```
1
2
3
4
5
6   #define TAM_LISTA 8
7
8   enum {
9     ASCENDENTE = 0,
10    DESCENDENTE = 1
11  };
12
13  void print(int list[]){ // funcion que imprime los valores de una lista
14    printf("\n[ ");
15    for (size_t i = 0; i < TAM_LISTA; i++) {
16      printf("%d ", list[i]);
17      if ( !(i == TAM_LISTA-1) ){
18        printf(", ");
19      }
20    }
21    printf(" ]\n");
22  }
23
24  void Merge(int lista[] , int aux[] , int direccion , size_t primero, size_t ultimo){
25    size_t x0, x1, x2, x3, mid, index = primero; // creamos los valores a usar como indices
26    mid = (primero + ultimo)/2; // ubicamos el punto medio de la lista recibida
27    x0 = index; // inicio
28    x1 = mid; // primera mitad
29    x2 = mid+1; // segunda mitad
30    x3 = ultimo; // fin del arreglo
31
32    while(x0 <= x1 && x2 <= x3){ // mientras los indices no salgan del limite
33      if(lista[x0] > lista[x2] && direccion == DESCENDENTE){
34        aux[index] = lista[x0];// cambiamos el indice con el valor mayor
35        ++index;
36        ++x0;
37      }else if(lista[x0] < lista[x2] && direccion == ASCENDENTE){
38        aux[index] = lista[x0]; // cambiamos el indice con el valor menor
39        ++index;
40        ++x0;
41      }else{
42        aux[index] = lista[x2]; // este bloque de codigo se ejecuta en caso de que
43        ++index; // el valor de las anteriores comprobaciones no corresponda
44        ++x2;
45      }
46    }
47
48    while(x0 <= x1){ // nuevamente recorremos y acomodamos
49      aux[index] = lista[x0];
50      ++index;
51      ++x0;
52    } // la primera mitad
53
54    while(x2 <= x3){ // igual con la segunda mitad
55      aux[index] = lista[x2];
56      ++index;
57      ++x2;
58    }
59
60    for (size_t i = primero; i < ultimo; ++i) { // por ultimo
61      lista[i] = aux[i]; // regresamos el valor del auxiliar a la lista principal
62    }
63  }
64
65  void MergeSort(int lista[] , int aux[] , int direccion , size_t primero , size_t ultimo){
66    if(primero < ultimo){ // checamos que los valores de los indices sean diferentes
67      size_t mitad = (primero + ultimo)/2; // creamos el indice de la mitad
68      // de izquierda a derecha va a checar la primera mitad e ira entrando nuevamente
        hasta llegar a un solo elemento
```

```
69        MergeSort(lista, aux, direccion , primero, mitad); // partimos la lista a la mitad
          derecha
70        // de izquierda a derecha va a checar la segunda mitad y hara lo mismo
71        MergeSort(lista , aux , direccion , mitad+1 , ultimo); // partimos la lista a la
          mitad izquierda
72        // al llegar a un valor que se deba ordenar llama la funcion y ordena esa parte
73        Merge(lista , aux, direccion , primero , ultimo); // mezclamos el resultado y
          regresamos a la funcion de invocacion
74      }
75    }
76
77    void Ordenar(int lista[] , int num_elems , int direccion){
78      int aux[num_elems]; // creamos una lista auxiliar y la enviamos tambien
79      MergeSort(lista, 0, num_elems-1, aux, direccion);
80    }
81
82    int main(){
83      int lista[TAM_LISTA] = {12,9,23,2,15,19,20,5};
84
85      print(lista); // imprimimos la lista original
86
87      Ordenar(lista , TAM_LISTA , DESCENDENTE); // ordenamos de manera
88
89      print(lista); // imprimimos la lista final
90
91      return 0;
92    }
93
```