Задание №7 в рамках вычислительного практикума.
Постановка замерного эксперимента

Студент: Краснов Леонид

Группа: ИУ7-21Б

Table of Contents

Цель

Провести сравнение производительности работы программы по двум плоскостям.

Задание

Подготовка программы к тестированию

Для выполнения задания используется программа №5 из лабораторной работы №2 по курсу СИ. Программа вычисляет и выводит на экран значение

X[0] * *Y[0]* + *X[1]* * Y[1] + ... + X[k - 1] * *Y[k - 1]*
X - отрицательные элементы в порядке следования
Y - положительные элементы в обратном порядке
k = min(p, q)
p - количество положительных элементов
q - количество отрицательных элементов

При этом пользователь вводит кол-во элементов и сами элементы

Исходный код программы

```c
#include <stdio.h>


#define N 10
#define INPUT_ERROR 1
#define NOELEMS_ERROR 2
#define SUCCESS 0


int count(int *xbeg, int *xend, int *ybeg, int *yend)
{
    int rez = 0;
    int k1 = (xend - xbeg);
    int k2 = (yend - ybeg);
    int k;
    if (k1 > k2)
    {
        k = k2;
    }
    else
    {
        k = k1;
    }
    while (k > 0)
    {
        rez += *xbeg * *ybeg;
        xbeg++;
        ybeg++;
        k--;
    }
    return rez;
}
```

```c
int main(void)
{
    int a[N], x[N], y[N], n;
    printf("Введите длину массива: ");
    if (scanf("%d", &n) != 1)
    {
        printf("Неверный ввод!\n");
        return INPUT_ERROR;
    }
    if (n > 10 || n < 1)
    {
        printf("Неверный ввод\n");
        return INPUT_ERROR;
    }
    int *pabeg = a, *paend = pabeg + n;
    printf("\nВведите элементы массива через пробел: ");
    while (pabeg < paend)
    {
        if (scanf("%d", pabeg) != 1)
            return INPUT_ERROR;
        pabeg++;
    }
    pabeg = a;
    int *pxbeg = x, p = 0;
    int *pybeg = y, q = 0;
    while (pabeg < paend)
    {
        if (*pabeg < 0)
        {
            *pxbeg = *pabeg;
            p++;
            pxbeg++;
        }
        if (*pabeg > 0)
        {
            *pybeg = *pabeg;
            q++;
            pybeg++;
        }
        pabeg++;
    }
    pxbeg = x;
    pybeg = y;
    // Реверсирование массива
    //ukazatel1 < y + q / 2 && ukazatel2 > y + q / 2
    for (int *ukazatel1 = y, *ukazatel2 = y + q - 1; ukazatel1 < y + q /
2; ukazatel1++, ukazatel2--)
    {
        int temp = *ukazatel2;
        *ukazatel2 = *ukazatel1;
        *ukazatel1 = temp;
    }
    if (p == 0 || q == 0)
    {
        printf("Нет отрицательных / положительных элементов!\n");
        return NOELEMS_ERROR;
    }
    int *pxend = pxbeg + p;
    int *pyend = pybeg + q;
    int rez = count(pxbeg, pxend, pybeg, pyend);
    printf("%d", rez);
```

```
    return SUCCESS;
}
```

Для того, чтобы провести тестирование по времени, надо обеспечить автоматическое заполнение массива элементами, так же необходимо, что бы в массиве присутствовали как положительные, так и отрицательные элементы. Для этого используется этот алгоритм получения рандомного числа, для последующей записи его в массив:

```
int a = rand() % 10;
if (a > 5)
{
    a *= -1;
}
else if (a < 5)
{
    a *= 2;
}
```

Так как тестируемый алгоритм является линейным и работает достаточно быстро, то вместо миллисекунд, используются микросекунды.

Эта функция возвращает количество микросекунд с некоторого момента:

```
// Получение времени в микросекундах
unsigned long long microseconds_now(void)
{
    struct timeval val;
    if (gettimeofday(&val, NULL))
    {
        return (unsigned long long) - 1;
    }
    return val.tv_sec * 1000000ULL + val.tv_usec; //  Получение
микросекунд из секунд и микросекунд
}
```

Так как каждый раз программа выполняется разное время, то для более точного значения, она запускается несколько раз, тогда среднеарифметическое время выполнения будет более точным:

```
#!/bin/bash
gcc -o app.exe main.c
sum=0
loop=1000
for (( i=1; i <= $loop; i++ ))
do
    ./app.exe > "trash.txt"
    sum=$[ $sum+$? ]
done
calc() { awk "BEGIN{print $*}"; }
calc $sum / $loop
rm "trash.txt"
```

Так же для проведения экспериментов необходимо, чтобы через консоль можно было задавать количество элементов в массиве.

```
int main(int argc, char *argv[])
```

Так как в программе до этого использовался define N, то теперь это значение можно менять при запуске программы.

```
if (argc != 2)
{
    printf("Неверный параметр!\n");
    return ERROR;
}
int N = atoi(argv[1]);
```

1. Разные способы работы с элементами одномерного массива:

a) использование операции индексации a[i];

Целевая функция count с использованием операции индексации

Программа program_1.c

```
int count(const int x_arr[N], int k1, const int y_arr[N], int k2)
{
    int rez = 0;
    int k;
    if (k1 > k2)
    {
        k = k2;
    }
    else
    {
        k = k1;
    }
    for (int i =0; i < k; i++)
    {
        rez += x_arr[i] * y_arr[i];
    }
    return rez;
}
```

b) формальная замена операции индексации на выражение *(a + i);

Целевая функция count с формальной заменой операции индексации

Программа: program_02.c

```
int count(const int x_arr[N], int k1, const int y_arr[N], int k2)
{
    int rez = 0;
    int k;
    if (k1 > k2)
    {
        k = k2;
    }
    else
    {
        k = k1;
    }
    for (int i = 0; i < k; i++)
    {
```

```
        rez += *(x_arr + i) * *(y_arr + i);
    }
    return rez;
}
```

c) использование указателей для работы с массивом.

