1. Dado el siguiente procedimiento

```
proc dic07(a:Array [1..N] de Entero, b: Entero)
{Pre: 1≤b ≤ N }
    var
        i,j: Entero
    alg
        i:=b
        mientras  i >= 1
            desde j=1 hasta i
               a[i]:=j+a[j]
            fdesde
            i:=i/3
        fmientras
    fin
```

a) Cómo escogería el tamaño del problema para este procedimiento
b) Indique el caso mejor y peor para este procedimiento.
c) Determinar el tiempo de ejecución T(n) para dicho procedimiento.

Respuesta/Justificación:

a) El tamaño del problema depende del parámetro b, que determina la cantidad de veces que se realiza el bucle más externo e indirectamente el bucle interno. Por tanto **n=b**.
b) No existe caso mejor ni peor que otros para este procedimiento pues desde un punto de vista cualitativo no existen entradas que determinen que tarde más o menos.
c) Considerando las operaciones elementales

$$T(n) = 1+ \sum_{i=1}^{\lfloor \log_3 n \rfloor +1} (1+ 1 + \sum_{j=1}^{i} (1+4+2) +1+2)+1= 2+ \sum_{i=1}^{\lfloor \log_3 n \rfloor +1} (5+ 7i) = 2 + 5(\lfloor \log_3 n \rfloor +1)+$$

$$+ 7/2(1+ \lfloor \log_3 n \rfloor +1)(\lfloor \log_3 n \rfloor +1)$$

2. Se desea diseñar un algoritmo de divide y vencerás para tratar un array de ***N*** elementos. **Determinar el orden de complejidad de las siguientes opciones**:

a) Dividir el array en dos partes iguales, hacer dos llamadas recursivas y combinar las soluciones parciales con un algoritmo lineal.
b) Dividir el array en 4 partes iguales, hacer tres llamadas recursivas y combinar las soluciones parciales con un algoritmo de coste constante.
c) Dividir en array en dos partes una con la tercera parte y el otro con las dos terceras partes, y hacer una llamada recursiva y combinar las soluciones parciales con un algoritmo de coste constante.

Respuesta/Justificación:

El ejercicio se puede realizar usando las fórmulas vistas en clase o calculando las soluciones a las ecuaciones recurrentes que determinan cada caso. Se hará por las fórmulas:

$a < b^K$ entonces $O(n^k)$
$a = b^K$ entonces $O(n^k \log n)$
$a > b^K$ entonces $O(n^{\log_b a})$

a) En este caso a=2, b=2 y k=1 entonces $a=b^k$ y por tanto $O(n^k \log n)$
b) En este caso a=3, b=4 y k=0 entonces $a>b^k$ y por tanto $O(n^{\log_4 3})$
c) En este caso a=1, b=3 (caso mejor) o b=3/2 (caso peor) y k=0 entonces $a=b^k$ y por tanto O(log n)

Problema de Prácticas – Torneo deportivo mediante Play-Offs

En la liga ACB de baloncesto, para decidir el campeón de cada año, tras la competición regular se juega un torneo mediante la modalidad de Play-Offs. Esta modalidad consiste en varias rondas eliminatorias donde los equipos se enfrentan entre ellos según su clasificación en la competición regular. De esta forma, dado un número N de equipos (con N potencia de 2), el primer clasificado jugará contra el último, el segundo contra el penúltimo, y así sucesivamente. Una vez resuelta esta tanda de eliminatorias, los siguientes emparejamientos se realizarán de la misma forma, enfrentando a los ganadores de la eliminatoria anterior.

