Complejidad de los ejercicios de Coding Bat (Recursión 1 y Recursión 2)

1. powerN

1.1 Copiar el código en Word

```
public int powerN(int base, int n) {
if(n==0){
return 1;
}
return base*powerN(base, n-1);
}
```

1.2 Identificar quién es el tamaño del problema (llamado también "n")

El tamaño del problema es encontrar el resultado de base^n.

1.3 Etiquetar cuánto se demora cada línea

```
public int powerN(int base, int n) {
if(n==0){
return 1; // constante
}
return base*powerN(base, n-1) // constante * T(n-1);
}
```

1.4 Escribir la ecuación de recurrencia

$$f(x) = \begin{cases} c1 & si \quad n == 0 \\ \\ T(n-1) * c_1 \end{cases}$$

T(n) = c_1*T(n-1)

1.5 Resolver la ecuación con Wolfram Alpha

T(n) = c_1*T(n-1)
T(n) = c_1 * c_1^(n-1)

1.6 Aplicar la notación O a la solución de la ecuación

T(n) es O(c_1 * c_1^(n-1))
T(n) es O(c_1^n)

1.7 Explicar en palabras
La complejidad asintótica (es decir, para valores grandes de n) para el peor de los casos (es decir, en el que el algoritmo hace (uno o más operaciones) para el algoritmo de calcular la potencia N de una base cualquiera es O(c_1^n)

2 count8
2.1 copiar el código en word

```
public int count8(int n) {
  int cont=0;
  if(n==0)return 0;
  if(n%100==88)cont++;
  if(n%10==8)cont++;
  return cont+count8(n/10);

}
```

2.2 identificar quien es el llamado del problema (llamado también "n")
dado un entero no negativo se debe contar las ocurrencias de 8, teniendo en cuenta que si un 8 está seguido de otro cuentan el doble.

2.3 etiquetar cuanto se demora cada línea

```
public int count8(int n) {
  int cont=0;
  if(n==0)return 0;  // constante
  if(n%100==88)cont++;
  if(n%10==8)cont++;
  return cont+count8(n/10);  // T(n-1)

}
```

2.4 escribir la ecuación de recurrencia
T(n) = c_1 + T(n-1)

2.5 Resolver la ecuación con wolframAlpha
T(n) = c_1*n+c_1

2.6 Aplicar la notación O a la solución de la ecuación
T(n) es O(c_1*n+c_1)
T(n) es O(n)

2.7 explicar en palabras

La complejidad asintótica (es decir, para valores grandes de n) para el peor de los casos (es decir, en el que el algoritmo hace (uno o más operaciones) para el algoritmo que encuentra las veces en las que está en 8 es O(n)

3.0 BunnyEars2

3.1. Copiar el código en word

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

3.2 Identificar quien es el llamado del problema (llamado también"n")

El llamado del el problema es cuando se ingresa un número entero que representa la cantidad de conejos

3.3 Etiquetar cuanto se demora cada línea

```
public int bunnyEars2(int bunnies) {
  if(bunnies == 0 ) return 0;  // constante C_1
  else if (bunnies % 2 == 0) return 3 + bunnyEars2(bunnies - 1); // C_2 + T(n-1)
  else return 2 + bunnyEars2(bunnies - 1);  // C_3 T(n-1)
}
```

3.4 Escribir la ecuación de recurrencia

$T(n) = C_1 + C_2 + T(n-1) + C_3 + T(n-1)$

3.5 Resolver la ecuación con wolfram alpha

$T(n) = c_1\ 2^{(n - 1)} + (C_1 + C_2 + C_3)\ (2^n - 1)$ (c_1 is an arbitrary parameter)

3.6 Aplicar la notación O a la ecuación

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

3.7 Explicar en palabras

El algoritmo recibe un número entero que representa la cantidad de conejos y cuenta cuántas orejas hay en total por ellos, además tiene la característica que por cada dos conejos, uno de ellos contará como si tuviera tres orejas

4.0 Array11

4.1. Copiar el código en word

```
public int array11(int[] nums, int index) {
  if(index == nums.length){
  return 0;
 }
 if (nums[index] == 11){
  return 1 + array11(nums, index+1);
 }
 else return array11(nums,index+1);
}
```

4.2 Identificar quien es el llamado del problema (llamado también"n")

Dado un arreglo de enteros se debe contar recursivamente las veces que aparece el número once en él

4.3 Etiquetar cuanto se demora cada línea

```
public int array11(int[] nums, int index) {
  if(index == nums.length){
  return 0;  // C_1 constante
 }
 if (nums[index] == 11)
  return 1 + array11(nums, index+1);  // C_2 + T(n-1)
 }
 else return array11(nums,index+1);  // T(n-1)
}
```

4.4 Escribir la ecuación de recurrencia

$T(n) = C_1 + C_2\ T(n-1) + T(n-1)$

4.5 Resolver la ecuación con wolfram alpha

$T(n) = c_1\ 2^{(n-1)} + (C_1 + C_2)\ (2^n - 1)$ (c_1 is an arbitrary parameter)

