Оглавление

Общая инфраструктура исследования

Каталоги

1. apps содержит весь набор необходимых исполняемых файлов.

2. c_files содержит набор файлов с исходным кодом.

3. data содержит датасеты исследований.

4. graphs содержит графики различных типов.

5. prepdata содержит подготовленные данные для построения графиков.

Скрипты

1. build_apps.sh создает полный набор исполняемых файлов для заранее заданных длин массивов. Переменная SIZES определяет эти длины, для которых будут созданы исполняемые файлы.

2. update_data.sh добавляет некоторые измерения в датасет исследования. Первый аргумент определяет, для каких датасетов требуется уточнение данных. Он может принимать два значения: “all”, что указывает на добавление данных для всех наборов, или имя конкретного набора, для которого необходимо выполнить уточнение датасета. Второй параметр опционален и используется для указания количества итераций, которые следует выполнить для каждого наборов данных.

3. make_preproc.py подготавливает данные из набора для первичного анализа.

4. make_postproc.py строит кусочно-линейный график зависимости времени выполнения от числа элементов массива, с ошибками, с усами.

5. go.sh выполняет последовательный вызов четырех предыдущих скриптов и предоставляет данные исследования. В него входят определенные константы: MIN_ITER и MAX_ITER, которые определяют минимальное и максимальное количество итераций, которые должны быть выполнены при обновлении данных, и COUNT, количество новых данных, получаемых при каждой итерации.

6. get_rse.py вычисляет относительную стандартную ошибку среднего времени для одного набора данных.

7. table_rse.py предназначен для вывода результатов анализа данных в виде таблицы.

Все скрипты представлены в Приложении.

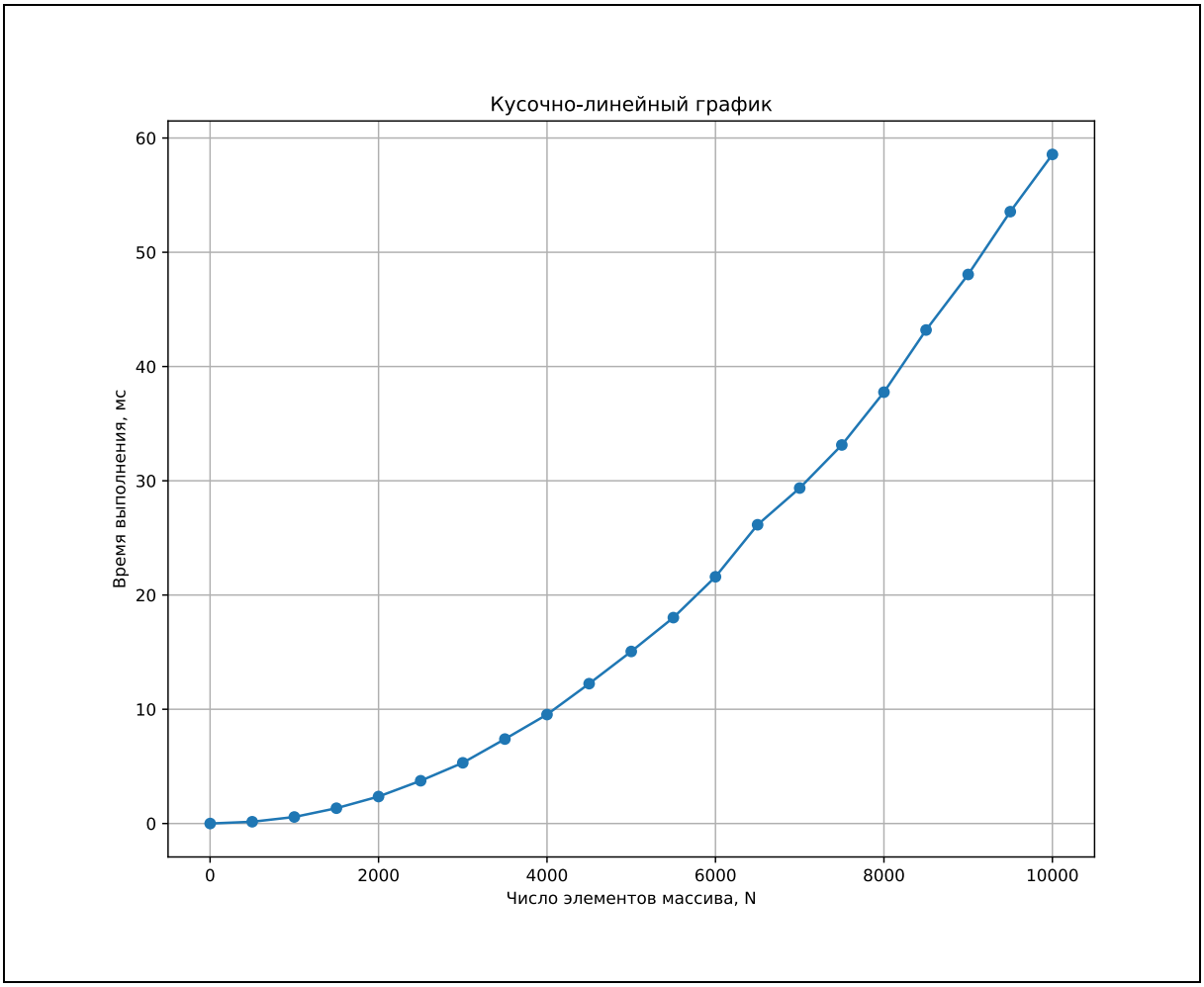
Сортировка одномерного массива с использованием операции индексации a[i]