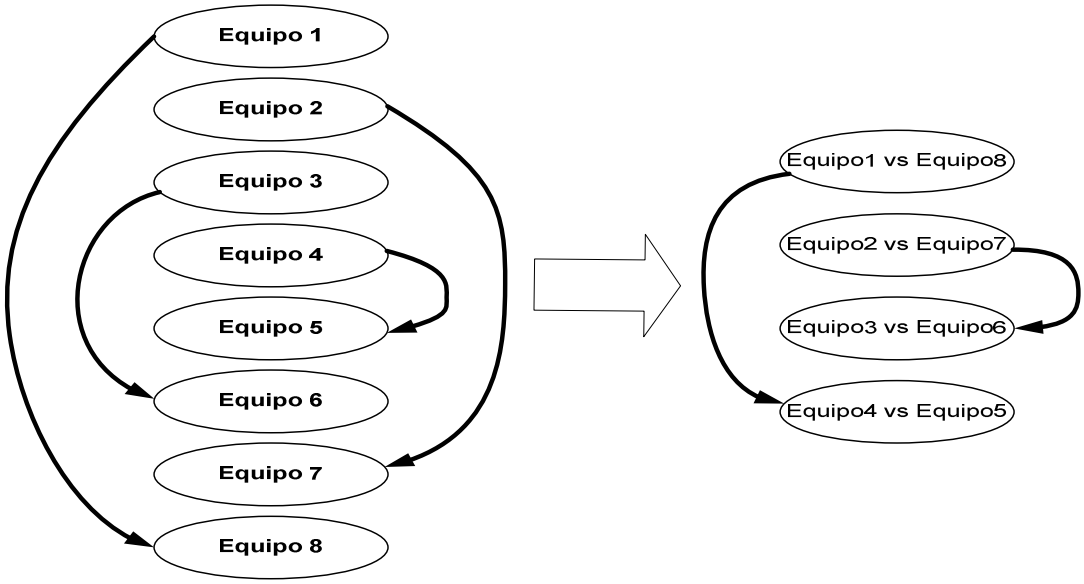

Figura 1: Ejemplo del mecanismo de Play-Offs

Como vemos en la figura 1, tras la primera tanda de eliminatorias, el ganador del partido entre el Equipo1 y el Equipo8 se enfrentará al ganador del partido entre el Equipo4 y el Equipo5. Es decir, se repite la misma mecánica que para la primera tanda de eliminatorias.

Dado un número de equipos N (que supondremos potencia de 2) queremos mostrar, mediante una cadena de texto, una representación del cuadro de enfrentamientos entre los N equipos. En la figura 2 tenemos un ejemplo para N=8.

SE PIDE:

- Completar el código de la clase `PlayOffsDyV`, para resolver este problema mediante la técnica de divide y vencerás.

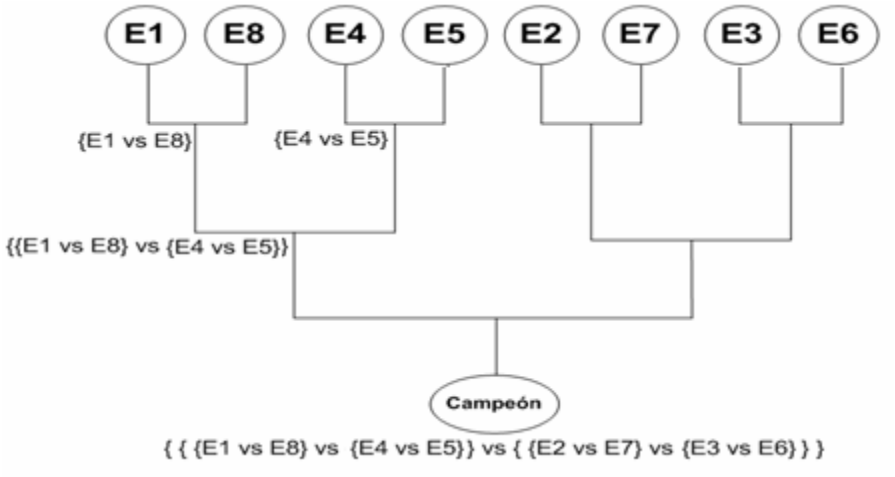

Figura 2: Ejemplo de un cuadro de enfrentamientos, y su representación como String.

A TENER EN CUENTA:

- El caso base se alcanza cuando sólo tenemos dos equipos. Habrá que construir una cadena del tipo: {EquipoX vs EquipoY}
- Para dividir el problema, generamos dos subproblemas con la mitad de equipos que el original. **La división se hará repartiendo los equipos de dos en dos en cada subproblema. Habrá que tratar aparte el primer y último elemento del problema original, que irán ambos en el primer subproblema**. Un ejemplo para 8 equipos lo tenemos en la figura 3.
- La operación de combinación tan solo tendrá que unir las dos subcadenas solución de cada uno de los grupos, con el formato mostrado en la figura 2.

