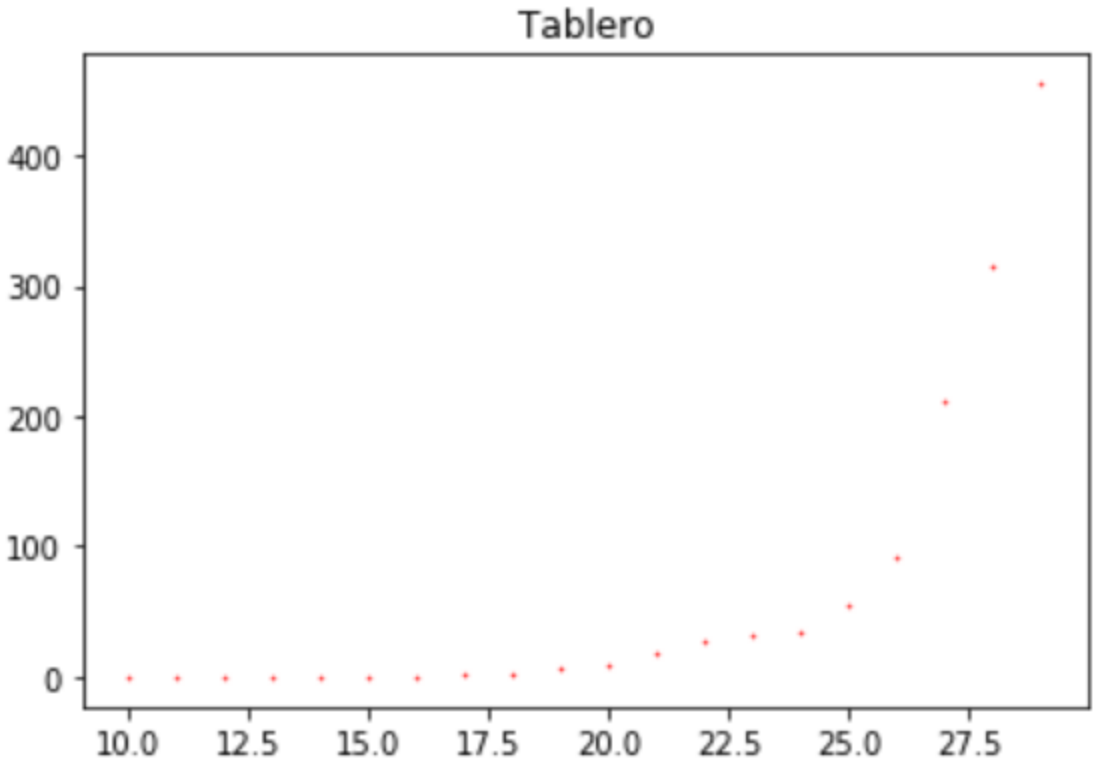
Informe de laboratorio 01.

3.1 La complejidad asintótica del ejercicio 1.2 es O(2^n)

3.2



Evidentemente el tiempo que el algoritmo tarda en calcular la cantidad de formas diferentes que existen para llenar un tablero de 2xn centímetros cuadrados con piezas de 1x2 centímetros cuadrados es una función exponencial, de lo cual podemos concluir que entre más grande sea la variable n, mucho más tiempo tardará en hacerse dicho cálculo. Para n = 50, el algoritmo demoraría aproximadamente 312,75 horas.

3.3 La complejidad del algoritmo es O(2^n), y no es viable usarlo en Puerto Antioquia en 2020 con contenedores que miden miles de centímetros porque se demoraría años en hacer el cálculo.

