2.4 – 2.5

Recursion 1

```
public  int factorial(int n) {              //c1
      if(n==0){                             //c2
         return 1;                          //c3
      }else{
         return n*(factorial(n-1));     //c4 + T(n-1))
      }
   }
```

T(n) = T(n -1) +c

T(n) = O(n)

```
public static int fibonacci(int n) {                          //c1
      if(n<=1){                                               //c2
         return n;                                            //c3
      }else{
         return (fibonacci(n-2)+ fibonacci(n-1));          // c4 + T(n-2) + T(n-1)
      }
   }
```

T(n) = c4 + T(n-2) + T(n-1)

T(n) = O(2^n)

```
public static int bunnyEars(int bunnies) {  // c1
      if(bunnies == 0){                    //c2
         return 0;                        //c3
      }else{
         return 2+(bunnyEars(bunnies-1));    //c4 + T(2 + (n-1))
      }
   }
```

T(n) = c4 + T(2 + (n-1))

T(n) = c + T(n + 1)

T(n) = O(n)

```
public int bunnyEars2(int bunnies) {
      if(bunnies == 0){                    //c1
         return 0;                        //c2
      }else if(bunnies % 2 == 1){          //c3
         return 2 + bunnyEars2(bunnies-1);   //c4 + T(2 + (n-1))
      }else {
         return 3 + bunnyEars2(bunnies-1);   //c5 + T(3 + (n-1))
      }
   }
```

T(n) = c4 + T(2 + (n-1)) + c5 + T(3 + (n-1))

T(n) = O(n)

```
public int triangle(int rows) {
      if(rows == 0){         //c1
         return 0;          //c2
      }else{
         return rows + 1 * (triangle(rows-1));    //c3 + T(n -1)
      }
   }
```

T(n) = c3 + T(n -1)

T(n) = O(n)

Recursión 2

```
public boolean groupSum6(int start, int[] nums, int target){
      if (start >= nums.length)          //c1
         return (target == 0);          //c2
      if (groupSum6(start+1, nums, target - nums[start])) return true;       //c3 + T(n -1)
      if (nums[start] != 6 && groupSum6(start+1, nums, target)) return true;    //c4 + T(n-1)
      return false;                    //c5
   }
```

T(n) = T(n-1) + T(n+1) + c

n = start

T(n) = O(2^n)

```
public boolean groupNoAdj(int start, int[] nums, int target) {
    if (start >= nums.length)          //c1
      return (target == 0);          //c2
    if (groupNoAdj(start+1, nums, target)) return true;        //c3 + T(n-1)
    if (groupNoAdj(start+2, nums, target-nums[start])) return true; //c4 + T(n-2)
    return false; //c5
  }
```

T(n) = T(n-1) + T(n-2) + c

n = start

T(n) = (2^n)

```
private boolean checkOne(int start, int[] nums) {
    if (start == 0) return true;        //c1
    if (start > 0 && nums[start-1] % 5 == 0 && nums[start] == 1)  //c2 + T(n -1)
      return false;   //c3
    else
      return true;   //c4
  }
```

T(n) = T(n -1) + c

T(n) = c.n + c1

n = start

T(n) = O(n)

```
public boolean groupSum5(int start, int[] nums, int target) {
    if (start >= nums.length) return (target == 0);        //c1
    if (groupSum5(start+1, nums, target-nums[start]) && checkOne(start, nums)) // c2 + T(n-1)
      return true;       //c3
    if (nums[start] % 5 != 0 && groupSum5(start+1, nums, target)) return true; // c4 + T(n-1)
    return false;      //c5
  }
```

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

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

n = start

T(n) = O(2^n)

```
private void altArray(int[] nums) {
      for (int i = 0; i < nums.length; i++) {  //c.n
         if (i > 0 && nums[i] == nums[i-1]) { //c2
            nums[i-1] += nums[i];  //c3
            if (i+1 < nums.length && nums[i] != nums[i+1])
               nums[i] = 0;  //c4
            else if (i == nums.length-1) // c5
               nums[i] = 0; //c6
         }
      }
   }
```

T(n) = O(1) + O(n)

n = nums

T(n) = O(n)

```
public boolean groupSumClump(int start, int[] nums, int target) {
      altArray(nums);     //O(2 ^n)
      if (start >= nums.length) return target == 0;       //c2
      if (groupSumClump(start+1, nums, target-nums[start])) return true;      //c3 + T(n-1)
      if (groupSumClump(start+1, nums, target)) return true;         //c4 + T(n – 1)
      else return false;      //c5
   }
```

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

n = start

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

T(n) = O(2^n)

```
private boolean recArray ( int[] nums, int index, int sum1, int sum2 ) {
    if ( index >= nums.length ) {      //c1
       return sum1 == sum2;          //c2
    }
    int value = nums[index];       //c3
    return (recArray(nums, index + 1, sum1 + value, sum2) ||
       recArray(nums, index + 1, sum1, sum2 + value)); //c4 + T(n -1) + T(n-1)
  }
```

T(n) = c4 + T(n -1) + T(n-1)

n = index

T(n) = O(2^n)

```
public boolean splitArray(int[] nums) {
    int index = 0;
    int sum1 = 0;
    int sum2 = 0;
    return recArray(nums, index, sum1, sum2); // O(2^n)
  }
```

T(n) = O(2^n)

2.3

Dado un arreglo de enteros, retorna true o false si es posible escoger un subconjunto de esos enteros, de tal manera que la suma de los elementos de ese subconjunto sea igual a target. El parámetro start funciona como un contador y representa un índice en el arreglo de números nums.

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

Se llama recursivamente incrementando los valores de start y de target hasta que start se hace mayor o igual a el tamaño del arreglo y ahí la función empieza a retornar los valores a los que equivale cada cambio de valor cuando se llamaba recursivamente al método si este alguna vez es igual al target que se ingresó como parámetro retorna true