Задание № 4.1

Введение:

Цель работы – изучить различные способы замера времени выполнения программы и провести сравнение производительности работы программы по трём плоскостям.

Задание №1

defines.h

```
#ifndef __DEFINES_H__
#define __DEFINES_H__

#include <sys/time.h>
#include <time.h>
#include <stdio.h>
#include <stdlib.h>

#define NUM_OF_TESTS 20
#define NSEC_IN_SEC 1e9
#define USEC_IN_SEC 1e6
#define MSEC_IN_SEC 1e3
#define OK 0
#define ERROR 1

#endif
```

clock_gettime_test.c

```
#include "defines.h"

int main(int argc, char **argv)
{
    struct timespec req, res;
    struct timespec start_time, end_time;
    double avg = 0;
    int rc;
    if (argc != 2)
    {
```

```
        rc = ERROR;
        printf("Incorrect arguments\n");
    }
    else
    {
        rc = OK;
        req.tv_nsec = strtol(argv[1], NULL, 10) %
((int)MSEC_IN_SEC) * USEC_IN_SEC;
        req.tv_sec = strtol(argv[1], NULL, 10) /
((int)MSEC_IN_SEC);
        for (int i = 0; i < NUM_OF_TESTS; i++)
        {
            clock_gettime(CLOCK_REALTIME, &start_time);
            nanosleep(&req, &res);
            clock_gettime(CLOCK_REALTIME, &end_time);
            avg += (end_time.tv_nsec - start_time.tv_nsec) /
NSEC_IN_SEC + (end_time.tv_sec - start_time.tv_sec);
        }
        avg /= NUM_OF_TESTS;
        printf("clock_gettime: nanosleep(%s ms) - %lf s\n",
argv[1], avg);
    }
    return rc;
}
```

clock_test.c

```
#include "defines.h"

int main(int argc, char **argv)
{
    struct timespec req, res;
    clock_t start_time, end_time;
    double avg = 0;
    int rc;
    if (argc != 2)
    {
        rc = ERROR;
        printf("Incorrect arguments\n");
    }
```

```
    else
    {
        rc = OK;
        req.tv_nsec = strtol(argv[1], NULL, 10) %
((int)MSEC_IN_SEC) * USEC_IN_SEC;
        req.tv_sec = strtol(argv[1], NULL, 10) /
((int)MSEC_IN_SEC);
        for (int i = 0; i < NUM_OF_TESTS; i++)
        {
            start_time = clock();
            nanosleep(&req, &res);
            end_time = clock();
            avg += (double)(end_time - start_time) /
CLOCKS_PER_SEC;
        }
        avg /= NUM_OF_TESTS;
        printf("clock: nanosleep(%s ms) - %lf s\n", argv[1], avg);
    }
    return rc;
}
```

gettimeofday_test.c

```
#include "defines.h"

int main(int argc, char **argv)
{
    struct timespec req, res;
    struct timeval start_time, end_time;
    double avg = 0;
    int rc;
    if (argc != 2)
    {
        rc = ERROR;
        printf("Incorrect arguments\n");
    }
    else
    {
        rc = OK;
```

```
        req.tv_nsec = strtol(argv[1], NULL, 10) %
((int)MSEC_IN_SEC) * USEC_IN_SEC;
        req.tv_sec = strtol(argv[1], NULL, 10) /
((int)MSEC_IN_SEC);
        for (int i = 0; i < NUM_OF_TESTS; i++)
        {
            gettimeofday(&start_time, NULL);
            nanosleep(&req, &res);
            gettimeofday(&end_time, NULL);
            avg += (end_time.tv_usec - start_time.tv_usec) /
USEC_IN_SEC + (end_time.tv_sec - start_time.tv_sec);
        }
        avg /= NUM_OF_TESTS;
        printf("gettimeofday: nanosleep(%s ms) - %lf s\n",
argv[1], avg);
    }
    return rc;
}
```

rdtsc_test.c

```
#include "defines.h"
#include <x86gprintrin.h>

int main(int argc, char **argv)
{
    struct timespec req, res;
    unsigned long long start_time, end_time;
    unsigned long long avg = 0;
    int rc;
    if (argc != 2)
    {
        rc = ERROR;
        printf("Incorrect arguments\n");
    }
    else
    {
        rc = OK;
        req.tv_nsec = strtol(argv[1], NULL, 10) %
((int)MSEC_IN_SEC) * USEC_IN_SEC;
```

```
        req.tv_sec = strtol(argv[1], NULL, 10) /
((int)MSEC_IN_SEC);
        for (int i = 0; i < NUM_OF_TESTS; i++)
        {
            start_time = __rdtsc();
            nanosleep(&req, &res);
            end_time = __rdtsc();
            avg += end_time - start_time;
        }
        avg /= NUM_OF_TESTS;
        printf("__rdtsc: nanosleep(%s ms) - %lld ticks\n",
argv[1], avg);
    }
    return rc;
}
```

Результаты исследования:

	10 ms	50 ms	100 ms	1000 ms
clock_gettime	0.011020 s	0.051405 s	0.102937 s	1.004083 s
clock	0.000039 s	0.000055 s	0.000061 s	0.000056 s
gettimeofday	0.015704 s	0.051203 s	0.101594 s	1.011435 s
__rdtsc	27271910 ticks	127376319 ticks	257141141 ticks	2521421176 ticks

Задание №2

main.h

```
#ifndef __MAIN_H__
#define __MAIN_H__

#include <stdio.h>
#include <time.h>
#include <math.h>
#include <string.h>
#include <stdlib.h>

#define OK 0
#define ERROR 1
```

```
#define NUM_OF_TESTS 200
#define NSEC_IN_SEC 1e9
#define SORTED_KEY "s"

#endif
```

brackets.c

```
#include "main.h"

int check_arr_sorted(int, char **);
double calc_sort_time(int *, int);
void sort(int *, int);
void shift(int *, int, int);
void gen_test(int *, int, int);
void output(double *);

int main(int argc, char **argv)
{
    int length = ARRAY_LENGTH, arr[ARRAY_LENGTH], is_arr_sorted;
    double time_results[NUM_OF_TESTS];
    is_arr_sorted = check_arr_sorted(argc, argv);
    for(int i = 0; i < NUM_OF_TESTS; i++)
    {
        gen_test(arr, length, is_arr_sorted);
        time_results[i] = calc_sort_time(arr, length);
    }
    output(time_results);
    return OK;
}

int check_arr_sorted(int argc, char **argv)
{
    int rc = 0;
    if (argc != 1)
    {
        if (strcmp(argv[1], SORTED_KEY) == 0)
        {
            rc = 1;
        }
    }
```

```
    return rc;
}

void gen_test(int *arr, int length, int is_arr_sort)
{
    for (int i = 0; i < length; i++)
    {
        arr[i] = rand() % 10;
    }
    if (is_arr_sort)
    {
        sort(arr, length);
    }
}

double calc_sort_time(int *arr, int length)
{
    struct timespec start_time, end_time;
    clock_gettime(CLOCK_REALTIME, &start_time);
    sort(arr, length);
    clock_gettime(CLOCK_REALTIME, &end_time);
    return (end_time.tv_nsec - start_time.tv_nsec) / NSEC_IN_SEC +
(end_time.tv_sec - start_time.tv_sec);
}

void output(double *time_results)
{
    for (int i = 0; i < NUM_OF_TESTS; i++)
    {
        printf("%lf\n", time_results[i]);
    }
}

void sort(int *arr, int length)
{
    int flag_index, temp;
    for (int i = 1; i < length; i++)
    {
        temp = arr[i];
        if (arr[i] < arr[0])
        {
            shift(arr, 0, i);
```

```
            arr[0] = temp;
        }
        else
        {
            flag_index = i - 1;
            for (flag_index = i - 1; arr[flag_index] > arr[i]; )
                flag_index--;
            flag_index++;
            shift(arr, flag_index, i);
            arr[flag_index] = temp;
        }
    }
}

void shift(int *arr, int i_beg, int i_end)
{
    for (int i_cur = i_end; i_cur >= i_beg; i_cur--)
    {
        arr[i_cur] = arr[i_cur - 1];
    }
}
```

