МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

“Московский государственный университет геодезии и картографии”

(МИИГАИК)

Факультет геоинформатики и информационной безопасности

Кафедра геоинформационных систем и технологий

**Лабораторная работа №1**

**"Библиотечные реализации алгоритмов и их скорость"**

Проверил: Выполнил:

Лебедев Евгений Денисович Студент группы: 2024-ФГИИБ-ПИ-1б

Центнер Валерий Викторович

Москва 2024

**Вариант 28**

Формулировка задания:

Создать алгоритм по теме:

Оценка вычислительной сложности и потребления памяти,

удаление элемента из конца одномерного Массива.

Определение структуры данных, с которой работаю:

Структура данных, с которой я работаю – массив.

Массив – структура данных, хранящая набор значений, которые принимают значения из указанного диапазона.  
Операция, которую я произвожу – удаление элемента.

Произвожу я её с помощью метода “ pop\_back() “.

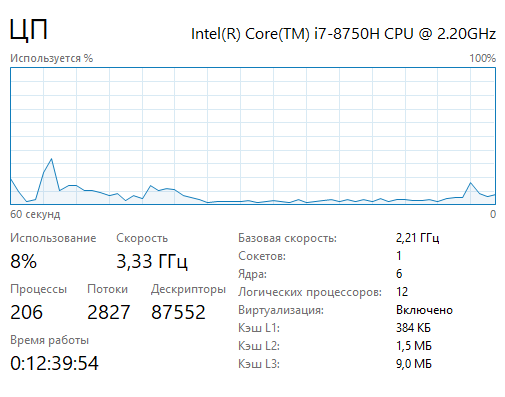
Описание ожидаемого результата:

Алгоритм должен корректно работать и считать время своей работы.

Я думаю, что алгоритм будет вести себя линейно, так как массив одномерный и программа должна удалять элементы последовательно.

Справка о процессоре на моём ноутбуке:

Intel Core i7-8750H — это высококлассный мобильный шестиядерный процессор семейства Coffee Lake. Процессор работает на частоте от 2.1 до 4.1 ГГц (4 ГГц для 4 активных ядер и 3.9 ГГц для всех 6 ядер) и способен обрабатывать до 12 потоков.



Листинги кода реализации:

Основной код программы :  
  
#include <iostream>

#include <vector>

#include <chrono>

// замеряем времечко

long long measureTime(void (\*func)(std::vector<int>&), std::vector<int>& vec) {

auto start = std::chrono::high\_resolution\_clock::now();

func(vec);

auto end = std::chrono::high\_resolution\_clock::now();

return std::chrono::duration\_cast<std::chrono::milliseconds>(end - start).count();

}

// запрашиваем инпут пользователя

long long getUserInput() {

long long value;

std::cout << "Enter the size of the array: ";

while (!(std::cin >> value) || value <= 0) {

std::cout << "Invalid input! Please enter a positive number: ";

std::cin.clear();

while (std::cin.get() != '\n');

}

return value;

}

// делаем сам массив

std::vector<int> createArray(int size) {

std::vector<int> vec;

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

vec.push\_back(i);

}

return vec;

}

// убираем элементы массива

void removeElements(std::vector<int>& vec) {

while (!vec.empty()) {

vec.pop\_back();

}

}

int main() {

int size = getUserInput();

std::vector<int> numbers = createArray(size);

long long timeTaken = measureTime(removeElements, numbers);

std::cout << "Time taken to remove elements: " << timeTaken << " ms" << std::endl;

return 0;

}

===================================================

Код для юнит-тестов :

#include "pch.h"

#include "CppUnitTest.h"

#include "../CodeHomework1/CodeHomework1.cpp"

using namespace Microsoft::VisualStudio::CppUnitTestFramework;

namespace UnitTest1

{

TEST\_CLASS(UnitTest1)

{

public:

// проверка, что createArray корректно создаёт массив указанного размера

TEST\_METHOD(TestCreateArraySize)

{

int size = 5;

std::vector<int> arr = createArray(size);

Assert::AreEqual(size, static\_cast<int>(arr.size()), L"Array size should match input size");

}

// проверка, что createArray создаёт массив с правильными значениями

TEST\_METHOD(TestCreateArrayValues)

{

std::vector<int> arr = createArray(5);

for (int i = 0; i < 5; i++)

{

Assert::AreEqual(i, arr[i], L"Array values should be sequential from 0");

}

}

// проверка работы removeElements (массив должен быть пустым после удаления)

TEST\_METHOD(TestRemoveElements)

{

std::vector<int> arr = createArray(10);

removeElements(arr);

Assert::IsTrue(arr.empty(), L"Array should be empty after removeElements");

}

// проверка работы removeElements на пустом массиве (не должно быть ошибки)

TEST\_METHOD(TestRemoveEmptyArray)

{

std::vector<int> arr;

removeElements(arr);

Assert::IsTrue(arr.empty(), L"Removing from an empty array should not cause issues");

}

// тест на создание и очистку массива из 1 млн. элементов

TEST\_METHOD(TestOneMillion)

