Árbol de costo mínimo

Le llamaremos **costo** de un árbol con valores enteros en los nodos, a la suma de los costos de sus nodos

$$Costo(Arbol) = \sum Costo(Nodo)$$

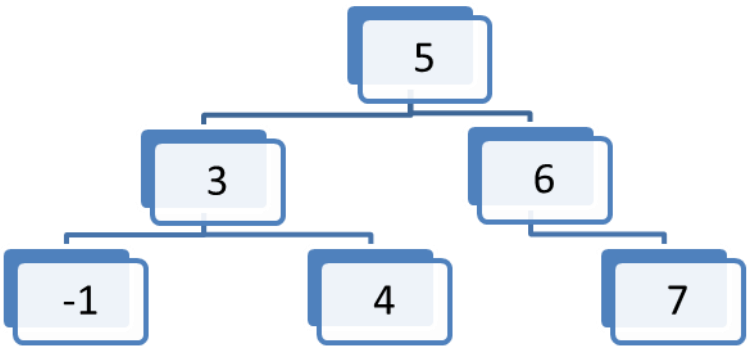
siendo el costo de un nodo el producto del valor entero del nodo multiplicado por 2^k donde k es la profundidad del nodo.

$$Costo(Nodo) = Nodo.Valor * 2^{Nodo.Profundidad}$$

$$Costo(Arbol) = \sum Nodo.Valor * 2^{Nodo.Profundidad} \mid para\ cada\ nodo\ del\ árbol$$

Consideramos que la raíz tiene profundidad cero.

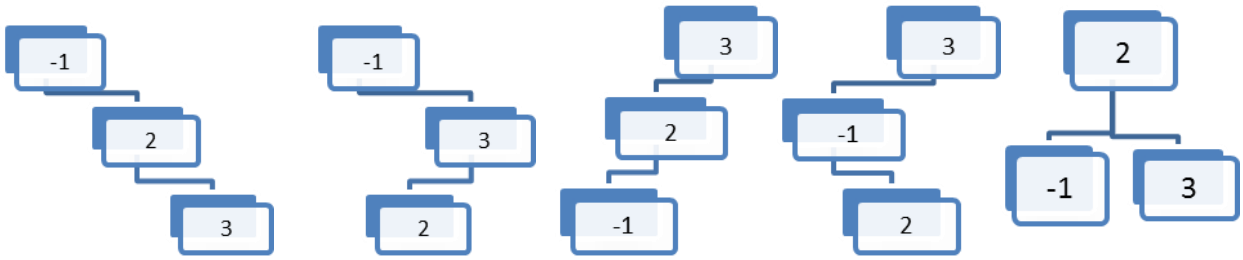
Ejemplo:



$$Costo = 5 * 2^0 + 3 * 2^1 + 6 * 2^1 + -1 * 2^2 + 4 * 2^2 + 7 * 2^2 = 63$$

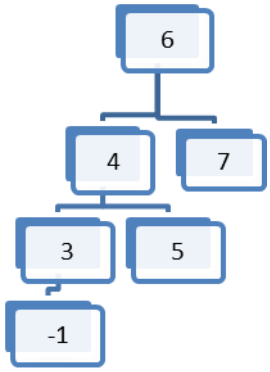
El problema a programar consiste en que a partir de un conjunto de números enteros, usted deberá construir un árbol binario ordenado con esos enteros como valores de los nodos y que el árbol resultante sea el de menor costo posible.

Por ejemplo, a partir del conjunto $\{2, -1, 3\}$se pueden formar los siguientes árboles binarios ordenados:



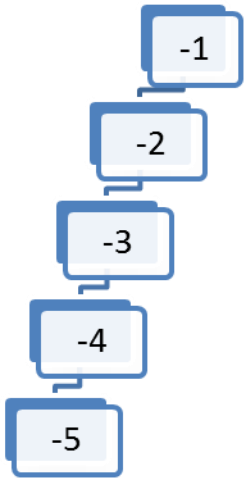
Los cuales tienen los costos 15, 13, 3, 5 y 6 respectivamente, por lo que el árbol binario ordenado de costo mínimo sería el tercero.

Para el conjunto del primer ejemplo, $\{-1, 3, 4, 5, 6, 7\}$, el árbol binario de costo mínimo es:



Con costo $c = 6 * 2^0 + 4 * 2^1 + 7 * 2^1 + 3 * 2^2 + 5 * 2^2 + -1 * 2^3 = 52$

Para $\{-1, -2, -3, -4, -5\}$ el árbol binario de costo mínimo es:



Con costo $c = -1 * 2^0 + -2 * 2^1 + -3 * 2^2 + -4 * 2^3 + -5 * 2^4 = -129$

Usted deberá implementar el método:

```
public static ArbolBinarioOrdenado<int> ConstruirArbolCostoMin(int[] elementos)
```

que retornará el árbol con menor costo que se puede formar con los números que contenga el parámetro *elementos.*

NOTA*:* Se garantiza que en *elementos* no habrá números repetidos, no estará vacío y no será *null* .

La clase `ArbolBinarioOrdenado<T>` está en la *dll* Weboo.Arboles.dll dentro del namespace Weboo.Arboles. Tanto en la plantilla de solución, como en el probador, usted puede referirse a la clase por su nombre.

```
public class ArbolBinarioOrdenado<T> where T :IComparable<T>
    {
        /// <summary>
        /// Constructor del árbol binario ordenado.
        /// </summary>
        /// <param name="valor"> Valor que se almacena en la raíz</param>
        /// <param name="hijoDerecho">Hijo derecho de la raíz del árbol</param>
        /// <param name="hijoIzquierdo">Hijo izquierdo de la raíz del árbol</param>
        public ArbolBinarioOrdenado(T valor,ArbolBinarioOrdenado<T> hijoDerecho,ArbolBinarioOrdenado<T>
hijoIzquierdo);

        /// <summary>
        /// Hijo derecho de la raíz del árbol
        /// </summary>
        public ArbolBinarioOrdenado<T> HijoDerecho { get;protected set; }

        /// <summary>
        /// Hijo izquierdo de la raíz del árbol
        /// </summary>
        public ArbolBinarioOrdenado<T> HijoIzquierdo { get; protected set; }

        /// <summary>
        /// Valor almacenado en la raíz del árbol
        /// </summary>
        public T Valor { get;protected set; }

        /// <summary>
        /// Recorrido en PreOrden del árbol
        /// </summary>
        /// <returns> IEnumerable con los elementos del recorrido en preorden </returns>
        public IEnumerable<T> PreOrden();

        /// <summary>
        /// Recorrido en PostOrden del árbol
        /// </summary>
        /// <returns>IEnumerable con los elementos del recorrido en postorden</returns>
        public IEnumerable<T> PostOrden();

        /// <summary>
        /// Recorrido en EntreOrden
        /// </summary>
        /// <returns>IEnumerable con los elementos del recorrido en entreorden</returns>
        public IEnumerable<T> EntreOrden();

    }
```