# Laboratorio Nro. 3: LinkedLists y ArrayLists

**3) Simulacro de preguntas de sustentación de Proyectos**

1. Cálculo de complejidad de los ejercicios 1.1, 1.2, 1.3 y 1.4 para ArrayList y LinkedLists.

\*Desarrollo en el código.

|  |  |  |
| --- | --- | --- |
|  | **ArrayList** | **LinkedList** |
| **1.1 (multiply)** | O(n) | O(n) |
| **1.2 (smartInsert)** | O(n) | O(n) |
| **1.3 (wherePivot)** | O(n) | O(n) |
| **1.4 (ejercicio4)** | O(m\*m+m\*n) | O(m\*n) |

1. Explicación de 2.1 y 2.2

**Explicación 2.1**

**Explicación 2.2**

\*leer el problema en <http://codeforces.com/problemset/problem/313/B>

Primero se intentó que cada vez que se solicitara una operación tuviera que calcularse con un ciclo que revisara si dos caracteres adyacentes coincidían. Sin embargo, esta solución era muy lenta, con una complejidad de O(n), así que hubo que intentar resolver el problema de otro modo. Así, se decidió hacer otro arreglo que guardara la cantidad de veces que se cumplía Si = Si+1 desde la posición inicial (0) hasta el tamaño del string (n), arreglo al que llamamos acum. Lo cual hizo que las preparaciones fueran O(n), pero que la complejidad por cada query fuera O(1) y solo requiriera hacer acum[right]-acum[left].

1. Cálculo de complejidades 2.1 y 2.2

\*En el código

**2.1** O(n) -> El procesamiento de cada línea.

**2.2** O(n) -> preparativos. O(1) -> consultas.

1. ¿Qué es m y qué es n en el numeral 3.3?

**2.1** n corresponde a la cantidad de caracteres en la línea.

**2.2** (en los preparativos) n corresponde al tamaño del arreglo.

**4) Simulacro de Parcial**

1. C
2. C
   1. q.size() > 1
   2. <=
   3. q.remove()
   4. q.remove()