	UNIVERSIDAD EAFIT ESCUELA DE INGENIERÍA DEPARTAMENTO DE INFORMÁTICA Y SISTEMAS	Código: ST245
		Estructura de Datos 1

Laboratorio Nro. 1: Recursión

María Paula Chaparro Muñoz
Universidad Eafit
Medellín, Colombia
mpchaparrm@eafit.edu.co

Juan José Parra Díaz
Universidad Eafit
Medellín, Colombia
jjparrad@eafit.edu.co

2) Simulacro de preguntas de sustentación de Proyectos

3. El ejercicio GroupSum5 consiste realizar un algoritmo que decida si en un arreglo de enteros, es posible hallar la suma de un número determinado contando todos los múltiplos de cinco que se encuentren y omitiendo los unos (1) que los siguen. La solución a este problema comienza revisando si ya se llegó al final del arreglo, caso en el que, dependiendo de si se ha alcanzado la suma deseada, se retorna verdadero o falso.

Después se revisa si el número actual es múltiplo de cinco. Si lo es, se revisa si el siguiente es uno y si así es, se ignora, sino, se busca la posibilidad de hacer la suma deseada, restándole el múltiplo de cinco con los demás elementos del arreglo.

En caso de que el número no sea múltiplo de cinco, se revisa si la suma es posible incluyendo y no incluyendo al número actual. Al final, si no se retorna ningún verdadero, entonces se retorna un falso para mostrar que no es posible.

4.

Recursión 1

```
public int factorial(int n) {  
    if (n==1) {                                C1  
        return 1;                             C2  
    }  
    else if(n==2) {                            C3  
        return 2;                             C4  
    }  
    else return n * factorial(n-1);            T(n-1)  
}
```

$$T(n) = C + T(n-1)$$

$$T(n) = C' + C*n$$

$$T(n) = O(C' + C*n)$$

$$T(n) = O(n)$$

```
public int bunnyEars(int bunnies) {  
    if (bunnies == 0) {  
        return 0;  
    }  
    else return 2 + bunnyEars(bunnies-1); T(n-1)  
}
```

$$T(n) = C + T(n-1)$$

$$T(n) = C' + C*n$$

$$T(n) = O(C' + C*n)$$

$$T(n) = O(n)$$

```
public int fibonacci(int n) {  
    if(n == 0){  
        return 0;  
    }  
    else if (n == 1){  
        return 1;  
    }  
    else return fibonacci(n-2) + fibonacci(n-1); T(n-2) + T(n-1)  
}
```

$$T(n) = C + T(n-1) + T(n-2)$$

$$T(n) = C' + C*2^n$$

$$T(n) = O(C' + C*2^n)$$

$$T(n) = O(2^n)$$

```
public int bunnyEars2(int bunnies) {  
    if(bunnies == 0){  
        return 0;  
    }  
    else if(bunnies % 2 != 0){  
        return 2 + bunnyEars2(bunnies-1);  
    }  
    else return 3 + bunnyEars2(bunnies-1);  
}
```

$$T(n) = C + T(n-1)$$

$$T(n) = C' + C*n$$

$$T(n) = O(C' + C*n)$$

$$T(n) = O(n)$$


```
public int triangle(int rows) {  
    if(rows == 0){  
        return 0;  
    }  
    if (rows == 1){  
        return 1;  
    }  
    else return rows + triangle(rows-1);  
}
```

$$T(n) = C + T(n-1)$$

$$T(n) = C' + C*n$$

$$T(n) = O(C' + C*n)$$

$$T(n) = O(n)$$

	UNIVERSIDAD EAFIT ESCUELA DE INGENIERÍA DEPARTAMENTO DE INFORMÁTICA Y SISTEMAS	Código: ST245
		Estructura de Datos 1

Recursión 2

```

public boolean groupSum5(int start, int[] nums, int target) {
    if(start >= nums.length){                                C1
        return target == 0;                                  C2
    }

    if(nums[start] % 5 == 0){                                C3
        if(start + 2 <= nums.length && nums[start + 1] == 1){ C4
            nums[start+1] = 0;                                C5
        }
        return groupSum5(start + 1, nums, target - nums[start]); T(n-1)

    } else {                                                  C6
        return groupSum5(start + 1, nums, target - nums[start]) || T(n-1)
        groupSum5(start + 1, nums, target);                  T(n-1)
    }
}

```

Este código recorre el arreglo desde la posición start, y a través de llamados recursivos, aumenta la posición mientras busca la respuesta. Explicación más a fondo en el punto 2.3.

$$T(n) = C + 2T(n-1)$$

$$T(n) = C' * 2^{n-1} + C(2^n - 1)$$

$$T(n) = O(C' * 2^{n-1} + C(2^n - 1))$$

$$T(n) = O(2^n)$$

```
public boolean groupSum6(int start, int[] nums, int target) {  
    if(start >= nums.length){  
        return target == 0;  
    }  
  
    if(nums[start] % 6 == 0){  
        return groupSum6(start + 1, nums, target - nums[start]);  
    } else {  
        return groupSum6(start + 1, nums, target - nums[start]) ||  
            groupSum6(start + 1, nums, target);  
    }  
}
```

