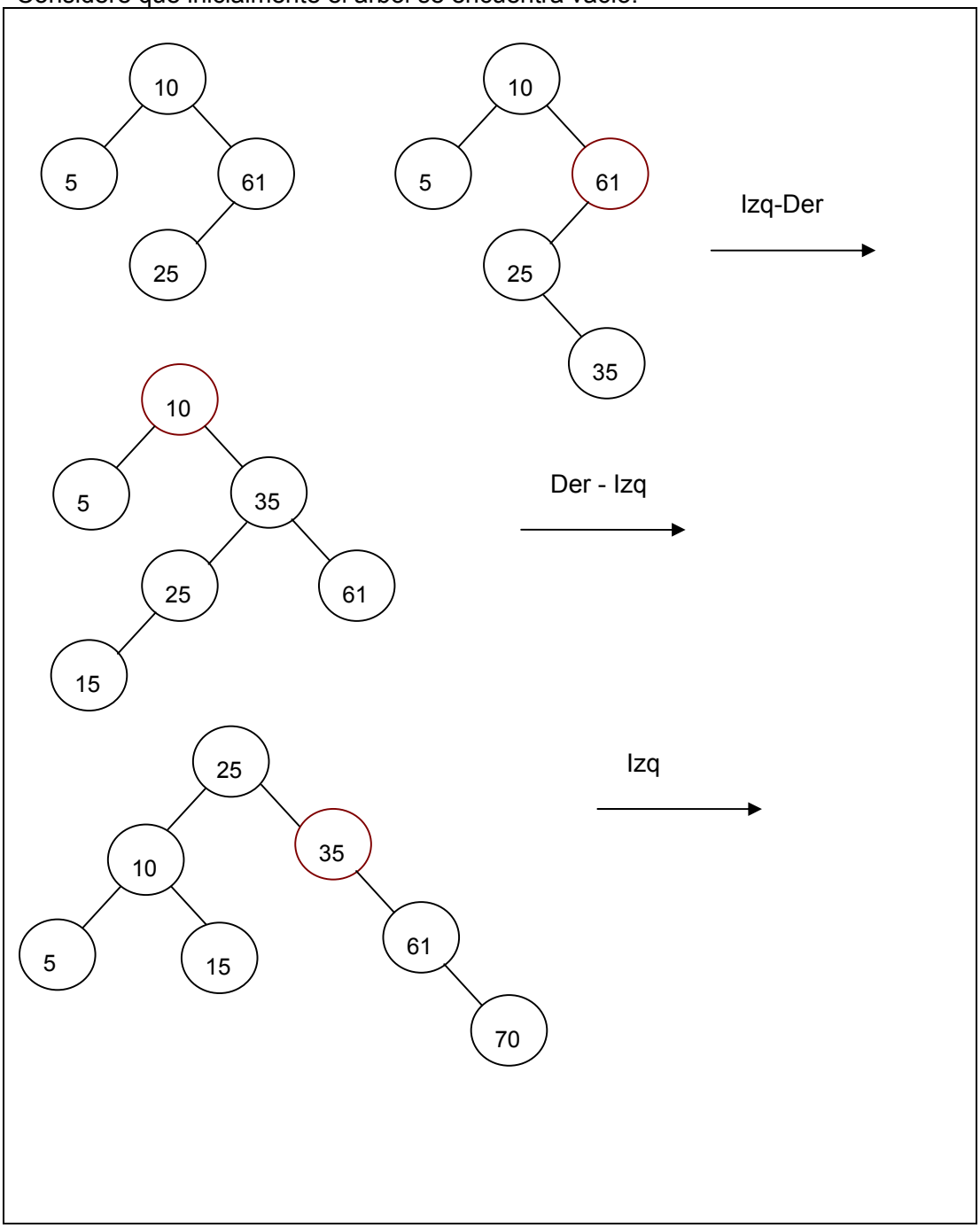
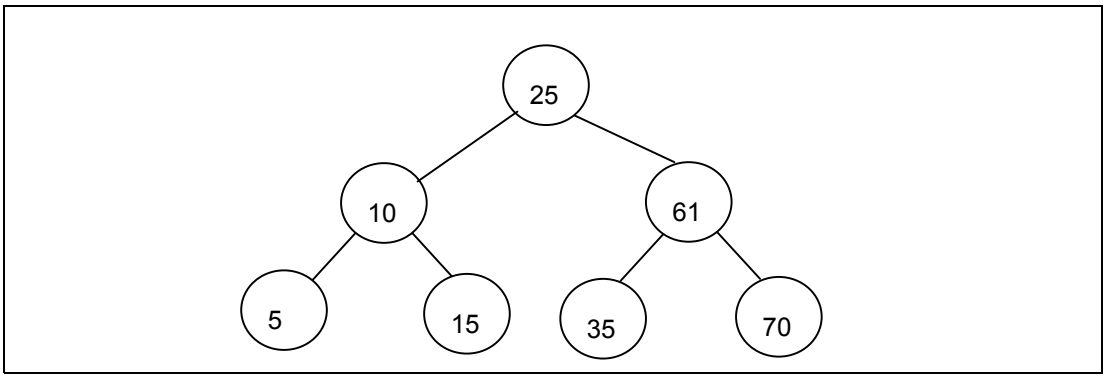
Primer punto (40 puntos)