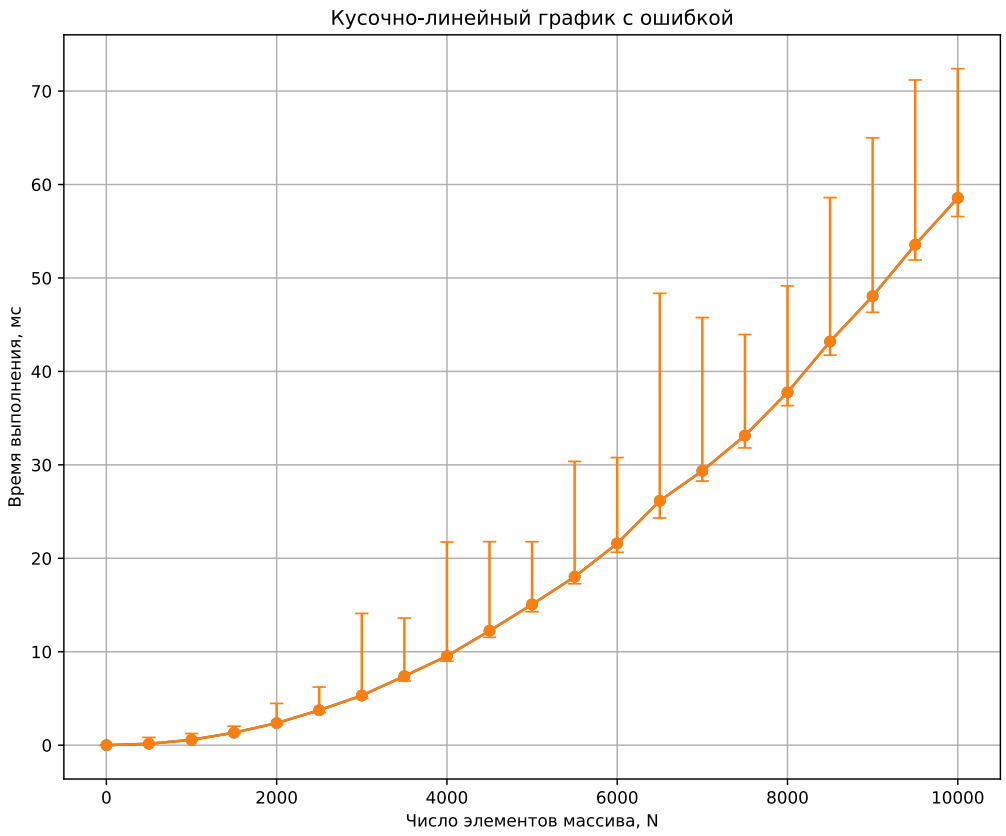
```
…
void insertion_sort(int arr[], size_t n)
{
   for (size_t I = 1; I < n; i++)
   {
      int item = arr[i];
      int j = I;
      while ((j > 0) && (item < arr[j - 1]))
      {
         arr[j] = arr[j - 1];
         j--;
      }
      arr[j] = item;
   }
}
…
```