Целевая функция count с использованием указателей

Программа program_3.c

```
int count(int *xbeg, int *xend, int *ybeg, int *yend)
{
    int rez = 0;
    int k1 = (xend - xbeg);
    int k2 = (yend - ybeg);
    int k;
    if (k1 > k2)
    {
        k = k2;
    }
    else
    {
        k = k1;
    }
    while (k > 0)
    {
        rez += *xbeg * *ybeg;
        xbeg++;
        ybeg++;
        k--;
    }
    return rez;
}
```

2. Разные уровни оптимизации Os, O0, O1, O2, O3.

1. **build_apps.sh, вызвав который, можно получить весь набор необходимых исполняемых файлов.**

Имеется 3 программы programm1.c, programm2.c, programm3.c. Для сборки исполняемых файлов с разными уровнями оптимизации используется скрипт make_apps.sh, который собирает 15 исполняемых файлов и помещает их в папку apps.

Код скрипта

```
#!/bin/bash


opts="O1 O2 O3 O0 Os"
programms="programm_1 programm_2 programm_3"
for i in $programms; do
    for opt in $opts; do
        gcc -std=c99 -Wall -Werror -Wpedantic -Wextra \
        -"${opt}" \
        "$i".c -o ./apps/"${i}"_"${opt}".exe
        echo "${i}"_"${opt}".exe готово
    done
done
```

2. **update_data.sh, вызвав который, можно добавить некоторые данные в датасет экспериментов.**

Имеется 15 исполняемых файлов вида: programm_[номер_версии_программы]_[уровень_оптимизации].exe, при этом каждую программу нужно запустить с различным размером массива. Скрипт update_data.sh запускает все исполняемые файлы с заданными размерами массива, записывая или дополняя соответствующие файлы data.

Код скрипта

```
#!/bin/bash

programms="programm_1 programm_2 programm_3"
sizes=""
opts="O1 O2 O3 O0 Os"
count=100

i=300000

while [ $i -le 600000 ]
do
    sizes="$sizes $i"
    ((i+=15000))
done

if [ ! -z $1 ]; then
    count=$1
fi
if [ ! -z $2 ]; then
    sizes=$2
fi
if [ ! -z $3 ]; then
    programms=$3
fi
if [ ! -z $4 ]; then
    opts=$4
fi


for prog in $programms; do
    for co in $(seq "$count"); do
        for opt in $opts; do
            for i in $sizes; do
                echo -n -e "${prog}_""${opt}_""${i}\t $co/$count \r"
                ./apps/"${prog}"_"${opt}".exe
"${i}">>./data/"${prog}"_"${opt}"_"${i}".txt
            done
        done
    done
done
```

У скрипта есть возможность дополнять отдельные данные с помощью флагов при запуске

bash update_data.sh [Кол-во_тестов] [кол-во_элементов] [имя_программы] [оптимизация]

3. **make_preproc.sh|py, вызвав который, можно подготовить данные из набора, провести первичный анализ: посчитать среднее арифметическое, медианное, найти максимум и минимум, вычислить нижний и верхний квартили.**

Код программы

```
import os
import shutil


#  Функция принимает имя файла и возвращает массив целых чисел из этого файла
def read(str):
    with open(str, "r") as f:
        mass = []
        for line in f:
            a = int(line)
            mass.append(a)
    return mass


def find_sizes(folder):
    sizes = []
    files = os.listdir(folder)
    for i in range(len(files)):
        file = files[i]
        file = file[14:]
        file = file.replace(".txt", "", 1)
        sizes.append(int(file))
    sizes = list(set(sizes))
    sizes.sort()
    for i in range(len(sizes)):
        sizes[i] = str(sizes[i])
    # print(sizes)
    return sizes


#  Функция принимает массив и возвращает среднеарифметическое
def average(arr):
    sum = 0
    for i in range(len(arr)):
        sum += arr[i]
    return sum/len(arr)


#  Функция принимает массив и возвращает его медианное значение
def find_median(arr):
    arr.sort()
    # print(arr)
    if len(arr) == 1:
        return arr[0]
    if len(arr) % 2 == 0:
        return arr[len(arr)//2]
    return (int(arr[int(len(arr)/2) - 1]) + int(arr[int(len(arr)/2)])) / 2
```

```
#  Находит нижний квартиль
def finde_lower_quartile(arr):
    a = len(arr)
    #  граница 25%
    a //= 4
    return arr[a]


#  Находит верхний квартиль
def find_upper_quartile(arr):
    a = len(arr)
    #  Граница 75%
    a //= 4
    a *= 3
    return arr[a]


#  Функция удаляет подготовленные данные с прошлых эксперементов
def del_old(folder):
    shutil.rmtree(folder)
    os.mkdir(folder)


#  Записывает полученные данные в файл
def save_prep_data(stri, avg, med, mini, maxi, up_quart, low_quart):
    folder = './prep_data/'
    file = folder + stri
    with open(file, "w") as f:
        f.write(str(avg) + '\n')
        f.write(str(med) + '\n')
        f.write(str(mini) + '\n')
        f.write(str(maxi) + '\n')
        f.write(str(low_quart) + '\n')
        f.write(str(up_quart))
        file = "\r" + file



folder = './data/'
folder_out = './prep_data'

#  Удаляет предыдущие данные
del_old(folder_out)

programm = ['programm_1', 'programm_2', 'programm_3']
option = ['O0', 'O1', 'O2', 'O3', 'Os']
for i in range(len(option)):
    option[i] = "_" + option[i]

size = find_sizes(folder)
for i in range(len(size)):
    size[i] = "_" + size[i]

ex = '.txt'
n = 0
print("Подготовка данных")
for prog in range(len(programm)):
    for opt in range(len(option)):
        for siz in range(len(size)):
            stri = programm[prog] + option[opt] + size[siz] + ex
```

```
            arr = read(folder + stri)
            avg = average(arr)
            med = find_median(arr)
            low_quart = finde_lower_quartile(arr)
            up_quart = find_upper_quartile(arr)
            mini = arr[0]
            maxi = arr[len(arr) - 1]
            save_prep_data(stri, avg, med, mini, maxi, up_quart,
low_quart)
            n += 1
            progress = str(n) + ' / ' + str(len(programm) * len(option) *
len(size)) + '\r'
            print(progress, end="")
print()
```