a) (20 puntos) Dibuje paso a paso (incluyendo rotaciones) el árbol **AVL** que se obtiene al realizar las siguientes inserciones:

10, 5, 61, 25, 35, 15, 70

Considere que inicialmente el árbol se encuentra vacío.

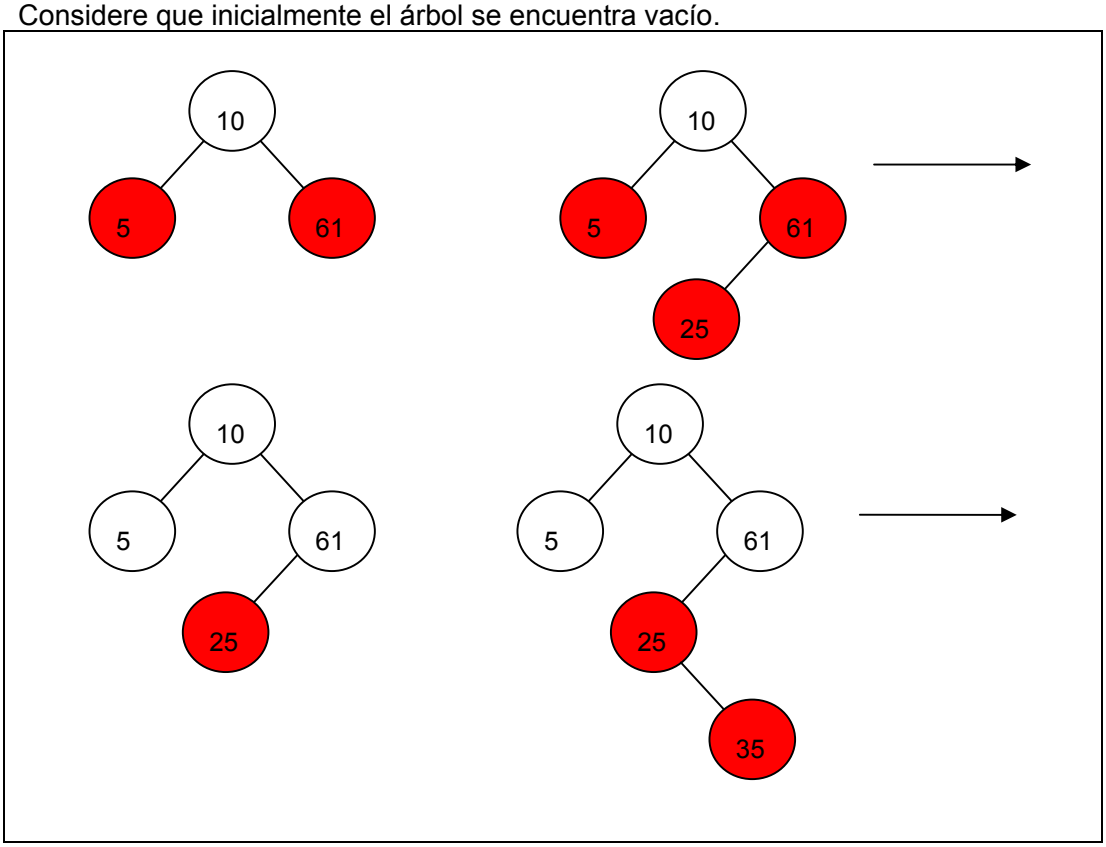




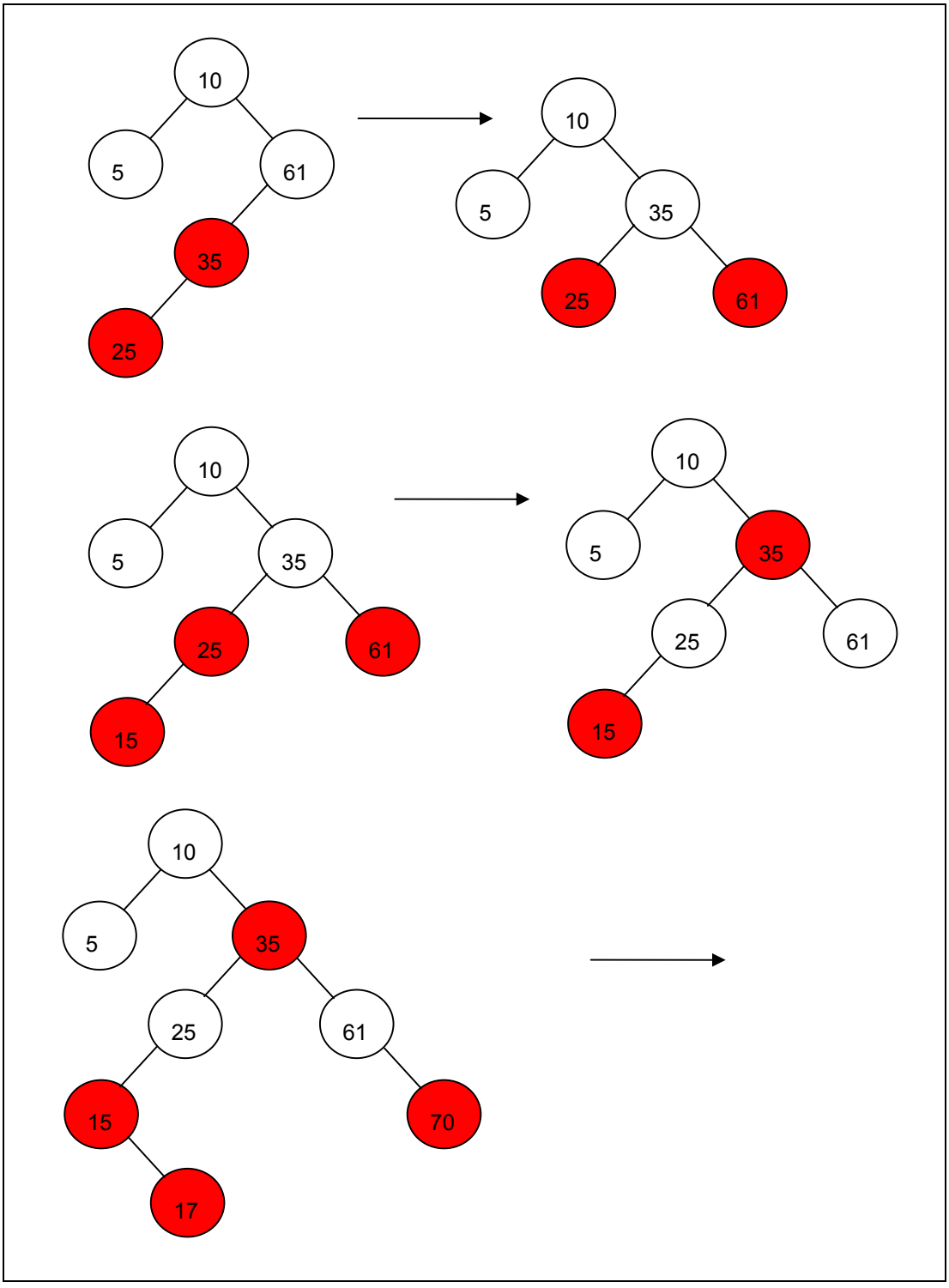
b) (20 puntos) Dibuje paso a paso (incluyendo cambios de color y rotaciones) el árbol **Rojo-Negro** que se obtiene al realizar las siguientes inserciones:

