**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ**

**РОССИЙСКОЙ ФЕДЕРАЦИИ**

Федеральное государственное бюджетное образовательное учреждение

высшего образование

«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

Плесовских Арсений Андреевич

**ВАРИАНТ 21**

**Формулировка задания**

Оценка вычислительной сложности и потребления памяти: добавление элемента в Список

**Определение структуры данных, с которой работаете. Определение операции.**

Мы будем работать со связным списком. Это структура данных, где каждый элемент (узел) содержит два компонента:

1. Данные — непосредственно значения элемента.

2. Указатель на следующий элемент — ссылка на следующий узел в списке.

Операция, которую мы будем рассматривать, — это добавление элемента в конец списка.

**Описание ожидаемого результата**

Я считаю, что программа будет работать исправно, но чем больше элементов будет добавляться к списку, тем дольше будет измеряться время программы. Алгоритм будет вести себя линейно. Мои предположения, что первые замеры программы будут в миллисекундах, а 100000 итераций и более будут занимать секунды или даже минуты.

**Справка о процессоре ПК**

Intel(R) Core™ i3-8100 3.60 GHz

Четырехядерный процессор начального уровня 8-го поколения. Встроенное графическое ядро.

**Листинг кода**

#include <iostream>

#include <chrono>

#include <vector>

#include <cassert>

using namespace std;

// Структура для узла связанного списка

struct Node {

int data;

Node\* next;

Node(int value) : data(value), next(nullptr) {}

};

// Класс для связанного списка

class LinkedList {

public:

LinkedList() : head(nullptr) {}

// Функция для добавления элемента в конец списка

void append(int value) {

Node\* newNode = new Node(value);

if (!head) {

head = newNode;

}

else {

Node\* current = head;

while (current->next) {

current = current->next;

}

current->next = newNode;

}

}

// Функция для вычисления размера списка

int size() const {

int count = 0;

Node\* current = head;

while (current) {

count++;

current = current->next;

}

return count;

}

private:

Node\* head;

};

// Функция для измерения времени выполнения добавления элементов

void measureTime() {

vector<int> sizes = { 10, 100, 1000, 10000, 100000, 1000000, 10000000 }; // примеры разных размеров

for (int size : sizes) {

LinkedList list;

// Засекаем время добавления элементов

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

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

list.append(i);

}

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

chrono::duration<double> duration = end - start;

cout << "Size: " << size << " Time taken: " << duration.count() << " seconds" << endl;

}

}

int main() {

// Тестирование добавления элементов

measureTime();

return 0;

}

**UNIT-тесты**

#include "pch.h"

#include "CppUnitTest.h"

#include "../№1 Библиотечные реализации алгоритмов и их скорость/main.cpp"

using namespace Microsoft::VisualStudio::CppUnitTestFramework;

namespace UnitTest1

{

TEST\_CLASS(UnitTest1)

{

public:

TEST(LinkedListTest, AddSingleElement) {

LinkedList list;

list.append(5);

ASSERT\_EQ(list.size(), 1);

}

// Тест на добавление нескольких элементов

TEST(LinkedListTest, AddMultipleElements) {

LinkedList list;

list.append(1);

list.append(2);

list.append(3);

ASSERT\_EQ(list.size(), 3);

}

// Тест на добавление 1000 элементов

TEST(LinkedListTest, Add1000Elements) {

LinkedList list;

for (int i = 0; i < 1000; ++i) {

list.append(i);

}

ASSERT\_EQ(list.size(), 1000);

}

int main(int argc, char\*\* argv) {

::testing::InitGoogleTest(&argc, argv);

return RUN\_ALL\_TESTS();

}

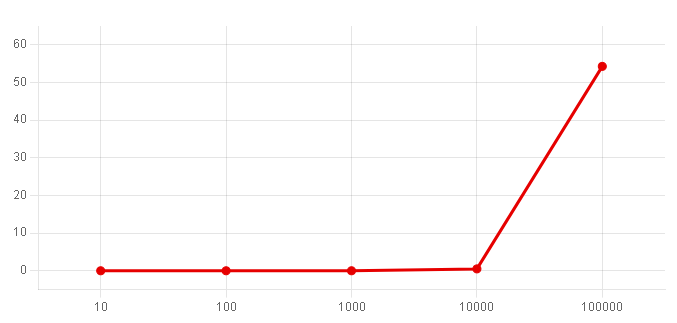
{

}

};

}

График с измерениями OX-итерации OY-секунды



Выводы

В общем и целом всё как я и ожидал, до 10000 итераций алгоритм прошел почти моментально, но меня удивило, что 100000 итераций я уже ждал 54 секунды. Линейная зависимость времени от количества итераций была предсказуема, следовательно алгоритм О.

P.S Миллион итераций я так и не дождался (ждал около 7-ми минут)