1. Quicksort
2. Improving Quicksort
3. Sortowanie Shella

1. Quicksort

Początkowo, mamy nieposortowaną tablicę t[] elementów:

Y	?	?	?	?	?	?

Następnie reorganizujemy uporządkowanie elementów następująco:
Wybieramy dowolny element Y w tablicy t[] (np. pierwszy) i sprawdzamy czy pozostałe (np. kolejne) elementy są mniejsze niż Y. Jeśli w trakcie przebiegania (np. przez zmienną i) element i-ty okaże się mniejszy niż Y to zamieniamy element następny po Y z elementem i-tym.

Y	Y_{+1}>Y	>Y	>Y	t[i]<Y	?	?

↓

Y	t[i]	>Y	>Y	Y_{+1}>Y	?	?

Następnie zamieniamy element Y z t[i]:

t[i]	Y	>Y	>Y	Y_{+1}>Y	?	?

Po czym powtarzamy kroki opisane powyżej odnośnie elementu Y, zwiększając indeks i do momentu gdy będzie on równy indeksowi z krańca tablicy.

W wyniku przyłożenia algorytmu Quicksort do tablicy t[]:

← Quicksort →						

Y						

dostajemy tablicę t[] złożoną z dwóch podtablic z elementem Y-ek pomiędzy nimi:

<Y	<Y	Y	Y>=	Y>=	Y>=	Y>=

do których przykładamy ponownie algorytm Quicksort:

← Quicksort →			← Quicksort →			

<Y	<Y	Y	Y>=	Y>=	Y>=	Y>=

Ciągłe przykładanie algorytmu Quicksort odbywa się poprzez samowywołanie funkcji Quicksort (rekurencja). Przykładanie algorytmu Quicksort kończy się w momencie gdy rozmiar podtablicy spadnie do jednego elementu. Podtablice są ustalane przez indeksy numerujące element początkowy i końcowy podtablicy (indeksy te są przekazywane do funkcji Quicksort).

Przykład 1 (C++).

[quickSort1.cpp]

```
//quickSort1.cpp
//demonstrates simple version of quick sort
//Robert Lafore - Teach Yourself Data Structures And Algorithms In 24 hours
#include <iostream>
#include<vector>
#include<cstdlib>                       //for random numbers
#include<ctime>                         //for random numbers
using namespace std;

class ArrayIns
{
    private:
        vector<double> theVect;          //vector of doubles
        int nElems;                       //number of data items

    public:
        ArrayIns(int max) : nElems(0)        //constructor
        {
            theVect.resize(max);              //size the vector
        }

        void insert(double value)            //put element into array
        {
            theVect[nElems] = value;          //insert it
            nElems++;                         //increment size
        }

        void display()                       //displays array contents
        {
            cout << "A=";
            for(int j=0; j<nElems; j++)       //for each element,
                cout << theVect[j] << " ";    //display it
            cout << endl;
        }

        void quickSort()                     //sort array
        {
            recQuickSort(0, nElems-1);        //call recursive sort
        }

        void recQuickSort(int left, int right)  //recursive sort
        {
            if(right-left <= 0)                  //if size <= 1,
                return;                          //   already sorted
            else                                 //size is 2 or large
            {
                double pivot = theVect[right];      //rightmost item
                                                    //partition range
                int partition = partitionIt(left, right, pivot);
                recQuickSort(left, partition-1);    //sort left side
                recQuickSort(partition+1, right);   //sort right side
            }
        }  //end recQuickSort()

        int partitionIt(int left, int right, double pivot)
        {
            int leftMark = left-1;           //left    (after ++)
```

```
            int rightMark = right;           //right-1 (after --)
            while(true)
            {                                //find bigger item
                while( theVect[++leftMark] < pivot )
                    ;                        //   (nop)
                //find smaller item
                while(rightMark > 0 && theVect[--rightMark] > pivot)
                    ;                        //   (nop)
                if(leftMark >= rightMark)    //if pointers cross,
                    break;                   //   partition done
                else                         //not crossed, so
                    swap(leftMark, rightMark);  //  swap elements
            }  //end while(true)
            swap(leftMark, right);           //restore pivot
            return leftMark;                 //return pivot location
        }  //end partitionIt()

        void swap(int dex1, int dex2)       //swap two elements
        {
            double temp = theVect[dex1];     //A into temp
            theVect[dex1] = theVect[dex2];   //B into A
            theVect[dex2] = temp;            //temp into B
        }  //end swap
};  //end class ArrayIns
//--------------------------------------------------------------
int main()
{
    time_t aTime;
    int maxSize = 16;                   //array size
    ArrayIns arr(maxSize);              //create array
    srand( static_cast<unsigned>(time(&aTime)) );  //seed random

    for(int j=0; j<maxSize; j++)        //fill array with
    {                                   //random numbers
        double n = rand() % 99;
        arr.insert(n);
    }
    arr.display();                      //display items
    arr.quickSort();                    //quicksort them
    arr.display();                      //display them again
    return 0;
}  //end main()
```

2. Improving Quicksort

Przykład 2 (C++).