index.c

```
#include "main.h"

int check_arr_sorted(int, char **);
double calc_sort_time(int *, int);
void sort(int *, int);
void shift(int *, int, int);
void gen_test(int *, int, int);
void output(double *);

int main(int argc, char **argv)
{
    int length = ARRAY_LENGTH, arr[ARRAY_LENGTH], is_arr_sorted;
    double time_results[NUM_OF_TESTS];
    is_arr_sorted = check_arr_sorted(argc, argv);
    for(int i = 0; i < NUM_OF_TESTS; i++)
    {
        gen_test(arr, length, is_arr_sorted);
```

```
            time_results[i] = calc_sort_time(arr, length);
        }
        output(time_results);
        return OK;
}

int check_arr_sorted(int argc, char **argv)
{
        int rc = 0;
        if (argc != 1)
        {
            if (strcmp(argv[1], SORTED_KEY) == 0)
            {
                rc = 1;
            }
        }
        return rc;
}

void gen_test(int *arr, int length, int is_arr_sort)
{
        for (int i = 0; i < length; i++)
        {
            arr[i] = rand() % 10;
        }
        if (is_arr_sort)
        {
            sort(arr, length);
        }
}

double calc_sort_time(int *arr, int length)
{
        struct timespec start_time, end_time;
        clock_gettime(CLOCK_REALTIME, &start_time);
        sort(arr, length);
        clock_gettime(CLOCK_REALTIME, &end_time);
        return (end_time.tv_nsec - start_time.tv_nsec) / NSEC_IN_SEC +
(end_time.tv_sec - start_time.tv_sec);
}

void output(double *time_results)
```

```
{
    for (int i = 0; i < NUM_OF_TESTS; i++)
    {
        printf("%lf\n", time_results[i]);
    }
}

void sort(int *arr, int length)
{
    int flag_index, temp;
    for (int i = 1; i < length; i++)
    {
        temp = *(arr + i);
        if (*(arr + i) < *arr)
        {
            shift(arr, 0, i);
            *arr = temp;
        }
        else
        {
            flag_index = i - 1;
            for (flag_index = i - 1; *(arr + flag_index) > *(arr +
i); )
                flag_index--;
            flag_index++;
            shift(arr, flag_index, i);
            *(arr + flag_index) = temp;
        }
    }
}

void shift(int *arr, int i_beg, int i_end)
{
    for (int i_cur = i_end; i_cur >= i_beg; i_cur--)
    {
        *(arr + i_cur) = *(arr + i_cur - 1);
    }
}
```

point.c

```
#include "main.h"

int check_arr_sorted(int, char **);
double calc_sort_time(int *, int);
void sort(int *, int);
void shift(int *, int *);
void gen_test(int *, int, int);
void output(double *);

int main(int argc, char **argv)
{
    int length = ARRAY_LENGTH, arr[ARRAY_LENGTH], is_arr_sorted;
    double time_results[NUM_OF_TESTS];
    is_arr_sorted = check_arr_sorted(argc, argv);
    for(int i = 0; i < NUM_OF_TESTS; i++)
    {
        gen_test(arr, length, is_arr_sorted);
        time_results[i] = calc_sort_time(arr, length);
    }
    output(time_results);
    return OK;
}

int check_arr_sorted(int argc, char **argv)
{
    int rc = 0;
    if (argc != 1)
    {
        if (strcmp(argv[1], SORTED_KEY) == 0)
        {
            rc = 1;
        }
    }
    return rc;
}

void gen_test(int *arr, int length, int is_arr_sort)
{
    for (int i = 0; i < length; i++)
    {
```

```
            arr[i] = rand();
        }
        if (is_arr_sort)
        {
            sort(arr, length);
        }
}

double calc_sort_time(int *arr, int length)
{
    struct timespec start_time, end_time;
    clock_gettime(CLOCK_REALTIME, &start_time);
    sort(arr, length);
    clock_gettime(CLOCK_REALTIME, &end_time);
    return (end_time.tv_nsec - start_time.tv_nsec) / NSEC_IN_SEC +
(end_time.tv_sec - start_time.tv_sec);
}

void output(double *time_results)
{
    for (int i = 0; i < NUM_OF_TESTS; i++)
    {
        printf("%lf\n", time_results[i]);
    }
}

void sort(int *arr_beg, int length)
{
    int *arr_flag = NULL, temp, *arr_end = arr_beg + length;
    for (int *arr_cur = arr_beg + 1; arr_cur < arr_end; arr_cur++)
    {
        temp = *arr_cur;
        if (*arr_cur < *arr_beg)
        {
            shift(arr_beg, arr_cur);
            *arr_beg = temp;
        }
        else
        {
            for (arr_flag = arr_cur - 1; *arr_flag > *arr_cur;)
                arr_flag--;
            arr_flag++;
```

```
            shift(arr_flag, arr_cur);
            *arr_flag = temp;
        }
    }
}

void shift(int *sh_beg, int *sh_end)
{
    for (int *sh_cur = sh_end; sh_cur > sh_beg; sh_cur--)
    {
        *sh_cur = *(sh_cur - 1);
    }
}
```

Программы должны компилироваться с ключом -D, определяющим константу ARRAY_LENGTH, которая отвечает за длину генерируемых массивов. При запуске программ можно добавить ключ s для генерации отсортированных массивов. На выход программы выводят массив чисел, равных продолжительности выполнения алгоритма сортировки в секундах.

build_apps.sh

```
#!/bin/bash

if [ $# -eq 1 ]; then
    value="$1"
else
    value="500"
fi

for name in *.c; do
    if ! gcc "$name" -o "${name%%.c}_O0.exe" -Wall -Werror -Wextra
-pedantic -O0 -DARRAY_LENGTH="$value" ; then
        exit 1
    fi
    if ! gcc "$name" -o "${name%%.c}_O2.exe" -Wall -Werror -Wextra
-pedantic -O2 -DARRAY_LENGTH="$value" ; then
        exit 1
    fi
done
```

Скрипт производит сборку 6 программ. Скрипту можно передать числовое значение, которое будет использоваться в качестве константы для определения длин массивов, по умолчанию константе задаётся значение 500.

update_data.sh

```
#!/bin/bash

arr_len=-1
is_sorted="shuffled"
file_name=""
opt=""
file_dir="${0//update_data.sh/}"

if [ $# -gt 0 ]; then
    if [ -f "$1" ] && [[ "$1" =~ .*\.c$ ]]; then
        file_name="$1"
        for arg in "$@"; do
            if [[ "$arg" =~ ^[0-9]+$ ]]; then
                arr_len=$arg
                ./"$file_dir"build_apps.sh "$arg"
            elif [[ "$arg" =~ ^-O[02]$ ]]; then
                opt="${arg#"-"}"
            elif [[ "$arg" == "-s" ]]; then
                is_sorted="sorted"
            fi
        done
    else
        echo No such .c file
    fi
else
    echo No arguments
fi

if [ "$file_name" != "" ] && [ ! "$arr_len" -eq -1 ] && [ "$opt"
!= "" ]; then
    file_name_with_opt="${file_name%".c"}"_"$opt"
    if [ -d "$file_dir$file_name_with_opt" ]; then
        if [ -d "$file_dir$file_name_with_opt/$is_sorted" ]; then
```

```
            if [ ! -d
"$file_dir$file_name_with_opt/$is_sorted/data" ]; then
                mkdir
"$file_dir$file_name_with_opt/$is_sorted/data"
            fi
        else
            mkdir "$file_dir$file_name_with_opt/$is_sorted"
            mkdir "$file_dir$file_name_with_opt/$is_sorted/data"
        fi
    else
        mkdir "$file_dir$file_name_with_opt"
        mkdir "$file_dir$file_name_with_opt/$is_sorted"
        mkdir "$file_dir$file_name_with_opt/$is_sorted/data"
    fi
    if [ "$is_sorted" == "sorted" ]; then
        ./"$file_name_with_opt.exe" -s >>
"$file_dir$file_name_with_opt/$is_sorted/data/$arr_len.txt"
    else
        ./"$file_name_with_opt.exe" >>
"$file_dir$file_name_with_opt/$is_sorted/data/$arr_len.txt"
    fi
else
    exit 1
fi
exit 0
```