3.4

```
public boolean groupSum5(int start, int[] nums, int target) {
  if (start >= nums.length)
    return (target == 0);
  if((start<nums.length-1)&&(nums[start]%5 == 0)&&(nums[start+1]==1))
    return groupSum5(start + 2, nums, target - nums[start]);
  if((start<nums.length-1)&&(nums[start]%5 == 0)&&(nums[start+1]!=1))
    return groupSum5(start + 1, nums, target - nums[start]);
  else
    return groupSum5(start + 1, nums, target - nums[start])||(groupSum5(start + 1, nums, target));
}
```

El caso base del ejercicio es cuando el entero encargado de recorrer el arreglo (start) es igual o mayor a la longitud del arreglo, en caso de que start sea mayor a la longitud del arreglo, solo es verdadero si la variable target es 0, porque no se necesitaría ninguna combinación para llegar al objetivo. En el primer paso recursivo se revisa que start este dentro de la longitud del arreglo, se pregunta si el arreglo en la posición start es múltiplo de 5 y si la posición inmediatamente posterior a start es 1, en caso de que todo lo anterior se cumpla, se invoca recursivamente el método groupSum5 pero comenzando desde la posición inmediatamente posterior a la que tiene el 1, y a la variable target se le resta el numero múltiplo de 5, ignorando el 1. El siguiente paso recursivo se comprueba si start esta dentro del arreglo, si el numero en la posición start es múltiplo de 5 y si el numero siguiente a este es diferente de 1, si todo lo anteriormente descrito se cumple, se llama recursivamente al método groupSum5, pero como parámetros se le pasa la posición siguiente a start y a la variable target se le resta el numero múltiplo de 5. Finalmente, si no se cumplió con ninguna de las condiciones anteriores, se invoca nuevamente el método groupSum5 y se divide en dos casos, sumando en ambos 1 a start (start + 1) con la diferencia de que en un caso al objetivo se le resta el numero en la posición start, y en el otro caso no.

3.5

Nombre del ejercicio (Recursion1)	**Complejidad**
countHi	O(n) n = Longitud del String
countX	O(n) n = Longitud del String
powerN	O(C^n-1) n = Exponente
count7	O(log n) n = Entero
sumDigits	O(log n) n = Entero

Nombre del ejercicio (Recursion2)	**Complejidad**
groupSum6	O(2^n)
groupNoAdj	O(2^n)
groupSum5	O(2^n)
groupSumClump	O(2^n)
splitArray	O(2^n)

3.6

Nombre del ejercicio (Recursion1)	**Complejidad**
countHi	O(n) **n = Longitud del String**
countX	O(n) **n = Longitud del String**
powerN	O(C^n-1) **n = Exponente**
count7	O(log n) **n = Entero**
sumDigits	O(log n) **n = Entero**

Nombre del ejercicio (Recursion2)	Complejidad
groupSum6	O(2^n) n=longitud del arreglo
groupNoAdj	O(2^n) n=longitud del arreglo
groupSum5	O(2^n) n=longitud del arreglo
groupSumClump	O(2^n) n=longitud del arreglo
splitArray	O(2^n) n=longitud del arreglo

Simulacro parcial

4.1 (start+1, nums, target)

4.2 a) $T(n)=T(n/2)+C$

4.3.1 n-a, a, b, c

4.3.2 res, solucionar (n-b, a, b, c)

4.3.3 res, solucionar (n-c, a, b, c)

4.4 e) La suma de los elementos del arreglo a y es $O(n)$

4.5.1 Línea 1: return 1

Línea 2: n-1

Línea 3: n-2

4.5.2 b. $T(n)=T(n-1)+T(n-2)+C$

4.6.1 return 0

4.6.2 sumaAux (n, i+1)

4.7.1 comb(S, i+1, t-s[i])

4.7.2 comb(S, i+1, t)

4.8.1 return 0

4.8.2 ni + nj

4.9 C. 22

4.10 B. 6

4.11.1 (n-2) + lucas(n-1)

4.11.2 c. *T(n)=T(n−1)+T(n−2)+c, que es O(*2^n*)*

4.12.1 return 0

4.12.2 Math.max(fi, fj)

4.12.3 sat