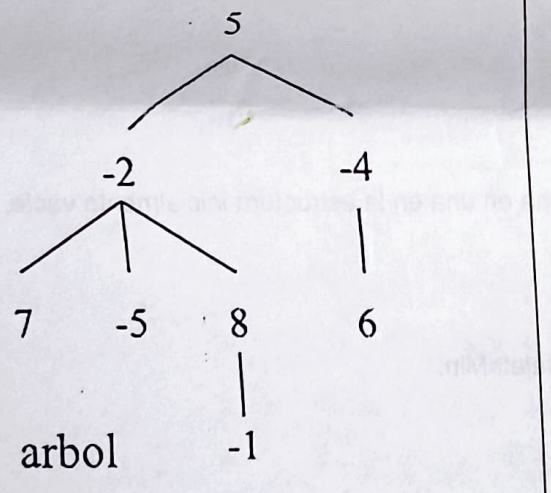
AyED 2025 Árboles y Heap - 1era fecha - 10/05/2025 Tema 2

Apellido	Nombre	Legajo	Corrigió
DOBAL	Federico	DNI=47222936	Jorge
Ejercicio 1: 5	**Ejercicio 2:** 1,5	**Ejercicio 3:** 3,5	**Total:** 10

Ejercicio 1 (5 puntos).
Defina una clase **ParcialArboles** que contenga el siguiente método:

```
public static List<Integer> caminoSignoAlternante(GeneralTree<Integer> arbol)
```

que recibe un árbol general de valores enteros y devuelve una lista con los valores de los nodos del **camino de mayor costo** (suma total de los valores) desde la raíz hasta una hoja, tal que los valores de los nodos **alternen** entre positivos y negativos en cada paso. Esto significa que cada nodo del camino debe tener signo opuesto al de su padre (por ejemplo, si un nodo tiene valor positivo, su hijo inmediato debe tener valor negativo y viceversa).
Si existen varios caminos con el mismo costo máximo que cumplen con la alternancia de signos, se debe devolver el **primer camino** encontrado en un recorrido de izquierda a derecha.
En caso de no existir ningún camino desde la raíz hasta una hoja que cumpla con esta condición, el método debe devolver una lista vacía.
Nota: se considera al número 0 como un valor positivo a efectos de la alternancia.
Ejemplo



1. 5 → -2 → 7
   + → - → + ✓ Alternancia válida. Costo = 10
2. 5 → -2 → 8 → -1
   + → - → + → - ✓ Alternancia válida. Costo = 10
3. 5 → -4 → 6
   + → - → + ✓ Alternancia válida. Costo = 7
4. 5 → -2 → -5
   + → - → - ✗ (No alterna positivo/negativo)

- Los caminos 1, 2 y 3 cumplen con la alternancia entre positivo y negativo.
- Los caminos 1 y 2 tienen el mismo costo máximo
- En un recorrido de izquierda a derecha, el camino 5 → -2 → 7 es el primero que se encuentra y el que tiene que retornar el método.

Observación importante:

- La alternancia puede comenzar desde cualquier valor en la raíz, ya sea positivo o negativo. Lo importante es que a partir del segundo nodo (el primer hijo), los valores deben alternar la condición respecto al valor del nodo padre.
  - Por ejemplo, si la raíz del árbol fuera -5 (negativo), un hijo con valor 2 (positivo) permitiría un inicio de camino alternante válido.

Tenga en cuenta que:
1. No puede agregar variables de instancia ni de clase a la clase **ParcialArboles** .
2. Debe respetar la clase y la firma del método indicado.
3. Puede definir todos los métodos y variables locales que considere necesarios.
4. Todo método que no esté definido en la sinopsis de clases debe ser implementado.

Ejercicio 2 (1.5 puntos).

a. ¿Cuál es la expresión infija de la siguiente expresión postfija: ABC*+AB+C*/?:

(a) (A*B+C)/(A+B*C)
(b) (A+B*C)/((A+B)*C)
(c) (A+B*C)/(A+B*C)
(d) (A*B+C)/(A+B)*C

b.- Construya una Min-Heap a partir de la siguiente secuencia, usando el algoritmo BuildHeap, **10 20 8 3 2 7 24 35**, ~~¿cómo queda el recorrido preorden de la heap?~~ ¿Cómo queda la Min-Heap?

