awФЕДЕРАЛЬНОЕ АГЕНТСТВО СВЯЗИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ТЕЛЕКОММУНИКАЦИЙ ИМ. ПРОФ. М.А. БОНЧ-БРУЕВИЧА»

(СПбГУТ)

Факультет инфоРМАЦИОННЫХ ТЕХНОЛОГИЙ И ПРОГРАММНОЙ ИНЖЕНЕРИИ (ИТПИ)

кафедра программной инженерии и вычислительной техники (пиивт)

Дисциплина: «Алгоритмы и структуры данных»

Лабораторная работа №2

Тема: «Методы поиска»

Отчёт

Выполнили:

Коньков М. Д.

Семенихин А. Р.

Подпись \_\_\_\_\_\_\_\_\_\_\_\_\_\_

Принял:

Дагаев А. В.

Подпись \_\_\_\_\_\_\_\_\_\_\_\_\_\_

Санкт-Петербург

2025г.

# Цель работы:

Целью работы является ознакомление с алгоритмами одномерной оптимизации и методикой оценки эффективности алгоритмов.

# Описание алгоритмов:

Метод дихотомии систем основан на поиске находящегося в интервале неопределённости, экстремума функции, одной переменной путем деления пополам интервала, на котором находится экстремум[1].

Алгоритм поиска экстремума методом дихотомии состоит из двух групп блоков:

1. Поиск интервала неопределенности;
2. Поиск экстремума на интервале неопределенности с установленной точностью.

На первой этапе вычисляется x0=(a+b)/2.

Далее определяется значение функции в этой точке:

1. если f(x0)< 0, то [a,x0];
2. если наоборот, то [x0,b].

То есть происходит сужение интервала. Таким образом, в результате формируется последовательность xi, где i - номер иттерации.

Вычисления прекращаются, когда разность b-a меньше требуемой погрешности.

# Пошаговое сравнение:

В таблице 1 отображена пошаговая работа рекурсивного варианта дихотомического алгоритма при . В таблице 1: Пошаговая работа рекурсивного дихотомического алгоритма.

*Таблица 1*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Границы промежутка /  Номер шага | l | m | r | Примечание |
| 1 | 0 | 4 | 2 |  |
| 2 | 0 | 1 | 2 | Сдвиг правой границы. |
| 3 | 0 | 0,5 | 1 | Сдвиг правой границы. |
| 4 | 0,5 | 0,75 | 1 | Сдвиг левой границы. |
| 5 | 0,5 | 0,625 | 0,75 | Сдвиг правой границы. |
| 6 | 0,625 | 0,6875 | 0,75 | Сдвиг левой границы. |
| 7 | 0,625 | 0,65625 | 0,6875 | Сдвиг правой границы. |
| 8 | 0,65625 | 0,671875 | 0,6875 | Останов. |

В таблице 2 отображена пошаговая работа итеративного варианта дихотомического алгоритма при . В таблице 2: Пошаговая работа итеративного дихотомического алгоритма.

*Таблица 2*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Границы промежутка /  Номер шага | l | m | r | Примечание |
| 1 | 0 | 4 | 2 |  |
| 2 | 0 | 1 | 2 | Сдвиг правой границы. |
| 3 | 0 | 0,5 | 1 | Сдвиг правой границы. |
| 4 | 0,5 | 0,75 | 1 | Сдвиг левой границы. |
| 5 | 0,5 | 0,625 | 0,75 | Сдвиг правой границы. |
| 6 | 0,625 | 0,6875 | 0,75 | Сдвиг левой границы. |
| 7 | 0,625 | 0,65625 | 0,6875 | Сдвиг правой границы. |
| 8 | 0,65625 | 0,671875 | 0,6875 | Останов. |

По результатам пошагового сравнения, рекурсивный и итеративный алгоритм выполняются за одинаковое число шагов.

# Описание программы:

Программа для выполнения настоящей работы обеспечивает выполнение процедур поиска подстроки в фоновом режиме и подсчёт временных затрат на работу каждого из алгоритмов. Имеется возможность указать число итераций, в течение которых будут сняты показания времени.