4.6 Aplicar la notación O a la ecuación

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

4.7 Explicar en palabras

Dado un arreglo de números enteros el programa calcula las veces en las que el número once aparece en él, para ello por medio de recursión se hace un desplazamiento por todos los valores del arreglo y de esta manera encontrar las apariciones del mencionado número. el llamado recursivo puede expresarse como T(n-1) en las fórmulas a pesar de que en el algoritmo es T(n+1) debido a que el espacio en el arreglo decrece con cada aumento que se le hace a la variable que lleva el control de posición

5.0 sumDigits

5.1. Copiar el código en word

```
public int sumDigits(int n) {
  if(n%10==0 && n/10==0) return n/10;
  return (n%10)+sumDigits(n/10);
}
```

5.2 Identificar quien es el llamado del problema (llamado también"n")

Dado un entero se debe calcular la suma de todos los dígitos que lo conforman

5.3 Etiquetar cuanto se demora cada línea

```
public int sumDigits(int n) {
  if(n%10==0 && n/10==0) return n/10; // C_1 constant
  return (n%10)+sumDigits(n/10); // C_2 + T(n-1)
}
```

5.4 Escribir la ecuación de recurrencia

$T(n) = C_1 + C_2 + T(n-1)$

5.5 Resolver la ecuación con wolfram alpha

$T(n) = c_1 + (C_1 + C_2)\, n$ (c_1 is an arbitrary parameter)

5.6 Aplicar la notación O a la ecuación

$T(n) = O(n)$

5.7 Explicar en palabras

El algoritmo recibe un número entero positivo y calcula el valor que resulta al sumar todos sus dígitos, para ello se apoya de operaciones como la división y el módulo para extraer los valores uno a uno, además de hacer uso de la recursión

RECURSIÓN 2.

groupSum6

1.1 Copiar el código en Word

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

1.2 Identificar quién es el tamaño del problema (llamado también “n”)

El tamaño del problema es sumar un subgrupo de enteros de un arreglo para que den el mismo valor que el target, pero con la restricción de que si hay un 6 en el arreglo se debe usar.

1.3 Etiquetar cuánto se demora cada línea

```
public static boolean groupSum6(int start, int[] nums, int target) {
if(start >= nums.length){
return target==0;                    //Constante
}
if(nums[start] == 6){
return groupSum6(start+1,nums,target-nums[start]);  // T(n-1)
}
if(groupSum6(start+1,nums,target-nums[start])){.        //T(n-1) + constante
return true;
}
if(groupSum6(start+1,nums,target)){.                    // T(n-1) + constante
return true;
}
```

return false;
}

1.4 Escribir la ecuación de recurrencia

$$f(x) = \begin{cases} c1 & si \quad start => nums.length \\ T(n-1)+c_1 & si \quad nums[start] == 6 \\ T(n-1)+c_2 & \\ T(n-1)+c3 & \end{cases}$$

T(n) = T(n-1)+T(n-1)+c_1+T(n-1)+c_2+T(n-1)+c_3

1.5 Resolver la ecuación con Wolfram Alpha

T(n) = T(n-1)+T(n-1)+c_1+T(n-1)+c_2+T(n-1)+c_3

$$T(n) = c_1\, 4^{n-1} + \frac{1}{3}\,(c_1 + c_2 + c_3)\left(4^n - 1\right)$$ (c_1 is an arbitrary parameter)

1.6 Aplicar la notación O a la solución de la ecuación

8^n

1.7 Explicar en palabras

La complejidad asintótica (es decir, para valores grandes de n) para el peor de los casos (es decir, en el que el algoritmo hace (uno o más operaciones) para el algoritmo que encuentra un subgrupo de elementos en arreglo tal que sumados den el target y con la restricción de que si en el arreglo hay un número 6 se tendrá que utilizar es 8^n

2. groupSumClump

2.1 copiar el código en word

```
public boolean groupSumClump(int start, int[] nums, int target) {

  if( start>= nums.length){
    return target == 0;
  }
  else if(start+1 < nums.length &&nums[start]==nums[start+1]){
    return groupSumClump(start + 2, nums, target - (nums[start]+nums[start+1])) ||
    groupSumClump(start + 2, nums, target);
  }
  else {
    return groupSumClump(start + 1, nums, target - nums[start]) ||
```

```
    groupSumClump(start + 1, nums, target);

  }
}
```

2.2 Identificar quién es el tamaño del problema (llamado también “n”)

el tamaño del problema es crear un subgrupo del arreglo que sume lo mismo que el target, con la condición de que si hay números idénticos seguidos se toman todos ellos o no

2.3 Etiquetar cuánto se demora cada línea

```
public boolean groupSumClump(int start, int[] nums, int target) {

  if( start>= nums.length){
    return target == 0;  // constante
  }
  else if(start+1 < nums.length &&nums[start]==nums[start+1]){
    return groupSumClump(start + 2, nums, target - (nums[start]+nums[start+1])) ||
    groupSumClump(start + 2, nums, target);  // 2T(n-1)
  }
  else {
    return groupSumClump(start + 1, nums, target - nums[start]) ||
    groupSumClump(start + 1, nums, target);  // 2T(n-1)

  }
}
```