Скрипт добавляет данные в датасет. На вход необходимо передать имя .c файла, количество чисел в массиве, уровень оптимизации. Также можно передать ключ -s, чтобы выполнить замер для отсортированного массива.

go.sh

```
#!/bin/bash

max_arr_size=10000
step_arr_size=500
file_dir="${0//go.sh/}"
./build_apps.sh
```

```
for name in *.exe; do
    arr_size=500
    no_ext_name="${name%%.exe}"
    if [ ! -d "$file_dir$no_ext_name" ]; then
        mkdir "$file_dir$no_ext_name"
    fi
    if [ ! -d "$file_dir$no_ext_name/shuffled" ]; then
        mkdir "$file_dir$no_ext_name/shuffled"
    fi
    if [ ! -d "$file_dir$no_ext_name/sorted" ]; then
        mkdir "$file_dir$no_ext_name/sorted"
    fi
    if [ "$1" == "-r" ]; then
        rm "$file_dir$no_ext_name/shuffled/data" -r
        rm "$file_dir$no_ext_name/sorted/data" -r
    fi
    if [ ! -d "$file_dir$no_ext_name/shuffled/data" ]; then
        mkdir "$file_dir$no_ext_name/shuffled/data"
    fi
    if [ ! -d "$file_dir$no_ext_name/sorted/data" ]; then
        mkdir "$file_dir$no_ext_name/sorted/data"
    fi
    while [ $arr_size -le $max_arr_size ]; do
        ./build_apps.sh "$arr_size"
        ./"$name" >>
"$file_dir$no_ext_name/shuffled/data/$arr_size.txt"
        ./"$name" "s" >>
"$file_dir$no_ext_name/sorted/data/$arr_size.txt"
        arr_size=$(( arr_size + step_arr_size ))
    done
done

python3 make_preproc.py
python3 make_postproc.py
```

Скрипт запускает замеры времени выполнения алгоритма для разных размеров массивов, сохраняет и запускает обработку данных. Скрипт может запускаться с ключом -r для очистки предыдущих замеров.

make_preproc.py

```
from statistics import mean, median, variance, stdev, quantiles
import os
import sys
from math import sqrt

cur_dir = sys.argv[0][:-15] + "./"

def preproc_data(input_file_dir, output_file_dir):
    time_data = []
    arr_length = int(input_file_dir[input_file_dir.rfind('/') +
1:-4])
    with open(input_file_dir, "r") as input, open(output_file_dir,
"w") as output,\

open(output_file_dir[:output_file_dir.rfind('preproc_data')] +
"stat.txt", "a") as stat:
        for time in input:
            time_data.append(float(time))
        mean_data = mean(time_data)
        stdev_data = stdev(time_data)
        output.write(f"Arr length={arr_length:d}\n")
        output.write(f"Mean={mean_data:.6g}\n")
        output.write(f"Median={median(time_data):.6g}\n")
        output.write(f"Variance={variance(time_data):.6g}\n")
        output.write(f"Sandard deviation={stdev_data:.6g}\n")
        output.write(f"Max time={max(time_data)}\n")
        output.write(f"Min time={min(time_data)}\n")
        output.write(f"Lower
quartile={quantiles(time_data)[0]:.6g}\n")
        output.write(f"Medium
quantile={quantiles(time_data)[1]:.6g}\n")
        output.write(f"Upper
quartile={quantiles(time_data)[2]:.6g}\n")
        if (arr_length % 1000 == 0):

stat.write(f"{arr_length:d}\t{mean_data:.2g}\t{stdev_data /
sqrt(arr_length) / mean_data * 100:.2g}%\n")

def sort_stat(stat_file_dir):
    arr_stat = []
```

```
    with open(stat_file_dir, "r+") as f:
        for line in f:
            split_1_i = line.find("\t")
            split_2_i = line.rfind("\t")
            try:
                arr_stat.append((int(line[:split_1_i]),
float(line[split_1_i + 1: split_2_i]), float(line[split_2_i:-2])))
            except TypeError:
                print("EXEPTION")
                return 1
    for i in range(len(arr_stat)):
        for j in range(len(arr_stat) - i - 1):
            if (arr_stat[j][0] > arr_stat[j + 1][0]):
                tmp = arr_stat[j]
                arr_stat[j] = arr_stat[j + 1]
                arr_stat[j + 1] = tmp
    with open(stat_file_dir, "w") as f:
        for elem in arr_stat:

f.write(f"{elem[0]:d}\t{elem[1]:.2g}\t{elem[2]:.2g}%\n")




for file in os.listdir(cur_dir):
    if (os.path.isdir(cur_dir + file)):
        sort_types = ("/shuffled", "/sorted")
        for sort_type in sort_types:
            if (os.path.isdir(cur_dir + file + sort_type)):
                if (not os.path.isdir(cur_dir + file + sort_type +
"/preproc_data")):
                    os.mkdir(cur_dir + file + sort_type +
"/preproc_data")
                if (os.path.isdir(cur_dir + file + sort_type +
"/data")):
                    f = open(cur_dir + file + sort_type +
"/stat.txt", "w")
                    f.close()
                    for data_file in os.listdir(cur_dir + file +
sort_type + "/data"):
                        if os.path.isfile(cur_dir + file +
sort_type + "/data/" + data_file) and data_file[-4:] == ".txt":
```

```
                                preproc_data(cur_dir + file +
sort_type + "/data/" + data_file,
                                             cur_dir + file + sort_type
+ "/preproc_data/preprocessed_" + data_file)
                        sort_stat(cur_dir + file + sort_type +
"/stat.txt")
```

Программа обрабатывает данные, полученные при замерах времени выполнения алгоритма.

make_postproc.py

```
import matplotlib.pyplot as plt
import matplotlib.markers
import os
import sys

cur_dir = sys.argv[0][:-16] + "./"

def get_preprocesed_data(preproc_file_path, field):
    with open(preproc_file_path, "r") as preproc_file:
        element = None
        for line in preproc_file:
            if (line.find(field) == 0):
                element = float(line[line.find("=") + 1:])
                break
        return element

def get_data(data_file_path):
    with open(data_file_path, "r") as preproc_file:
        elements = []
        for line in preproc_file:
                elements.append(float(line))
        return elements

def struct_preprocesed_data_in_points(dir_path, field):
    points = []
    for preproc_file in os.listdir(dir_path):
        preproc_file_path = dir_path + preproc_file
```

```
        points.append([get_preprocesed_data(preproc_file_path,
"Arr length"),
                       get_preprocesed_data(preproc_file_path,
field)])
    points.sort()
    x_vals = []
    y_vals = []
    for point in points:
        y_vals.append(point[1])
        x_vals.append(int(point[0]))
    return x_vals, y_vals

def struct_data_in_points(dir_path):
    points = []
    for data_file in os.listdir(dir_path):
        data_file_path = dir_path + data_file
        size = int(data_file[:-4])
        points.append([size, get_data(data_file_path)])
    points.sort()
    x_vals = []
    y_vals = []
    for point in points:
        y_vals.append(point[1])
        x_vals.append(int(point[0]))
    return x_vals, y_vals

def draw_line_plot(is_sorted):
    if (is_sorted):
        is_sorted_dir_name = "/sorted/"
    else:
        is_sorted_dir_name = "/shuffled/"
    markers = (".", "v", "^", "<", ">", "8", "s", "p", "h")
    for opt in ("O0", "O2"):
        labels = []
        plt.figure(figsize=(12, 10), dpi=80)
        marker_i = 0
        for file in os.listdir(cur_dir):
            if (os.path.isdir(cur_dir + file + is_sorted_dir_name)
and (opt in file)):
                x_vals, y_mean_vals =
struct_preprocesed_data_in_points(cur_dir + file +
is_sorted_dir_name + "preproc_data/", "Mean")
```

