Punto 2.3

En el ejercicio groupSum5 en el primer if se pregunta si el parámetro start es mayor a la longitud del arreglo y si esta condición se cumple retorna 0; esto con el fin de evitar un overflow.

El siguiente if pregunta si el valor en el arreglo en la posición del parámetro start es un múltiplo de 5. Si lo es el siguiente if pregunta si el parámetro start es menor a la longitud del arreglo menos uno (el -1 es porque si el parámetro es igual a la longitud del arreglo al compararlo con strat+1 daría overflow) y si el parámetro start mas uno es igual a 1; en caso de que se cumplan estas condiciones se aumenta el parámetro start en 2 (ya que según el problema no se puede tomar un numero múltiplo de 5 que tenga en la posición de su derecha un 1) y se resta al target el valor en la posición start.

Si el arreglo en la posición start no es múltiplo de 5 simplemente se le resta al parámetro target el valor del arreglo en la posición start y se le suma uno a start.

Por último, se retorna el llamado recursivo del método con start + 1 y el parámetro target igual o con start + 1 y el parámetro target menos el valor de la posición start.

Punto 2.4

Factorial, BunnyEars, triangle = $T(n) = n + 1$

Fibonacci = $T(n) = 2n + 1$

BunnyEars2 = $2n + 2$

groupSum6, splitArray= $T(n) = n^2 + 2$

groupNoAdj = $T(n) = n^2 + 1$

groupSum5 = $T(n) = n^n + 3$

groupSumClamp = $T(n) = n^2 * n + 2$

Punto 2.5

La variable n es el número de elementos con los que se ejecuta el algoritmo.

Punto 3.1

Que es muy importante tener presente en todo momento el tamaño del problema y la condición de parada al hacer ejercicios con recursión. Ya que si no se tienen en cuenta estos dos factores puede causar un stack overflow.

Punto 3.2

40, no se puede calcular con valores tan grandes porque el crecimiento de Fibonacci es de tipo exponencial lo que aumenta en gran medida el tiempo cada que aumentan los datos.

Punto 3.3

Con ciclos sería más eficiente ya que no hay que hacer llamadas recursivas.

Punto 3.4

Los ejercicios de recursión 1 estaban sencillos en comparación con los de recursión 2 ya que en recursión 2 se necesitaba más de un llamado recursivo para resolver el ejercicio lo cual hace más complicada la solución y menos eficiente el algoritmo.

Punto 4.

1) start + 1, nums, target

2) b

3)

3.1 int res = solucionar (n - a, a, b, c, c) +1;

3.2 res = Math.max (res, solucionar (n - b, a, b, c) +1);

3.3 res = Math.max (res, solucionar (n – c, a, b, c)+1);

4) e

5)

5.1

Linea 2: return n;

Linea 3: n-1

Linea 4: n-2

5.2 b

6)

6.1 return 0;

6.2 return(n.charAt(i)-'0') + n.charAt(i+1);

7)

7.1 return comb (S, i+1, t – S [i])
7.2 comb (S, i+1, t)