Программа выполняет первичный анализ и записывает результаты в соответствующие файлы в папку prep_data.

Файлы записываются построчно на одной строке - 1 параметр

1. Среднеарифметическое
2. Медианное
3. Минимальное
4. Максимальное
5. Верхний квартиль
6. Нижний квартиль

4. **make_postproc.sh|py, вызвав который, можно получить указанные ниже графики.**

Код программы

```
import os
import shutil

import matplotlib.pyplot as plt

#  Получение информации из переданной строки о том, что содержится в
файле
def whats_in_file(file):
    sp_file = file.split("_")
    prog_num = sp_file[1]
    opt_lvl = sp_file[2]
    col_elems = sp_file[3]
    col_elems = col_elems.partition('.')[0]
    file = 'post_prep_data_' + prog_num + '_' + opt_lvl + '.txt'

    return prog_num, opt_lvl, col_elems, file


#  Плучение информации из файла
def params_from_file(folder_input, file):
    val = []
    with open(folder_input + file) as f:
        for line in f:
            val.append(float(line))
```

```
    return val


#  Функция подготавливает данные для дальнейшего построения графика
def linal_graph_data(files_input, folder_input, folder_out):
    print("____Таблица данных для кусочного графика____")
    print("col_elems\t time")
    files_input.sort()
    for file in files_input:
        avg = params_from_file(folder_input, file)[0]
        prog_num, opt_lvl, col_elems, file = whats_in_file(file)
        with open(folder_out + file, 'a') as f:
            f.write(str(col_elems) + ' ' + str(avg) + '\n')

    files_table = os.listdir(folder_out)
    files_table.sort()
    for file_tab in files_table:
        print("__tab data", file_tab, "__")
        with open(folder_out + file_tab) as f:
            for line in f:
                print(line, end="")


#  Сортировка массивов X и Y по X
def sort_2(arr1, arr2):
    for i in range(len(arr1)):
        for j in range(len(arr1) - 1):
            if arr1[j] > arr1[j + 1]:
                s = arr1[j]
                arr1[j] = arr1[j + 1]
                arr1[j + 1] = s
                s = arr2[j]
                arr2[j] = arr2[j + 1]
                arr2[j + 1] = s


#  Запись подготовленных данных в массивы по X и по Y для линейного
графика
def linal_graph_to_mass(folder):
    x_array = []
    y_array = []
    files_input = os.listdir(folder)
    files_input.sort()
    for file in files_input:
        file = folder + file
        x_arg = []
        y_arg = []
        with open(file, "r") as f:
            for line in f:
                args = line.split(' ')
                x_arg.append(int(args[0]))
                y_arg.append(float(args[1]))
        sort_2(x_arg, y_arg)
        x_array.append(x_arg)
        y_array.append(y_arg)
    return x_array, y_array, files_input
```

```
#  Строит линейный график
def plot_linal_graph(x_array, y_array, labels_array):
    #  Удаление лишнего из лэйблов
    for i in range(len(labels_array)):
        labels_array[i] = labels_array[i].replace(".txt", "")
        labels_array[i] = labels_array[i].replace("post_prep_", "",
1)
        labels_array[i] = labels_array[i].replace("data", "prog")
    fig, ax = plt.subplots()
    #  Набор параметров для отображения графиков
    linestyle_array = ['-', '--', '-.', ':', (0, (3, 1, 1, 1, 1, 1))]
    markers = ['^', 's', 'X']
    #  Генерация 15 видов линий
    difference = []
    for j in range(len(markers)):
        for i in range(len(linestyle_array)):
            typ = [linestyle_array[i], markers[j]]
            difference.append(typ)
    #  Построение графиков
    for i in range(len(x_array)):
        ax.plot(x_array[i], y_array[i], label=labels_array[i],
linestyle=difference[i][0], marker=difference[i][1])

    #  Добавляем подписи к осям:
    ax.set_xlabel('Кол-во элементов')
    ax.set_ylabel('Время (микросекунды)')

    #  Добавление легенды, масштабной сетки и вывод графиков на поле
    ax.legend()
    ax.grid()

    #  Сохранение графика
    fig.savefig('line_plot.svg')


#  Функция удаляет подготовленные данные с прошлых эксперементов
def del_old(folder):
    shutil.rmtree(folder)
    os.mkdir(folder)


#  Функция подготавливает данные для графика с ошибкой
def error_graph_data(files_input, folder_input, folder_out):
    print("___Таблица график с ошибкой___")
    print("col_elems\t avg\t max\t min\t")
    files_input.sort()
    for file in files_input:
        params = params_from_file(folder_input, file)
        avg = params[0]
        maxi = params[3]
        mini = params[2]
        prog_num, opt_lvl, col_elems, file = whats_in_file(file)
        if opt_lvl == 'O2':
            with open(folder_out + file, 'a') as f:
                f.write(str(col_elems) + ' ' + str(avg) + ' ' +
str(maxi - avg) + ' ' + str(avg - mini) + '\n')
                # print(str(col_elems) + ' ' + str(avg) + ' ' +
str(maxi) + ' ' + str(mini))
```

```
    files_table = os.listdir(folder_out)
    files_table.sort()
    for file_tab in files_table:
        print("__tab data", file_tab, "__")
        with open(folder_out + file_tab) as f:
            for line in f:
                line = line.split()
                print(line[0], line[1], int(float(line[2]) +
float(line[1])), int(float(line[3]) + float(line[1])))

#  Сортирует 3 массива по первому
def sort3(arr1, arr2, arr3):
    for i in range(len(arr1)):
        for j in range(len(arr1) - 1):
            if arr1[j] > arr1[j + 1]:
                s = arr1[j]
                arr1[j] = arr1[j + 1]