(a) 2 3 7 20 8 10 24 35
(b) 2 3 7 10 20 8 24 35
(c) 2 3 7 20 10 8 24 35
(d) 2 3 7 8 20 10 24 35

c.- Considere las siguientes sentencias:

(i) La altura de un árbol general es el máximo nivel de alguna de sus hojas
(ii) Un árbol binario lleno tiene hojas en más de un nivel
(iii) Un árbol binario completo puede almacenarse en un arreglo

¿Cuál de las opciones es correcta?

(a) (i) y (ii)
(b) Sólo (i)
(c) Sólo (ii)
(d) (i) y (iii)
(e) (ii) y (iii)

Ejercicio 3 (3.5 puntos).

a.- *Construya una* ***Min-Heap*** *con las siguientes claves insertándolas de una en una en la estructura inicialmente v*
muestre cada uno de los pasos seguidos para construirla

150 100 20 16 70 85 140 54 139

b.- En la Min-Heap, construida anteriormente, ejecute una operación de DeleteMin.

DNI 47222436 | Federico Dobal | HOJA Nº 1/2 | FECHA 10/5/25

1)

```
Public class ParcialArboles{
    Private GeneralTree<Integer> tree;
    Public static List<Integer> caminoSignoAlternante(GeneralTree
                                                      <Integer>
                                                      arbol) {
        List<Integer> camino = new LinkedList<>();
        if(arbol != null && !arbol.isEmpty())
            caminoHelper(arbol, camino, new LinkedList<>(),
                         INTEGER.MIN_VALUE, 0);
        return camino;
    }

    Private static int caminoHelper(GeneralTree<Integer> nodo, List<Integer>
                                    caminoMax, List<Integer> caminoAct,
                                    int sumaCaminoMax, int sumaCaminoAct){

        SumaCaminoAct += nodo.getData();
        CaminoAct.add(nodo.getData);
        if(nodo.isLeaf()){
            if (sumaCaminoAct > sumaCaminoMax){
                SumaCaminoMax = sumaCaminoAct;
                caminoMax.clear();
                caminoMax.addAll(caminoAct);
            }
        } else{
            For(GeneralTree<Integer> hijo: nodo.getChildren()){
                if( (hijo.getData() <0 && nodo.getData()>=0)||
                    (hijo.getData()>=0 && nodo.getData()<0)){
                    SumaCaminoMax = CaminoHelper(hijo, caminoMax,
                                    caminoAct, SumaCaminoMax,
                                    SumaCaminoAct);
        }}
    }
```

