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

Laboratorio Nro. 1: Recursión.

Manuela Valencia Toro

Universidad Eafit
Medellín, Colombia
mvalenciat@eafit.edu.co

2)

3. El algoritmo ArraySum5 está hecho con recursión y su funcionamiento se basa en una condición de parada que define cuando debe detenerse el algoritmo y otra que permite tomar primero todos los múltiplos de 5 y ver si después de estos hay o no un uno, en caso de haberlo simplemente se adelanta una posición sin sumarlo.

Después de haber verificado si cada posición es o no múltiplo de 5, pasa a sumar los demás números que existan en el arreglo, pudiendo tomarlos o no, es decir, pasa por cada posición calculando la suma con el numero o sin el numero en el que está “parado”, hasta que al final “start” sea igual a la longitud del arreglo, e imprima si “target” es o no 0.

4. Recursion 1

```

1. public int factorial(int n) {
    if(n==1){
        return 1;}    //C
return n*factorial(n-1); //C2 + T(n-1)
    }
//T(n)={C1, n<=1;
C2 + T(n-1), n>1.
//T(n)=Cn + C1
//T(n)= O(n)

```

```
2. public int bunnyEars(int bunnies) {  
    if(bunnies==0){ return 0; }    //C1  
return 2+ bunnyEars(bunnies-1); } //C2 + T(n-1)
```

```
//T(n)= {C1, n<=1;  
C2 + T(n-1), n>1.  
//T(n)=Cn + C1  
//T(n)= O(n)
```

```
3. public int  ibonacci(int n) {  
    if(n<=1){ return n;}    //C1  
return  ibonacci (n-2)+  ibonacci(n-1);  
    //C2+ T(n-2)+T(n-1)  
    }
```

```
T(n)= { C1, n <=1  
C2 + T(n-2) + T(n-1), n>1.  
T(n)= O(2^n)
```

```
4. public int bunnyEars2(int bunnies) {  
    int a=2;    //c1  
    if(bunnies % 2 ==0){ //c2  
        a=a +1; //c3  
    }  
    if(bunnies==0){ //c4  
        return 0; //c5  
    }  
return a + bunnyEars2(bunnies-1); //c6 + t(n-1)  
    }
```

```
//T(n)= {C1 , n%2==0;  
C2, n== 0  
C3 + T(n-1), n>0 && n%2 >0.  
//T(n)=Cn + C1  
//T(n)= O(n)
```

```
5. public int triangle(int rows) {  
    int a=rows;    //c1  
    if(rows<=1){    //c2  
        return rows;    //c3  
    }  
    return a + triangle(rows-1); //c4 + t (n-1)  
    }  
    //T(n)= {C1, n<=1;  
    C2 + T(n-1), n>1.  
    //T(n)=Cn + C1  
    //T(n)= O(n)
```

