```
//=======================================
// Auteur:   © 2017 Cliff Huylebroeck
// Compiler: MSL
// Project:  TestQuickSort
// Bestand:  TestQuickSort.h
// Implemen: TestQuickSort.cp
//=======================================

#pragma once

int main();

void DoeAlles();

void Sorteer(z4 aElement,
             p_z4 elementen);

void SorteerOplopend(z4 aElement,
                     p_z4 elementen);

void SorteerAflopend(z4 aElement,
                     p_z4 elementen);

void SorteerIdentiek(z4 aElement,
                     p_z4 elementen);

void SorteerWillekeurig(z4 aElement,
                        p_z4 elementen);

void SorteerOplopendPointers(z4 aElement,
                             p_z4 elementen);

void SorteerAflopendPointers(z4 aElement,
                             p_z4 elementen);

void SorteerIdentiekPointers(z4 aElement,
                             p_z4 elementen);

void SorteerWillekeurigPointers(z4 aElement,
                                p_z4 elementen);

void VoegSamen(z4 aElement,
               p_z4 elementen);
```

```
//=======================================
// Auteur:   © 2017 Cliff Huylebroeck
// Compiler: MSL
// Project:  TestQuickSort
// Bestand:  TestQuickSort.cp
//=======================================

#include "TestQuickSort.h"

#include "Controle.h"
#include "MergeSort.h"
#include "QuickSort.h"
#include "QuickSortPointers.h"
#include "TestData.h"

int main()
{
    cout << "Start van het programma.";
    cout << endl;

    cout << "Typ Command-Q om te stoppen.";
    cout << endl;

    ::ProfilerInit(0,
                   4,
                   5000,
                   100);

    DoeAlles();

    ::ProfilerDump("\pProfiler");
    ::ProfilerTerm();

    cout << "Einde van het programma.";
    cout << endl;

    return 0;
}

void DoeAlles()
{
    Quickdraw::InitialiseerRandom();
    c_z4 aElement=1<<20;
    cp_z4 elementen=new z4[aElement];
    if (!elementen)
```

```
        {
        Exit("Te weinig geheugen.");
        }
    MergeSort::Init_klasse(aElement);

    Sorteer(aElement,
            elementen);

    delete [] elementen;
    MergeSort::Exit_klasse();
}

void Sorteer(c_z4 aElement,
             cp_z4 elementen)
{
    SorteerOplopend(aElement,
                    elementen);
    SorteerAflopend(aElement,
                    elementen);
    SorteerIdentiek(aElement,
                    elementen);
    SorteerWillekeurig(aElement,
                       elementen);
    SorteerOplopendPointers(aElement,
                            elementen);
    SorteerAflopendPointers(aElement,
                            elementen);
    SorteerIdentiekPointers(aElement,
                            elementen);
    SorteerWillekeurigPointers(aElement,
                               elementen);
    VoegSamen(aElement,
              elementen);
}

void SorteerOplopend(c_z4 aElement,
                     cp_z4 elementen)
{
    TestData::MaakOplopend(aElement,
                           elementen);
    QuickSort::Sorteer(aElement,
                       elementen);
    Controle::Controleer(aElement,
                         elementen);
}
```

```
void SorteerAflopend(c_z4 aElement,
                     cp_z4 elementen)
{
    TestData::MaakAflopend(aElement,
                           elementen);
    QuickSort::Sorteer(aElement,
                       elementen);
    Controle::Controleer(aElement,
                         elementen);
}

void SorteerIdentiek(c_z4 aElement,
                     cp_z4 elementen)
{
    TestData::MaakIdentiek(aElement,
                           elementen);
    QuickSort::Sorteer(aElement,
                       elementen);
    Controle::Controleer(aElement,
                         elementen);
}

void SorteerWillekeurig(c_z4 aElement,
                        cp_z4 elementen)
{
    TestData::MaakWillekeurig(aElement,
                              elementen);
    QuickSort::Sorteer(aElement,
                       elementen);
    Controle::Controleer(aElement,