NOTA

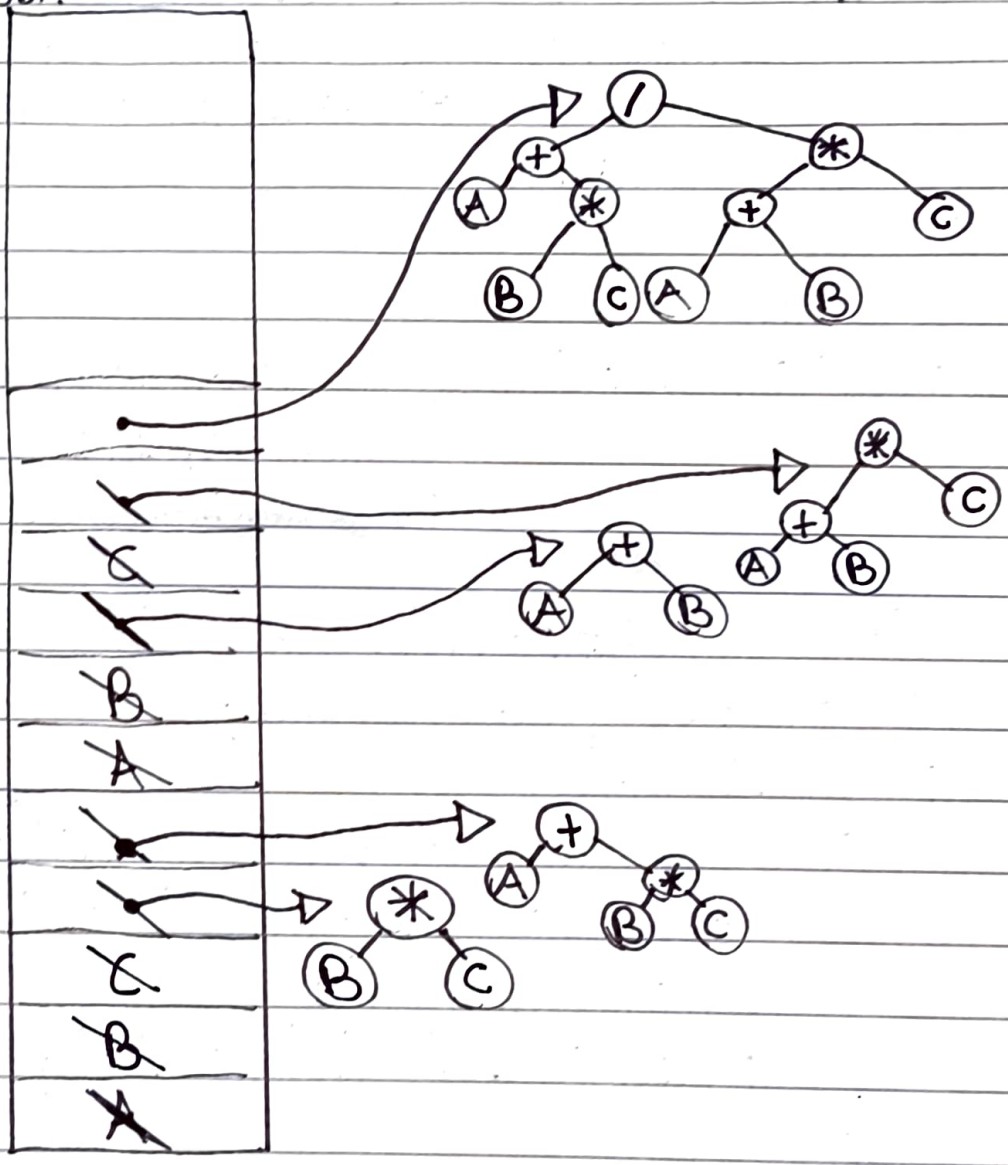
```
        CaminoAct.removeLast();
        return sumaCaminoMax;
    }
}
```

2)

a) Exp = ABC * + AB + C * /

Construyo el árbol:

Pila =

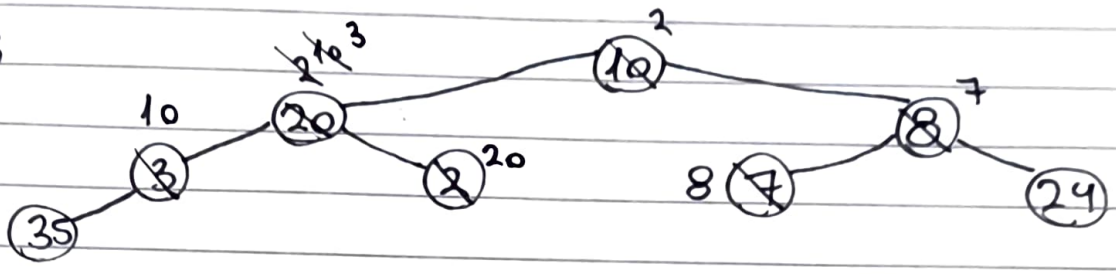


Escribo la exp. del árbol: (A + (B * C)) / ((A + B) * C)

