Estimación Orden de Crecimiento:

Para Split():

La operación elemental representativa para este método es:

```
while (rapido != null && rapido.siguiente != null) {
    rapido = rapido.siguiente.siguiente;
    lento = lento.siguiente;
}
```

El método utiliza 2 punteros para recorrer la lista doblemente enlazada, “rápido” y “lento”
rápido avanza 2 veces por cada iteración, del bucle termina cuando rápido llega al final de la lista, por lo tanto la operación dentro del bucle (O.E.R.) se ejecutará n/2 siendo n el tamaño del arreglo.

Finalmente el orden de crecimiento es lineal O(N)

Para Merge():

Este método tiene 3 bucles while:

```
while (actual1 != null && actual2 != null) {
    if (actual1.persona.getNombres().compareTo(actual2.persona.getNombres()) <=
0) {
        resultado.agregarAlFinal(actual1.persona);
        actual1 = actual1.siguiente;
    } else {
        resultado.agregarAlFinal(actual2.persona);
        actual2 = actual2.siguiente;
    }
}
```

Por cada iteración, uno de los punteros actual 1 o actual 2 y avanza al siguiente nodo, el número de iteraciones está dado por la longitud de l1 y l2 ya que el bucle termina cuando ambos lleguen al final de la lista, entonces el número de veces que se ejecutará el bucle será n+m siendo n la longitud de l1 y m la longitud de l2:

```
while (actual1 != null) {
    resultado.agregarAlFinal(actual1.persona);
    actual1 = actual1.siguiente;
}

while (actual2 != null) {
    resultado.agregarAlFinal(actual2.persona);
    actual2 = actual2.siguiente;
}
```

Para cada uno de estos bucles se delimita la iteración hasta que los punteros lleguen al final de la lista con la que iteran siendo así n y m las veces que se ejecuta cada bucle

Entonces el orden de crecimiento de este método será O(N + M)

Para MergeSort():

```
ListaDobleEnlazada[] partes = split();
ListaDobleEnlazada l1 = partes[0];
ListaDobleEnlazada l2 = partes[1];

ListaDobleEnlazada listaOrdenada1 = l1.mergesort();
ListaDobleEnlazada listaOrdenada2 = l2.mergesort();
```

En este método se hace un llamado al split para separar las listas y posteriormente un llamado recursivo al mergesort, como la lista se divide a la mitad en cada

nivel, hay log N niveles de recursión

```
return merge(listaOrdenada1, listaOrdenada2);
```

Finalmente se hace un llamado a merge para combinar las mitades ya ordenadas con 2 sublistas de tamaño n/2, por lo tanto tiene un tiempo lineal O(N), haciendo que el orden de crecimiento total del método mergesort sea O(NlogN)