Задание 4

По зданию используем три способа работы с элементами одномерного массива.

- Доступ по индексу. В нашем варианте данные хранятся под именем idx

index_sort.c

```c
#include "sort.h"

void insertion_sort(int *a, size_t size)
{
    int j;
    for (size_t i = 1; i < size; i++)
    {
        int el = a[i];
        j = i - 1;
        while (j >= 0 && a[j] > el)
        {
            a[j + 1] = a[j];
            j--;
        }
        a[j + 1] = el;
    }
}
```

- Доступ по формальной замене операции индексации. Под именем fi(formal_index)

formal_index_sort.c

```c
#include "sort.h"

void insertion_sort(int *a, size_t size)
{
    int j;
    for (size_t i = 1; i < size; i++)
    {
        int el = *(a+i);
        j = i - 1;
        while (j >= 0 && *(a + j) > el)
        {
            *(a + j + 1) = *(a + j);
            j--;
        }
        *(a + j + 1) = el;
    }
}
```

- Использование указателей для работы с массивом. Под именем ptr

pointer_sort.c

```c
#include "sort.h"

void insertion_sort(int *a, size_t size)
{
    int *j;
    for (int *i = a + 1; i < a + size; i++)
    {
        int el = *i;
        j = i - 1;
        while (j >= a && *j > el)
        {
            *(j + 1) = *j;
            j--;
        }
        *(j + 1) = el;
    }
}
```

В задании сказано, что измерения проводятся с внешней и внутренней архитектурами. Для выполнения были написаны 2 программы.

Первая производит подсчёт rse внутри цикла и выводит результаты измерения в stdout, пока величина rse > 1.

main1.c

```c
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#include <string.h>

#include "statistic.h"
#include "errors.h"
#include "sort.h"

#define MIN_ITER 20
#define MAX_ITER 10000
#define REPEATS 30
#ifndef NMAX
#error Not defined array size NMAX
#endif

void fill_rand_arr(int *arr, size_t size);
unsigned long long calc_elapsed_time(const struct timespec *beg, const struct timespec *end);
```

```c
int main()
{
    int arr[NMAX];
    int src[NMAX];
    unsigned long long times[MAX_ITER];
    srand(time(NULL));
    fill_rand_arr(src, NMAX);

    struct timespec beg = {0, 0};
    struct timespec end = {0, 0};
    unsigned long long time = 0;
    int n = 0;
    size_t size = 0;
    double average_v = 1;
    double dispersion_v = 1;
    double standart_deviation_v = 1;
    double standart_error_v = 1;
    double rse_value = rse(standart_error_v, average_v);
    // "Разогрев"
    for (size_t i = 0; i < REPEATS; i++)
    {
        memcpy(arr, src, sizeof(src));
        clock_gettime(CLOCK_MONOTONIC_RAW, &beg);
        insertion_sort(arr, size);
        clock_gettime(CLOCK_MONOTONIC_RAW, &end);
    }

    for (size_t i = 0; (i < MAX_ITER && rse_value > 1) || i <= MIN_ITER;i++)
    {
        memcpy(arr, src, sizeof(src));
        clock_gettime(CLOCK_MONOTONIC_RAW, &beg);
        insertion_sort(arr, NMAX);
        clock_gettime(CLOCK_MONOTONIC_RAW, &end);
        arr[0] = arr[1];
        arr[1] = 1234;
        time = calc_elapsed_time(&beg, &end);
        times[i] = time;
        size++;
        printf("%llu\n", time);
        average(times, size, &average_v, &n);
        dispersion(times, size, &average_v, &dispersion_v, &n);
        standart_deviation(&dispersion_v, &standart_deviation_v);
        standart_err(&standart_deviation_v, n, &standart_error_v);
        rse_value = rse(standart_error_v, average_v);
    }

    return ERR_OK;
}

// вычисляет разность времени в микросекундах
```

```
unsigned long long calc_elapsed_time(const struct timespec *beg, const struct
timespec *end)
{
    return ((unsigned long long)
        (end->tv_sec - beg->tv_sec) * 1000 * 1000 * 1000 +
                        (end->tv_nsec - beg->tv_nsec)) / 1000;
}

void fill_rand_arr(int *arr, size_t size)
{
    for (size_t i = 0; i < size; i++)
        arr[i] = rand();
}
```

Во втором варианте в теле программы выводится лишь 1 измерение.

main2.c

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

#include "errors.h"
#include "sort.h"

#ifndef NMAX
#error Not defined array size NMAX
#endif

#define REPEATS 10

void fill_rand_arr(int *arr, size_t size);
unsigned long long calc_elapsed_time(const struct timespec *beg, const struct
timespec *end);

int main(void)
{
    int a[NMAX];
    int src[NMAX];
    size_t size = sizeof(a) / sizeof(a[0]);
    unsigned long long time_tmp = 0;

    struct timespec beg = {0, 0};
    struct timespec end = {0, 0};

    srand(time(NULL));
    fill_rand_arr(src, NMAX);

    // "Разогрев"
```

```
    for (size_t i = 0; i < REPEATS; i++)
    {
        memcpy(a, src, sizeof(src));
        clock_gettime(CLOCK_MONOTONIC_RAW, &beg);
        insertion_sort(a, size);
        clock_gettime(CLOCK_MONOTONIC_RAW, &end);
    }
    memcpy(a, src, sizeof(src));
    clock_gettime(CLOCK_MONOTONIC_RAW, &beg);
    insertion_sort(a, size);
    clock_gettime(CLOCK_MONOTONIC_RAW, &end);
    time_tmp = calc_elapsed_time(&beg, &end);


    printf("%llu\n", time_tmp);
    return ERR_OK;
}


unsigned long long calc_elapsed_time(const struct timespec *beg, const struct
timespec *end)
{
    return ((unsigned long long)
        (end->tv_sec - beg->tv_sec) * 1000 * 1000 * 1000 +
                        (end->tv_nsec - beg->tv_nsec)) / 1000;
}

void fill_rand_arr(int *arr, size_t size)
{
    for (size_t i = 0; i < size; i++)
        arr[i] = rand();
}
```

Скрипт build_apps.sh запускает сборку каждой вариации программы для разных размерностей массива.

build_apps.sh