                arr1[j + 1] = s

                s = arr2[j]
                arr2[j] = arr2[j + 1]
                arr2[j + 1] = s

                s = arr3[0][j]
                arr3[0][j] = arr3[0][j + 1]
                arr3[0][j + 1] = s

                s = arr3[1][j]
                arr3[1][j] = arr3[1][j + 1]
                arr3[1][j + 1] = s


#  Записывает данные из файлов по массивам
def error_graph_to_mass(folder):
    x_array = []
    y_array = []
    errors_array = []
    files_data = os.listdir(folder)
    files_data.sort()
    for file in files_data:
        file = folder + file
        x_arg = []
        y_arg = []
        errors_args = []
        err_max = []
        err_min = []
        with open(file, "r") as f:
            for line in f:
                args = line.split(' ')
                x_arg.append(int(args[0]))
                y_arg.append(float(args[1]))
                err_max.append(float(args[2]))
                err_min.append(float(args[3]))
        errors_args.append(err_min.copy())
        errors_args.append(err_max.copy())
        sort3(x_arg, y_arg, errors_args)
        x_array.append(x_arg)
```

```
        y_array.append(y_arg)
        errors_array.append(errors_args)
    return x_array, y_array, errors_array, files_data


#  Строит график ошибок
def plot_error_graph(x_array, y_array, errors_array, labels_array):
    #  Удаление лишнего из лэйблов
    for i in range(len(labels_array)):
        labels_array[i] = labels_array[i].replace(".txt", "")
        labels_array[i] = labels_array[i].replace("post_prep_", "",
1)
        labels_array[i] = labels_array[i].replace("data", "prog")
    fig, ax = plt.subplots()
    #  Набор параметров для отображения графиков
    markers = ['^', 's', 'X']
    linestyle_array = ['-', '--', '-.']
    #  Построение графиков
    for i in range(len(x_array)):
        ax.errorbar(x_array[i], y_array[i], yerr=errors_array[i],
label=labels_array[i], marker=markers[i],
                    linestyle=linestyle_array[i])
    #  Добавляем подписи к осям:
    ax.set_xlabel('Кол-во элементов')
    ax.set_ylabel('Время (микросекунды)')
    #  Добавление легенды, масштабной сетки и вывод графиков на поле
    ax.legend()
    ax.grid()

    #  Сохранение графика
    fig.savefig('error_plot.svg')


# График с усами (среднее, максимум, минимум; нижний, средний и
верхний квартили) для варианта обработки
# «через квадратные скобки» при уровне оптимизации O3

#  Записывает данные из data в массив
def boxplot_prep_data(folder_prep_data, folder_for_table):
    print("__Таблица данных для графика с усами__")
    print("col_elems\tavg\t med\t min\t max\t up_quart\t low_quart")
    files_table = os.listdir(folder_for_table)
    files_table.sort()
    for file in files_table:
        prog_num, opt_lvl, col_elems, file1 = whats_in_file(file)
        params = params_from_file(folder_prep_data, file)
        avg = params[0]
        median = params[1]
        maxi = params[3]
        mini = params[2]
        up_quart = params[4]
        low_quart = params[5]
        if opt_lvl == "O2" and prog_num == '1':
            print(col_elems, avg, median, mini, maxi, up_quart,
low_quart, sep='\t')
    data_array = []
    elements_data = []
    times = []
```

```
    for file in files:
        time_data = []
        time_data.clear()
        #  Сбор данных
        prog_num, opt_lvl, col_elems, file1 = whats_in_file(file)
        #  Запись подготовленных данных
        if opt_lvl == 'O3' and prog_num == '1':
            with open(folder_prep_data + file, "r") as f:
                for line in f:
                    time_data.append(float(line))
                times.append(time_data)
            elements_data.append(col_elems)

    data_array.append(times.copy())
    data_array.append(elements_data.copy())
    sort_2(data_array[0], data_array[1])
    return data_array


def plot_boxplot(arr):
    fig, ax = plt.subplots()
    #  Набор параметров для отображения графиков
    #  Построение графиков

    # Creating plot
    ax.boxplot(arr[0])

    #  Добавляем подписи к осям:
    # ax.set_xlabel('Кол-во элементов')
    ax.set_ylabel('Время (микросекунды)')

    #  Добавление легенды, масштабной сетки и вывод графиков на поле
    ax.grid()

    #  Сохранение графика
    fig.savefig('boxplot.svg')


#  Заданные папки
folder_in = "prep_data/"
folder_lingraph = "post_prep_lingraph_data/"
folder_errorgraph = "errorgraph_data/"
folder_boxplot = "post_prep_box_plot_data/"
folder_data = 'data/'

#  Массив в котором хранятся имена файлов из папки folder_in
files = os.listdir(folder_in)

#  Линейный график

#  Удаление старых данных
del_old(folder_lingraph)

#  Подготовка данных для построения графика
linal_graph_data(files, folder_in, folder_lingraph)

#  Построение графика по подготовленным данным
x, y, labels = linal_graph_to_mass(folder_lingraph)
```

```
plot_linal_graph(x, y, labels)

#  График с ошибкой

#  Удаление старых данных
del_old(folder_errorgraph)

#  Подготовка данных для построения графика
error_graph_data(files, folder_in, folder_errorgraph)

#  Построение графика по подготовленным данным
x, y, errors, labels = error_graph_to_mass(folder_errorgraph)
plot_error_graph(x, y, errors, labels)

#  Ящик с усами


#  Подготовка данных для построения графика
data_arr = boxplot_prep_data(folder_data, folder_in)

plot_boxplot(data_arr)


#  Показать все построенное
plt.show()
```