Recursion 2.

```
1. public boolean groupSum6(int start, int[] nums, int target) {  
    if (start >= nums.length)  
        return target == 0; // c2  
    if (nums[start] == 6) {  
        return groupSum6(start + 1, nums, target - nums[start]); // c3 + T(n-1)  
    }  
    return groupSum6(start + 1, nums, target - nums[start]) //C4 + T(n-1)  
    || groupSum6(start + 1, nums, target); // C5  
    }  
    T(n)=C2, n >= nums.length;  
    C3 + T(n-1), nums[n] ==6;  
    C4 + T(n-1), nums[n]<>6  
    T(n)=C*2^(n-1) + C((2^n) - 1)  
    T(n)=O(n)=O(2^n-1) + O(2^n)  
    O(n)=O(2^(n-1) + 2^n)  
    O(n)=O(2^n)
```

```
2. public boolean groupNoAdj(int start, int[] nums, int target) {  
    if (start >= nums.length) return target == 0; C1, start >= nums.length  
    return groupNoAdj(start + 2, nums, target - nums[start]) // C2 + T(n-2)  
    || groupNoAdj(start + 1, nums, target); C3  
}
```

```
//T(n)=C1, start >= nums.length;  
C2 + T(n-2);  
C3;  
//T(n)= C(n-1)  
T(n)=O(n)=O(C*(n-1))  
O(n)= O(n-1)  
O(n)=O(n)
```

```
3. public boolean groupSum5(int start, int[] nums, int target) {  
    if (start >= nums.length) return target == 0; //c1  
    if (nums[start] % 5 == 0) { //c2  
        if (start < nums.length - 1 && nums[start + 1] == 1){ //c3  
            return groupSum5(start + 2, nums, target - nums[start]); //C4+T(n-2)  
        } else{  
            return groupSum5(start + 1, nums, target - nums[start]); //C5+T(n-1)  
        }  
    }  
    return groupSum5(start + 1, nums, target - nums[start]) //C6 + T(n-1)  
    || groupSum5(start + 1, nums, target);  
}
```

```
//T(n)= C2, n >=nums.length;  
C4 + T(n-2), n< nums.length - 1 && nums[start + 1] == 1;  
C5 + T(n-1), n>= nums.length - 1 && nums[start + 1] <> 1;  
C6 + T(n-1);  
T(n)= C + 2^n-1  
T(n)=O(n)=(C+ 2^n/2)  
O(n)=O(2^n)
```

```
4. public boolean groupSumClump(int start, int[] nums, int target) {
    if (start >= nums.length) return target == 0; // C1
    if (start < nums.length - 1 && nums[start] == nums[start + 1]) {
        return groupSumClump(start + 2, nums, target - (nums[start] + nums[start + 1]))
            // C2 + T(n-2) ||
        groupSumClump(start + 2, nums, target); // C3
    } else {
        return groupSumClump(start + 1, nums, target) || // C4 + T(n-1)
        groupSumClump(start + 1, nums, target - nums[start]); C5
    }
    // T(n) = C1, n >= nums.length
    C2 + T(n-2), n < nums.length - 1 && nums[n] == nums[n + 1]
    C4 + T(n-1), n >= nums.length - 1 && nums[n] <> nums[n + 1]
    T(n) = C + T(n-1) + T(n-2)
    T(n) = O(2^n)
```

```
5. public boolean splitArray(int[] nums) {
    int suma1 = 0;
    int suma2 = 0;
    int start = 0;
    if (ayuda(nums, suma1, suma2, start) == true) { // O(n)
        return true; // C1
    } else {
        return false; // C2
    }
    T(n) = O(n) = O(n)
```

```
public boolean ayuda(int [] nums,int suma1,int suma2,int start){
    if(start>=nums.length){
        return suma1==suma2; //C1
    }
    return ayuda(nums,suma1+nums[start],suma2,start+1) ||
    ayuda(nums,suma1,suma2+nums[start],start+1); //C2 + T(n-1)
}
```

$T(n) = C1, \text{ start} \geq \text{nums.length}$
 $C2 + T(n-1), \text{ start} < \text{nums.length}$
 $T(n) = C + T(n-1)$
 $T(n) = Cn + C$
 $O(n) = O(n)$

5 Cada uno de los métodos anteriores presenta una complejidad especial, la cual varía principalmente a partir de las funciones que un algoritmo realiza. En cada uno de estos algoritmos la “n” presente en la complejidad representa la variable que será disminuida en cada llamado recursivo, lo que a su vez representa la notación de las veces que cierto proceso se repite mientras se está ejecutando el algoritmo.

3) Simulacro de preguntas de sustentación de Proyectos

	N=30	N=40	N=45	N=50
R Fibonacci	23 milisegundos	3771 milisegundos	33925 milisegundos	412584 milisegundos

1.