При запуске программы следует ввести длину строки и количество итераций согласно инструкциям. По завершении работы процедур сортировки на экран будет выведено среднее время работы обоих алгоритмов. Для завершения работы программы следует ввести символ q, для продолжения работы — любой другой символ.

Программа создана с использованием языка программирования C++ в текстовом редакторе с конфигурацией LazyVim (Neovim) для работы в системе под управлением ОС Linux.

# Полученные результаты:

Результаты оценки временной сложности алгоритмов (в миллисекундах) представлены в таблице 3 и отображены на рисунке 1. Далее показана **зависимость времени работы алгоритмов одномерной оптимизации от заданной точности вычислений (в миллисекундах).**

*Таблица 3*

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Точность / Алгоритм** | 10⁻⁴ | 10⁻⁵ | 10⁻⁶ | 10⁻⁷ | 10⁻⁸ | 10⁻⁹ |
| **Рекурсивный** | 0,000158 | 0,000191 | 0,000206 | 0,000234 | 0,000275 | 0,000303 |
| **Итерационный** | 0,000135 | 0,000158 | 0,000172 | 0,000203 | 0,000218 | 0,000231 |

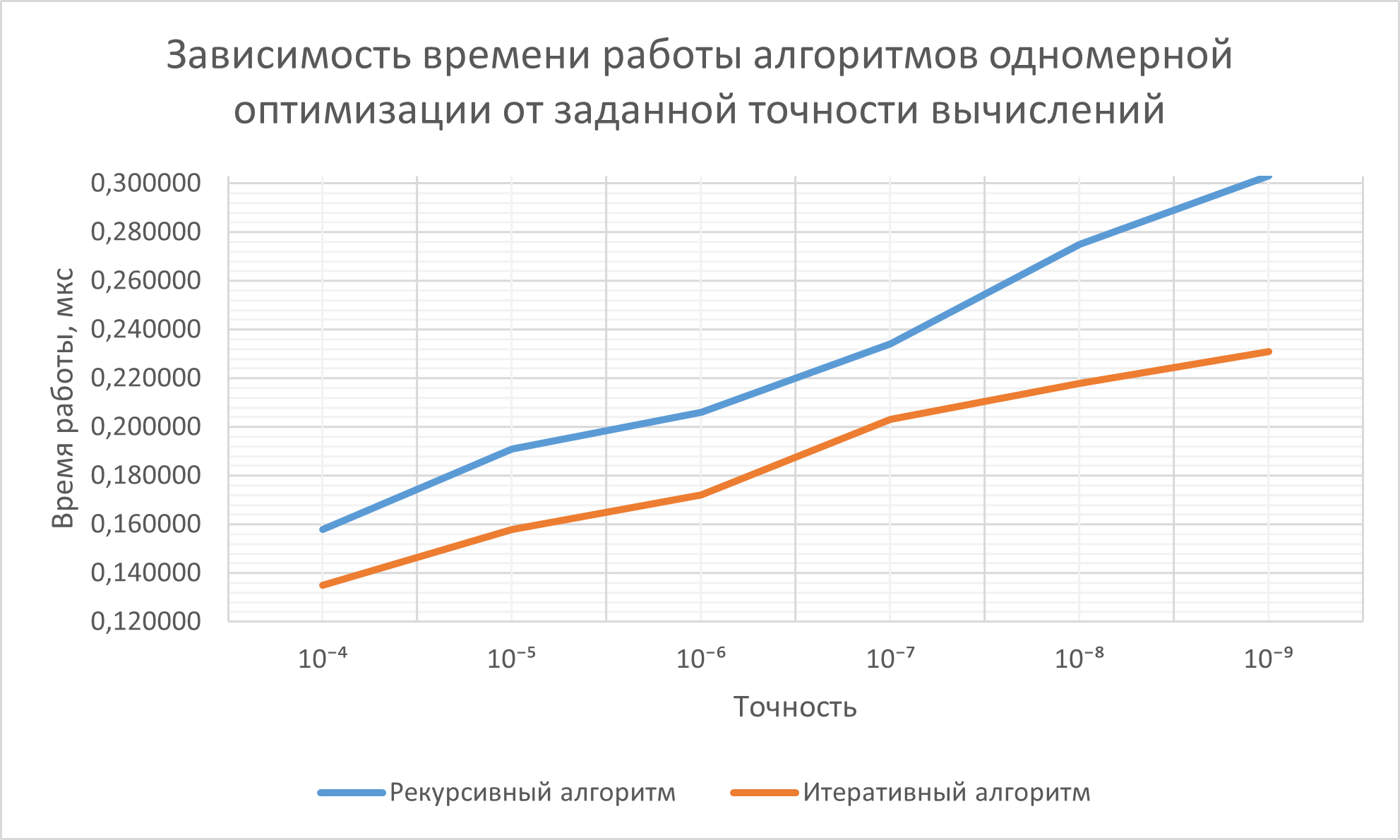


Рис. 1. Зависимость времени работы алгоритмов одномерной оптимизации от заданной точности вычислений.

Данные, отображенные в таблице и на рисунке, показывают, что рекурсивный и итеративный дихотомические алгоритмы имеют одинаковую асимптотическую сложность; тем не менее, итеративный вариант работает в среднем быстрее.

# Вывод:

В ходе проведённой лабораторной работы получены следующие результаты:

1. Изучен дихотомический алгоритм одномерной оптимизации.
2. Создана программа, позволяющая выполнить оценку временной сложности алгоритмов.
3. Проведена оценка временной сложности итерационного и рекурсивного варианта дихотомического алгоритма.
