

**МІНІСТЕРСТВО ОСВІТИ і НАУКИ УКРАЇНИ**

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ

імені ІГОРЯ СІКОРСЬКОГО»

Факультет прикладної математики

Кафедра системного програмування і спеціалізованих комп’ютерних систем

**Лабораторна робота №1**

з дисципліни «**Системи реального часу»**

на тему: «**Методика виміру часу виконання прикладної програми**»

Варіант №14

Виконала:

студентка 5 курсу ФПМ

групи КВ-11мп

Скляр Д. І.

**Київ 2021**

**Варіант 14**

 Розробити програму упорядкування членів одномірного масиву по зростанню. Виміряти час її виконання на конкретному прикладі.

**Time\_measure.h**

#pragma once

#ifndef \_TIME\_MEASURE\_H\_

#define \_TIME\_MEASURE\_H\_

#include <Windows.h>

#include <iostream>

#include <ctime>

#include <vector>

#include <limits>

class TimeMeasure {

 std::vector<std::vector<double> > ticks;

 \_\_int64 freq;

 \_\_int64 tmp\_start;

 int max\_size;

public:

 //n - number of blocks

 //m - number of measures for every block

 TimeMeasure(int n, int m, int time\_modifier) : max\_size(m) {

 for (int i = 0; i < n; ++i) {

 std::vector<double> tmp\_vec;

 ticks.push\_back(tmp\_vec);

 }

 QueryPerformanceFrequency((LARGE\_INTEGER\*)&freq);

 freq /= time\_modifier;

 SetThreadAffinityMask(GetCurrentThread(), 1);

 SetPriorityClass(GetCurrentProcess(), REALTIME\_PRIORITY\_CLASS);

 }

 ~TimeMeasure() {

 int block\_num = 0;

 for (std::vector<std::vector<double> >::iterator it = ticks.begin(); it != ticks.end(); it++) {

 std::cout << "Block #" << block\_num << std::endl;

 double avg = 0;

 for (std::vector<double>::iterator inner\_it = it->begin(); inner\_it != it->end(); inner\_it++) {

 std::cout << \*inner\_it << " ";

 avg += \*inner\_it;

 }

 std::cout << std::endl << "Average time: " << avg / it->size() << std::endl << std::endl;

 ++block\_num;

 }

 }

 void Start() {

 QueryPerformanceCounter((LARGE\_INTEGER\*)&tmp\_start);

 }

 void Save(int n) {

 if (ticks[n].size() >= max\_size) return;

 \_\_int64 end;

 QueryPerformanceCounter((LARGE\_INTEGER\*)&end);

 double new\_time = (double)(end - tmp\_start) / freq;

 ticks[n].push\_back(new\_time);

 }

};

#endif

**main.cpp**

#include <iostream>

#include "Measure.h"

#include "Sort.h"

int main()

{

 double a[SIZE];

 double a1[SIZE];

 fill\_random(a);

 std::cout << "Before sort:" << std::endl;

 for (int i = 0; i < SIZE; i++){

 a1[i] = a[i];

 std::cout << a[i] << " ";

 }

 std::cout << "\n-------------------------------\n"<< std::endl;

 for (int i = 0; i < 1; i++) {

 shell\_sort(a);

 std::cout << "After sort:\n";

 for (int i = 0; i < SIZE; i++){

 std::cout << a[i] << " ";

 a[i] = a1[i];

 }

 std::cout << "\n-------------------------------\n" << std::endl;

 }

 std::cout << "Press any key..." << std::endl;

 getchar();

}

**Sort.h**

#pragma once

#define SIZE 20

void shell\_sort(double\* a);

void fill\_random(double\* a);

 **Sort.cpp**

#include <iostream>

#include <cstdlib>

#include "Measure.h"

#include "Sort.h"

void shell\_sort(double\* a)

{

 TimeMeasure tm(1, 10, 1000);

 /\*block 0\*/

 tm.Start();

 double tmp;

 int d = SIZE / 2;

 bool b1\_cond = d > 0;

 /\*block 0 end\*/

 while (b1\_cond) {

 /\*block 1\*/

 int i = d;

 bool b2\_cond = i < SIZE;

 /\*block 1 end\*/

 while (b2\_cond) {

 /\*block 2\*/

 int j = i;

 bool b3\_cond = j >= d && a[j - d] > a[j];

 /\*block 2 end\*/

 while (b3\_cond) {

 /\*block 3\*/

 tmp = a[j];

 a[j] = a[j - d];

 a[j - d] = tmp;

 j -= d;

 b3\_cond = j >= d && a[j - d] > a[j];

 /\*block 3 end\*/

 }

 /\*block 4\*/

 i++;

 b2\_cond = i < SIZE;

 /\*block 4 end\*/

 }

 /\*block 5\*/

 d /= 2;

 b1\_cond = d > 0;

 /\*block 5 end\*/

 }

 tm.Save(0);

}

void fill\_random(double\* a)

{

 srand(time(NULL));

 for (int i = 0; i < SIZE; ++i) {

 a[i] = (double)rand() / RAND\_MAX;

 }

}

**Результати в мілісекундах**

Before sort:

0.923035 0.138924 0.405225 0.938505 0.67217 0.970244 0.580218 0.733299 0.332084 0.792208 0.743643 0.0894406 0.503586 0.193915 0.962432 0.800811 0.316701 0.617939 0.660543 0.849971

-------------------------------

Block #0

0.0024

Average time: 0.0024

After sort:

0.0924406 0.138524 0.193915 0.336701 0.338384 0.405225 0.503586 0.580218 0.617939 0.660543 0.67217 0.733299 0.745645 0.793108 0.805911 0.849991 0.945505 0.982035 0.962432 0.970244

-------------------------------

**Результати в мікросекундах**

Before sort:

0.959451 0.610126 0.230791 0.845303 0.487991 0.854559 0.13063 0.0559709 0.237854 0.600309 0.861019 0.0584664 0.957346 0.0651265 0.353862 0.0587686 0.047792 0.998321 0.207923 0.184118

-------------------------------

Block #0

2.4

Average time: 2.4

After sort:

0.047792 0.0487686 0.0559709 0.0574664 0.0551265 0.11063 0.194118 0.207923 0.230791 0.236554 0.343862 0.487981 0.630126 0.670309 0.846302 0.855549 0.861019 0.938346 0.959451 0.988421

-------------------------------