2° RECUPERATORIO DE ALGORITMICA Y PROGRAMACION II - 2021

1) Agregar a la clase **ArrayList.java** el siguiente método:

```
/**
 * Inserta todos los elementos de la lista especificada en la lista actual en la
 * posición indicada. Desplaza el elemento actual a la posición indicada (si
 * hay) y todos los elementos siguientes a la derecha (incrementando sus
 * índices). Los nuevos elementos aparecerán en la lista en el orden en que
 * están en la lista pasada como argumento.
 *
 * @param index índice en el cual se inserta el primer elemento desde la lista
 *              especificada
 * @param list  lista que contiene los elementos a ser adicionados
 * @throws IndexOutOfBoundsException lanza la excepción si el índice esta fuera de rango
 * (index < 0 || index > size)
 */
/**
 * Ejemplo:
 *
 * Dada la lista: [A, B, C, D, E] y la lista: list = [1, 2]
 *
 * addList(2,list)
 *
 * Retorna la lista: [A, B, 1, 2, C, D, E]
 *
 */
void addAll(int index, List<E> list)
```

Realizar un programa que llame a ***addList*** con distintos parámetros probando las distintas condiciones que se pueden presentar y atrapar sus excepciones.

Nota: se evaluará la eficiencia del algoritmo. Por ejemplo no es válido llamar al método ***add*** ya que la solución tendría una complejidad de $O(n^2)$

2) Agregar a la clase **LinkedPositionalList.java** el siguiente método:

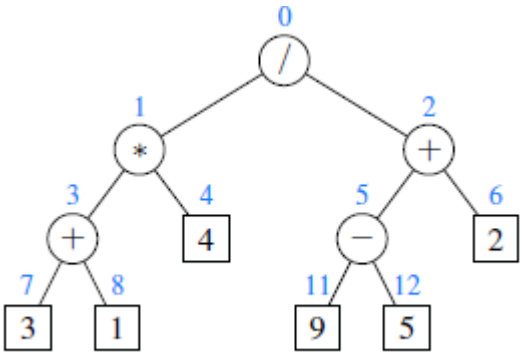
```
/**
 * Intercambia todos los elementos de la lista sin modificar sus posiciones
 *
 */
/**
 * Ejemplo:
 *
 * Dada la lista: [H, O, L, A]
 *
 * reverse()
 *
 * Se obtiene la lista: [A, L, O, H]
 *
 * Nota: no se modifican las posiciones. En este ejemplo, en la primer posición
 * está el elemento H después de llamar a reverse() en la misma posición esta el
 * elemento A
 */
void reverse();
```

Realizar un programa que llame a ***reverse*** con distintas listas (lista con cantidad de elementos pares, cantidad de elementos impares, lista vacía, etc.) y mostrar los resultados. Se evaluará la eficiencia del algoritmo.

3) Agregar a la clase **LinkedBinaryTree.java** el siguiente método:

```
/**
 * Retorna una lista que contiene la representación de un árbol binario.
 *
 * @return lista que contiene la representación de un árbol binario.
 */
public List<E> ListBinaryTree()
```

Por ejemplo, dado el siguiente árbol:



El método retorna una lista con los siguientes elementos:

/	*	+	+	4	−	2	3	1	NULL	NULL	9	5	NULL	NULL
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

Para su desarrollo considere la posibilidad de utilizar una lista auxiliar para almacenar las referencia a los nodos (`List<Node<E>>`) inicializado en null con el tamaño de un árbol binario lleno.

Árbol binario lleno: se dice que un árbol binario está lleno si es un árbol binario de altura k que tiene 2^{k+1} - 1 nodos. (Para esta definición consideramos que la altura de la raíz se considera cero)

Por ejemplo si la altura es 3, la cantidad de nodos es 2^4-1 = 15

Dado el padre es fácil determinar la posición que ocupará el hijo dentro de la lista:

```
Hijo izquierdo: 2 * posición del padre + 1
Hijo derecho: 2 * posición del padre + 2
```

IMPORTANTE:

1. Los enunciados no deberían dejar lugar a dudas de los ejercicios a resolver y preguntas a responder. De todas maneras si surge alguna consulta del enunciado enviar la misma al **Foro** que está en la **sección Evaluación**. Las preguntas serán respondidas dentro de los 30 minutos de realizadas.

2. Enviar un WhatsApp al grupo **solo** si tienen algún inconveniente (no pueden entrar al foro, no reciben la respuesta en el tiempo indicado, no pueden subir las soluciones, etc.)

3. Para la parte práctica del parcial, subir solamente los archivos con extensión .java. Entregar un directorio por ejercicio. Al copiar el directorio a Eclipse el proyecto debe funcionar sin cambios. Verificar antes de entregar.