10, 5, 61, 25, 35, 15, 70, 17, 30

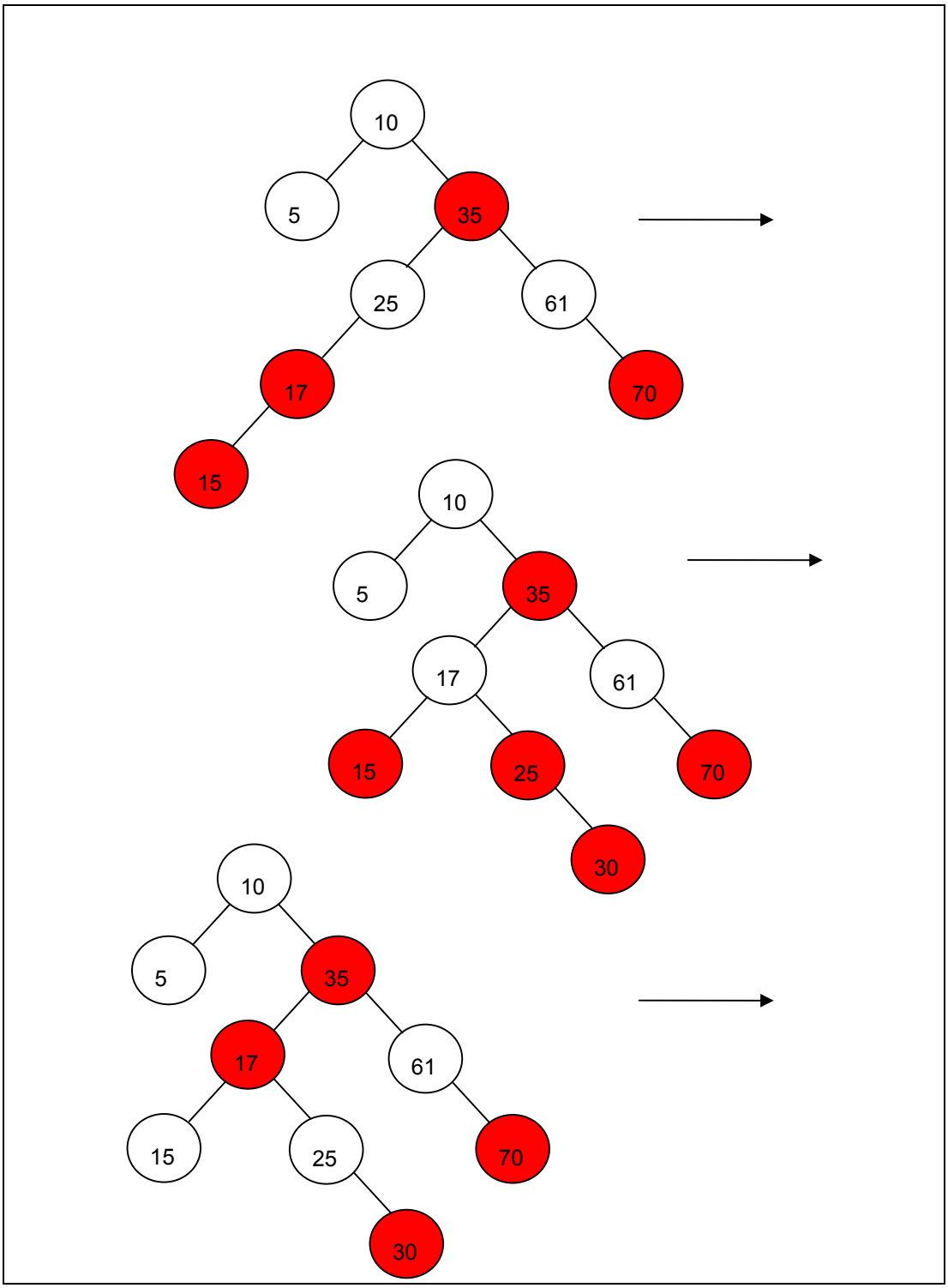
Considere que inicialmente el árbol se encuentra vacío.

