Ejercicio 1

Se desea **diseñar** un algoritmo recursivo no final denominado ***calculoSept2011*** de acuerdo con el esquema propuesto en clase (con las funciones **esCasoBase, solucionBase, siguiente** y **combina**) para calcular el sumatorio desde un valor de $\alpha=i$ hasta *n* de la expresión **(i!/α!)*(2*A[α]*B[α])**. Siendo A y B dos vectores de *n* enteros cada uno e *i* un valor comprendido entre *1* y *n.*

$$calculoSept2011(A,B,i,n) = \sum_{\alpha=i}^{n} (i!/\alpha!) * (2 * A[\alpha] * B[\alpha])$$

Tenga en cuenta que esta expresión es equivalente a:

$$calculoSept2011(A,B,i,n) = 2 * A[i] * B[i] + (1/(1+i)) * \sum_{\alpha=i+1}^{n} ((i+1)!/\alpha!) * (2 * A[\alpha] * B[\alpha])$$

SE PIDE

a) Realizar la implementación en C del esquema propuesto en clase para el diseño recursivo no final para este problema y los métodos correspondientes de dicho esquema:

```
float calculoSept2011(          ) {
        // TO DO
}
logico esCasoBase(              ){
   // TO DO
}
float solucionBase(             ){
   // TO DO
}

..... siguiente(                ){
   // TO DO
}

float combina(                  ){
   // TO DO
}
```

b) **Cómo debe llamarse a la función implementada** si queremos calcular el sumatorio pedido desde la posición 3 a la 6 de dos vectores A y B dados de 6 enteros cada uno.

c) Indicar con respecto a la complejidad algorítmica: ¿Qué es **tamaño del problema**?

d) **Describa el caso mejor y el caso peor** de la implementación realizada de ***calculoSept2011*** y calcule el **T(n) para cada uno de los dos casos**.

Tiempo estimado: 45 min. Puntuación: 3.0 puntos

Solución del Ejercicio número 1.

Se supone existe un fichero con los tipos definidos

```
typedef enum{false, true} logico;
typedef struct{
    float *a;
    float *b;
    int i;
    int n;
} CuatroDatos;
```

```
float calculoSept2011(CuatroDatos n){
    float r, rSig;
    CuatroDatos nSig;

    if (esCasoBase(n)) {
        r = solucionBase(n);
    }
    else {
        nSig = siguiente(n);
            rSig = calculoSept2011(nSig);
            r = combina(n,rSig);
    }
    return r;
}

logico esCasoBase(CuatroDatos q){
      return(q.i==q.n);
}
float solucionBase(CuatroDatos q){
      return(2*q.a[q.i]*q.b[q.i]);
}

CuatroDatos siguiente(CuatroDatos q){
      CuatroDatos qsig={q.a,q.b,q.i+1,q.n};
      return(qsig);
}

float combina(CuatroDatos q,float rSig){
      float p= (float) 1/(q.i+1);
      float r= 2*q.a[q.i]*q.b[q.i] + p*rSig;
      return(r);
}
```

Apartado b) Se debe hacer la llamada para i uno menos que el valor dado y para n igual. Para el caso de la llamada para ir de 3 a 6, en *q.i* debe ser instanciado a 2 y en *q.j* de CuatroDatos debería instanciarse a 5. Ejemplo:

```
CuatroDatos d;
 float x[7]={2.0,3.0,4.0,6.0,1.0,8.0,23.0};
 float y[7]={4.0,6.0,9.0,1.0,6.0,5.0,12.0};
 d.a=x;
 d.b=y;
 d.i=2;
 d.n=5;
 float r=calculoSept2011(d);
```

Apartado c) El tamaño del problema es ***n-i+1***

Apartato d) Los caso mejor y peor son el mismo disponer de dos vectores y hacer el recorrido desde i hasa n, y la complejidad del caso mejor y peor por tanto es la misma:

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

Por tanto $T(n)=k1*n+k2 \in \Theta(n)$. Es un algoritmo lineal