```
min_NMAX=100
max_NMAX=10000
step_NMAX=500

versions="1 2"
sizes=$(seq 0 $step_NMAX $max_NMAX)
sizes="$min_NMAX ${sizes:1}"

rm -f ./apps/*.exe || exit

if ! [ -d ./apps ]; then
    mkdir apps
fi
```

```
echo start compilation

for ver in $versions; do
    for size in $sizes; do
        if [ "$ver" == "1" ]; then
            gcc -std=gnu99 -Wall -Werror -Wpedantic -DNMAX="${size}"\
            -Wextra -Wfloat-conversion -Wfloat-equal -o0\
               -Wvla  -o  ./apps/app"${ver}"_"${size}"_fi.exe  ./src/main"${ver}".c
./src/formal_index_sort.c ./src/statistic.c -lm

            gcc -std=gnu99 -Wall -Werror -Wpedantic -DNMAX="${size}"\
            -Wextra -Wfloat-conversion -Wfloat-equal -o0\
              -Wvla  -o  ./apps/app"${ver}"_"${size}"_ptr.exe  ./src/main"${ver}".c
./src/pointer_sort.c ./src/statistic.c -lm

            gcc -std=gnu99 -Wall -Werror -Wpedantic -DNMAX="${size}"\
            -Wextra -Wfloat-conversion -Wfloat-equal -o0\
              -Wvla  -o  ./apps/app"${ver}"_"${size}"_idx.exe  ./src/main"${ver}".c
./src/index_sort.c ./src/statistic.c -lm
        fi
        if [ "$ver" == "2" ]; then
            gcc -std=gnu99 -Wall -Werror -Wpedantic -DNMAX="${size}"\
            -Wextra -Wfloat-conversion -Wfloat-equal -o0\
               -Wvla  -o  ./apps/app"${ver}"_"${size}"_fi.exe  ./src/main"${ver}".c
./src/formal_index_sort.c

            gcc -std=gnu99 -Wall -Werror -Wpedantic -DNMAX="${size}"\
            -Wextra -Wfloat-conversion -Wfloat-equal -o0\
              -Wvla  -o  ./apps/app"${ver}"_"${size}"_ptr.exe  ./src/main"${ver}".c
./src/pointer_sort.c

            gcc -std=gnu99 -Wall -Werror -Wpedantic -DNMAX="${size}"\
            -Wextra -Wfloat-conversion -Wfloat-equal -o0\
              -Wvla  -o  ./apps/app"${ver}"_"${size}"_idx.exe  ./src/main"${ver}".c
./src/index_sort.c
        fi
    done
done
echo succefully compilated
```

Скрипт update_data.sh был разделен на 2 скрипта. Первый update_data_inmeasure.sh запускает только версии программы, которые считают rse внутри. А скрипт update_data_outmeasure.sh запускает только версии программы, которые считают rse снаружи. Скрипт update_data.sh запускает выше приведенные скрипты.

update_data_inmeasure.sh

```
min_NMAX=100
```

```
max_NMAX=10000
step_NMAX=500

versions="1 2"
sizes=$(seq 0 $step_NMAX $max_NMAX)
sizes="$min_NMAX ${sizes:1}"
count=20
if ! [ -d ./data ]; then
    mkdir data
fi

# main1.c
for size in $sizes;do
    if [ ! -e ./apps/app1_"${size}"_fi.exe ]; then
        gcc -std=c99 -Wall -Werror -Wpedantic -DNMAX="${size}"\
        -Wextra -Wfloat-conversion -Wfloat-equal -o0 \
                    -Wvla    -o    ./apps/app1_"${size}"_fi.exe    ./src/main1.c
./src/formal_index_sort.c
    fi

    if [ ! -e ./apps/app1_"${size}"_ptr.exe ]; then
        gcc -std=c99 -Wall -Werror -Wpedantic -DNMAX="${size}"\
        -Wextra -Wfloat-conversion -Wfloat-equal -o0 \
        -Wvla -o ./apps/app1_"${size}"_ptr.exe ./src/main1.c ./src/pointer_sort.c
    fi

    if [ ! -e ./apps/app1_"${size}"_idx.exe ]; then
        gcc -std=c99 -Wall -Werror -Wpedantic -DNMAX="${size}"\
        -Wextra -Wfloat-conversion -Wfloat-equal -o0 \
        -Wvla -o ./apps/app1_"${size}"_idx.exe ./src/main1.c ./src/index_sort.c
    fi

    if [ ! -e ./data/data1_"${size}"_fi.txt ];then
        touch ./data/data1_"${size}"_fi.txt || exit
    fi
        ./apps/app1_"${size}"_fi.exe >> ./data/data1_"${size}"_fi.txt || exit

    if [ ! -e ./data/data1_"${size}"_idx.txt ];then
        touch ./data/data1_"${size}"_idx.txt || exit
    fi
    ./apps/app1_"${size}"_idx.exe >> ./data/data1_"${size}"_idx.txt || exit

    if [ ! -e ./data/data1_"${size}"_ptr.txt ]; then
        touch ./data/data1_"${size}"_ptr.txt || exit
    fi
    ./apps/app1_"${size}"_ptr.exe >> ./data/data1_"${size}"_ptr.txt || exit

done
```

update_data_outmeasure.sh

```
min_NMAX=100
max_NMAX=10000
step_NMAX=500

versions="1 2"
sizes=$(seq 0 $step_NMAX $max_NMAX)
sizes="$min_NMAX ${sizes:1}"
count=20
if ! [ -d ./data ]; then
    mkdir data
fi

# main2.c
for size in $sizes;do
    if [ ! -e ./apps/app2_"${size}"_fi.exe ]; then
        gcc -std=c99 -Wall -Werror -Wpedantic -DNMAX="${size}"\
        -Wextra -Wfloat-conversion -Wfloat-equal -o0 \
                    -Wvla    -o    ./apps/app2_"${size}"_fi.exe    ./src/main2.c
./src/formal_index_sort.c
    fi

    if [ ! -e ./apps/app2_"${size}"_ptr.exe ]; then
        gcc -std=c99 -Wall -Werror -Wpedantic -DNMAX="${size}"\
        -Wextra -Wfloat-conversion -Wfloat-equal -o0 \
        -Wvla -o ./apps/app2_"${size}"_ptr.exe ./src/main2.c ./src/pointer_sort.c
    fi

    if [ ! -e ./apps/app2_"${size}"_idx.exe ]; then
        gcc -std=c99 -Wall -Werror -Wpedantic -DNMAX="${size}"\
        -Wextra -Wfloat-conversion -Wfloat-equal -o0 \
        -Wvla -o ./apps/app2_"${size}"_idx.exe ./src/main2.c ./src/index_sort.c
    fi