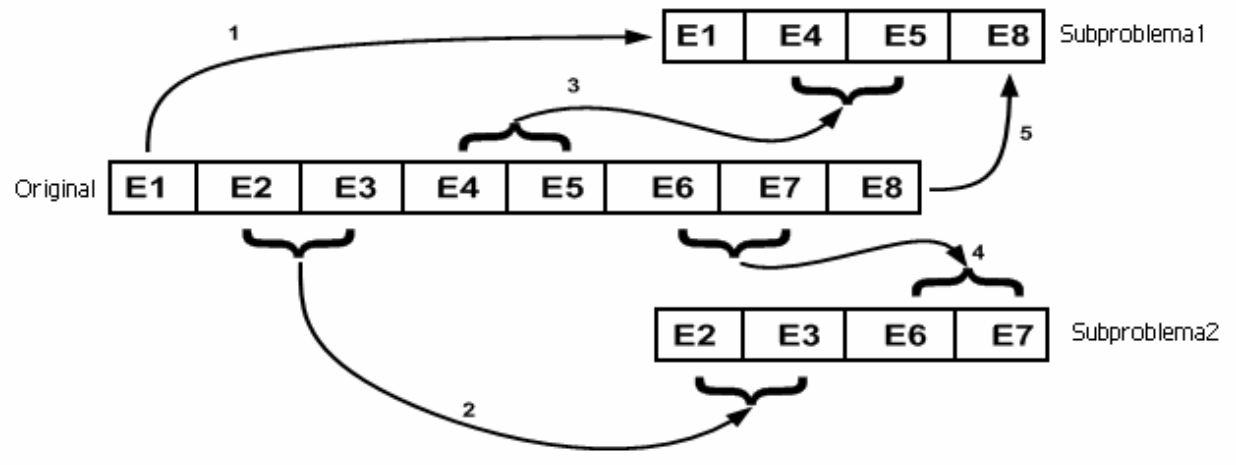

Figura 3: Ejemplo de división de un problema en dos subproblemas

```
public class ProblemaPlayOffs {
      private List<String> equipos;

      //Suponemos que el número de elementos de la lista eqs es potencia de 2
      public ProblemaPlayOffs(List <String> eqs) {
            equipos = eqs;
      }
}

public class PlayOffsDyV extends EsquemaDYV implements EstrategiaSolucion {

    private List <String> equipos;
    private Problema pb;
    private Solucion sol;

      public PlayOffsDyV(ProblemaPlayOffs pcl) {
            super();
            equipos = pcl.getEquipos();
            pb = new ProblemaEliminatoria(equipos);
            sol = new SolucionEliminatoria();
      }


      public void procesamientoFinal() {}
      public void procesamientoInicial() {}

      public void solucion() {
            sol = this.dYV(pb);


      }

      protected boolean esCasoBase(Problema p) {
            ProblemaEliminatoria pc = (ProblemaEliminatoria)p;

            return (pc.clasificados.size() == 2);
      }

      protected Solucion resuelveCasoBase(Problema p) {
            SolucionEliminatoria sol = new SolucionEliminatoria();
            ProblemaEliminatoria pc = (ProblemaEliminatoria)p;

            sol.enfrentamientos = "{" + pc.clasificados.get(0) +
                  "  vs  " + pc.clasificados.get(1) + "}";

            return sol;


      }

      protected Problema[] divide(Problema p) {
         ProblemaEliminatoria pc = (ProblemaEliminatoria)p;

         ProblemaEliminatoria[] problemas = new ProblemaEliminatoria[2];
         problemas[0] = new ProblemaEliminatoria();
         problemas[1] = new ProblemaEliminatoria();

         int i, bandera = 1;
```

```
      //Trato el primer elemento aparte
      problemas[0].clasificados.add(pc.clasificados.get(0));


      //El resto de elementos siguen la regla comentada en el enunciado
      for(i = 1; i < pc.clasificados.size()-1; i = i+2) {
        problemas[bandera].clasificados.add(pc.clasificados.get(i));
        problemas[bandera].clasificados.add(pc.clasificados.get(i+1));

        if(bandera == 0)
            bandera = 1;
        else
            bandera = 0;
      }

      //Trato el último elemento aparte
      problemas[0].clasificados.add(pc.clasificados.get(i));


      return problemas;


    }

    protected Solucion combina(Solucion[] s) {

      SolucionEliminatoria sol = new SolucionEliminatoria();

      sol.enfrentamientos = "{" +
            ((SolucionEliminatoria)s[0]).enfrentamientos + "  vs  " +
            ((SolucionEliminatoria)s[1]).enfrentamientos + "}";

      return sol;


    }

  class ProblemaEliminatoria extends Problema {
      List <String> clasificados;

      ProblemaEliminatoria(List <String> eq) {
          clasificados = new ArrayList<String>();
          for(int i = 0; i < eq.length; i++) {
                clasificados.add(eq.get(i));
          }
      }

      ProblemaEliminatoria() {
         clasificados = new ArrayList<String>();
      }
  }

  class SolucionEliminatoria extends Solucion {
      String enfrentamientos;

      SolucionEliminatoria() {
         enfrentamientos = "";
      }
  }
}
```

Tiempo: 45 minutos