```cpp
#include <iostream>
#include <vector>
#include <cstdlib>

using namespace std;

template <typename T>
void merge(vector<T> &v, unsigned int l, unsigned int m, unsigned int r, vector
<T> &h){
    cout << "Mergen van v["<<l<<","<<m<<"] en v[" << m+1 << "," << r << "]" <<
endl;
    // Voeg de deeltabellen v[l,m] en v[m+1,r] (grenzen inclusief) samen
    // tot één gesorteerde deeltabel v[l,r].

    // We nemen een kopie van de elementen v[l,m]
    for (unsigned int i = l; i <= m; i++){
        h[i-l] = v[i];
    }

    unsigned int i = l;  // huidige index in eerste deeltabel
    unsigned int j = m+1;// huidige index in tweede deeltabel
    unsigned int loc = l;// huidige index in resultaattabel

    // Zolang loc <= r zijn er nog elementen die een plaats moeten krijgen
    // Zolang i <= m zijn er nog elementen uit de eerste tabel die een
    // plaats moeten krijgen.
    // Zolang j <= r zijn er nog elementen uit de tweede tabel die een
    // plaats moeten krijgen.
    while (loc <= r && i <= m && j <= r){
        if (v[j] < h[i-l]){
            v[loc] = v[j];
            j++;
        }
        else{
            v[loc] = h[i-l];
            i++;
        }
        loc++;
    }
    // Hier komen we als loc > r: alle elementen zijn geplaatst, dan
    // moeten de volgende lussen niet uigevoerd worden, of als alle
    // elementen uit één deeltabel geplaatst zijn maar er nog elementen
    // in de andere deeltabel zijn. We kunnen nu gewoon de andere deeltabel
    // overlopen en ze direct in de resultaattabel plaatsen.
    while (loc <= r && i <= m){
        v[loc] = h[i-l];
        loc++;
        i++;
    }
    while (loc <= r && j <= r){
        v[loc] = v[j];
        loc++;
        j++;
    }
}

template <typename T>
void mergesort(vector<T> &v, unsigned int l, unsigned int r, vector<T> &h){
    cout << "Sorteren van tabel v["<<l<<","<<r<<"]" << endl;
    // Sorteer de elementen in vector v met index = l tot en met index = r
    if (l < r){
        // Als er meerdere elementen tussen l en r zitten
        // splitsen we de tabel in twee en sorteren we elk deel afzonderlijk
        unsigned int m = l + (r-l)/2;
```

```
        mergesort(v, l, m, h);
        mergesort(v, m+1, r, h);
        // De tabellen v[l,m] en v[m+1,r] (grenzen inclusief) zijn gesorteerd
        // Nu moeten we ze samenvoegen.
        merge(v,l,m,r,h);
    }
}

template <typename T>
void mergesort(vector<T> &v ){
    // Maak een hulpvector aan.
    vector<T> h(v.size()/2);
    // Start het sorteren op de volledige tabel
    mergesort(v,0, v.size()-1,h);
}


/*
0 0 8 9 2 9 8 0 1 6
Sorteren van tabel v[0,9]
Sorteren van tabel v[0,4]
Sorteren van tabel v[0,2]
Sorteren van tabel v[0,1]
Sorteren van tabel v[0,0]
Sorteren van tabel v[1,1]
Mergen van v[0,0] en v[1,1]
Sorteren van tabel v[2,2]
Mergen van v[0,1] en v[2,2]
Sorteren van tabel v[3,4]
Sorteren van tabel v[3,3]
Sorteren van tabel v[4,4]
Mergen van v[3,3] en v[4,4]
Mergen van v[0,2] en v[3,4]
Sorteren van tabel v[5,9]
Sorteren van tabel v[5,7]
Sorteren van tabel v[5,6]
Sorteren van tabel v[5,5]
Sorteren van tabel v[6,6]
Mergen van v[5,5] en v[6,6]
Sorteren van tabel v[7,7]
Mergen van v[5,6] en v[7,7]
Sorteren van tabel v[8,9]
Sorteren van tabel v[8,8]
Sorteren van tabel v[9,9]
Mergen van v[8,8] en v[9,9]
Mergen van v[5,7] en v[8,9]
Mergen van v[0,4] en v[5,9]
0 0 0 1 2 6 8 8 9 9
*/
```