    if [ ! -e ./data/data2_"${size}"_fi.txt ];then
        touch ./data/data2_"${size}"_fi.txt
    fi
    while python3 ./scripts/rse.py ./data/data2_"${size}"_fi.txt;do
        ./apps/app1_"${size}"_fi.exe >> ./data/data2_"${size}"_fi.txt || exit
    Done

    if [ ! -e ./data/data2_"${size}"_idx.txt ];then
        touch ./data/data2_"${size}"_idx.txt
    fi
    while python3 ./scripts/rse.py ./data/data2_"${size}"_idx.txt;do
        ./apps/app1_"${size}"_idx.exe >> ./data/data2_"${size}"_idx.txt || exit
    done


    if [ ! -e ./data/data2_"${size}"_ptr.txt ]; then
```

```
        touch ./data/data2_"${size}"_ptr.txt
    fi
    while python3 ./scripts/rse.py ./data/data2_"${size}"_ptr.txt ;do
        ./apps/app1_"${size}"_ptr.exe >> ./data/data2_"${size}"_ptr.txt || exit
    done
done
```

Для наружного подсчета был написан скрипт rse.py

rse.py

```
import sys
import math
import statistics

def statistic(data) -> float:
    """Возвращает статистику по данным"""
    mean = statistics.mean(data)

    dispersion = statistics.variance(data)
    std_deviation = math.sqrt(dispersion)
    std_err = std_deviation / math.sqrt(len(data))

    return std_err / math.sqrt(len(data)) * 100
filename = sys.argv[1]
try:
    with open(filename, 'r') as file:
        data = list(map(float, file.readlines()))
except FileNotFoundError:
    exit(1)


if len(data) <= 1:
    exit(0)
if len(data) >= 250:
    exit(1)
if statistic(data) < 1 and len(data) >= 20:
    exit(1)
else:
    exit(0)
```

Скрипт make_preproc.py собирает данные из файлов, которые были созданы при выполнении. Он считывает статистику и создаёт .json файл в котором хранит нужную информацию.

make_preproc.py

```
import json
import os
import statistics
```

```
import math
os.chdir('./data')
files = os.listdir('./')
preproc_data = {'data1':{'idx':{}, 'ptr':{}, 'fi':{}}, 'data2':{'idx':{}, 'ptr':{},
'fi':{}}}

def statistic(data) -> tuple:
    """Возвращает статистику по данным"""
    quantiles = statistics.quantiles(data)
    mean = statistics.mean(data)
    maxim = max(data)
    minim = min(data)
    median = statistics.median(data)
    dispersion = statistics.variance(data)
    std_deviation = math.sqrt(dispersion)
    std_err = std_deviation / math.sqrt(len(data))
     return (mean, dispersion, std_deviation, std_err, maxim, minim, quantiles,
len(data), median, data)


for filename in files:
    file = filename.replace('.txt', '')
    name, nmax, sort_type = file.split('_')
    with open(filename, 'r') as file:
        data = list(map(float, file.readlines()))
        preproc_data[name][sort_type][nmax] = statistic(data)


os.chdir('./../')
with open("./prepdata/data_file.json", "w") as write_file:
    json.dump(preproc_data, write_file, indent=4)
```

Скрипт make_postproc.py выполняет отрисовку графиков на основании .json файла

make_postproc.py

```
import matplotlib.pyplot as plt
import json
import os
with open('./prepdata/data_file.json', 'r') as json_file:
    data = json.load(json_file)
os.chdir("./postprocdata")
graph_type = ('o-', '^-', 's-')

line_width = 0.2
marker_size = 0.5
box_plot_width = 0.5
# Линейный график
for data_type, data_type_value in data.items():
    plt.clf()
```

```
    plt.xlabel("Размерность, кол-во элементов")
    plt.ylabel("Время, наносекунды")
    if data_type == "data1":
        name_type = "in_measure"
    else:
        name_type = "out_measure"
    plt.title(f"{name_type}_linear")
    plt.grid()
    i = 0
    for access_type, access_type_value in data_type_value.items():
        if access_type == "idx":
            access_name = "indexation"
        elif access_type == "ptr":
            access_name = "pointer_access"
        else:
            access_name = "formal_indexation"
        x = []
        y = []
        for key in sorted([int(x) for x in access_type_value.keys()]):
            x.append(key)
            y.append(access_type_value[str(key)][0])

       plt.plot(x, y, graph_type[i], linewidth=line_width, markersize=marker_size,
label=f'{access_name}')
        i += 1
    plt.xticks([i for i in range(0, max(x)+1000, 1000)], rotation=45)
    plt.legend()
    plt.savefig(f"{name_type}_linear.svg", bbox_inches='tight')

# Линейный график с ошибкой
for data_type, data_type_value in data.items():
    plt.clf()
    if data_type == "data1":
        name_type = "in_measure"
    else:
        name_type = "out_measure"


    plt.grid()
    for access_type, access_type_value in data_type_value.items():
        plt.clf()
        if access_type == "idx":
            access_name = "indexation"
        elif access_type == "ptr":
            access_name = "pointer_access"
        else:
            access_name = "formal_indexation"

        plt.xlabel("Размерность, кол-во элементов")
        plt.ylabel("Время, наносекунды")
        x = []
```

```
        y_av = []
        y_min = []
        y_max = []
        y_av_err = []
        for key in sorted([int(x) for x in access_type_value.keys()]):
            x.append(key)
            y_av.append(access_type_value[str(key)][0])
            y_min.append(access_type_value[str(key)][5])
            y_max.append(access_type_value[str(key)][4])
            y_av_err.append(access_type_value[str(key)][3])
        plt.grid()
                    plt.plot(x,    y_av,    graph_type[0],    linewidth=line_width,
markersize=marker_size, label=f'{access_name} average')
                   plt.plot(x,    y_min,    graph_type[1],    linewidth=line_width,
markersize=marker_size, label=f'{access_name} min')
                   plt.plot(x,    y_max,    graph_type[2],    linewidth=line_width,
markersize=marker_size, label=f'{access_name} max')
        plt.title(f"{name_type}_{access_name}_err")
        for i in range(len(x)):
            plt.errorbar(x[i], y_av[i], y_av_err[i], color='red')
        plt.legend()
        plt.savefig(f"err_{name_type}_{access_name}.svg", bbox_inches='tight')
    plt.xticks([i for i in range(0, max(x)+1000, 1000)], rotation=45)
# График с усами
for data_type, data_type_value in data.items():
    plt.clf()