[quickSort2.cpp]

```
//quickSort2.cpp
//demonstrates quick sort with median-of-three partitioning
//Robert Lafore - Teach Yourself Data Structures And Algorithms In 24 hours
#include <iostream>
#include <vector>
#include <cstdlib>                      //for random numbers
#include <ctime>                        //for random numbers
using namespace std;

class ArrayIns
{
    private:
        vector<double> theVect;          //vector of doubles
        int nElems;                       //number of data items

    public:
        ArrayIns(int max) : nElems(0)        //constructor
        {
            theVect.resize(max);              //size the vector
        }

        void insert(double value)            //put element into array
        {
            theVect[nElems] = value;          //insert it
            nElems++;                         //increment size
        }

        void display()                       //displays array contents
        {
            cout << "A=";
            for(int j=0; j<nElems; j++)       //for each element,
                cout << theVect[j] << " ";     //display it
            cout << endl;
        }

        void quickSort()                     //sort array
        {
            recQuickSort(0, nElems-1);        //call recursive sort
        }

        void recQuickSort(int left, int right)  //recursive sort
        {
            int size = right-left+1;
            if(size <= 3)                      //manual sort if small
                manualSort(left, right);
            else                               //quicksort if large
            {
                double median = medianOf3(left, right);
                int partition = partitionIt(left, right, median);
                recQuickSort(left, partition-1);
                recQuickSort(partition+1, right);
```

```
        }
    }  //end recQuickSort()

    double medianOf3(int left, int right)
    {
        int center = (left+right)/2;
        //order left & center
        if( theVect[left] > theVect[center] )
            swap(left, center);
        //order left & right
        if( theVect[left] > theVect[right] )
            swap(left, right);
        //order center & right
        if( theVect[center] > theVect[right] )
            swap(center, right);
        swap(center, right-1);           //put pivot on right
        return theVect[right-1];         //return median value
    }  //end medianOf3()

    void swap(int dex1, int dex2)       //swap two elements
    {
        double temp = theVect[dex1];     //A into temp
        theVect[dex1] = theVect[dex2];   //B into A
        theVect[dex2] = temp;            //temp into B
    }  //end swap(

    //partition a range
    int partitionIt(int left, int right, double pivot)
    {
        int leftMark = left;             //right of first elem
        int rightMark = right - 1;       //left of pivot
        while(true)
        {
            while( theVect[++leftMark] < pivot ) //find bigger
                ;                                //   (nop)
            while( theVect[--rightMark] > pivot ) //find smaller
                ;                                 //   (nop)
            if(leftMark >= rightMark)     //if pointers cross,
                break;                    //   partition done
            else                          //not crossed, so
            swap(leftMark, rightMark); //swap elements
        }  //end while(true)
        swap(leftMark, right-1);          //restore pivot
        return leftMark;                  //return pivot location
    }  //end partitionIt()

    void manualSort(int left, int right)
    {
        int size = right-left+1;
        if(size <= 1)
            return;                       //no sort necessary
        if(size == 2)
        {
            if( theVect[left] > theVect[right] )
                swap(left, right);
            return;
        }
        else  //size==3, so 3-sort left, center (right-1) & righ
        {
            if( theVect[left] > theVect[right-1] )
                swap(left, right-1);               //left, center
```

```
                if( theVect[left] > theVect[right] )
                    swap(left, right);                  //left, right
                if( theVect[right-1] > theVect[right] )
                    swap(right-1, right);               //center, righ
            }
        }  //end manualSort()
};  //end class ArrayIns
//--------------------------------------------------------------
int main()
{
    time_t aTime;
    int maxSize = 16;               //array size
    ArrayIns arr(maxSize);          //create the array
    srand( static_cast<unsigned>(time(&aTime)) );  //seed rando

    for(int j=0; j<maxSize; j++)    //fill array with
    {                               //random numbers
        double n = rand() % 100;    //<0, 99>
        arr.insert(n);
    }
    arr.display();                  //display items
    arr.quickSort();                //quicksort them
    arr.display();                  //display them again
    return 0;
}  //end main()
```

3. Sortowanie Shella

Przykład 3 (C++).

[ShellSort.cpp]

```
//ShellSort.cpp
//demonstrates Shell's sort
#include <iostream>
#include <vector>
#include <iomanip>

using namespace std;
//--------------------------------------------------------------
class ArrayShell
{
    private:
        vector<double> v;                   //vector v
        int nElems;                         //number of data items

    public:
        ArrayShell(int max) : nElems(0)       //constructor
        {
            v.resize(max);                  //size the vector
        }

        void insert(double value)           //put element into array
        {
            v[nElems] = value;              //insert it
            nElems++;                       //increment size
        }

        void display()                      //displays array contents
        {
            for(int j=0; j<nElems; j++)     //for each element,
                cout << setw(2) << v[j] << " ";        //display it
            cout << endl;
        }

        void exch(double& x, double& y){
            int t=x;
            x=y;
            y=t;
        }

        void ShellSort()
        {
            int h = 1;
            while (h < nElems/3)
                h = 3*h + 1;                    //1, 4, 13, 40, 121, 364,
...
            while (h >= 1)
            {
                for(int i=h;i<nElems;i++)      // h-sortowanie tablicy
                {
                    cout<<"h="<<h<<" ";
                    display();
```

```
                    for(int j=i;j>=h&&v[j]<v[j-h];j-=h)
                        exch(v[j],v[j-h]);
                }
                h/=3;
            }
        }  //end ShellSort()

};  //end class ArrayShell
//--------------------------------------------------------------
int main()
{
    int maxSize = 100;              //array size
    ArrayShell arr(maxSize);        //create array

    arr.insert(77);                 //insert 10 items
    arr.insert(99);
    arr.insert(44);
    arr.insert(55);
    arr.insert(22);
    arr.insert(88);
    arr.insert(11);
    arr.insert(00);
    arr.insert(66);
    arr.insert(33);

    cout<<"    ";
    arr.display();                  //display items
    cout<<"h-sortowanie:"<<endl;
    arr.ShellSort();                //insertion-sort them
    cout<<"    ";
    arr.display();                  //display them again
    return 0;
}  //end main()
```

Opracowano na podstawie:
Robert Lafore – "Teach Yourself Data Structures And Algorithms In 24 Hours"