Facultad Politécnica – Ingeniería Informática
Algoritmos y Estructura de Datos III – 2025 – 2do.Periodo – Sección TQ/TR

Tarea 3-U3 (45p)

Trabajo a entregar por GRUPO vía EDUCA. Indique claramente los integrantes de su grupo. El código debe poder probarse sin usar ningún IDE desde la línea de comandos. No incluya ninguna clase en package. Debe poder compilarse y ejecutarse sin ningún cambio.

Considere el archivo BST.java que es una implementación sencilla de BST (El mismo utilizado en la práctica de laboratorio).

Cada ejercicio contiene el nombre de archivo que debe respetar. Deben indicar en cada archivo: Código de grupo, Integrantes y el ejercicio al que corresponde. Debe incluir el archivo de código de honor, siempre, indicando la tarea que están presentando. Todos los comentarios colocar en el código fuente.

Recuerde que se entrega en un archivo comprimido con los archivos esperados. Solo incluir los fuentes (.java)

Ejercicio 1 (30p) `BST1.java`

Resolver el siguiente ejercicio que consta de varias partes. Incluya las pruebas necesarias para cada uno de los ítems. Por cada uno de los ítems indique el costo asintótico, además incluir la documentación para su generación con javadoc.

Convierta su código a Generic, de forma que ahora pueda asegurar que los datos guardados en el BST sean de un solo tipo que implemente la interface *Comparable*. Considere que no deben existir valores duplicados (si se intenta agregar uno simplemente ignorar). Se asume que debe incluir `agregar(T valor)`. Extienda las funcionalidades que incluya:

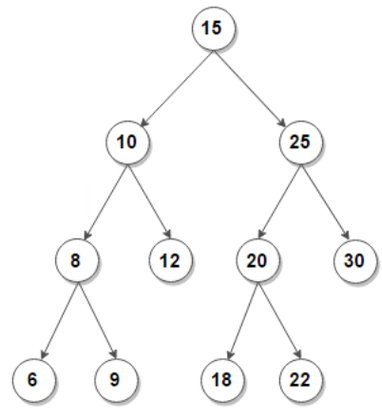
```
a) implementación con iterador para usarse en el for-each
b) void eliminar( T valor ) //eliminar valor del árbol
c) T sucesor ( T valor )     //retorna el sucesor de valor o null si no existe
d) T predecesor ( T valor ) //retorna el predecesor de valor o null si no existe
e) boolean esCompleto()     //retorna true o false si el árbol es completo o no
f) boolean esLleno()         //retorna true o false si el árbol es lleno o no
g) void imprimirPorNiveles() // imprime el árbol por niveles. Una línea por nivel y
   separados adecuadamente como el ejemplo de abajo:
```

```
          12
         /  \
        9    15
       / \     \
      8   10    16
```

Rúbrica	**Puntaje**
Realización de modificaciones para uso de Generic	1
Implementación de cada item (2p por cada uno)	14
Especificación del costo asintótico por cada ítem	7
Datos de pruebas de funcionamiento	6
Documentación para javadoc	2

Ejercicio 2 (5p) `BST2.java`

Dado un BST (con la cantidad de nodos n>1), escriba un algoritmo en Java, de coste temporal lineal y coste espacial constante, que cuente todos los subárboles no vacíos cuyas claves están entre un rango [a,b] inclusive. Por ejemplo, considere el BST de la derecha: Si se solicita la cantidad de subárboles no vacíos en el siguiente rango [5,20] la cantidad de subárboles es 6. Las raíces de los subárboles son las claves 6, 9, 8, 10, 12, 18. Nótese que aunque el nodo con clave 20 se encuentra dentro del rango no se tiene en cuenta debido a que 22 es una clave que conforma el subárbol con raíz 20, por lo que el subárbol completo no se puede considerar.



El método a implementar debe ser estático en la clase BST pero que sea genérico (como el ejercicio 1). Finalmente, incluya en el método main la construcción de al menos dos árboles BST diferentes y probar cada uno de ellos con al menos tres rangos válidos.

Calcule la T(n) de su algoritmo en el peor caso y su coste asintótico temporal en término de O grande. Fundamente.

Rúbrica	**Puntaje**
Implementación correcta	2
Datos de prueba de acuerdo a lo solicitado (en el método main)	1
Análisis de rendimiento encontrando T(n) y O.	2

Ejercicio 3 (10p) `ArbolBinario.java`
Implemente la función de búsqueda en un árbol binario (no precisamente de búsqueda (BST)) sin hacer uso de recursión ni estructuras auxiliares como pilas y colas. Demuestre el costo asintótico de su algoritmo. Su función recibe como parámetro la clave a buscar, y retorna el dato asociado a la clave buscada (ver abajo, el método buscar). Para ello cree un contenedor llamado `ArbolBinario` cuya definición se presenta abajo.

Incluya las pruebas de funcionamiento necesarias al menos con 20 nodos.

La clase `ArbolBinario` se definiría:

```
public class ArbolBinario<K,D> {

	private class Nodo {
		public K key;
		public D dato,
		public Nodo izq, der;
	}

	private Nodo raiz;

	ArbolBinario() {
		raiz = null;
	}

	void agregar (K clave, D dato ) {
		…
	}
	D buscar ( K clave ) {
		…
	}
}
```

Rúbrica	**Puntaje**
Implementación correcta de la clase de acuerdo a las especificaciones	5
Pruebas de funcionamiento (en el método main)	2
Demostración de costo asintótico de la búsqueda	3