    plt.xlabel("Размерность, кол-во элементов")
    plt.ylabel("Время, наносекунды")
    if data_type == "data1":
        name_type = "in_measure"
    else:
        name_type = "out_measure"
    access_name = "Indexation"
    plt.title(f"moustache_{name_type}_{access_name}")
    access_type_value = data_type_value['idx']
    x = []
    data = []
    medians = []
    y_av = []
    for key in sorted([int(x) for x in access_type_value.keys()]):
        x.append(key)
        data.append(access_type_value[str(key)][9])
        y_av.append(access_type_value[str(key)][0])

    plt.grid()
    plt.xticks(rotation=45)
    plt.boxplot(data, labels=x, showfliers=False, widths=box_plot_width)

    plt.savefig(f"moustache_{name_type}_{access_name}.svg", bbox_inches='tight')
```

Полученные результаты

Измерение внутри программы:

Index_sort

Размер	Время среднее, нс	Кол-во повторов	RSE
100	4	72	0,7
500	68,61	1368	0,73
1000	267,26	922	0,71
1500	555,79	42	0,84
2000	967,30	238	0,80
2500	1570,37	270	0,72
3000	2204,66	376	0,76
3500	2292,76	220	0,78
4000	4032,21	148	0,73
4500	4978,02	82	0,77

Fi_sort

Размер	Время среднее, нс	Кол-во повторов	RSE
100	2,03	180	0,72
500	61,64	42	0,15
1000	256,77	693	0,71
1500	559,57	322	0,75
2000	981,28	376	0,72
2500	1544,11	358	0,75
3000	2281,7	411	0,74
3500	2944,21	90	0,78
4000	3874,54	68	0,71
