Trabajo práctico 1 - Listas y recursión
Programación 3 - TUDAI

Ejercicio 1.
Implemente los métodos indicados del esqueleto de Lista desarrollado en Teoría (insertFront, extractFront, isEmpty, size, toString). Agregar también el método: T get(index)

Ejercicio 2.
Considerando la implementación de la Lista realizado en el ejercicio anterior, comparar la complejidad computacional contra un array en las siguientes operaciones:
1.- Insertar al principio. Lista O(1) - Array O(n)
2.- Buscar un elemento en una posición. Lista O(n) - Array O(1)
3.- Determinar la cantidad de elementos. Lista si esta implementado un contador de nodos es O(1), sino O(n) - Array O(1)

Ejercicio 3.
Implemente una Pila utilizando la Lista del ejercicio 1. A una pila se le pueden agregar elementos utilizando el método *push(T o)*. Para retirar elementos de la colección se utiliza el método *pop()*, que retorna el último elemento agregado a la colección y lo elimina de la misma. Es posible consultar el tope de la pila (sin eliminarlo) utilizando el método *top()*. Por último, es posible invertir el orden de los elementos de la pila mediante el método *reverse()*.

Ejercicio 4.
A la implementación de la clase Lista realizada en el ejercicio 1, agregue un método indexOf, que reciba un elemento y retorne el índice donde está almacenado ese elemento, o -1 si el elemento no existe en la lista.

Ejercicio 5.
A partir de la clase Lista implementada en el ejercicio 1, implemente el patrón iterator-iterable, para que la lista sea iterable. ¿Existe alguna ventaja computacional a la hora de recorrer la lista de principio a fin si se cuenta con un iterador?
La complejidad de usar iterador es O(n), donde n es el tamaño de la lista; mientras que la de no usarlo es O(n^2)

Ejercicio 6.
Escriba un procedimiento que dadas dos listas construya otra con los elementos comunes, suponiendo que: a) Las listas están desordenadas y la lista resultante debe quedar ordenada. b) Las listas están ordenadas y la lista resultante debe mantenerse ordenada.

Ejercicio 7.
Escriba una función que dadas dos listas construya otra con los elementos que están en la primera pero no en la segunda.

Ejercicio 8.
Considerando la implementación de Lista del ejercicio 1, realice una Lista doblemente vinculada.

Ejercicio 9.
Implemente un algoritmo recursivo que verifique si una cadena de texto es palindroma (capicua).

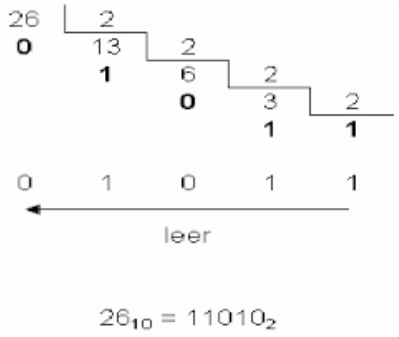
Ejercicio 10.
Implemente un algoritmo recursivo que determine si un arreglo de tamaño N está ordenado.

1. ¿Qué complejidad O tiene? (La complejidad en el peor caso)
2. ¿Trae algún problema hacerlo recursivo? ¿Cuál?
3. ¿Qué cambiaría si la estructura fuera una lista en lugar de un arreglo?

Ejercicio 11.
Implemente un algoritmo recursivo para buscar un elemento en un arreglo **ordenado ascendentemente**.

Ejercicio 12.
Implemente un algoritmo recursivo que convierta un número en notación decimal a su equivalente en notación binaria.

Recordatorio, por ejemplo convertir el 26 a binario:



$26_{10} = 11010_2$

Ejercicio 13.
Implemente un algoritmo recursivo que presente los primeros N términos de la secuencia de Fibonacci.

Por ej. los 6 primeros términos son: 0 1 1 2 3 5