Ejercicio 1

1. Dado un número de tipo ***long int*** se desea implementar en el lenguaje C la función que suma todos sus dígitos. Por ejemplo el número 7104546031 devolvería como resultado 31. El prototipo de la función podría ser:

   int sumaDigitos(long int n)

SE PIDE:

1. Diseñar las correspondientes funciones
   a. recursiva lineal no final
   b. recursiva final e indicar cómo sería la llamada inicial a dicha función.
2. ¿Cuál es el tamaño del problema?
3. ¿Calcular T(n) para el caso mejor y peor de cada una de las funciones?

Puntuación: 3.33 puntos

SOLUCIÓN

// Importante: La precondición de ambas funciones es $n \geq 0$

1.a. Recursivo Lineal No final

```
int sumaDigitos(long int n){
   int r;
   if(n < 10)
      r= n;
    else
       r= n%10+sumaDigitos(n/10);
   return(r);

}
```

1.b. Recursivo Final

```
int sumaDigitos(long int n){
        int r;
        r=sumaDigitosAux(n,0);
        return(r);

}
int sumaDigitosAux(long int n, int x)
{
     int r;
    if(n < 10)
          r = n+x;
    else {
      r= sumaDigitosAux(n/10,x+n%10);
   }
    return(r);
}
```

2. **Tamaño n = número de cifras que tiene el número de tipo long**

3. **Para las dos funciones no existe caso mejor ni peor, solamente hay un caso y su T(n) = T(n-1)+k de acuerdo con la fórmula si a =1 entonces $T(n)=c_1n+c_2$**

EJERCICIO_2

```
private void voraz() {
     while (problema.isAlternativa(estado)) {
          A alternativa = problema.getAlternativa(estado);
          estado.add(alternativa);
     }
     solucion = estado.getSolucion();
}
```

VZ	
Problema Mudanza	
Técnica: Voraz	
Tamaño: n = mueblesSinProcesar().size()	
Propiedades Compartidas	***listaMuebles**:List<Muebles>*
Propiedades del Estado	***mueblesSinProcesar:** List<Mueble>* ***mueblesTransportados**:* *List<Camion<Mueble>>*
Solución: Integer	
Alternativa: M_i = { M_i ∈ *mueblesSinProcesar* } (primer mueble que cumpla esta condición)	
Estado inicial: *mueblesSinProcesar = listaMuebles* *mueblesTransportados = new Lista<Camion<Mueble>>()* *mueblesTransportados.add(new Camion<Mueble>())*	

```
public boolean add(Mueble m) {
     Camion hueco = null;
     for (Camion c: mueblesTransportados){
          if (c.hayCapacidad(m))
               hueco = c;
     }
     if (hueco == null){
          hueco = new Camion();
          mueblesTransportados.add(hueco);
     }
     hueco.incluirMueble(m);
     this.mueblesSinProcesar.remove(m);
     return true;
}

public List<Camion> getSolucion() {
     return this.mueblesTransportados;
}

public Mueble getAlternativa(EstadoMudanza e) {
     return e.getMueblesSinProcesar().get(0);
}

public boolean isAlternativa(EstadoMudanza e) {
     return !e.getMueblesSinProcesar().isEmpty();
}
```

<table>
<tr><th colspan="2">FICHA</th></tr>
<tr><th colspan="2">Ejercicio de funciones con comportamiento similar</th></tr>
<tr><td colspan="2">Técnica: Divide y Vencerás sin memoria</td></tr>
<tr><td colspan="2">Representación: Cada problema un conjunto de parámetros</td></tr>
<tr><td colspan="2">Tamaño: (x2 – x1) / epsilon</td></tr>
<tr><td>Propiedades Compartidas</td><td>f1, Funcion
f2, Funcion
epsilon, Double</td></tr>
<tr><td>Propiedades Individuales</td><td>X1, Double [f1.getInicioX(), f1.getFinX()]
X2, Double [f2.getInicioX(), f2.getFinX()]</td></tr>
<tr><td colspan="2">Solucion: Boolean</td></tr>
<tr><td colspan="2">compSimilar (f1, f2, epsilon) = compSimDyV(f1.getInicioX(),f1.getFinX())

compSimDyV (x1,x2) =
signoIncY1(x1,x2) = signoIncY2(x1,x2) (x2 – x1) <= epsilon
compSimDyV(x1,k) && compSimDyV(k,x2) (x2 – x1) > epsilon

k = (x1 + x2)/ 2</td></tr>
<tr><td>Ecuación de Recurrencia</td><td>T(n) = 2T(n/2) + k</td></tr>
<tr><td>Complejidad</td><td>O(n)</td></tr>
</table>

```
Boolean  compSimDyV (Double x1, Double x2){
        Boolean res;
        If (x2-x1 <= epsilon){
                res = signoIncY(x1,x2);
        }else {
                Double k = (x1 + x2)/ 2
                Boolean r1 = compSimDyV(x1, k);
                Boolean r2 = compSimDyV(k, x2);
                res = r1 && r2;
        }
        return res;
}

Boolean signoIncY(Double x1, Double x2){
return
        (   (f1.getValue(x1) > f1.getValue(x2))  &&       (f2.getvalue(x1) > f2.getValue(x2))  )
                ||
        (   (f1.getValue(x1) < f1.getValue(x2))  &&  (f2.getvalue(x1) < f2.getValue(x2))   )
}
```

Otra opción más optima sería:

```
Boolean  compSimDyV (Double x1, Double x2){
        Boolean res;
        If (x2-x1 <= epsilon){
                res = signoIncY(x1,x2);
        }else {
                k = (x1 + x2)/ 2
                res = compSimDyV(x1, k) && compSimDyV(k, x2);
        }
        return res;
}
```

```
private Boolean dYV(ProblemaDyV<Boolean> p){
                    Boolean s;
                    if( p.esCasoBase()){
                              s = p.getSolucionCasoBase();
                    }else {
                       int numeroDeSubProblemas = p.getNumeroSubProblemas();
                      Boolean[] soluciones = Utiles.creaArray(tipoSolucion,numeroDeSubProblemas);
                     for(int i = 0; i < numeroDeSubProblemas; i++){
                              s = dYV(p.getSubProblema(i));
                              soluciones[i]=s;
                     }
                     s = p.combinaSoluciones(soluciones);
             }
            return s;
 }

Boolean esCasoBase(){
          return (x2-x1 <= épsilon);
}

Boolean getSolucionCasoBase(){
          return (   (f1.getValue(x1) > f1.getValue(x2))  &&       (f2.getvalue(x1) > f2.getValue(x2))  )
                    ||
          (   (f1.getValue(x1) < f1.getValue(x2))  &&  (f2.getvalue(x1) < f2.getValue(x2))   ) ;
}

Int getNumeroSubProblemas(){
          return 2;
}

ProblemaDyV<Boolean>  getSubproblema(int i){
          ProblemaDyV<Boolean> p;
          Double k = x1 + x2 / 2;
          switch(i){
                    case 0:
                    p = new ProblemaDyV(x1,k);
                     break;
                    case 1:
                    p = new ProblemaDyV(k,x2);
                    break;
                    default:
          }
          return p;
}

Boolean combinaSoluciones(Boolean[] soluciones){
          return (soluciones[0] && soluciones[1]);
}
```