                         elementen);
}

void SorteerOplopendPointers(c_z4 aElement,
                             cp_z4 elementen)
{
    TestData::MaakOplopend(aElement,
                           elementen);
    QuickSortPointers::Sorteer(aElement,
                               elementen);
    Controle::Controleer(aElement,
                         elementen);
}
```

```
void SorteerAflopendPointers(c_z4 aElement,
                             cp_z4 elementen)
{
    TestData::MaakAflopend(aElement,
                           elementen);
    QuickSortPointers::Sorteer(aElement,
                               elementen);
    Controle::Controleer(aElement,
                         elementen);
}

void SorteerIdentiekPointers(c_z4 aElement,
                             cp_z4 elementen)
{
    TestData::MaakIdentiek(aElement,
                           elementen);
    QuickSortPointers::Sorteer(aElement,
                               elementen);
    Controle::Controleer(aElement,
                         elementen);
}

void SorteerWillekeurigPointers(c_z4 aElement,
                                cp_z4 elementen)
{
    TestData::MaakWillekeurig(aElement,
                              elementen);
    QuickSortPointers::Sorteer(aElement,
                               elementen);
    Controle::Controleer(aElement,
                         elementen);
}

void VoegSamen(c_z4 aElement,
               cp_z4 elementen)
{
    TestData::MaakWillekeurig(aElement,
                              elementen);
    MergeSort::Sorteer(aElement,
                       elementen);
    Controle::Controleer(aElement,
                         elementen);
}
```

```
//=======================================
// Auteur:   © 2017 Cliff Huylebroeck
// Compiler: MSL
// Project:  TestQuickSort
// Bestand:  QuickSort.h
// Implemen: QuickSort.cp
//=======================================

#pragma once

class QuickSort
{
public:
    // Globale functies.
    static void Sorteer(z4 aElement,
                        p_z4 elementen);
private:
    static void Sorteer(z4 eerste,
                        z4 laatste,
                        p_z4 elementen);

    // Verboden.
    QuickSort();
};
```

```
//=======================================
// Auteur:   © 2017 Cliff Huylebroeck
// Compiler: MSL
// Project:  TestQuickSort
// Bestand:  QuickSort.cp
//=======================================

#include "QuickSort.h"

void QuickSort::Sorteer(c_z4 aElement,
                        cp_z4 elementen)
{
    if (aElement<2)
        {
        return;
        }
    Sorteer(0,
            aElement-1,
            elementen);
}

#pragma profile off

void QuickSort::Sorteer(c_z4 eerste,
                        c_z4 laatste,
                        cp_z4 elementen)
{
    z4 i=eerste;
    z4 j=laatste;
    c_z4 m=(i+j)>>1;
    c_z4 pivot=elementen[m];
    do
        {
        while (elementen[i]<pivot)
            {
            i++;
            }
        while (pivot<elementen[j])
            {
            j--;
            }
        if (i<=j)
            {
            c_z4 r=elementen[i];
            elementen[i]=elementen[j];
```

```
            elementen[j]=r;
            i++;
            j--;
            }
        }
    while (i<j);
    if (eerste<j)
        {
        Sorteer(eerste,
                j,
                elementen);
        }
    if (i<laatste)
        {
        Sorteer(i,
                laatste,
                elementen);
        }
}
```

```
//=======================================
// Auteur:   © 2017 Cliff Huylebroeck
// Compiler: MSL
// Project:  TestQuickSortPointers
// Bestand:  QuickSortPointers.h
// Implemen: QuickSortPointers.cp
//=======================================

#pragma once

class QuickSortPointers
{
public:
    // Globale functies.
    static void Sorteer(z4 aElement,
                        p_z4 elementen);
private:
    static void Sorteer(p_z4 eerste,
                        p_z4 laatste);

    // Verboden.