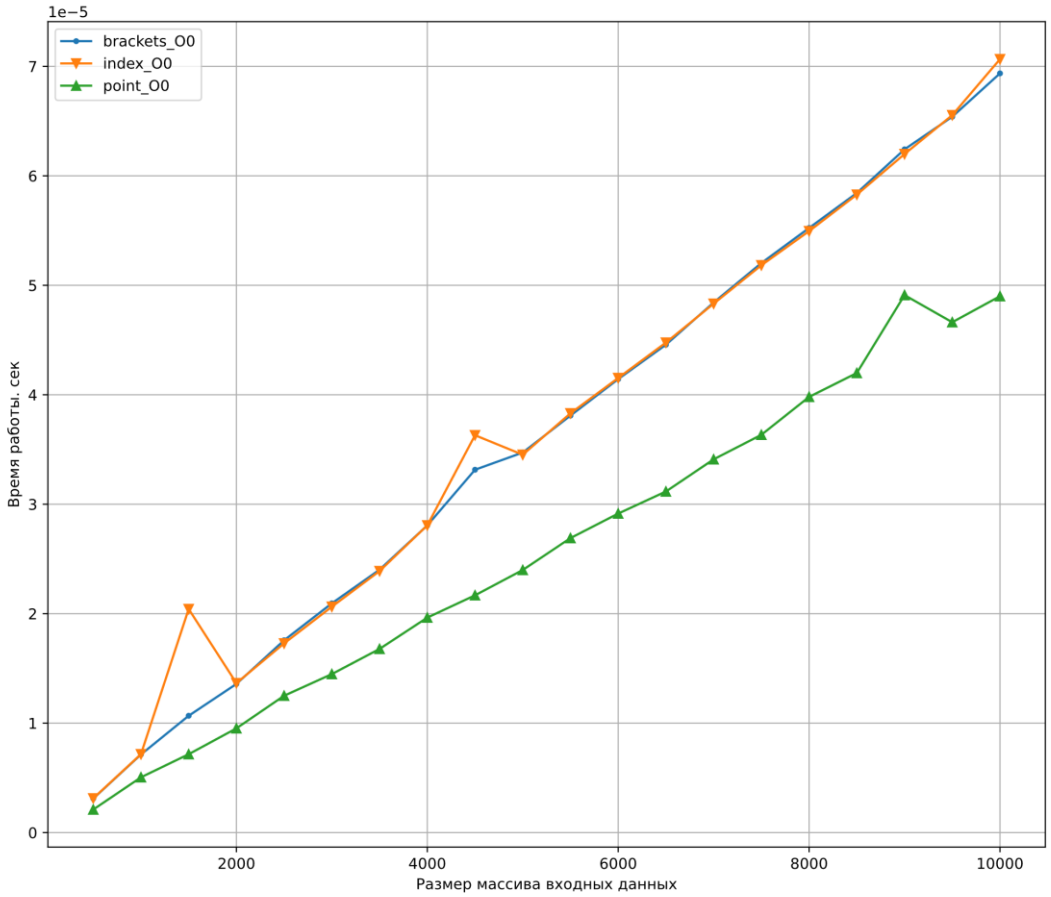
```
                plt.plot(x_vals, y_mean_vals, label=file,
marker=markers[marker_i])
                labels.append(file)
                marker_i += 1
        plt.ylabel("Время работы. сек")
        plt.xlabel("Размер массива входных данных")
        plt.legend(labels)
        plt.grid(True)
        plt.savefig(cur_dir+ is_sorted_dir_name[1:-1] + "_" + opt
+".svg")
        plt.close()

def draw_error_line_plot(opt):
    is_sorted_dir_name = "/shuffled/"
    for file in os.listdir(cur_dir):
        if (os.path.isdir(cur_dir + file + is_sorted_dir_name) and
(opt in file)):
            plt.figure(figsize=(12, 10), dpi=80)
            x_vals, y_mean_vals =
struct_preprocesed_data_in_points(cur_dir + file +
is_sorted_dir_name + "preproc_data/",
                                                        "Mean")
            _ ,y_max_vals =
struct_preprocesed_data_in_points(cur_dir + file +
is_sorted_dir_name + "preproc_data/",
                                                      "Max")
            _ ,y_min_vals =
struct_preprocesed_data_in_points(cur_dir + file +
is_sorted_dir_name + "preproc_data/",
                                                      "Min")
            size = len(x_vals)
            errors = []
            errors.append([y_mean_vals[i] - y_min_vals[i] for i in
range(size)])
            errors.append([y_max_vals[i] - y_mean_vals[i] for i in
range(size)])
            plt.errorbar(x_vals, y_mean_vals, yerr=errors, fmt=".-
", label=file)
            plt.ylabel("Время работы, сек")
            plt.xlabel("Число входных данных")
            plt.legend(labels=[file])
            plt.grid(True)
```

```
            plt.savefig(cur_dir + file + "_" +
is_sorted_dir_name[1:-1] + "_error_line_plot.svg")
            plt.close()


def draw_box_plot(opt):
    is_sorted_dir_name = "/shuffled/"
    for file in os.listdir(cur_dir):
        if (os.path.isdir(cur_dir + file + is_sorted_dir_name) and
(opt in file)):
            x_vals, y_vals = struct_data_in_points(cur_dir + file
+ is_sorted_dir_name + "data/")
            plt.figure(figsize=(12, 10), dpi=80)
            plt.boxplot(y_vals, whis=(0, 100), labels=x_vals,
showmeans=True,
                        meanline=True,
medianprops=dict(linewidth=0), meanprops=dict(linestyle='-'))
            plt.xticks(rotation="vertical")
            plt.ylabel("Время работы, сек")
            plt.xlabel("Число входных данных")
            plt.legend(labels=[file])
            plt.grid(True)
            plt.savefig(cur_dir + file + "_" +
is_sorted_dir_name[1:-1] + "_boxplot.svg")
            plt.close()

draw_line_plot(False)
draw_line_plot(True)
draw_error_line_plot("O2")
draw_box_plot("O2")
```

Программа преобразует обработанные данные в графики.

