using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Collections;

namespace ConsoleApplication1

{

class Program

{

static int type ;

static int pivotPointLeft = 0;

static int pivotPointRight = 0;

static int pivotPointRight2 = 0;

static int pivotPointMedian = 0;

static int scan = 0;

static int runningScan = 0;

static int endOfLeftList = 0;

static int endOfRightList = 0;

static int endOfLeftList1 = 0;

static int countPartition = 0;

static int countRightPartition = 0;

static bool swamp = true;

static void Main(string[] args)

{

int[] array ;

int[] arrayRightMost;

int[] arrayMedium;

string line = "";

ArrayList size = new ArrayList();

int i = 0;

Random ran ;

try

{

System.IO.StreamReader file =

new System.IO.StreamReader("number.txt");

while ( (line = file.ReadLine()) != null)

{

size.Add(Convert.ToInt32(line));

}

array = new int[size.Count];

arrayRightMost = new int[size.Count];

int[] arrayRightPivot = new int[size.Count];

arrayMedium = new int[size.Count];

ran = new Random();

while(i < size.Count)

{

array[i] = (int)size[i];

arrayRightMost[i] = (int)size[i];

arrayRightPivot[i] = (int)size[i];

arrayMedium[i] = (int)size[i];

i++;

}

Console.Write("\n");

type = 0;

quickSort(array);

Console.WriteLine("The number of comparisons for the left L -> R pivot are {0} \n", pivotPointLeft);

type = 3;

quickSort(arrayRightPivot);

Console.WriteLine("The number of comparisons with swap R -> L pivot and starting with L pivot are {0} \n", pivotPointRight2);

type = 1;

quickSort(arrayRightMost);

Console.WriteLine("The number of comparisons from the R pivot are {0} \n", pivotPointRight);

type = 2;

Console.Write("\n");

quickSort(arrayMedium);

Console.WriteLine("The number of comparisons with the median pivot are {0} \n", pivotPointMedian);

}

catch(Exception e)

{

Console.WriteLine(e.Message);

Console.WriteLine(e.Source);

// Console.WriteLine(e.StackTrace);

}

Console.ReadLine();

}

/\*\*

\* starts the quicksort alghoritm

\* @parm database

\* @param index

\* @return doQuickSort method

\*/

public static void quickSort(int[] array)

{

switch (type)

{

case 0:

doQuickSort(array, 0, array.Length);

break;

case 1:

doQuickSort(array, array.Length - 1, 0);

break;

case 2:

doQuickSort(array, 0, array.Length);

break;

case 3:

{

doQuickSort(array, 0, array.Length);

break;

}

}

}

/\*\*

\* It sorts the two halfs of the array

\* @param dataBase

\* @param start

\* @param end

\* @return dataBase

\*

\*/

private static void doQuickSort(int[] array,int start, int end)

{

switch (type)

{

case 0:

{

pivotPointLeft = quickSort1(array, start, end);

break;

}

case 1:

{

pivotPointRight = quickSort2(array, start, end);

break;

}

case 2:

{

pivotPointMedian = quickSort3(array, start, end);

break;

}

case 3:

{

pivotPointRight2 = quickSort1bis(array, start, end);

break;

}

}

}

/\*\*

\* it partition the two halfs of the array

\* and uses the swamo method to swap the elements

\* @param dataBase

\* @param start

\* @param end

\* @return endOflist

\*/

private static int quickSort1(int[] array,int start,int end)

{

if (start == end)

{

return 0;

}

int pivotValue;

int endOfLeftList;

pivotValue = array[start];

endOfLeftList = start + 1;

for (int scan = start + 1; scan < end; scan++)

{

if (array[scan] < pivotValue)

{

swap(array, endOfLeftList, scan);

endOfLeftList++;

}

}

swap(array, start, endOfLeftList - 1);

return (end - start - 1) + quickSort1(array, start, endOfLeftList - 1) + quickSort1(array, endOfLeftList, end);

}

private static int quickSort1bis(int[] array, int start, int end)

{

if (start == end)

{

return 0;

}

if (swamp == true)

{

swap(array, start, end - 1);

swamp = false;

}

int pivotValue;

int endOfLeftList1;

pivotValue = array[start];

endOfLeftList1 = start + 1;

for (int scan = start + 1; scan < end; scan++)

{

if (array[scan] < pivotValue)

{

swap(array, endOfLeftList1, scan);

endOfLeftList1++;

}

}

swap(array, start, endOfLeftList1 - 1);

return (end - start - 1) + quickSort1bis(array, start, endOfLeftList1 - 1) + quickSort1bis(array, endOfLeftList1, end);

}

/// <summary>

///

/// </summary>

/// <param name="array"></param>

/// <param name="start"></param>

/// <param name="end"></param>

/// <returns></returns>

private static int quickSort2(int[] array,int start,int end)

{

if (start <= end)

{

return 0;

}

int pivotValue;

pivotValue = array[start];

endOfRightList = start - 1;

for( int scan = start - 1; scan >= end; scan--)

{

if (array[scan] > pivotValue)

{

swap(array, scan, endOfRightList);

endOfRightList--;

}

}

swap(array,endOfRightList + 1,start);

return (start - end) + quickSort2(array, endOfRightList, end) + quickSort2(array, start, endOfRightList + 1);

}

/// <summary>

/// array medium

/// </summary>

/// <param name="array"></param>

/// <param name="start"></param>

/// <param name="end"></param>

/// <returns></returns>

private static int quickSort3(int[] array,int start,int end)

{

if (start == end)

{

return 0;

}

int pivotValue;

int endOfLeftList;

int median = medianValue(array,start,end - 1);

swap(array,start,median);

pivotValue = array[start];

endOfLeftList = start + 1;

for (int scan = start + 1; scan < end; scan++)

{

if (array[scan] < pivotValue)

{

swap(array, endOfLeftList, scan);

endOfLeftList++;

}

}

swap(array,start,endOfLeftList - 1);

return (end - start - 1) + quickSort3(array, start, endOfLeftList - 1) + quickSort3(array, endOfLeftList, end);

}

/\*\*

\* swaps the elements of the database

\* @param dataBase

\* @param a

\* @param b

\*/

private static void swap(int[] array, int a, int b)

{

int temp;

temp = array[a];

array[a] = array[b];

array[b] = temp;

}

public static void print(int[] array)

{

for(int i = 0; i < array.Length; i++)

{

Console.Write(array[i]+" ");

if( (i+1) % 40 == 0)

{

Console.Write("\n");

}

}

}

public static int medianValue(int[] array,int left, int right)

{

int center = (left + right) / 2;

// order left & center

if (array[left] > array[center])

swap(array,left, center);

// order left & right

if (array[left] > array[right])

swap(array ,left, right);

// order center & right

if (array[center] > array[right])

swap(array,center, right);

if (array.Length % 2 == 0)

{

swap(array, center, left);

return left;

}

else

{

swap(array, center, right - 1); // put pivot on right

return right - 1; // return median value

}

}

}

}