Respuesta: (b)

b) Exp. = 10 20 8 3 2 7 24 35

Tamaño = 8



i = (Tamaño / 2) = 4
Exp[i] = 3
Exp[i-1] = 8
Exp[i-1] = 20
Exp[i-1] = 10

Recorrido preorden:
2 3 10 35 20 7 8 24

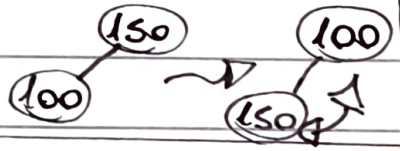
Respuesta: (b)

c) (d)

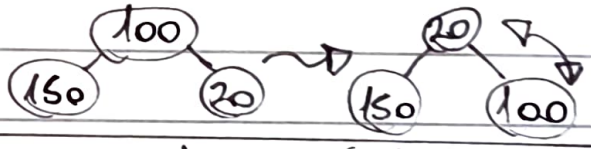
3)

a) Insert(150):

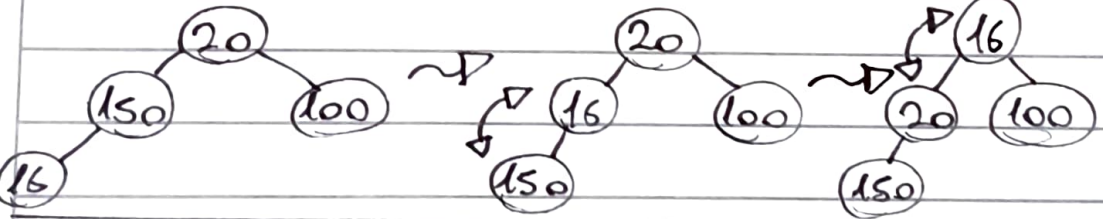
Insert(100):



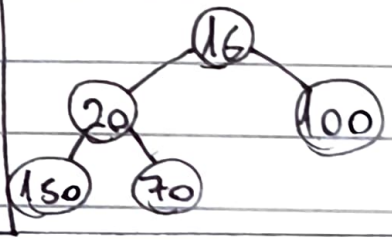
Insert(20):



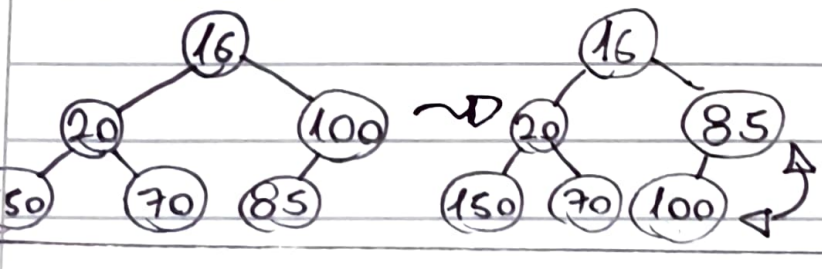
Insert(16):



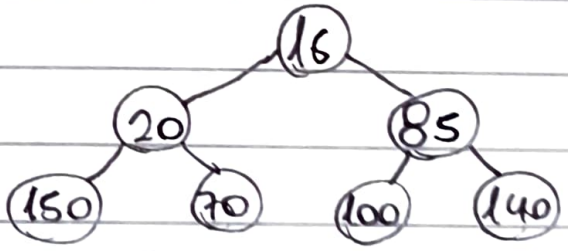
Insert(70):



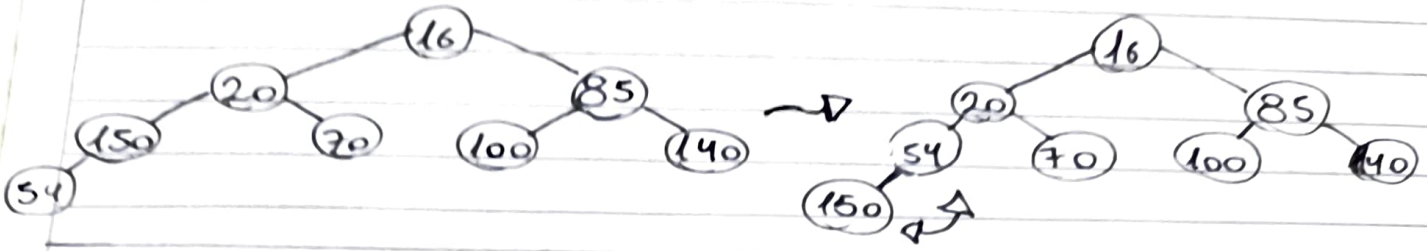
Insert(85):