Скрипт строит 3 графика, выводит их на экран, сохраняет их в папку скрипта и выводит в консоль данные, по которым их строит.

5. **go.sh, вызвав который, можно получить данные эксперимента (скрипт вызывает по очереди предыдущие четыре).**

Код скрипта

```
#!/bin/bash
mkdir "apps"
bash build_apps.sh
mkdir "data"
bash update_data.sh
mkdir "prep_data"
python3 make_preproc.py
mkdir "errorgraph_data"
mkdir "post_prep_lingraph_data"
python3 make_postproc.py
```

Графики

Из-за простоты и линейности замеряемой функции, для демонстрации графиков был выбран отрезок для замера времени от 300 000 элементов до 600 000 элементов с шагом 15 000 элементов.

1. **Обычный кусочно-линейный график зависимости времени выполнения в любых единицах измерения времени от числа элементов массива для всех 15 вариантов программы.**

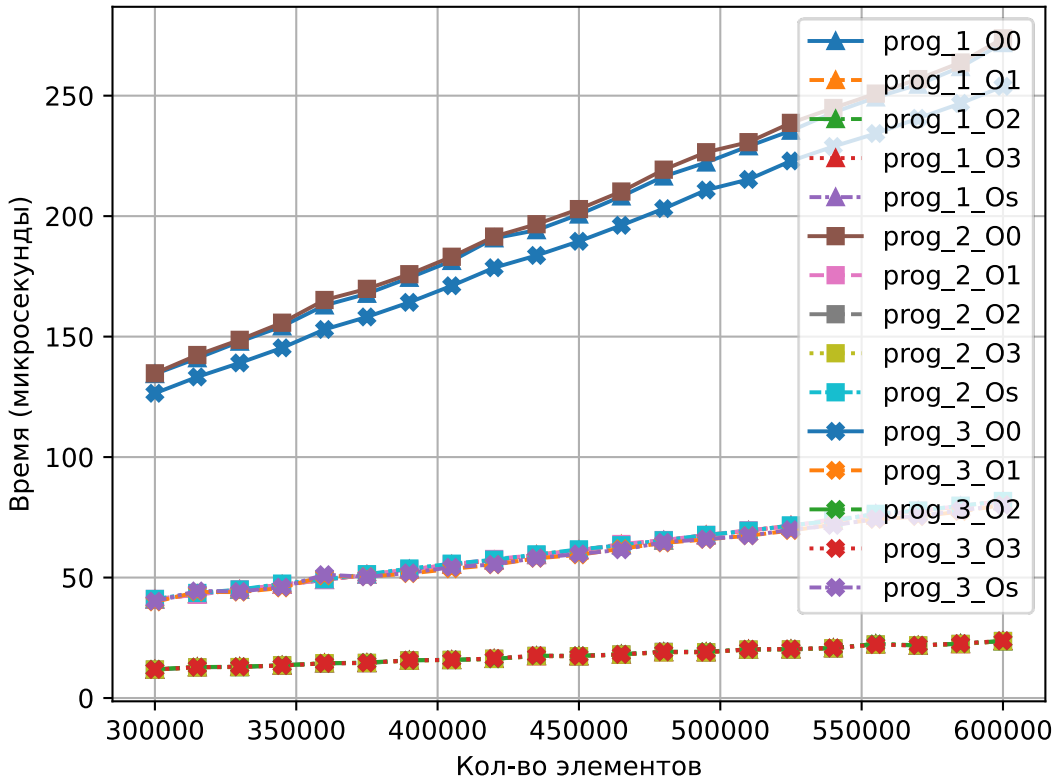

Таблица к кусочно-линейному графику

prog_1_O0		prog_1_O1		prog_1_O2		prog_1_O3		prog_1_Os	
col_elems	time	col_elems	time	col_elems	col_elems	time	col_elems	time	col_elems
300000	134.46	300000	40.89	300000	11.74	300000	11.78	300000	40.84
315000	141.03	315000	43.18	315000	12.85	315000	12.77	315000	43.03
330000	147.9	330000	44.97	330000	12.86	330000	12.96	330000	44.95
345000	154.26	345000	47.04	345000	13.53	345000	13.47	345000	47.0
360000	162.96	360000	49.04	360000	14.44	360000	14.41	360000	48.95
375000	167.77	375000	51.14	375000	14.52	375000	14.51	375000	51.18
390000	174.41	390000	53.29	390000	15.67	390000	15.57	390000	53.43
405000	181.41	405000	55.37	405000	15.76	405000	15.81	405000	55.33
420000	190.8	420000	57.19	420000	16.22	420000	16.32	420000	57.1
435000	194.18	435000	59.45	435000	17.48	435000	17.33	435000	59.32
450000	200.68	450000	61.34	450000	17.42	450000	17.32	450000	61.42
465000	208.19	465000	63.76	465000	18.19	465000	17.99	465000	63.44
480000	216.47	480000	65.32	480000	19.1	480000	19.02	480000	65.44