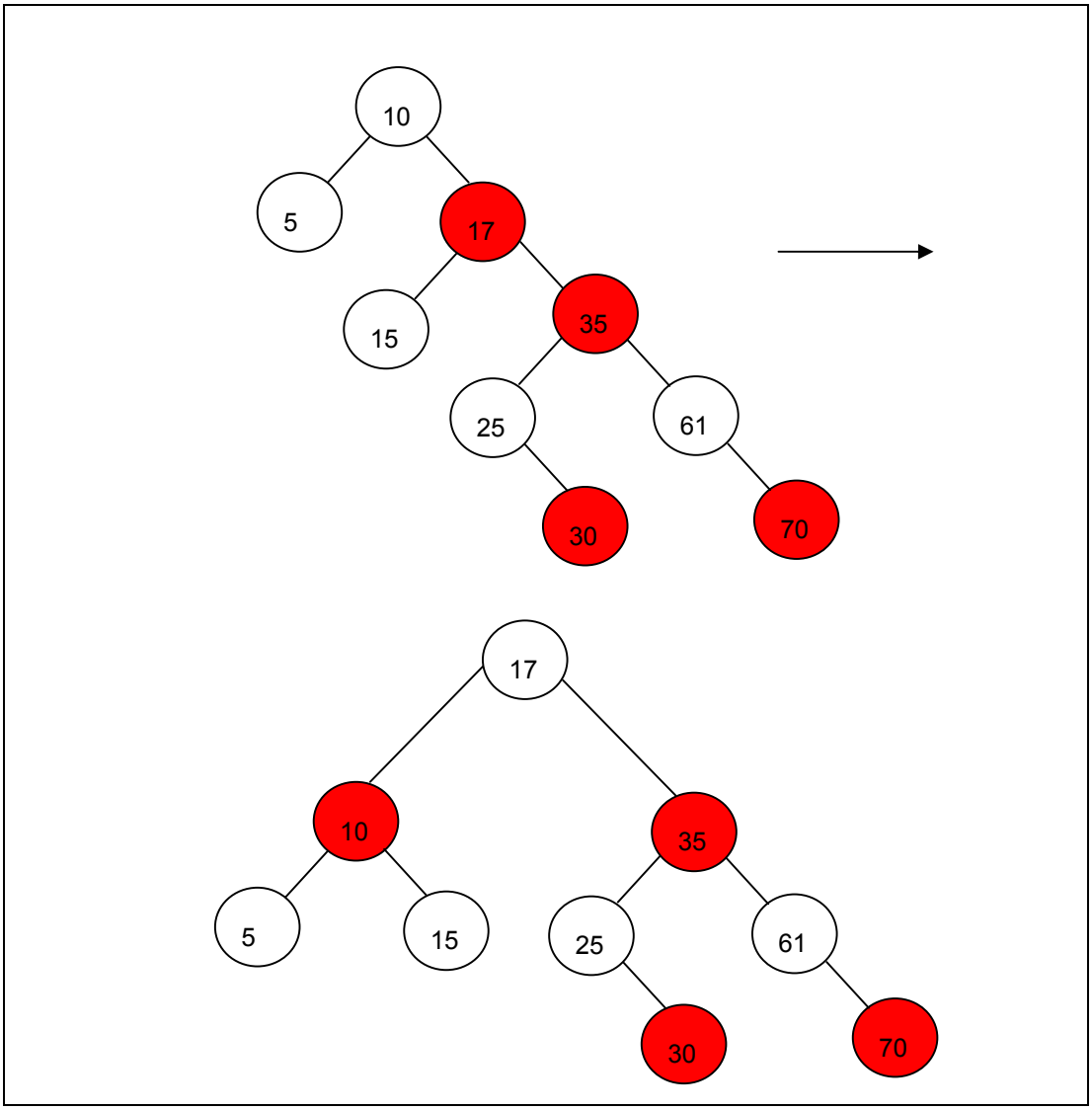


Universidad del Cauca - FIET
Ingeniería de Sistemas
Estructuras de Datos II
Segundo examen parcial
Agosto de 2007



Universidad del Cauca - FIET
Ingeniería de Sistemas
Estructuras de Datos II
Segundo examen parcial
Agosto de 2007