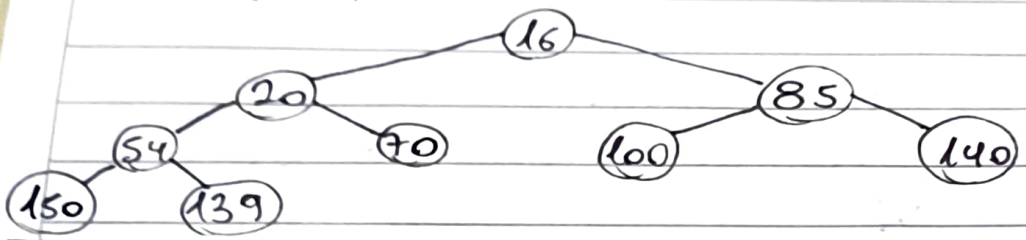
Insert(140):



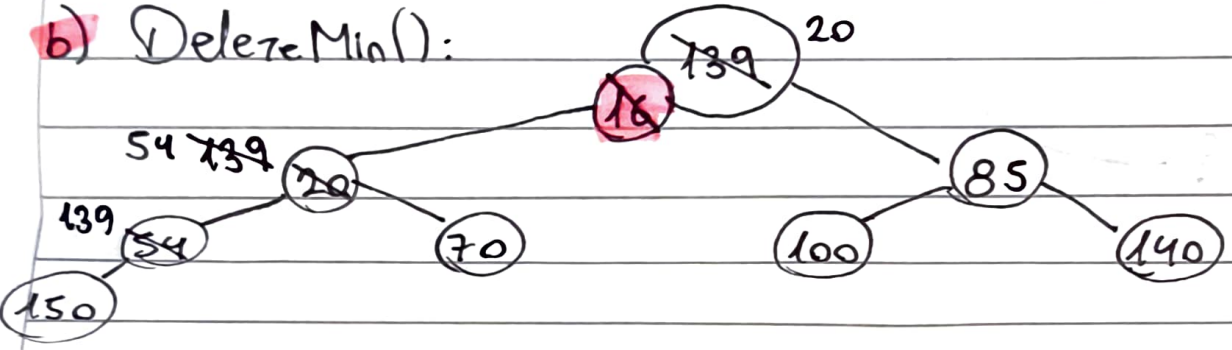
Insert (54):



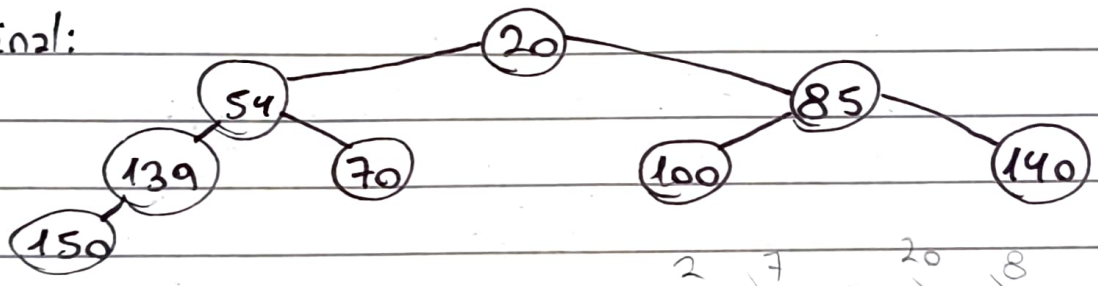
Insert (139):



b) DeleteMin():



MinHeap Final:



10 20 8 3 2 7 24 35