495000	222.27	495000	67.65	495000	19.07	495000	19.0	495000	67.47
510000	229.0	510000	69.47	510000	20.17	510000	20.16	510000	69.45
525000	235.46	525000	71.52	525000	20.28	525000	20.18	525000	71.56
540000	242.99	540000	73.76	540000	20.78	540000	20.66	540000	73.53
555000	249.27	555000	75.77	555000	22.52	555000	22.18	555000	76.48
570000	254.57	570000	77.63	570000	21.76	570000	21.79	570000	77.7
585000	261.87	585000	79.46	585000	22.62	585000	22.35	585000	79.7
600000	271.83	600000	81.85	600000	23.51	600000	23.66	600000	81.76
prog_2_O0		prog_2_O1		prog_2_O2		prog_2_O3		prog_2_Os	
col_elems	time	col_elems	time	col_elems	col_elems	time	col_elems	time	col_elems
300000	134.77	300000	41.13	300000	11.81	300000	11.83	300000	41.11
315000	142.35	315000	42.82	315000	12.77	315000	12.72	315000	43.6
330000	148.66	330000	45.11	330000	12.83	330000	12.85	330000	45.03
345000	155.77	345000	47.01	345000	13.6	345000	13.55	345000	47.48
360000	165.19	360000	49.05	360000	14.39	360000	14.47	360000	49.31
375000	169.82	375000	51.21	375000	14.55	375000	14.58	375000	51.39
390000	175.9	390000	53.3	390000	15.66	390000	15.54	390000	53.75
405000	183.14	405000	55.44	405000	15.83	405000	15.79	405000	55.75
420000	191.52	420000	57.71	420000	16.3	420000	16.25	420000	57.41
435000	196.65	435000	59.34	435000	17.5	435000	17.42	435000	59.62
450000	202.92	450000	61.24	450000	17.33	450000	17.34	450000	61.7
465000	210.21	465000	63.92	465000	18.09	465000	18.0	465000	63.4
480000	219.28	480000	65.57	480000	19.18	480000	18.97	480000	65.45
495000	226.52	495000	67.68	495000	19.0	495000	18.95	495000	67.69
510000	230.7	510000	69.62	510000	20.09	510000	20.11	510000	69.43
525000	238.7	525000	71.39	525000	20.16	525000	20.17	525000	71.65
540000	244.88	540000	74.35	540000	20.6	540000	20.63	540000	73.66
555000	250.85	555000	75.85	555000	22.23	555000	22.21	555000	76.44
570000	256.87	570000	77.73	570000	21.81	570000	21.84	570000	77.8
585000	263.76	585000	79.75	585000	22.46	585000	22.41	585000	79.79
600000	273.78	600000	81.6	600000	23.69	600000	23.6	600000	81.78
prog_3_O0		prog_3_O1		prog_3_O2		prog_3_O3		prog_3_Os	
col_elems	time	col_elems	time	col_elems	col_elems	time	col_elems	time	col_elems
300000	126.48	300000	39.89	300000	11.97	300000	11.85	300000	40.2
315000	133.26	315000	44.03	315000	12.86	315000	12.82	315000	44.45
330000	139.09	330000	43.95	330000	13.02	330000	12.96	330000	44.22
345000	145.33	345000	45.69	345000	13.59	345000	13.5	345000	46.03
360000	152.96	360000	50.71	360000	14.42	360000	14.38	360000	51.21

375000	158.16	375000	50.38	375000	14.77	375000	14.6	375000	50.31
390000	164.22	390000	51.64	390000	15.67	390000	15.69	390000	51.94
405000	171.11	405000	53.61	405000	15.92	405000	15.84	405000	54.28
420000	178.59	420000	55.3	420000	16.25	420000	16.44	420000	55.45
435000	183.69	435000	57.98	435000	17.57	435000	17.63	435000	58.11
450000	189.56	450000	59.41	450000	17.58	450000	17.45	450000	59.8
465000	196.19	465000	61.97	465000	18.15	465000	18.11	465000	61.52
480000	203.12	480000	64.35	480000	19.21	480000	19.2	480000	64.65
495000	210.83	495000	65.83	495000	19.28	495000	19.03	495000	65.96
510000	215.23	510000	67.37	510000	20.2	510000	20.23	510000	67.36
525000	222.91	525000	69.52	525000	20.29	525000	20.35	525000	69.87
540000	228.9	540000	71.94	540000	20.74	540000	20.8	540000	71.64
555000	234.29	555000	74.13	555000	22.48	555000	22.19	555000	74.37
570000	240.6	570000	75.33	570000	21.91	570000	21.85	570000	75.54
585000	246.79	585000	77.39	585000	22.54	585000	22.61	585000	77.91
600000	253.85	600000	80.15	600000	23.73	600000	23.77	600000	80.06