    QuickSortPointers();
};
```

```
//=======================================
// Auteur:   © 2017 Cliff Huylebroeck
// Compiler: MSL
// Project:  TestQuickSort
// Bestand:  QuickSortPointers.cp
//=======================================

#include "QuickSortPointers.h"

void QuickSortPointers::Sorteer(c_z4 aElement,
                                cp_z4 elementen)
{
    if (aElement<2)
        {
        return;
        }
    Sorteer(elementen,
            elementen+aElement-1);
}

#pragma profile off

void QuickSortPointers::Sorteer(cp_z4 eerste,
                                cp_z4 laatste)
{
    p_z4 i=eerste;
    p_z4 j=laatste;
    c_z4 aByte=static_cast<z4>(j)-static_cast<z4>(i);
    c_z4 m=(aByte bitand 0x7ffffff8)>>3;
    c_z4 pivot=eerste[m];
    do
        {
        while (*i<pivot)
            {
            i++;
            }
        while (pivot<*j)
            {
            j--;
            }
        if (i<=j)
            {
            c_z4 r=*i;
            *i=*j;
            *j=r;
```

```
            i++;
            j--;
            }
        }
    while (i<j);
    if (eerste<j)
        {
        Sorteer(eerste,
                j);
        }
    if (i<laatste)
        {
        Sorteer(i,
                laatste);
        }
}
```

```
//=======================================
// Auteur:   © 2017 Cliff Huylebroeck
// Compiler: MSL
// Project:  TestQuickSort
// Bestand:  MergeSort.h
// Implemen: MergeSort.cp
//=======================================

#pragma once

class MergeSort
{
private:
    // Globale data.
    static p_z4 s_temp1;
    static p_z4 s_temp2;

    // Globale functies.
public:
    static void Init_klasse(z4 aElement);
    static void Exit_klasse();
    static void Sorteer(z4 aElement,
                        p_z4 elementen);
private:
    static void Sorteer(z4 aElement,
                        z4 deelLengte,
                        pc_z4 invoer,
                        p_z4 uitvoer);

    // Verboden.
    MergeSort();
};
```

```
//=========================================
// Auteur:   © 2017 Cliff Huylebroeck
// Compiler: MSL
// Project:  TestQuickSort
// Bestand:  MergeSort.cp
//=========================================

#include "MergeSort.h"

p_z4 MergeSort::s_temp1=nil;

p_z4 MergeSort::s_temp2=nil;

void MergeSort::Init_klasse(c_z4 aElement)
{
    s_temp1=new z4[aElement];
    if (!s_temp1)
        {
        Exit("Te weinig geheugen.");
        }
    s_temp2=new z4[aElement];
    if (!s_temp2)
        {
        Exit("Te weinig geheugen.");
        }
}

void MergeSort::Exit_klasse()
{
    delete [] s_temp1;
    delete [] s_temp2;
}

void MergeSort::Sorteer(c_z4 aElement,
                        cp_z4 elementen)
{
    if (aElement<2)
        {
        return;
        }
    if (aElement==2)
        {
        if (*elementen>elementen[1])
            {
            c_z4 r=*elementen;
```

```
            *elementen=elementen[1];
            elementen[1]=r;
            }
        return;
        }
    z4 aSorteer=1;
    z4 maxElement=2;
    while (aElement>maxElement)
        {
        aSorteer++;
        maxElement<<=1;
        }
    z4 deelLengte=1;
    if (aSorteer bitand 1)
        {
        Sorteer(aElement,
                deelLengte,
                elementen,
                s_temp2);
        deelLengte<<=1;
        Sorteer(aElement,
                deelLengte,
                s_temp2,
                s_temp1);
        deelLengte<<=1;
        Sorteer(aElement,
                deelLengte,
                s_temp1,
                elementen);
        aSorteer-=3;
        }
    while (aSorteer)
        {
        Sorteer(aElement,
                deelLengte,
                elementen,
                s_temp1);
        deelLengte<<=1;
        Sorteer(aElement,
                deelLengte,
                s_temp1,
                elementen);
        deelLengte<<=1;
        aSorteer-=2;
        }
```

```
}

#pragma profile off

void MergeSort::Sorteer(c_z4 aElement,
                        c_z4 deelLengte,
                        pc_z4 invoer,
                        p_z4 uitvoer)
{
    c_z4 deelLengte2=deelLengte<<1;
    cpc_z4 teVer=invoer+aElement;
    while (invoer<teVer)
        {
        pc_z4 i1=invoer;
        cpc_z4 teVer1=i1+deelLengte;
        pc_z4 i2=teVer1;
        cpc_z4 teVer2=i2+deelLengte;
        while ((i1<teVer1) and (i2<teVer2))
            {
            if (*i1<=*i2)
                {
                *uitvoer++=*i1++;
                }
            else
                {
                *uitvoer++=*i2++;
                }
            }
        while (i1<teVer1)
            {
            *uitvoer++=*i1++;
            }
        while (i2<teVer2)
            {
            *uitvoer++=*i2++;
            }
        invoer+=deelLengte2;
        }
}
```

```
//=======================================
// Auteur:   © 2017 Cliff Huylebroeck
// Compiler: MSL
// Project:  TestQuickSort
// Bestand:  TestData.h
// Implemen: TestData.cp
//=======================================

#pragma once

class TestData
{
public:
    // Globale functies.