Результаты измерений

Для предварительно отсортированных массивов:



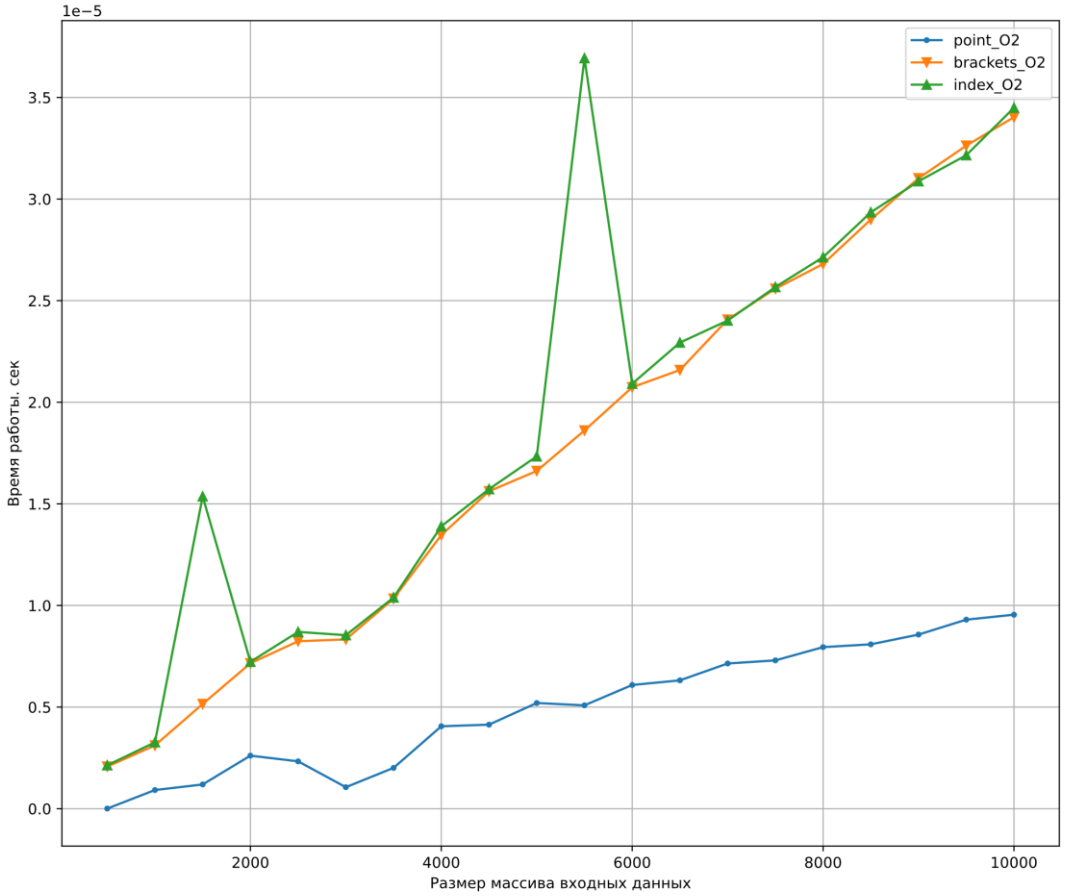
Использование операции индексации a[i] (brackets_O0)		
Длина массива	Время выполнения, с	Относительная стандартная ошибка среднего
1000	7.1e-06	0.41%
2000	1.4e-05	0.62%
3000	2.1e-05	0.34%
4000	2.8e-05	0.25%
5000	3.5e-05	0.25%
6000	4.1e-05	0.15%
7000	4.8e-05	0.17%
8000	5.5e-05	0.17%

9000	6.2e-05	0.16%
10000	6.9e-05	0.14%

Формальная замена операции индексации на выражение *(a + i) (index_O0)		
Длина массива	Время выполнения, с	Относительная стандартная ошибка среднего
1000	7.2e-06	0.74%
2000	1.4e-05	0.41%
3000	2.1e-05	0.27%
4000	2.8e-05	0.2%
5000	3.5e-05	0.33%
6000	4.2e-05	0.21%
7000	4.8e-05	0.19%
8000	5.5e-05	0.13%
9000	6.2e-05	0.2%
10000	7.1e-05	0.14%

Использование указателей для работы с массивом (point_O0)		
Длина массива	Время выполнения, с	Относительная стандартная ошибка среднего
1000	5.1e-06	0.14%
2000	9.5e-06	0.58%
3000	1.4e-05	0.33%
4000	2e-05	0.26%
5000	2.4e-05	0.19%
6000	2.9e-05	0.15%

7000	3.4e-05	0.17%
8000	4e-05	0.23%
9000	4.9e-05	2.1%
10000	4.9e-05	0.16%



Использование операции индексации a[i] (brackets_O2)		
Длина массива	Время выполнения, с	Относительная стандартная ошибка среднего
1000	3.1e-06	1.3%
2000	7.2e-06	0.34%
3000	8.3e-06	0.38%
4000	1.3e-05	0.29%

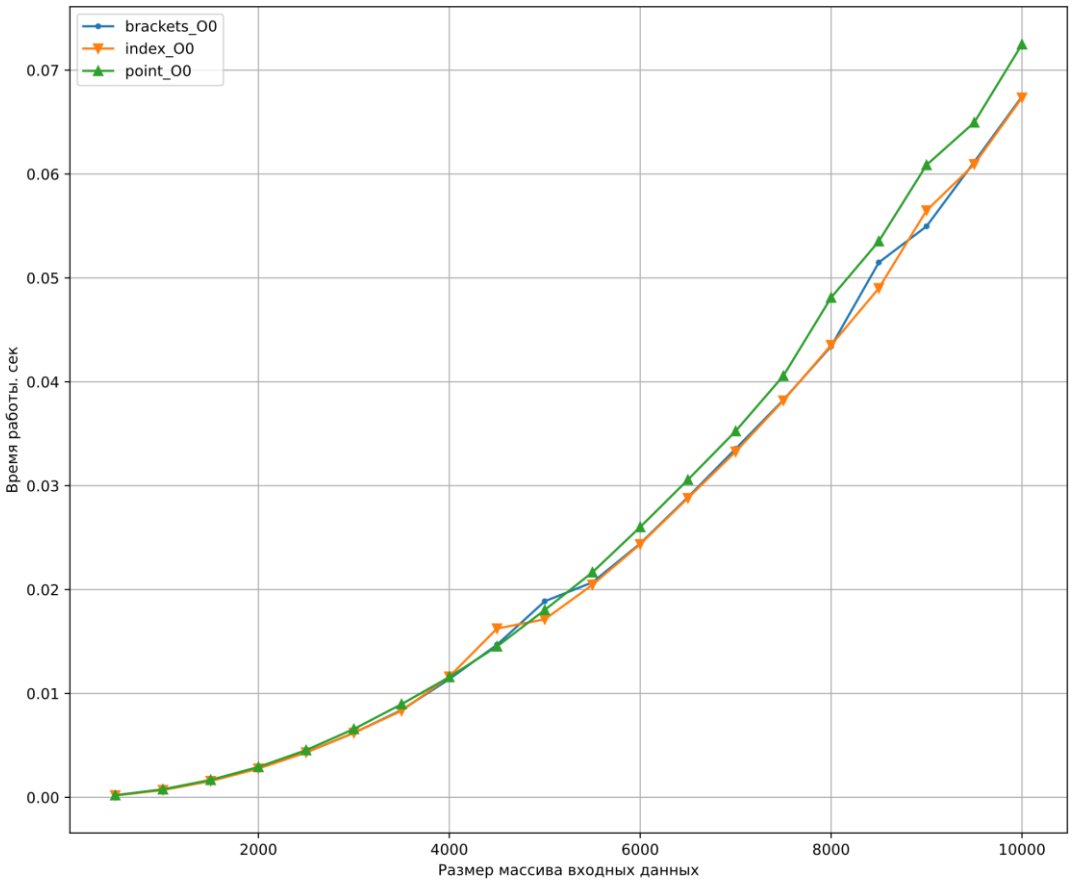
5000	1.7e-05	0.3%
6000	2.1e-05	0.24%
7000	2.4e-05	0.16%
8000	2.7e-05	0.22%
9000	3.1e-05	0.16%
10000	3.4e-05	0.081%

Формальная замена операции индексации на выражение *(a + i) (index_O2)		
Длина массива	Время выполнения, с	Относительная стандартная ошибка среднего
1000	3.3e-06	1.7%
2000	7.2e-06	0.52%
3000	8.5e-06	0.53%
4000	1.4e-05	0.33%
5000	1.7e-05	0.29%
6000	2.1e-05	0.18%
7000	2.4e-05	0.18%
8000	2.7e-05	0.17%
9000	3.1e-05	0.18%
10000	3.4e-05	0.13%