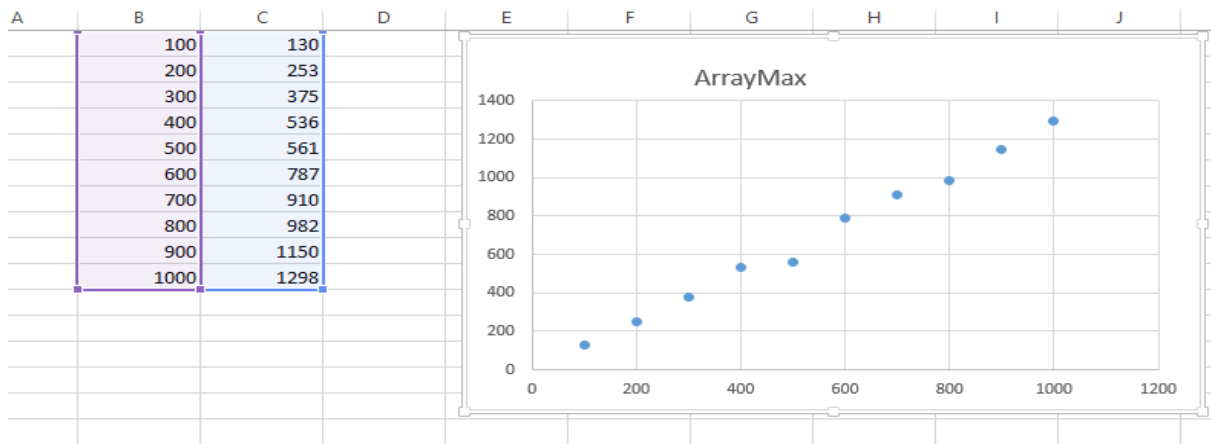
	N=1000	N=2000	N=3000	N=2500
R ArraySum	1515 milisegundos	3028 milisegundos	4568 milisegundos	3739 milisegundos

	N=1000	N=2000	N=2200	N=2500
R Array Maximun	1236 milisegundos	2544 milisegundos	2740 milisegundos	3095 milisegundos

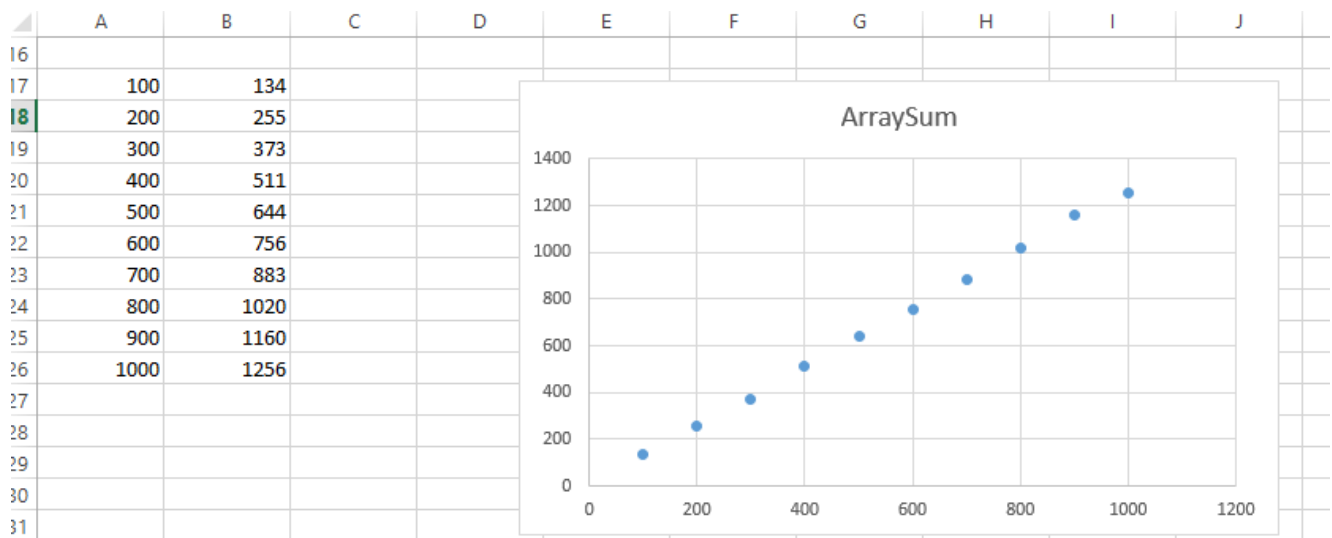
Nota: Para el algoritmo ArrayMax se usó un retardo de 1 nanosegundo.