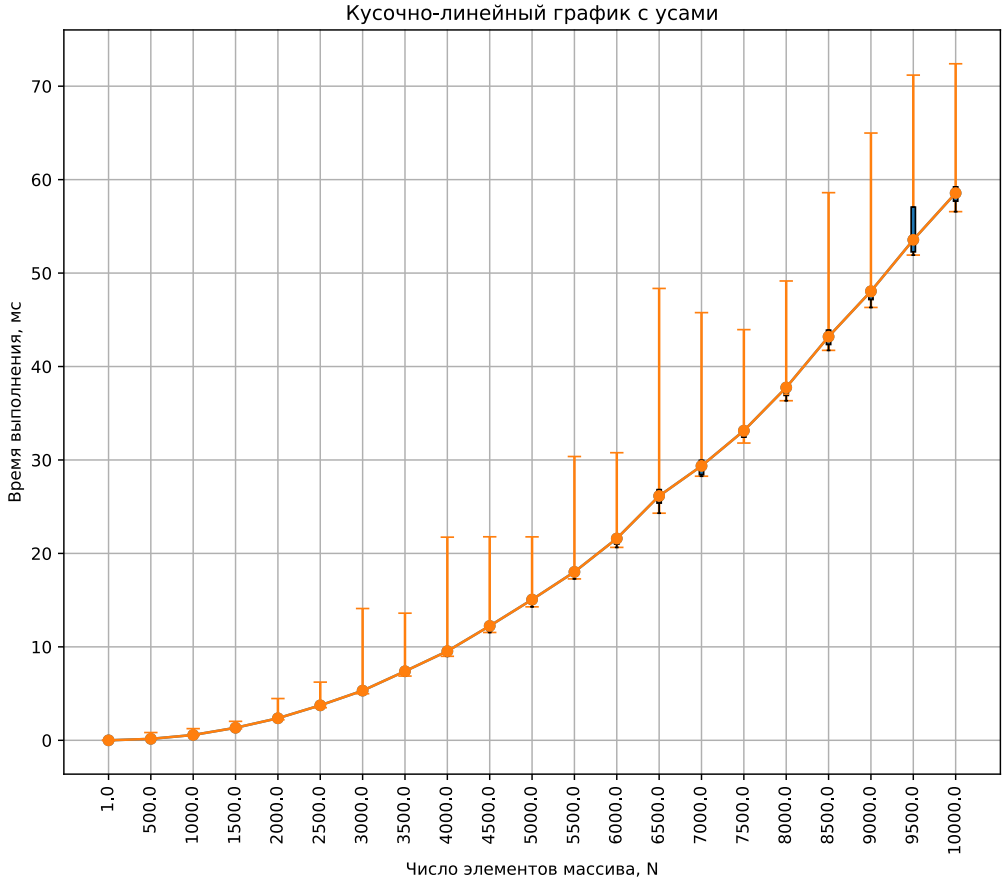
Инфраструктура измерения времени выполнения функции в самой программе

Размер, N	t, мс	Кол-во повторов	RSE, %
1	3.7e-05	1000	0.16
500	0.15	1000	0.68
1000	0.57	1000	0.37
1500	1.3	1000	0.23
2000	2.4	1000	0.21
2500	3.7	1000	0.21
3000	5.3	1000	0.27
3500	7.4	1000	0.19
4000	9.5	1000	0.24
4500	12	1000	0.21
5000	15	1000	0.12
5500	18	1000	0.14
6000	22	1000	0.13

6500	26	1000	0.24
7000	29	1000	0.12
7500	33	1000	0.12
8000	38	1000	0.099
8500	43	1000	0.11
9000	48	1000	0.092
9500	54	1000	0.1
10000	59	1000	0.096