Использование указателей для работы с массивом (point_O2)		
Длина массива	Время выполнения, с	Относительная стандартная ошибка среднего
1000	9.2e-07	1.8%
2000	2.6e-06	9.9%

3000	1.1e-06	0.45%
4000	4.1e-06	0.28%
5000	5.2e-06	0.49%
6000	6.1e-06	0.25%
7000	7.1e-06	0.24%
8000	8e-06	0.17%
9000	8.6e-06	0.25%
10000	9.5e-06	0.28%

Для случайно сгенерированных массивов:

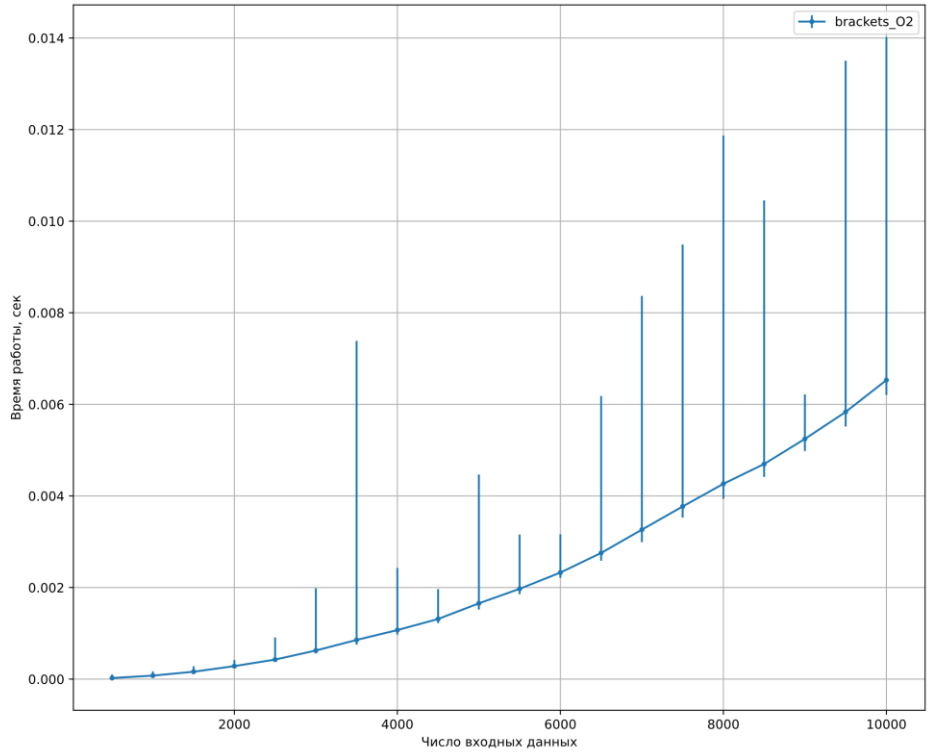


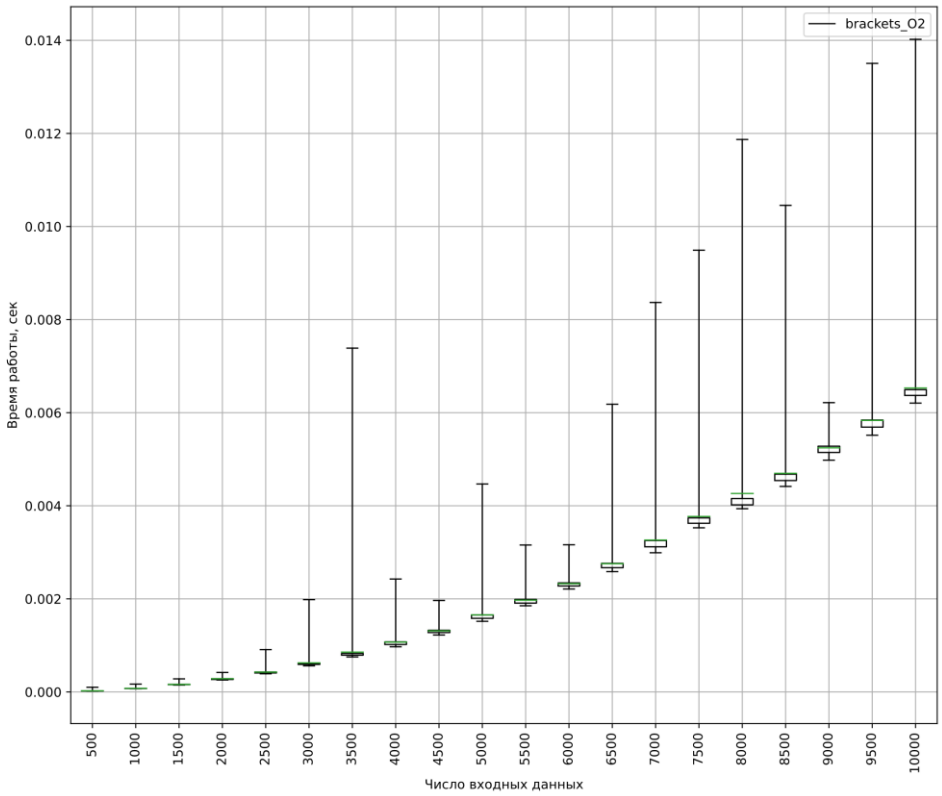
Использование операции индексации a[i] (brackets_O0)

Длина массива	Время выполнения, с	Относительная стандартная ошибка среднего
1000	0.00074	0.72%
2000	0.0028	0.36%
3000	0.0062	0.25%
4000	0.011	0.13%
5000	0.019	0.26%
6000	0.024	0.038%
7000	0.034	0.046%
8000	0.043	0.027%
9000	0.055	0.033%
10000	0.067	0.024%

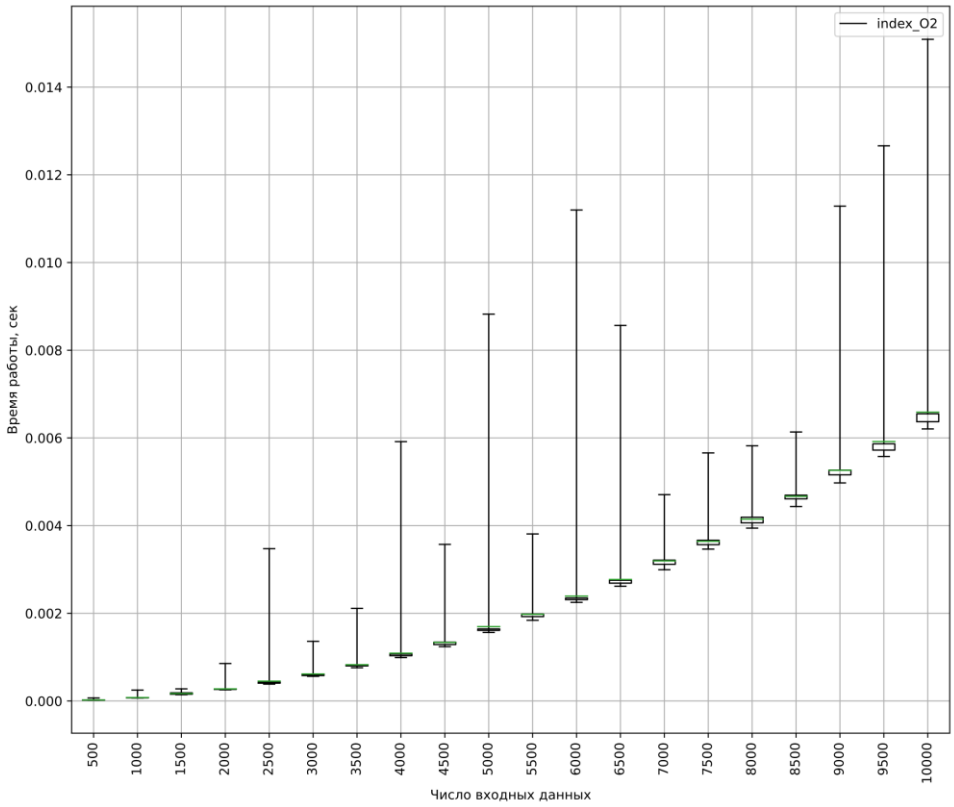
Формальная замена операции индексации на выражение *(a + i) (index_O0)		
Длина массива	Время выполнения, с	Относительная стандартная ошибка среднего
1000	0.0007	0.22%
2000	0.0028	0.099%
3000	0.0062	0.18%
4000	0.012	0.25%
5000	0.017	0.092%
6000	0.024	0.031%
7000	0.033	0.045%
8000	0.044	0.037%
9000	0.056	0.097%
10000	0.067	0.022%