C1

C2

C3

T(n-1)

T(n-1)

T(n-1)

Este código pregunta, en caso de haber acabado el recorrido, si se llegó al objetivo deseado. Continúa obligando a sumar el número en caso de que sea divisible por 6, y si no, busca otra forma de encontrar la suma buscada, sumando o no el número actual.

$$T(n) = C + 2T(n-1)$$

$$T(n) = C' * 2^{n-1} + C(2^n - 1)$$

$$T(n) = O(C' * 2^{n-1} + C(2^n - 1))$$

$$T(n) = O(2^n)$$

```
public boolean groupNoAdj(int start, int[] nums, int target) {  
    if(start >= nums.length){  
        return target == 0;  
    }  
    return groupNoAdj(start + 2, nums, target - nums[start]) ||  
        groupNoAdj(start + 1, nums, target);  
}
```

C1
C2
T(n-2)
T(n-1)

Este código pregunta, en caso de haber acabado el recorrido, si se llegó al objetivo deseado. Continúa haciendo dos posibles llamados recursivos, el primero hace que, si se suma el número actual, se salte el siguiente, mientras que el otro, dice que si no se suma el actual, el siguiente sí sea sumado.

$$T(n) = C + T(n-1) + T(n-2)$$

$$T(n) = C' + C \cdot 2^n$$

$$T(n) = O(C' + C \cdot 2^n)$$

$$T(n) = O(2^n)$$

```
public boolean groupSumClump(int start, int[] nums, int target) {
    if (start >= nums.length){           C1
        return target == 0;              C2
    }
    int i;                               C3
    int sum = 0;                          C4
    for (i = start; i < nums.length && nums[i] == nums[start]; i++){ C5*n
        sum += nums[i];                  C6*n
    }
    nums[start] = sum;                   C7
    return groupSumClump(i, nums, target - nums[start]) || T(n-1)
        groupSumClump(i, nums, target); T(n-1)
}
```

Este código pregunta, en caso de haber acabado el recorrido, si se llegó al objetivo deseado. Continúa determinando, a través de un ciclo, la cantidad de números repetidos que hay. Después hace dos posibles llamados recursivos, uno restando todos estos números y otro sin restar ninguno.

$$T(n) = C + 2T(n-1)$$

$$T(n) = C' * 2^{n-1} + C(2^n - 1)$$

$$T(n) = O(C' * 2^{n-1} + C(2^n - 1))$$

$$T(n) = O(2^n)$$


```
public boolean splitArray(int[] nums) {  
    return splitArray2(0, nums, 0, 0);  
}  
public boolean splitArray2(int start, int[] nums, int g1, int g2) {  
    if(start >= nums.length){  
        return g1 == g2;  
    }  
  
    return splitArray2(start+1, nums, g1 + nums[start], g2) ||  
           splitArray2(start+1, nums, g1, g2 + nums[start]);  
}
```

C1

C2

C3

T(n-1)

T(n-1)

Este código pregunta, en caso de haber acabado el recorrido, si los dos grupos de números son iguales. Si no son iguales, el código da dos posibles llamados recursivos, en uno, se agrega un número a el grupo uno y se continúa, mientras que en el otro se agrega al grupo 2.

$$T(n) = C + 2T(n-1)$$

$$T(n) = C' * 2^{n-1} + C(2^n - 1)$$

$$T(n) = O(C' * 2^{n-1} + C(2^n - 1))$$

$$T(n) = O(2^n)$$

5. 'n' y 'm' son las variables que son ingresadas a los métodos, para que de esta manera puedan hacerse de manera adecuada y teniendo una idea de lo largo que son estos mismos. Las variables anteriores representan la longitud del problema que se le va a ingresar a cada método, afectando de manera directa la complejidad del mismo.

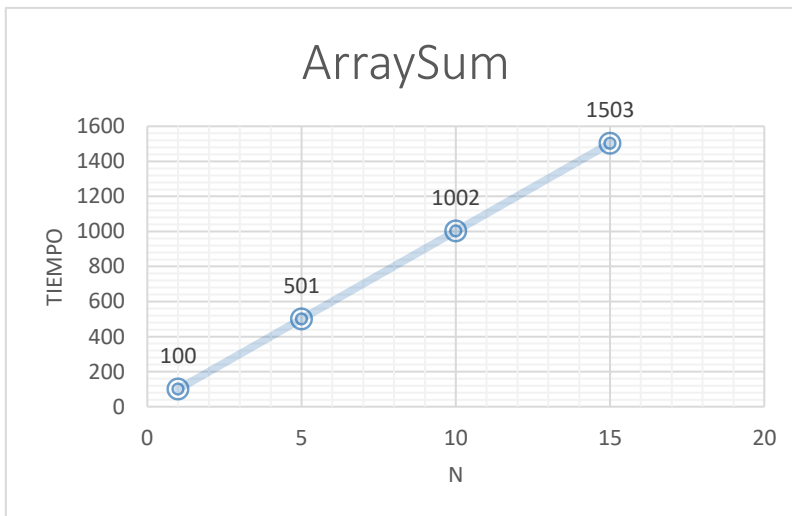
3) Simulacro de preguntas de sustentación de Proyectos

