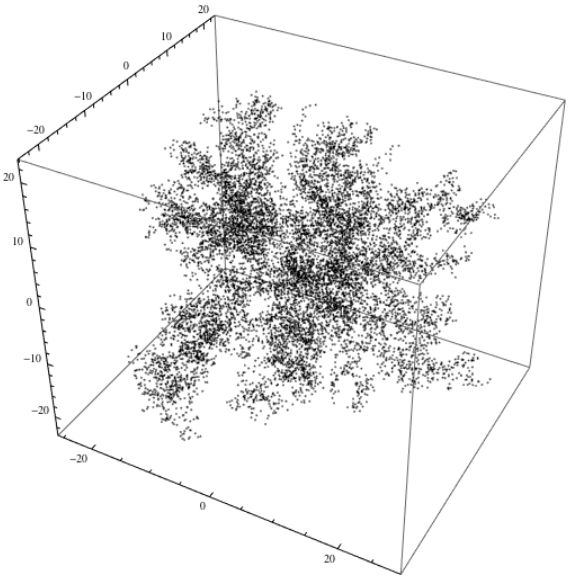
Ejercicio 3: Divide y Vencerás

En un espacio tridimensional, tenemos ubicados una nube de puntos definidos por tres coordenadas (x, y, z) respecto a un punto origen (0, 0, 0) dentro de este espacio.

Se desean almacenar estos puntos en un array a partir de un orden establecido. Este orden debe estar basado en la suma de sus coordenadas de mayor a menor valor de manera que en las primeras posiciones se encontrarán los puntos más alejados del origen y en las últimas los más cercanos.

Por tanto y, a partir de este modelado y siguiendo el esquema de la técnica de divide y vencerás, se pide resolver las siguientes cuestiones.



Nube de puntos tridimensional

a) Implementar la clase ComparadorPuntos necesaria para realizar la ordenación del array que almacena los puntos.

b) Implementar el siguiente método siguiendo la técnica de divide y vencerás. Este método recibe un array ordenado a partir del orden definido anteriormente y un punto y devuelve la posición en la que se encuentra el punto buscado o -1 en el caso de que no exista.

int devuelvePosicion(Punto[] array, Punto p)

c) Implementar un método principal que dado un array que contiene los puntos desordenados y un punto, haga uso de los apartados anteriores para ordenarlo y encontrar el punto deseado.

d) Indicar el tamaño del problema y el orden de complejidad del algoritmo **de manera justificada**.

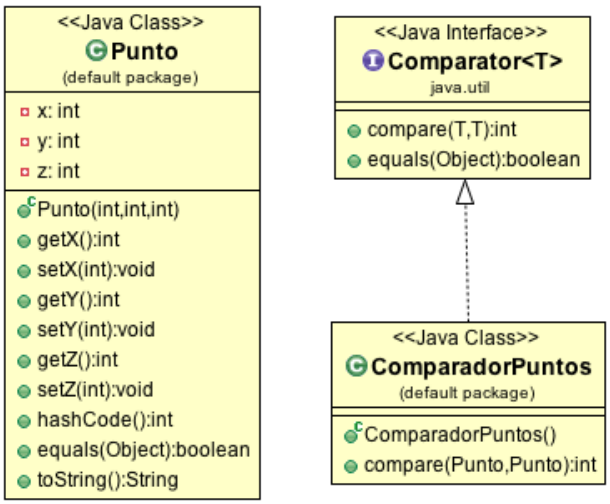


Diagrama UML

Tiempo estimado: 45 minutos.

Solución:

a)
```
public class ComparadorPuntos implements Comparator<Punto> {
    public ComparadorPuntos() {
    }

    public int compare(Punto p1, Punto p2) {
        int suma1 = Math.abs(p1.getX())
            + Math.abs(p1.getY())
            + Math.abs(p1.getZ());
        int suma2 = Math.abs(p2.getX())
            + Math.abs(p2.getY())
            + Math.abs(p2.getZ());

        return suma1 - suma2;
    }
}
```

b)
```
int devuelvePosicion(Punto[] array, Punto p) {
    return devuelvePosicionAux(array, p, 0, array.length);
}

int devuelvePosicionAux(Punto[] array, Punto p, int i, int j) {
    int res;

    if(i > j) {
        res = -1;
    }
    else {
        int mitad = (i + j) / 2;

        if(array[mitad].compareTo(p) == 0) {
            res = mitad;
        }
        else if(array[mitad].compareTo(p) > 0) {
            res = devuelvePosicionAux(array, p, mitad + 1, j);
        }
        else {
            res = devuelvePosicionAux(array, p, i, mitad - 1);
        }
    }

    return res;
}
```

c)
```
public static void main(String[] args) {
    Comparator<Punto> comparador = new ComparadorPuntos();
    Comparator<Punto> comparadorInvertido =
        Collections.reverseOrder(comparador);

    List<Punto> listaPuntos = Lists.newArrayList(puntos);

    Collections.sort(listaPuntos, comparadorInvertido);

    int posicion = devuelvePosicion((Punto[]) listaPuntos.toArray(), punto);
```

```
        if(posicion == -1) {
                System.out.println("El punto no se encuentra en el array.");
        }
        else {
                System.out.println("El punto se encuentra en la posición " + (posición
                        + 1));
        }
    }
```

d) El tamaño del problema es el tamaño del array y la complejidad es de orden $\theta(\log n)$.