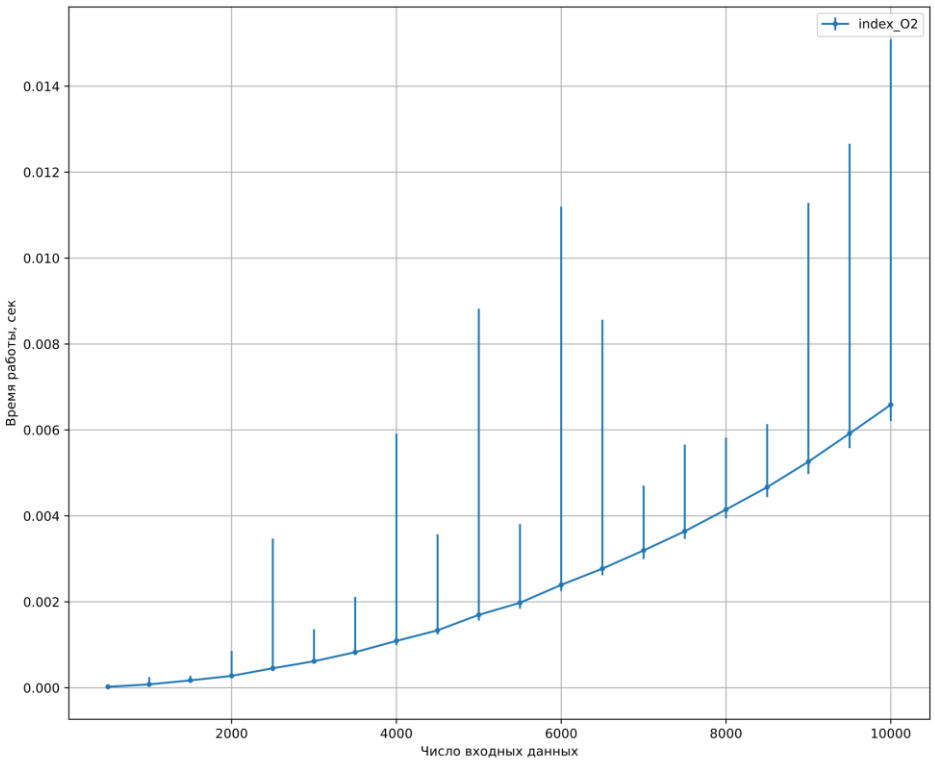
Использование указателей для работы с массивом (point_O0)		
Длина массива	Время выполнения, с	Относительная стандартная ошибка среднего
1000	0.00077	1.9%
2000	0.0029	0.27%
3000	0.0066	0.14%
4000	0.012	0.085%
5000	0.018	0.061%
6000	0.026	0.04%
7000	0.035	0.028%
8000	0.048	0.12%
9000	0.061	0.09%
10000	0.072	0.038%


brackets_O2
0.014
0.012
0.010
0.008
0.006
0.004
0.002
0.000
Время работы, сек
2000
4000
6000
8000
10000
Число входных данных

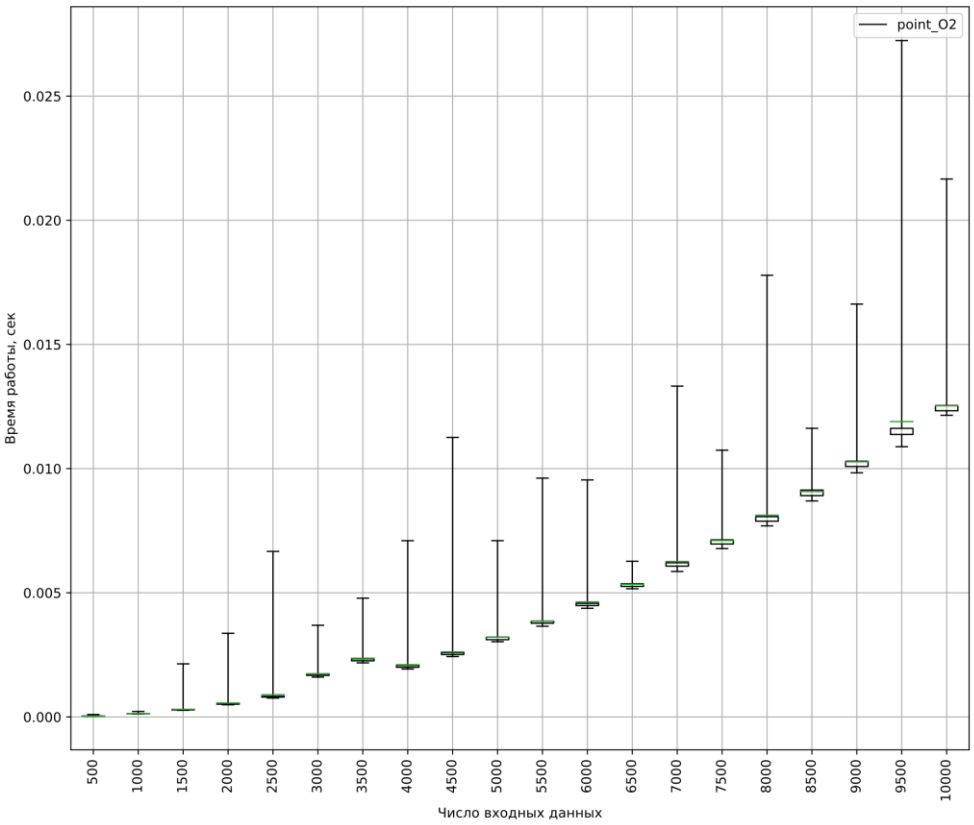

brackets_O2
0.014
0.012
0.010
0.008
0.006
0.004
0.002
0.000
Время работы, сек
500
1000
1500
2000
2500
3000
3500
4000
4500
5000
5500
6000
6500
7000
7500
8000
8500
9000
9500
10000
Число входных данных

