BorrarRaiz(ABB &A){
   ABB auxABB=A;
   if (auxABB->izq==NULL) {
      A=auxABB->der;
      delete auxABB;
      auxABB = NULL;
   }
   else {
   //puedo invocar BorrarMax porque ya se que el hijo izquierdo no es nulo
   A->valor=BorrarMax(auxABB->izq);
   }
}
```