EXAMEN Estructura de Datos y Algoritmos.

Febrero 2011

El examen suma 100 puntos. Para aprobar se necesitan 60.

Ejercicio 1. (35 puntos)

Considere la siguiente función en C:

```
bool F (int *A, unsigned int n)
{     bool res = true;
      int i, j;
      for (i=0; i<n; i++)
          for (j=0; j<n; j++)
               if (i<j)
                   res = res && (A[j]<A[i]);
      return res;
}
```

a) ¿Qué calcula/retorna (conceptualmente) la función F, dado un arreglo de enteros de tamaño n?.

b) Calcule el orden (O) de tiempo de ejecución para el peor caso de la función F. El cálculo puede ser realizado esquemáticamente sobre el código mismo.

c) El problema que resuelve F, ¿podría resolverse en un menor orden de tiempo de ejecución en el peor caso?. Justifique en caso negativo y en caso afirmativo escriba la función, indicando el orden (O).

a) Retorna true si el arreglo esta ordenado de forma decreciente y false en caso contrario.

b)

```
bool F (int *A, unsigned int n) O(n²)
{  bool res = true;
   int i, j;
   for (i=0; i<n; i++)     O(n²)
         for (j=0; j<n; j++)    O(n)
              if (i<j) O(1)
                  res = res && (A[j]<A[i]);  O(1)
   return res;
}
```

Respuesta $O(n^2)$

c) Si podría resolverse en un menor orden de tiempo de ejecución en el peor caso.
La siguiente función retorna el mismo resultado que la anterior y tiene O(n)

```
bool F (int *A, unsigned int n) O(n)
{  bool res = true;
   int i, j;
   for (i=0; i<n-1; i++) O(n)
         res = res && (A[i+1]<A[i]);   O(1)
   return res;
}
```

Ejercicio 2 (25 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:

InvParcial: 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, excepto al último, pero en orden inverso. Si L es una lista vacía, la función debe retornar la lista vacía.

Ejemplos:

Entradas	**Resultado de *InvParcial***
L = [2,7,8,3]	[8,7,2]
L = [2]	[]
L = []	[]

```
LISTA InvParcial(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 laux2 = NULL;//auxiliar para construir lres
      laux = l;
      if (laux != NULL){
            //pregunto por el siguiente para no copiar el último
            while(laux->sig != NULL){
            //creo nuevo nodo en la lista a retornar y lo agrego al principio
                  laux2 = lres;
                  lres = new struct nodoLista;
                  //copio información de l en lres
                  lres->info=laux->info;
                  lres->sig=laux2;
                  //avanzo l
                  laux = laux->sig;
            }
      }
      return lres;
}
```

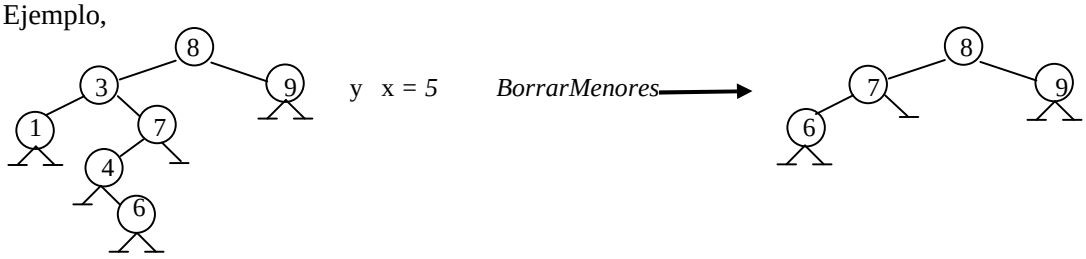
Ejercicio 3 (40 puntos)
Considere la siguiente definición del tipo ABB de los árboles binarios de búsqueda de enteros, en memoria dinámica:

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

typedef nodoABB *ABB;
```

Defina un procedimiento recursivo ***BorrarMenores*** en que dados un árbol binario de búsqueda de enteros A de tipo ABB y un entero x (que puede o no estar en A), elimine de A todos los elementos del árbol cuyos datos son menores a x. El resultado debe ser un árbol binario de búsqueda. El procedimiento debe evitar recorrer todo el árbol, de ser posible. Deberá liberarse la memoria de las celdas cuyos elementos sean eliminados.
Se sugiere usar un procedimiento auxiliar (que deberá ser implementado) que borre todo un árbol (subárbol).

Ejemplo,



Si x fuera igual a 10 en el ejemplo previo, el árbol quedaría vacío.

```
void BorrarArbol(ABB & A){
     if (A != NULL){
        if ((A->izq == NULL)&& (A->der == NULL)){
           delete A;
           A = NULL;
        }
        else {
             BorrarArbol(A->izq);
             BorrarArbol(A->der);
             delete A;
             A = NULL;
             }
     }
}

void BorrarMenores(ABB & A, int x){

    ABB arbolAux = NULL;

    if (A != NULL){

       if (A->dato == x)
          //Borro el izquierdo y terminé ya que los del derecho son mayores
          BorrarArbol(A->izq);
       else if (A->dato < x){
            //debo borrar la raiz y todo el subarbol izquierdo
               BorrarArbol(A->izq);
               arbolAux = A;
```

```
            A = A->der;
            delete arbolAux;
            arbolAux = NULL;
            BorrarMenores(A, x);
         }
         else if (A->dato > x){
         //la raiz es mayor debo chequear el subarbol izquierdo
            BorrarMenores(A->izq, x);
         }
    }

}
```