4500	5100,18	134	0,75

Pointer_sort

Размер	Время среднее, нс	Кол-во повторов	RSE
100	2,45	696	0,92
500	58,69	907	0,98
1000	202	1185	0,73
1500	411,64	133	0,84
2000	808,6	901	0,98
2500	1149,48	388	0,95
3000	1576,44	45	0,52
3500	2280,05	359	0,95
4000	2880,3	128	0,84
4500	3601,02	42	0,25

Измерение снаружи программы:

Index_sort

Размер	Время среднее, нс	Кол-во повторов	RSE
100	4,30	125	0,99
500	65,24	494	0,99
1000	255,82	370	1
1500	569,9	404	0,95
2000	1002,36	584	0,81
2500	1561,76	303	0,68
3000	2448,86	541	0,77
3500	3080,28	326	0,58
4000	4148,29	523	0,73
4500	4941,57	261	0,59

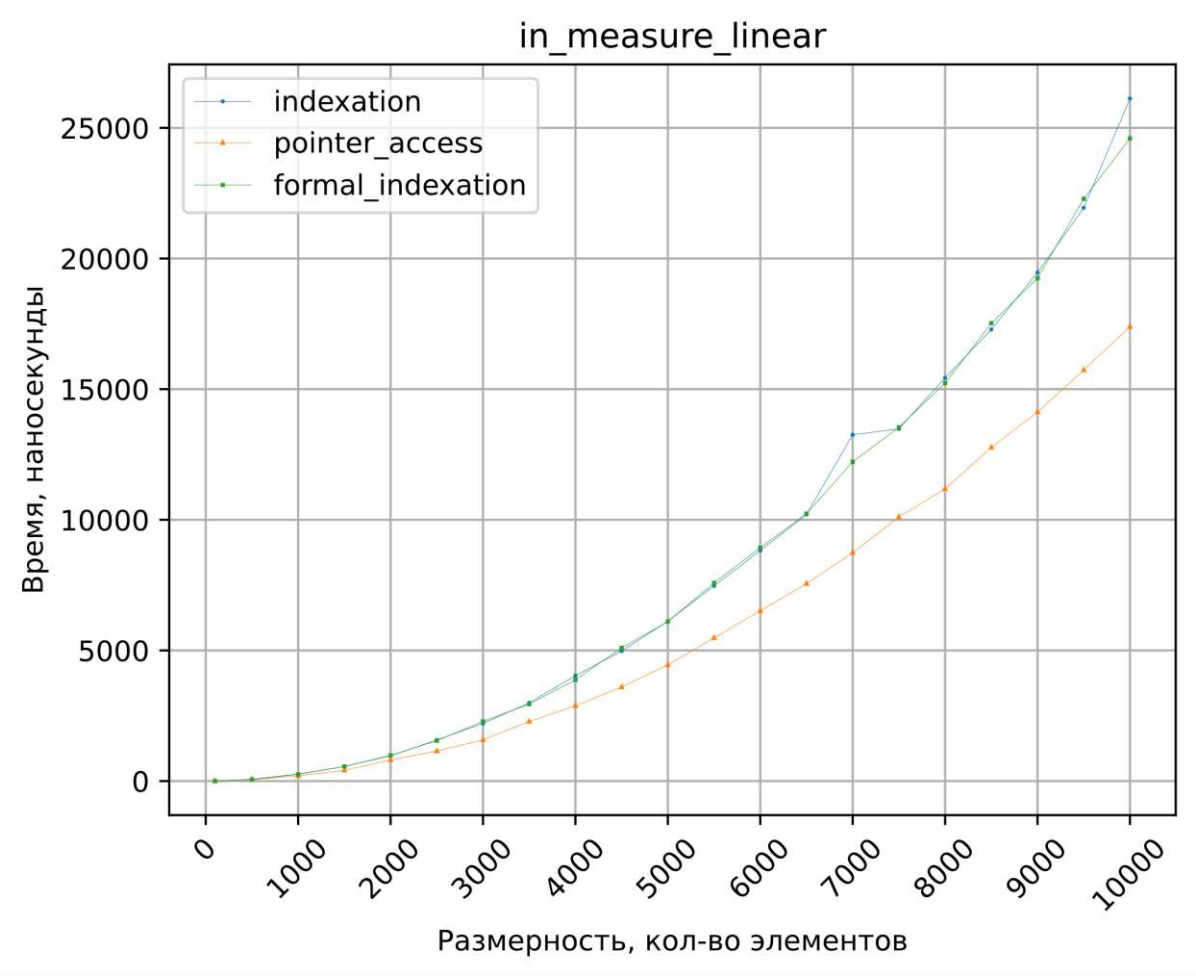
Fi_sort