    static void MaakOplopend(z4 aElement,
                             p_z4 elementen);
    static void MaakAflopend(z4 aElement,
                             p_z4 elementen);
    static void MaakIdentiek(z4 aElement,
                             p_z4 elementen);
    static void MaakWillekeurig(z4 aElement,
                                p_z4 elementen);
private:
    static z4 BerekenWillekeurig();

    // Verboden.
    TestData();
};
```

```
//=======================================
// Auteur:   © 2017 Cliff Huylebroeck
// Compiler: MSL
// Project:  TestQuickSort
// Bestand:  TestData.cp
//=======================================

#include "TestData.h"

#pragma profile off

void TestData::MaakOplopend(c_z4 aElement,
                            cp_z4 elementen)
{
    for (z4 i=0;i<aElement;i++)
        {
        elementen[i]=i;
        }
}

void TestData::MaakAflopend(c_z4 aElement,
                            cp_z4 elementen)
{
    c_z4 lElement=aElement-1;
    for (z4 i=0;i<aElement;i++)
        {
        elementen[i]=lElement-i;
        }
}

void TestData::MaakIdentiek(c_z4 aElement,
                            cp_z4 elementen)
{
    p_z4 p=elementen;
    cpc_z4 teVer=elementen+aElement;
    while (p<teVer)
        {
        *p=1;
        p++;
        }
}

void TestData::MaakWillekeurig(c_z4 aElement,
                               cp_z4 elementen)
{
```

```
    p_z4 p=elementen;
    cpc_z4 teVer=elementen+aElement;
    while (p<teVer)
        {
        *p=BerekenWillekeurig();
        p++;
        }
}

z4 TestData::BerekenWillekeurig()
{
    c_z2 x=Quickdraw::Random();
    c_z2 y=Quickdraw::Random();
    c_z4 willekeurig=(x<<16) bitor y;
    return willekeurig;
}
```

```
//=======================================
// Auteur:   © 2017 Cliff Huylebroeck
// Compiler: MSL
// Project:  TestQuickSort
// Bestand:  Controle.h
// Implemen: Controle.cp
//=======================================

#pragma once

class Controle
{
public:
    // Globale functies.
    static void Controleer(z4 aElement,
                           pc_z4 elementen);

    // Verboden.
private:
    Controle();
};
```

```
//=======================================
// Auteur:   © 2017 Cliff Huylebroeck
// Compiler: MSL
// Project:  TestQuickSort
// Bestand:  Controle.cp
//=======================================

#include "Controle.h"

#pragma profile off

void Controle::Controleer(c_z4 aElement,
                          cpc_z4 elementen)
{
    if (aElement<2)
        {
        return;
        }
    pc_z4 vorige=elementen;
    pc_z4 p=elementen+1;
    cpc_z4 teVer=elementen+aElement;
    while (p<teVer)
        {
        if (*p<*vorige)
            {
            Exit("Verkeerd gesorteerd.");
            }
        vorige=p;
        p++;
        }
}
```