4. По результатам оценки оба варианта дихотомического алгоритма имеют одинаковую асимптотическую сложность.

# Листинг кода:

#include <iostream>

#include <vector>

#include <time.h>

double f(double x)

{

return 2.755 \* x \* x - 3.7743 \* x + 1.2311;

}

double f2(double l, double r, double pre)

{

double m = (r - l) / 2.0 + l;

while (abs(f(r) - f(l)) > pre)

{

if (f(m - pre) < f(m + pre))

{

r = m;

}

else

{

l = m;

}

m = (r - l) / 2.0 + l;

}

return m;

}

double f1(double l, double r, double pre)

{

double m = (r - l) / 2.0 + l;

if (abs(f(r) - f(l)) <= pre)

{

return m;

}

if (f(m - pre) < f(m + pre))

{

m = f1(l, m, pre);

}

else

{

m = f1(m, r, pre);

}

return m;

}

int main()

{

srand(static\_cast<unsigned int>(time(NULL)));

char c = ' ';

while (c != 'q')

{

int r = 0;

double eps, rg;

std::cout << "Precision as power of 10:" << std::endl;

std::cin >> rg;

eps = pow(10.0, -1 \* rg);

std::cout << "Iterations:" << std::endl;

std::cin >> r;

double time\_f1\_sum = 0, time\_f2\_sum = 0;

const double a = 0, b = 4;

for (int k = 0; k < r; k++)

{

clock\_t start = clock();

f1(a, b, eps);

clock\_t end = clock();

double seconds = (double)(end - start) / CLOCKS\_PER\_SEC;

time\_f1\_sum += seconds;

start = clock();

f2(a, b, eps);

end = clock();

seconds = (double)(end - start) / CLOCKS\_PER\_SEC;

time\_f2\_sum += seconds;

}

double time\_1 = time\_f1\_sum / (double)r;

std::cout << "The time (recursion): " << time\_1 << " seconds" << std::endl;

double time\_2 = time\_f2\_sum / (double)r;

std::cout << "The time (iteration): " << time\_2 << " seconds" << std::endl;

std::cout << "Enter any char to continue; enter 'q' to exit." << std::endl;

std::cin >> c;

}

return 0;

}