Размер	Время среднее, нс	Кол-во повторов	RSE
100	5	29	0,99
500	64,65	628	0,74
1000	259,14	458	0,98
1500	599,22	496	0,70
2000	991,06	389	0,81
2500	1510,78	299	0,57
3000	2418,27	418	0,63
3500	3094,67	428	0,54
4000	4156,01	279	0,64
4500	4994,75	277	0,76

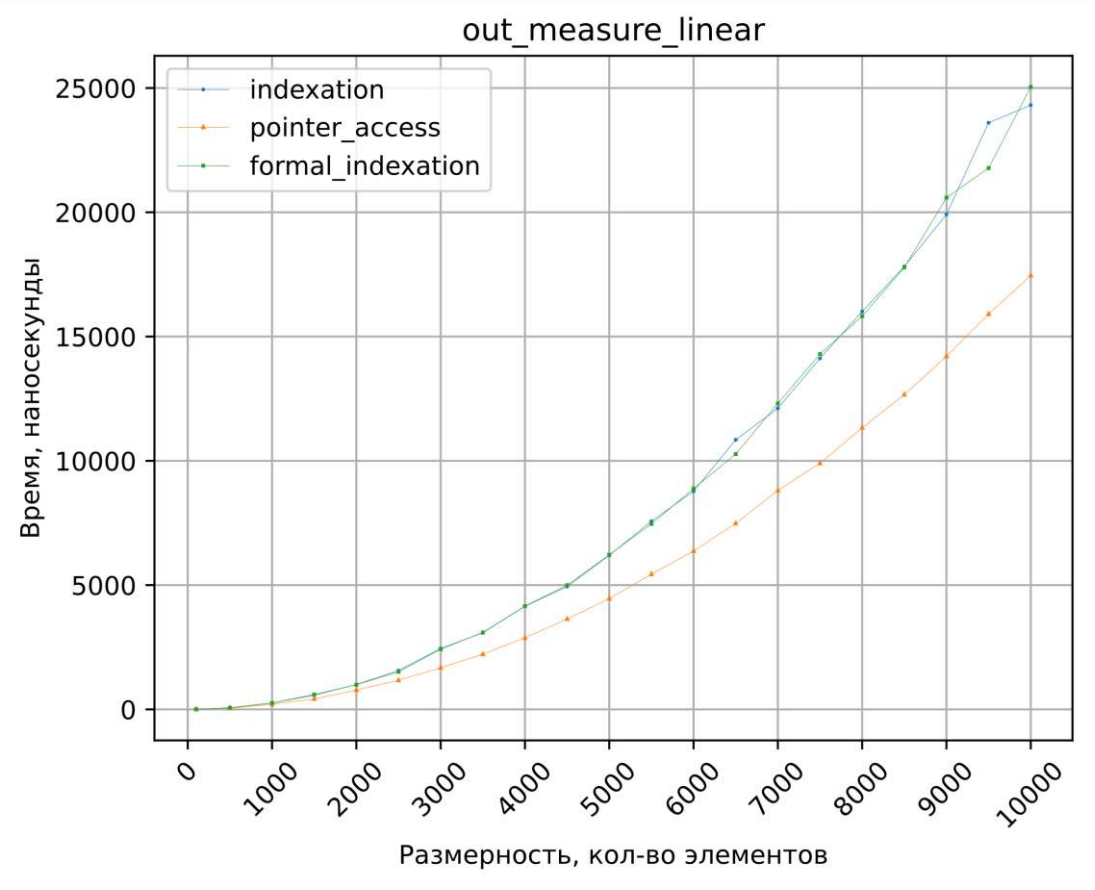
Pointer_sort

Размер	Время среднее, нс	Кол-во повторов	RSE
100	4,06	35	0,98
500	51,68	388	0,99
1000	206,00	695	0,99
1500	422,36	644	0,94
2000	779,15	570	0,90
2500	1172,74	379	0,60
3000	1673,31	358	0,62
3500	2221,42	360	0,81
4000	2880,77	269	0,50
4500	3635,41	250	0,58

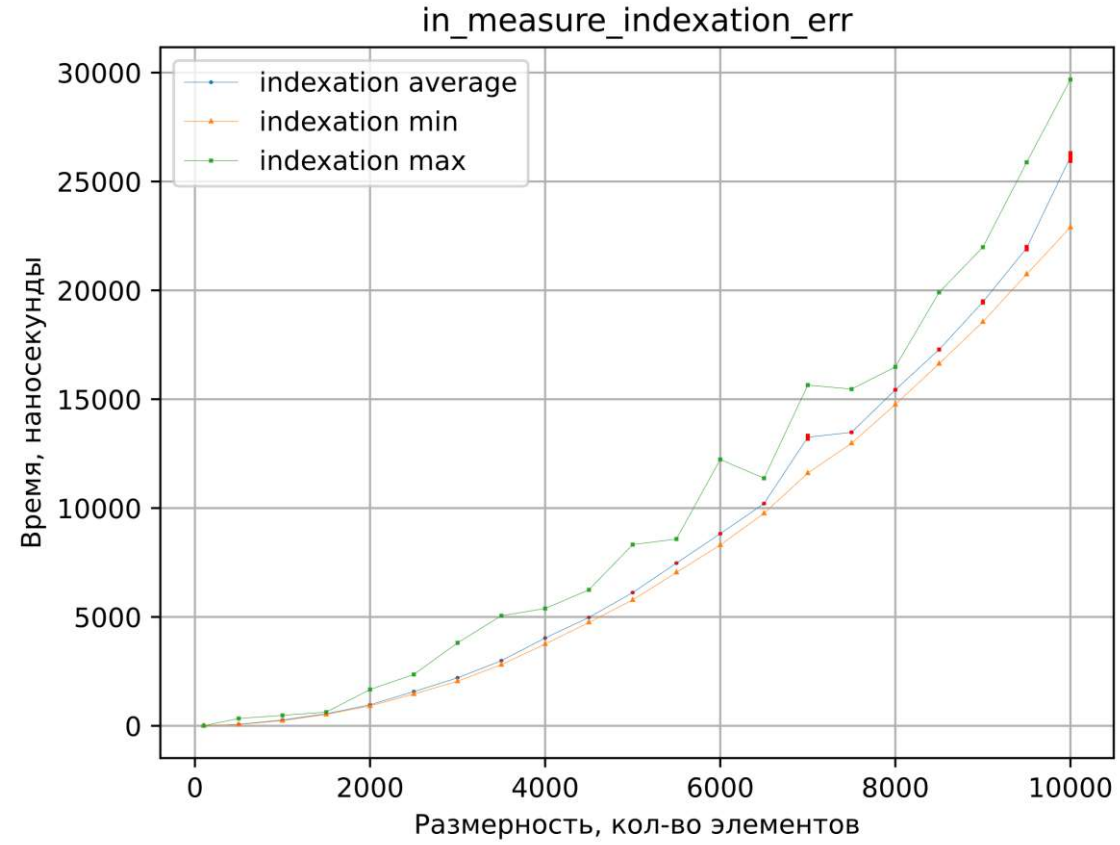
Линейный график внутреннее измерение



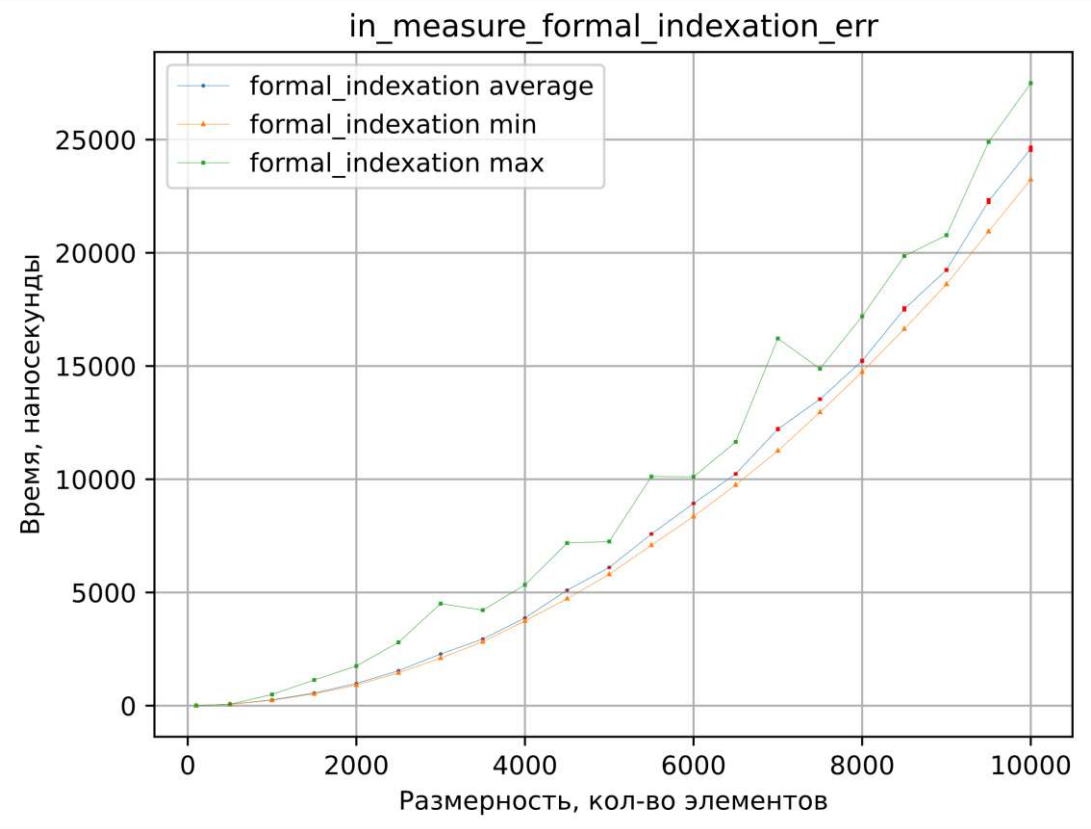