2. **Кусочно-линейный график с ошибкой (среднее, максимум, минимум) для всех вариантов обработки массива при уровне оптимизации 02.**

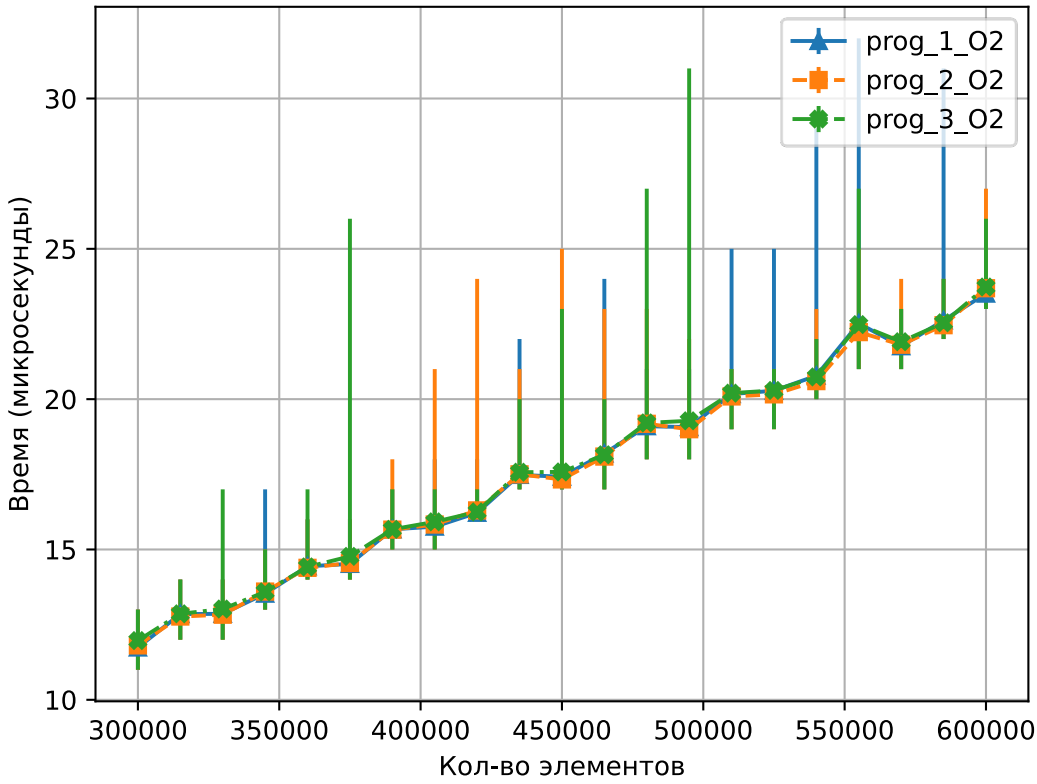

Таблица к кусочно-линейному графику ошибок

Prog_1_O2				Prog_2_O2				Prog_3_O2			
col_elems	avg	max	min	col_elems	avg	max	min	col_elems	avg	max	min
300000	11.74	13	11	300000	11.81	13	11	300000	11.97	13	11
315000	12.85	14	12	315000	12.77	14	12	315000	12.86	14	12
330000	12.86	14	12	330000	12.83	14	12	330000	13.02	17	12
345000	13.53	17	13	345000	13.6	15	13	345000	13.59	15	13
360000	14.44	16	14	360000	14.39	16	14	360000	14.42	17	14
375000	14.52	16	14	375000	14.55	16	14	375000	14.77	26	14
390000	15.67	17	15	390000	15.66	18	15	390000	15.67	17	15
405000	15.76	18	15	405000	15.83	21	15	405000	15.92	17	15
420000	16.22	18	16	420000	16.3	24	16	420000	16.25	17	16
435000	17.48	22	17	435000	17.5	21	17	435000	17.57	20	17
450000	17.42	19	17	450000	17.33	25	17	450000	17.58	23	17
465000	18.19	24	17	465000	18.09	23	17	465000	18.15	20	17
480000	19.1	21	18	480000	19.18	23	18	480000	19.21	27	18
495000	19.07	21	18	495000	19	22	18	495000	19.28	31	18
510000	20.17	25	19	510000	20.09	21	19	510000	20.2	21	19

525000	20.28	25	20	525000	20.16	21	19	525000	20.29	21	19
540000	20.78	29	20	540000	20.6	23	20	540000	20.74	22	20
555000	22.52	32	21	555000	22.23	25	21	555000	22.48	27	21
570000	21.76	23	21	570000	21.81	24	21	570000	21.91	23	21
585000	22.62	31	22	585000	22.46	24	22	585000	22.54	24	22
600000	23.51	25	23	600000	23.69	27	23	600000	23.73	26	23

3. **График с усами (среднее, максимум, минимум; нижний, средний и верхний квартили) для варианта обработки «через квадратные скобки» при уровне оптимизации 03.**