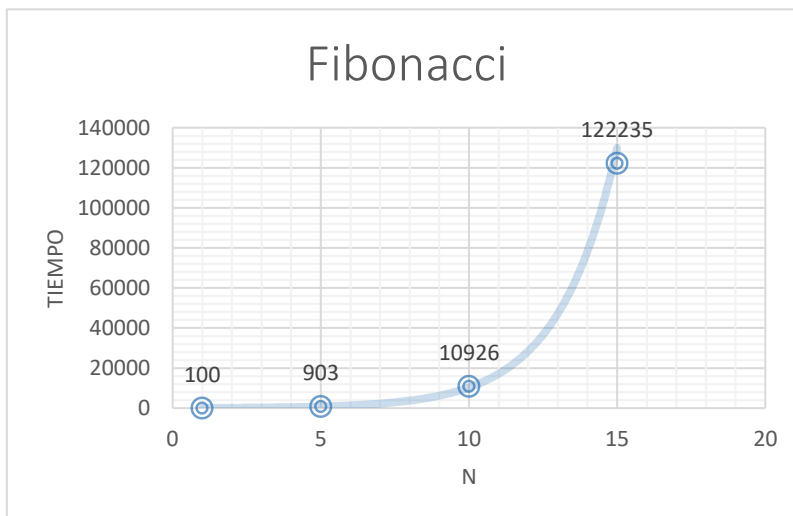
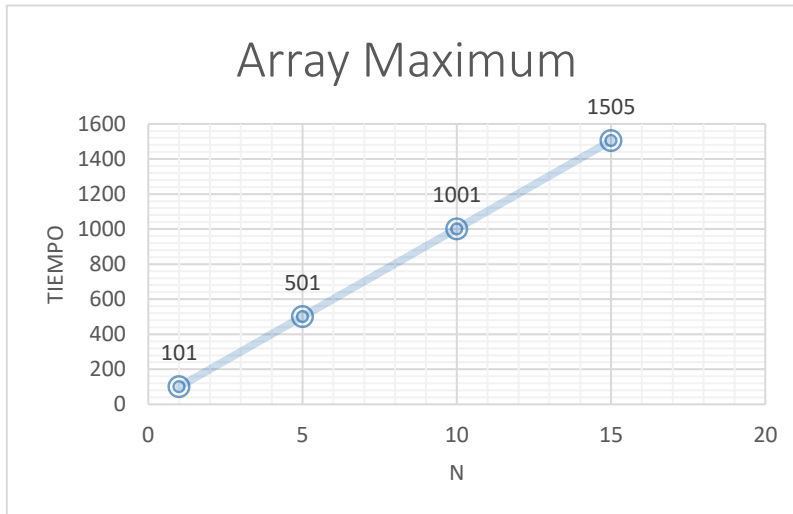
1.

	N = 1*	N = 5*	N = 10*	N = 15*
R Array sum	100	501	1002	1503
R Array maximum	101	501	1001	1505
R Fibonacci	100	903	10926	122235

*Con un retraso en cada llamado recursivo (para evitar errores de Stack Overflow)

2.





3. Así como los tiempos teóricos deberían dar, los tiempos de laboratorio se dieron, puesto que en los dos problemas con complejidad baja (ArraySum y ArrayMaximum) se dio un crecimiento lineal, el esperado, mientras que en el problema con mayor complejidad, el crecimiento se presentó de manera exponencial, es decir que aumentó mucho más precipitadamente a como lo hicieron los otros dos.

4. El Stack Overflow es un error que ocurre cuando se llena la memoria destinada a almacenar datos, esto quiere decir que se produce una cantidad tan grande de datos almacenados en variables en la memoria, que se llena el cupo que está destinado para esto.
5. 70, mas no por error de Stack Overflow, sino porque al hacerlo con este tamaño se demoró una cantidad de tiempo muy grande, por lo que era poco efectivo intentarlo con arreglos de mayor tamaño, puesto que sería imposible graficarlo. Por este motivo y por el StackOverflow es que no se puede ejecutar Fibonacci con 1 millón.
6. Una solución muy viable para calcular Fibonacci con valores grandes es hacer un proceso de Memorización en la memoria, el cual consiste en guardar los valores de Fibonacci requeridos para la realización de otros, y de esta manera, no tener que hacer el cálculo completo con ciertos valores, sino que se puede recuperar directamente desde la memoria.
7. Los problemas de CodingBat de Recursion1 son unos de una complejidad baja, los cuales consumen menos líneas de código y son más eficientes, mientras que los de Recursion2 tienen complejidades más altas, son más lentos y requieren más concentración para realizarlos.

4) Simulacro de Parcial

1. Start+1, nums, target
2. a
3.
 1. (n-a, b, c, a)
 2. *solucionar*(n-b, c, a, b), *solucionar*(n-c, a, b, c)
 3. *solucionar*(n-a, b, c, a), *solucionar*(n, a, b, c)