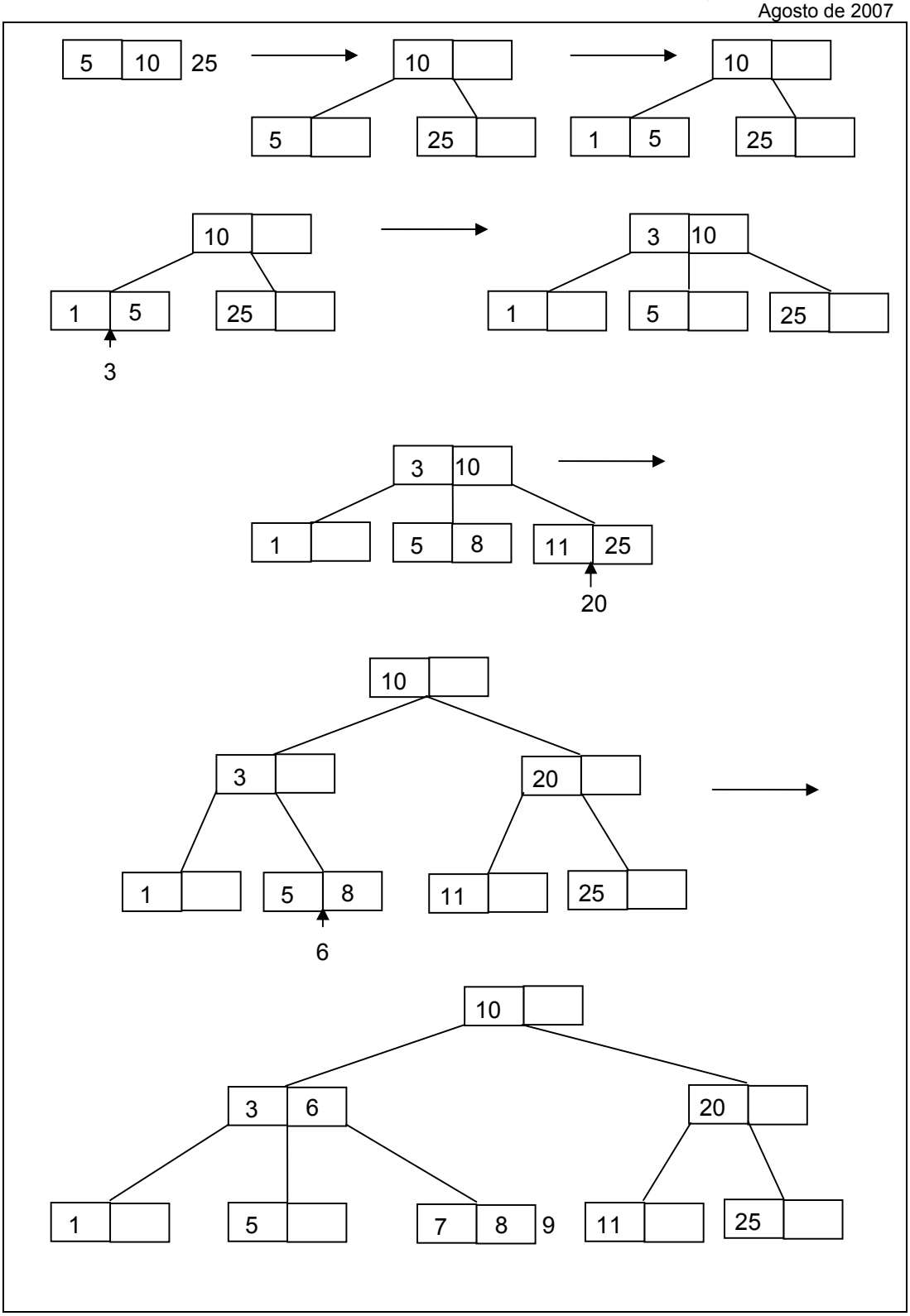


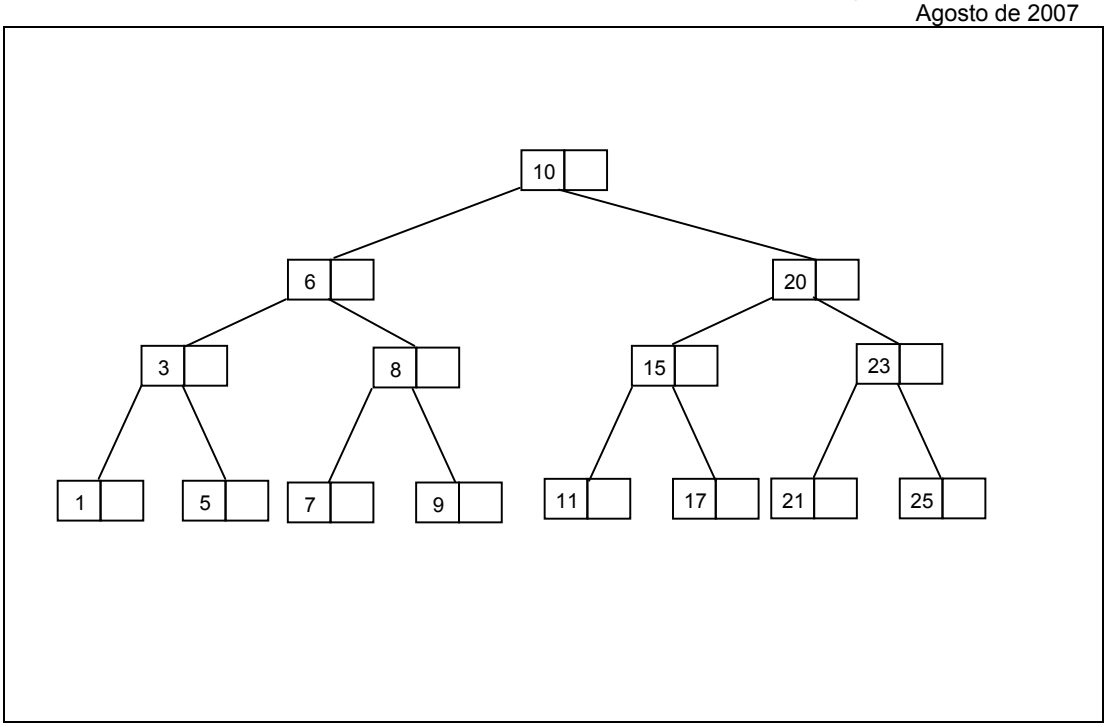
Segundo punto (60 puntos)

a) (30 puntos) Dibuje paso a paso el **árbol B de orden 3** que se obtiene al realizar las siguientes inserciones:

10, 5, 25, 1, 3, 8, 11, 20, 6, 7, 9, 15, 21, 23, 17

Considere que inicialmente el árbol se encuentra vacío.


5 10 25
10
5 25
10
1 5 25
10
1 5 25
3
3 10
1 5 25
3 10
1
5 8
11 25
20
10
3 20
1 5 8 11 25
6
10
3 6 20
1 5 7 8 9 11 25



b) (30 puntos) Desarrolle un algoritmo para buscar un dato en un árbol B de orden n, suponiendo que los datos dentro de un nodo se almacenan en una lista simple enlazada.

```
/* Buscar . Funcion para buscar un elemento en un arbol B.
  Parametros:
  a: Referencia a la raiz del arbol (nodo b)
  dato: Dato a buscar en el arbol */
buscar(a, dato) {
     baux = a /* Apuntador a la raiz del arbol (nodo b) */
     ant = NULL /* Apuntador a la lista enlazada dentro del nodo b*/
     encontrado=FALSE
     mientras baux != NULL and encontrado == FALSE
           /* Sacar el inicio de la lista de datos del nodo b*/
           aux = baux->lista_datos
           ant = NULL
           /* Comparar el dato con el primero de la lista */
           si dato == aux->dato
                 encontrado=TRUE
           sino si dato < aux->dato  /* Bajar al hijo izquierdo */
                    baux = aux->izq
           sino /* Recorrer la lista de datos */
                 ant = NULL
                 mientras aux != NULL and aux->dato < dato
                       ant = aux
                       aux = aux->sig
                 fin_mientras
```

```
                si aux != NULL /* Se puede encontrar
                          en este nodo b */
                     si dato == aux->dato /*Existe en este nodo? */
                        encontrado=TRUE
                     sino
                        baux = aux->izq
                     fin_si
                sino /* No se encuentra en la lista de datos,
                        bajar al hijo derecho */
                   si dato > ant->dato
                      baux = ant->der
                   fin_si
                fin_si
           fin_si
     fin_mientras
    retornar encontrado
fin_algoritmo
```