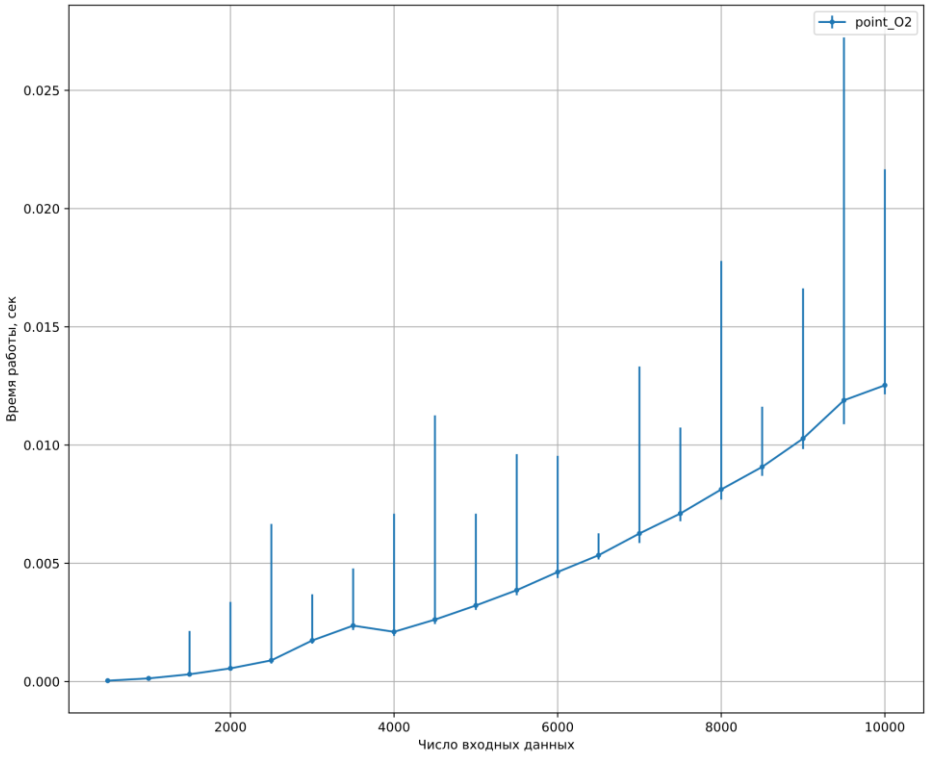
Использование операции индексации a[i] (brackets_O2)		
Длина массива	Время выполнения, с	Относительная стандартная ошибка среднего
1000	7.5e-05	0.41%
2000	0.00028	0.2%
3000	0.0006	0.26%
4000	0.001	0.14%
5000	0.0016	0.17%
6000	0.0023	0.054%
7000	0.0032	0.14%
8000	0.0041	0.19%
9000	0.0051	0.036%
10000	0.0064	0.087%


index_O2
0.014
0.012
0.010
0.008
0.006
0.004
0.002
0.000
Время работы, сек
500
1000
1500
2000
2500
3000
3500
4000
4500
5000
5500
6000
6500
7000
7500
8000
8500
9000
9500
10000
Число входных данных

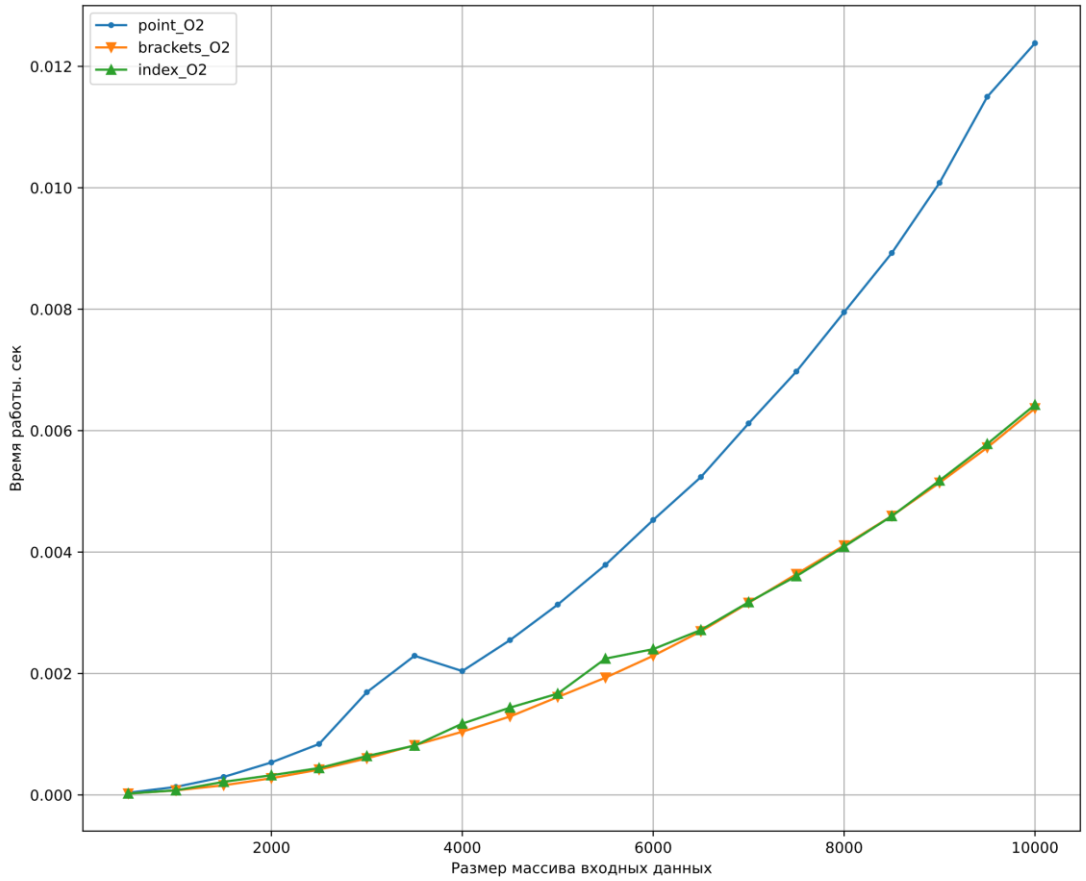

index_O2
0.014
0.012
0.010
0.008
0.006
0.004
0.002
0.000
Время работы, сек
2000
4000
6000
8000
10000
Число входных данных

Формальная замена операции индексации на выражение *(a + i) (index_O2)		
Длина массива	Время выполнения, с	Относительная стандартная ошибка среднего
1000	7.6e-05	0.66%
2000	0.00032	3.5%
3000	0.00064	0.61%
4000	0.0012	1%
5000	0.0017	0.37%
6000	0.0024	0.26%
7000	0.0032	0.075%
8000	0.0041	0.059%
9000	0.0052	0.071%
10000	0.0064	0.085%


point_O2
0.025
0.020
0.015
0.010
0.005
0.000
Время работы, сек
500
1000
1500
2000
2500
3000
3500
4000
4500
5000
5500
6000
6500
7000
7500
8000
8500
9000
9500
10000
Число входных данных


point_O2
0.025
0.020
0.015
0.010
0.005
0.000
Время работы, сек
2000
4000
6000
8000
10000
Число входных данных

Использование указателей для работы с массивом (point_O2)		
Длина массива	Время выполнения, с	Относительная стандартная ошибка среднего
1000	0.00013	0.28%
2000	0.00054	0.65%
3000	0.0017	0.27%
4000	0.002	0.27%
5000	0.0031	0.12%
6000	0.0045	0.13%
7000	0.0061	0.095%
8000	0.008	0.1%
9000	0.01	0.05%
10000	0.012	0.054%



Использование операции индексации a[i] (brackets_O2)		
Длина массива	Время выполнения, с	$\frac{\ln(t_{i+1}) - \ln(t_i)}{\ln(n_{i+1}) - \ln(n_i)}$
1000	7.5e-05	1.90
2000	0.00028	1.88
3000	0.0006	1.78
4000	0.001	2.11
5000	0.0016	1.99
6000	0.0023	2.14
7000	0.0032	1.86
8000	0.0041	1.85
9000	0.0051	2.16

Вывод:

Для отсортированных массивов (идеальный случай) самым быстрым способом обработки является способ через указатели.

А для случайно сгенерированных массивов самыми быстрыми способами обработки являются: обработка через оператор индексации и обработка через формальную замену оператора индексации.

Заменять серию экспериментов с одинаковым результатом на один нельзя, так как это может привести к искажению среднего арифметического значения, дисперсии, стандартного отклонения и так далее. Как итог, такие изменения могут привести к некорректным выводам об эффективности алгоритма.

Целью проведённых исследований было сравнение производительности работы алгоритма в различных условия, поэтому время заполнения массивов случайными значениями не учитывается.