{

int size = 1000000;

std::vector<int> arr = createArray(size);

Assert::AreEqual(size, static\_cast<int>(arr.size()), L"Array size should match input size for 1 million");

removeElements(arr);

Assert::IsTrue(arr.empty(), L"Array should be empty after removeElements for 1 million elements");

}

// тест на создание и очистку массива из 10 млн. элементов

TEST\_METHOD(TestTenMillion)

{

int size = 10000000;

std::vector<int> arr = createArray(size);

Assert::AreEqual(size, static\_cast<int>(arr.size()), L"Array size should match input size for 10 million");

removeElements(arr);

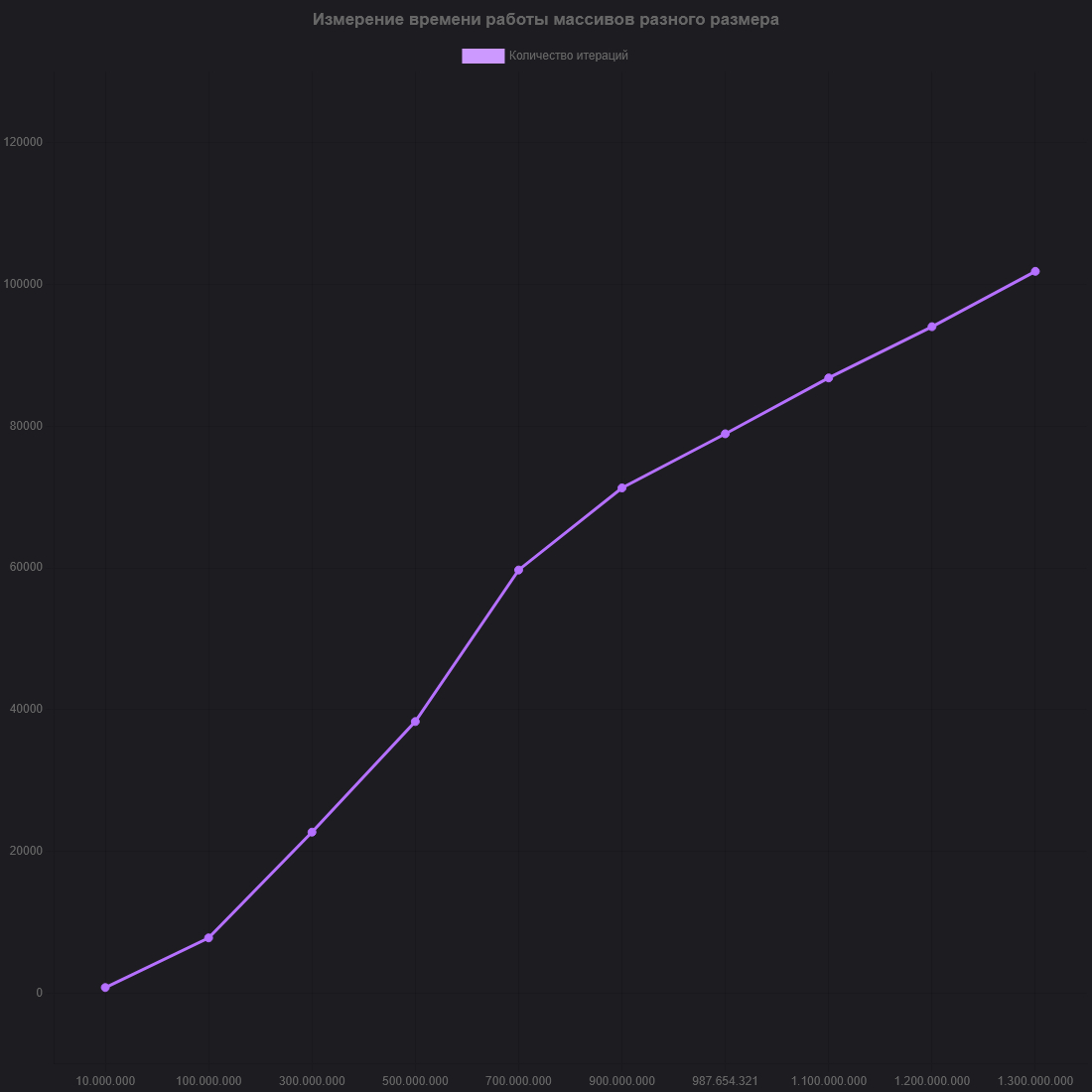
Assert::IsTrue(arr.empty(), L"Array should be empty after removeElements for 10 million elements");

}

};

}

График с измерениями (ось X – итерации, ось Y – время):



Выводы о измерениях:

Ожидания и измерения в целом сошлись, подобных результатов я и ожидал.

На маленьких итерациях ( <= 10 млн.) программа ведет себя крайне быстро, а потом кратно растёт время исполнения ( > 10 млн.).

Наблюдаю здесь линейную зависимость времени работы алгоритма, как я и предполагал.

Зависимость линейная, то есть алгоритм O(n).