Линейный график внешнее измерении



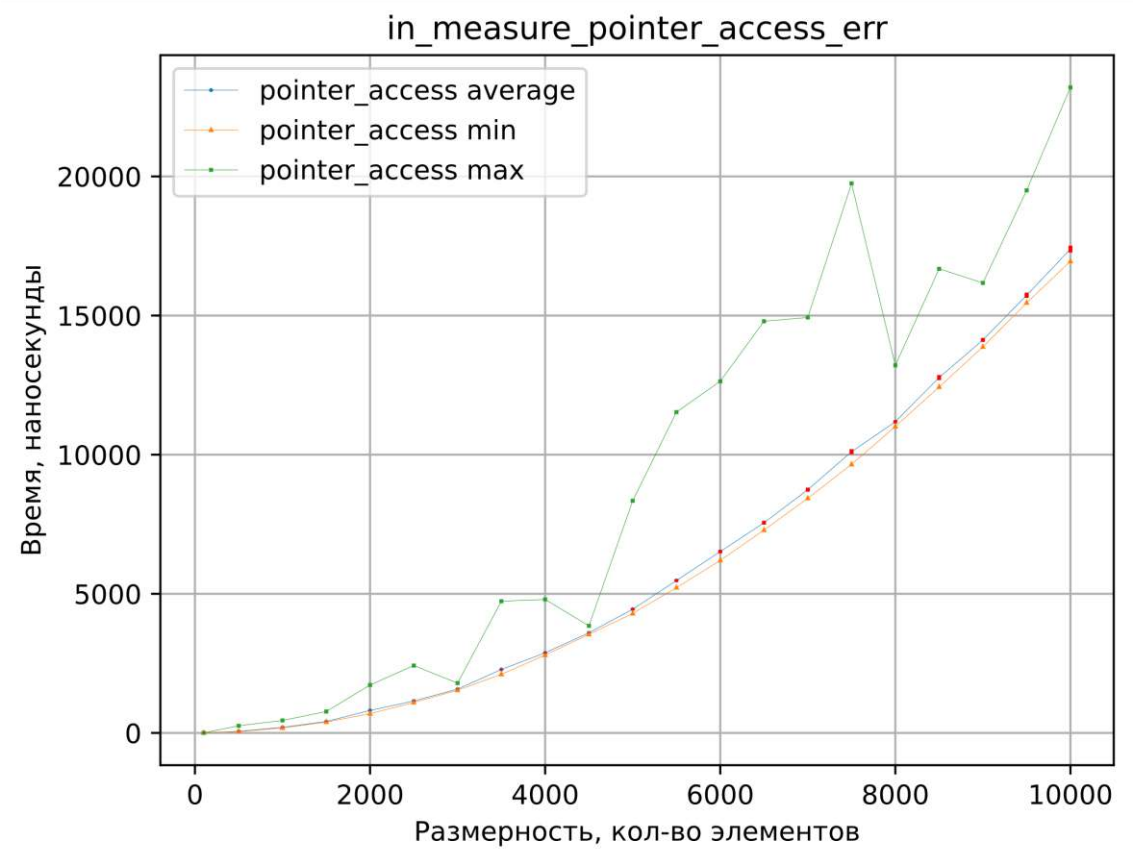
Линейный график с ошибкой внутреннее index_sort



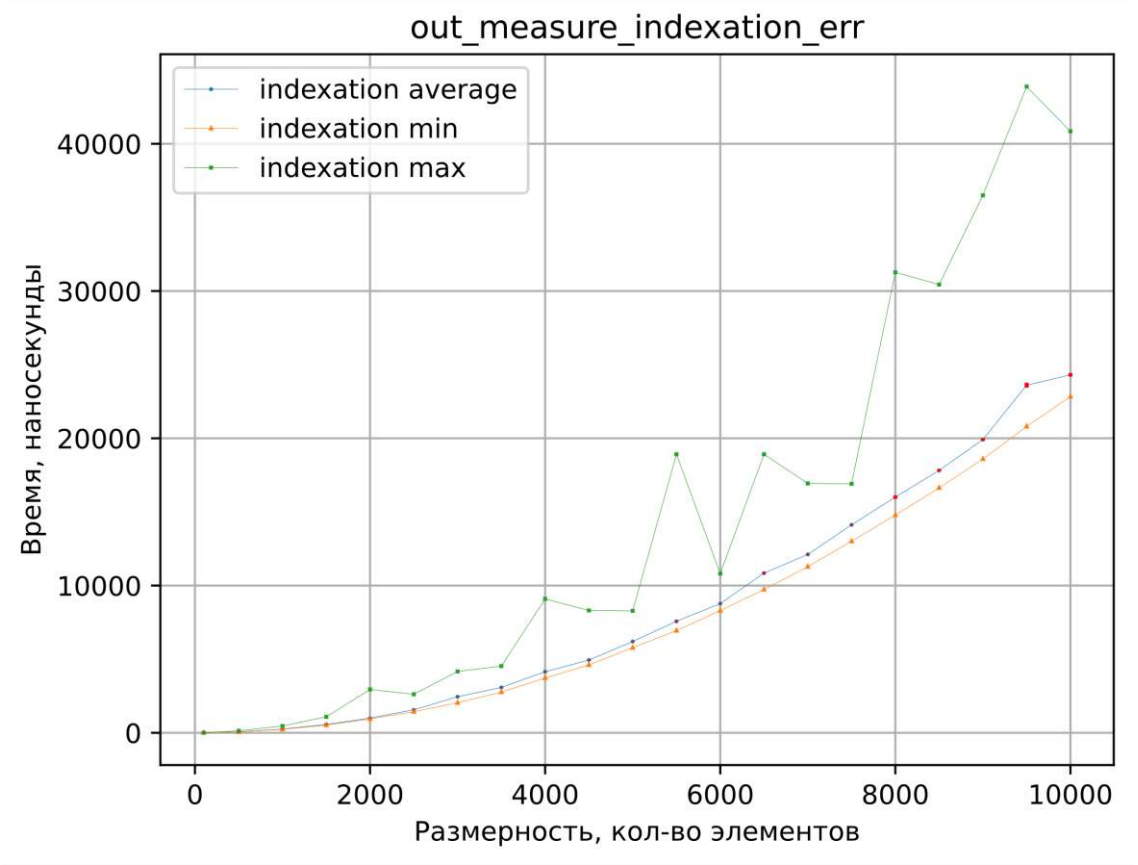
Линейный график с ошибкой внутреннее formal_indexation



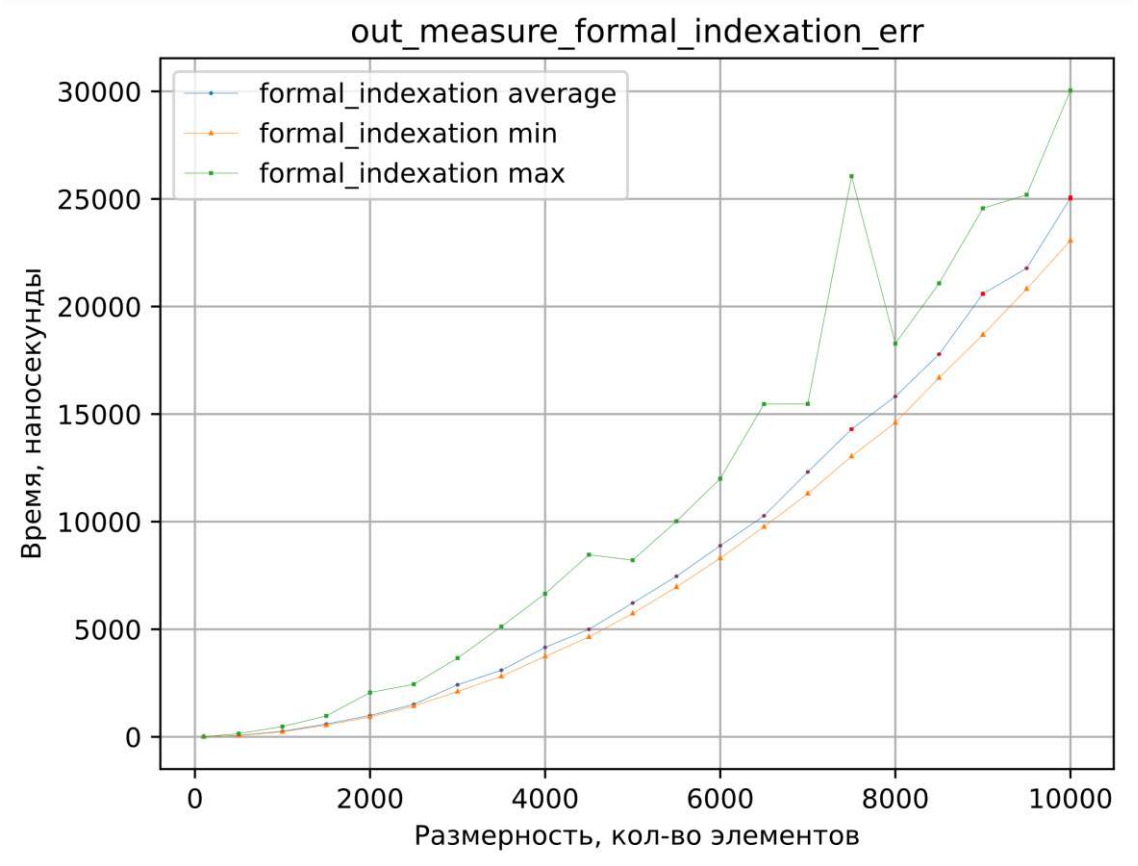
Линейный график с ошибкой внутреннее pointer_sort



Линейный график с ошибкой внешняя index_sort



Линейный график с ошибкой внешняя formal_indexation



Линейный график с ошибкой внешняя pointer_sort

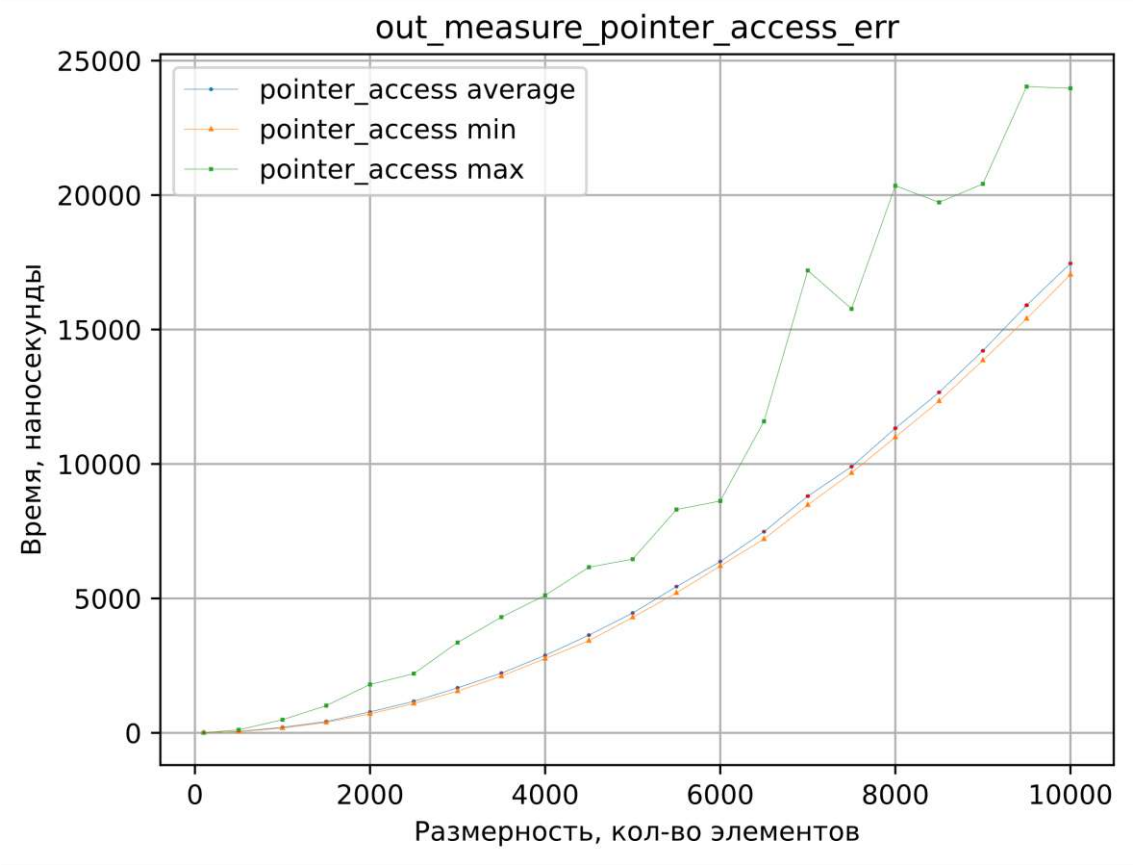


График с усами внутреннее измерение index_sort

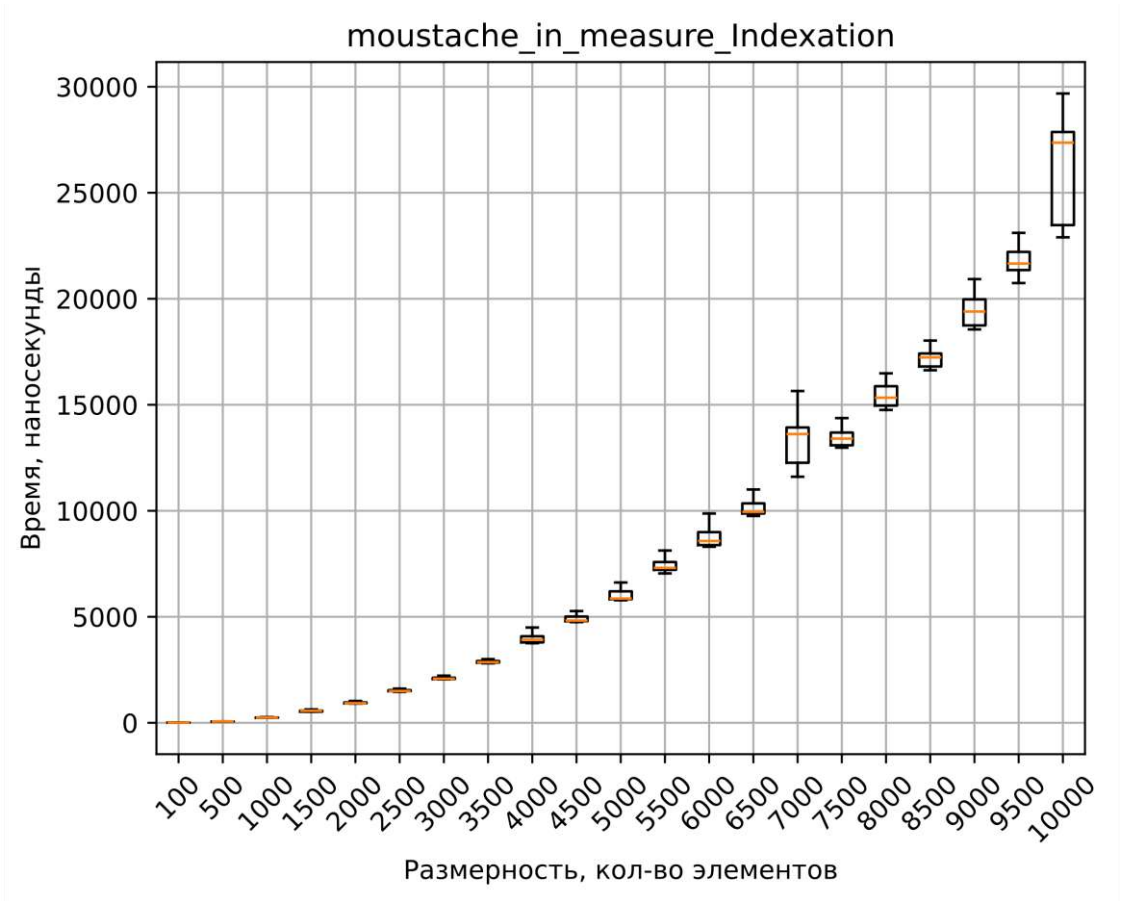


График с усами внешнее измерение index_sort