2.4 Escribir la ecuación de recurrencia

T(n)=c_1+4T(n-1)

2.5 Resolver la ecuación con Wolfram Alpha

$$T(n) = c_1\, 4^{n-1} + \frac{1}{3}\, c_1 \left(4^n - 1\right)$$ (c_1 is an arbitrary parameter)

2.6 Aplicar la notación O a la solución de la ecuación

T(n) es O(c_1 * 4^(n-1) + ⅓ * c_1 * (4^n-1))

T(n) es O(8^n)

2.7 Explicar en palabras

La complejidad asintótica (es decir, para valores grandes de n) para el peor de los casos (es decir, en el que el algoritmo hace (uno o más operaciones) para el algoritmo que encuentra un subgrupo de elementos en arreglo tal que sumados den el target y con la restricción de que si hay dos números idénticos seguidos se toman todos ellos o no se toman es O(8^n)

3.0 groupSum5
3.1 copiar el código en word

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

3.2 Identificar quien es el tamaño del problema (también llamado “n”)

El tamaño del problema radica en encontrar un subgrupo de números que sumados sean igual al número objetivo, con la condición de que si hay un número que sea múltiplo de 5 en dicho arreglo debe tomarse si o si pero si el siguiente número es uno debe ignorarse

3.3 Etiquetar cuanto se demora cada línea

```
public boolean groupSum5(int start, int[] nums, int target) {
if(start < nums.length){
  if(start > 0){
    if(nums[start - 1] % 5 == 0 && nums[start] == 1) return groupSum5(start +
1,nums,target); // T(n-1)
   }
  if(nums[start] % 5 == 0) return groupSum5(start+1,nums,target-nums[start]); // T(n-1)
  else return groupSum5(start+1,nums,target) ||
groupSum5(start+1,nums,target-nums[start]); //2T(n-1)
}
if(target == 0) return true; // C_1 constante
return false; // C_2 constante
}
```

3.4 Escribir la ecuación de recurrencia

$T(n) = T(n-1) + 2T(n-1) + T(n-1) + C_1 + C_2$
$T(n) = 4\,T(n-1) + C_1 + C_2$

3.5 Resolver la ecuación con wolfram alpha

T(n) = c_1 4^(n − 1) + 1/3 (C_1 + C_2) (4^n − 1) (c_1 is an arbitrary parameter)

3.6 Aplicar la notación O a la solución

T(n) = O(4^(n − 1) + (4^n − 1))

3.7 Explicar en palabras

El algoritmo recibe tres valores: 1) variable de desplazamiento. 2)Un arreglo de números. 3) un objetivo.
El algoritmo recorre el arreglo de números con ayuda de la variable de desplazamiento en busca de un subconjunto de números que sea igual al objetivo designado.
Cuenta con una condición especial, si encuentra un número que sea múltiplo de 5 debe agregarlo obligatoriamente al subconjunto y si el siguiente número es un uno debe ignorarlo por completo

4.0 splitArray
4.1 copiar el código en word

```
public boolean splitArray(int[] nums) {
  int index = 0;
  int suma1 = 0;
  int suma2 = 0;
  return auxArray(nums, index, suma1, suma2);
}

private boolean auxArray ( int[] nums, int index, int suma1, int suma2 ) {
  if ( index >= nums.length ) {
    return suma1 == suma2;
  }

  int value = nums[index];

  return (auxArray(nums, index + 1, suma1 + value, suma2) ||
    auxArray(nums, index + 1, suma1, suma2 + value));
}
```

4.2 Identificar quien es el tamaño del problema (también llamado “n”)

El tamaño del problema radica en separar los números del arreglo en dos grupos de tal manera que la suma de los dos subgrupos sea la misma

4.3 Etiquetar cuanto se demora cada línea

```
public boolean splitArray(int[] nums) {
  int index = 0;
  int suma1 = 0;
  int suma2 = 0;
  return auxArray(nums, index, suma1, suma2);
}

private boolean auxArray ( int[] nums, int index, int suma1, int suma2 ) {
  if ( index >= nums.length ) {
    return suma1 == suma2; // C_1 constante
  }

  int value = nums[index];

  return (auxArray(nums, index + 1, suma1 + value, suma2) ||
    auxArray(nums, index + 1, suma1, suma2 + value)); // 2T(n-1)
}
```

4.4 Escribir la ecuación de recurrencia

$T(n) = C_1 + 2T(n-1)$

4.5 Resolver la ecuación con wolfram alpha

T(n) = c_1 2^(n − 1) + C_1 (2^n − 1) (c_1 is an arbitrary parameter)

4.6 Aplicar la notación O a la solución

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

4.7 Explicar en palabras

El algoritmo se apoya en un método auxiliar que le permite manejar de manera más eficiente el problema, para ello se establecen dos llamados recursivos que tienen por objetivo dividir los valores del arreglo original de números para finalmente comparar e indicar que son iguales.