ArrayMax:

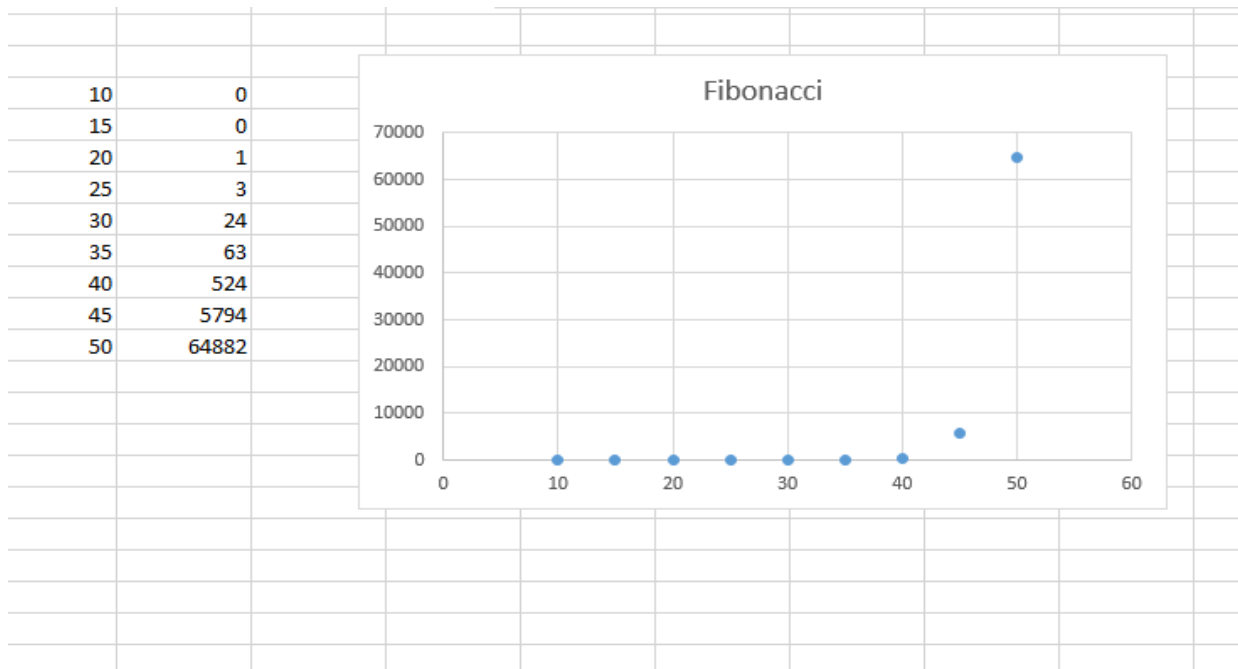
2.




ArraySum:



Fibonacci:



- Se puede concluir que las gráficas corresponden correctamente a la complejidad que posee cada algoritmo, es decir, la complejidad de arraymax y arraySum poseen una complejidad $O(n)$, y en las gráficas puede visualizarse una línea recta, e igualmente con fibonacci, su grafica representa una función exponencial y su notación es $O(2^n)$.
- El stack-Overflow es un error que indica un exceso de datos almacenados en la memoria, cuando este error aparece significa que el heap y la pila del computador están completamente llenos y no es posible almacenar más información por lo que la ejecución del programa se detiene.
- El valor más grande que pude calcular fue 50, esto Por su complejidad, que es $O(2^n)$, lo que hace que su ejecución sea bastante demorada. No se puede calcular para 1 millón por que con esta cantidad de datos el programa

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

lanza un stack-OverFlow, es decir, el computador no tiene suficiente espacio de almacenamiento para alojar tantos datos.

6. Por medio de la programación dinámica, que disminuye la complejidad de $O(2^n)$ a $O(n)$, posibilitando calcular el Fibonacci para valores grandes.
7. De la complejidad calculada en los algoritmos desarrollados de codingbat, se puede concluir que a diferencia de recursión 1, cada algoritmo de recursión 2 tiene como mínimo una complejidad $O(2^n)$, lo que los hace un poco lentos, y por ende dificulta poder tomar valores muy altos. Mientras que los de recursión 1, ya que tiene una complejidad baja posibilitan ingresar datos grandes arrojando un resultado en poco tiempo. En resumen, la complejidad de los algoritmos de recursión 2 es mayor que la de los algoritmos de recursión 1.

4) Simulacro de Parcial

1. start+1,nums,target.

2. A

3. 3.1

3.2

3.3

4. e