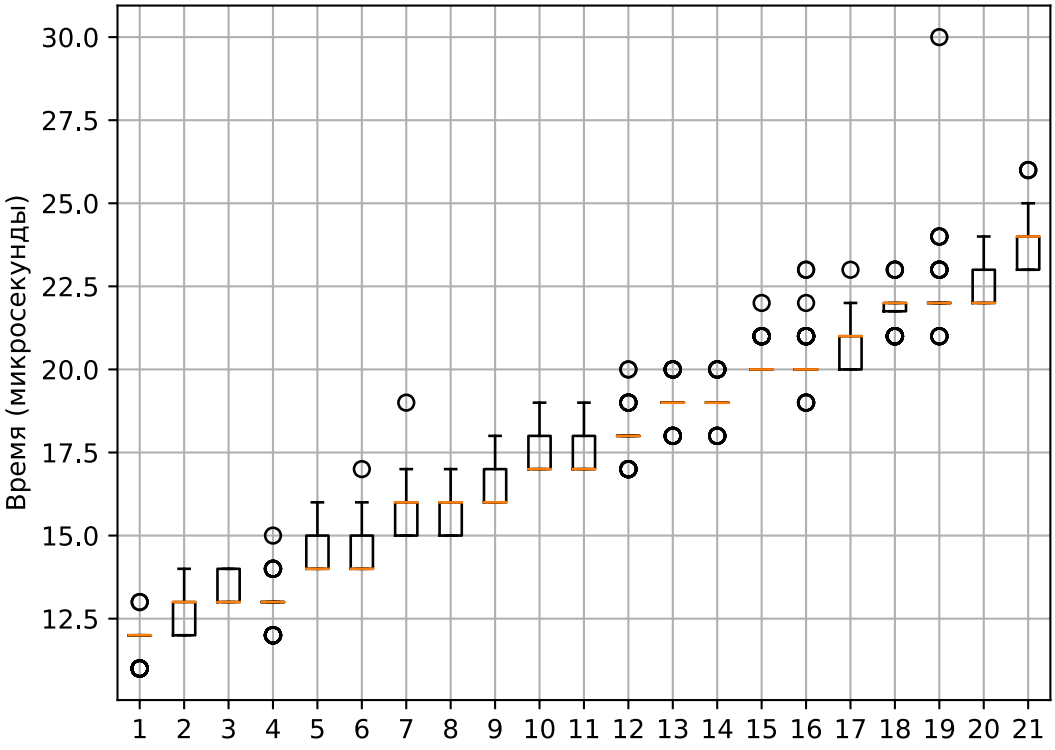

Таблица для графика с усами

data 1 O3						
col_elems	avg	med	min	max	up_quart	low_quart
300000	12	13	12	12	12	12
315000	12	13	13	13	13	13
330000	13	14	13	13	13	13
345000	13	13	13	13	13	13
360000	14	14	14	14	15	14
375000	15	14	16	14	15	14

390000	16	16	16	16	16	15
405000	16	16	16	15	16	16
420000	17	17	16	16	16	16
435000	18	17	18	17	18	17
450000	18	17	17	17	18	17
465000	18	18	18	17	19	17
480000	19	20	19	19	19	20
495000	19	19	19	19	19	19
510000	21	21	20	20	21	19
525000	20	21	20	20	20	20
540000	21	21	20	21	21	20
555000	22	23	22	22	22	22
570000	22	22	22	21	22	22
585000	23	24	22	22	23	23

Таблица для результатов обрабботки

Таблица для результатов обработки «через квадратные скобки» с уровнем оптимизации O2 со столбцами: длина массива n, время выполнения tn, величина $\frac{\ln(t_{i+1})-\ln(t_i)}{\ln(n_{i+1})-\ln(n_i)}$ для всех строк, кроме последней.

N	T	Заданная величина
300000	11.00	3
315000	13.00	0
330000	13.00	0
345000	13.00	2
360000	14.00	0
375000	14.00	3
390000	16.00	-2
405000	15.00	2
420000	16.00	2
435000	17.00	0
450000	17.00	2
465000	18.00	5
480000	21.00	-2
495000	20.00	3
510000	22.00	0
525000	22.00	-2
540000	21.00	2
555000	22.00	0
570000	22.00	2
585000	23.00	2

600000	24.00	

Ответы на вопросы

1. **Какой способ обработки быстрее и почему?**

Исходя из кусочно-линейного графика, можно сделать вывод, что особенности работы с массивом имеет значение только при уровне оптимизации O0, это доказывает график ошибок, где все 3 программы имеют уровень оптимизации O2, графики слипаются. Программа с формальной заменой индексации работает медленнее остальных, программа с индексацией работает немного быстрее, а программа с работой через указатель работает значительно быстрее. Это происходит из-за того, что:

- При работе через указатель используется прямой доступ к памяти.
- При работе через индекс используется операция индексации, а после операция доступа к памяти
- При формальной индексации используются арифметические операции, а прямой доступ к памяти

Таким образом, для доступа к элементу массиву через указатель программа производит одну операцию, а если действовать оставшимися двумя способами, то программа производит две операции для достижения того-же результата.

2. В датасете появился статистический выброс, причины которого очевидны, например, эксперимент был поставлен на другой машине. Порядок экспериментов не был известен заранее. Можно ли вырезать данные со статистическим выбросом из датасета?

Так как известна причина выброса, то можно.

3. **В датасете появился статистический выброс. Известно, что эксперимент был поставлен вчера, существует резервная копия датасета за позавчера. Можно ли вырезать данные со статистическим выбросом из датасета?**

Нет нельзя, можно удалить датасет и создать его заного, или разобраться в причине выброса.

4. **В датасете обнаружена серия экспериментов с одним результатом. Можно ли заменить её одним экспериментом?**

Нет, нельзя, так как эти результаты влияют на среднеарифметическое, медианное и квартили экспериментов.

5. **Если заполнение случайными числами массива (или любая другая инициализация) присутствует в каждом эксперименте, то почему Вы замеряете время только у целевого алгоритма?**

Так как инициализация и заполнение выполняется некоторое время, а задача измерить время выполнения целевого алгоритма, то для большей точности экспериментов, время замеряют только у целевого алгоритма.