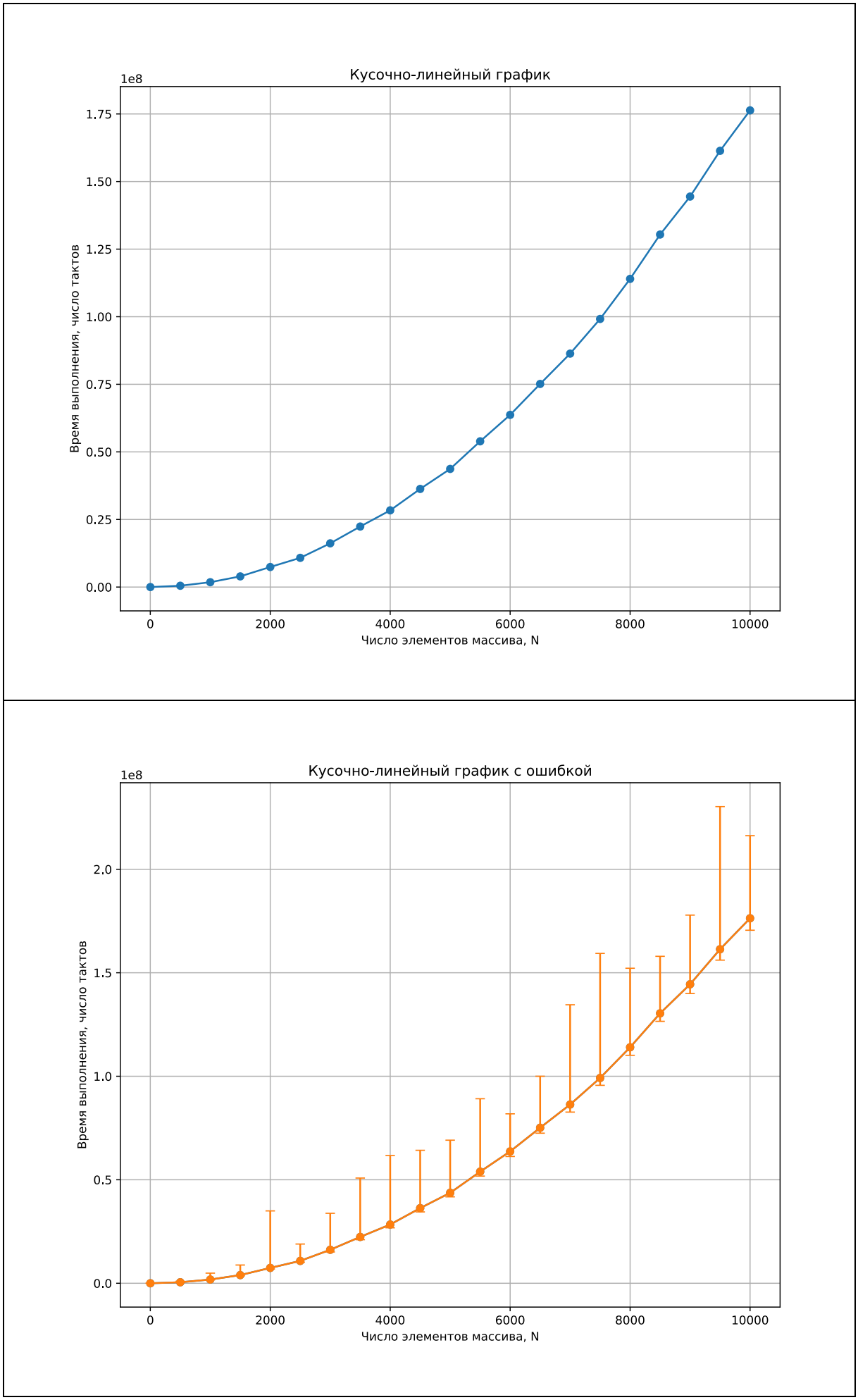


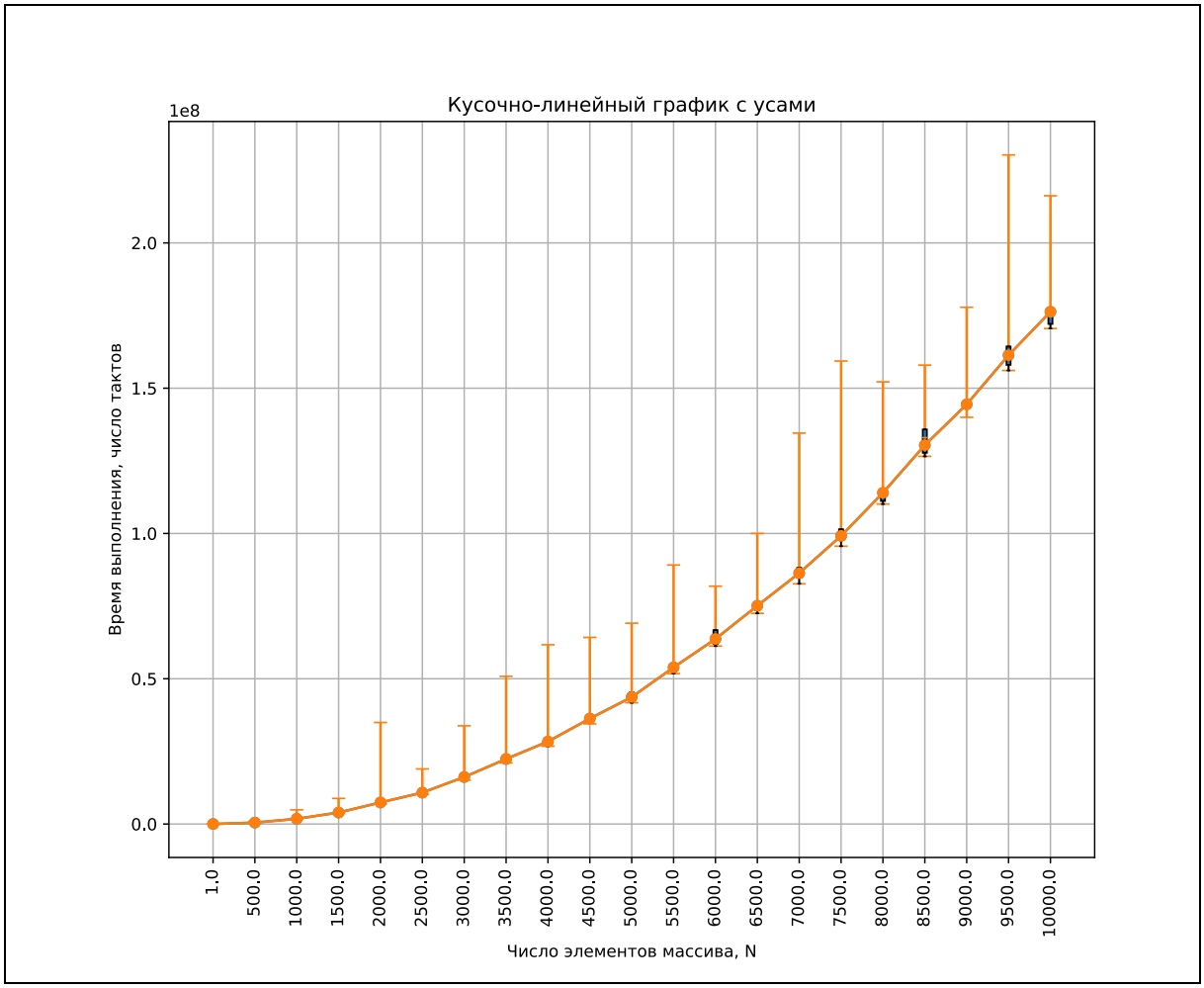

Кусочно-линейный график с ошибкой
Время выполнения, мс
Число элементов массива, N
0
10
20
30
40
50
60
70
0
2000
4000
6000
8000
10000


Кусочно-линейный график с усами
Время выполнения, мс
Число элементов массива, N
0
10
20
30
40
50
60
70
1.0
500.0
1000.0
1500.0
2000.0
2500.0
3000.0
3500.0
4000.0
4500.0
5000.0
5500.0
6000.0
6500.0
7000.0
7500.0
8000.0
8500.0
9000.0
9500.0
10000.0

Инфраструктура измерения времени выполнения функции в самой программе с использованием Time Stamp Counter

Размер, N	t, число тактов	Кол-во повторов	RSE, %
1	29	4536	1
500	4.8e+05	1000	0.47
1000	1.8e+06	1000	0.35
1500	3.9e+06	1000	0.24
2000	7.4e+06	1000	0.48
2500	1.1e+07	1000	0.2
3000	1.6e+07	1000	0.18
3500	2.2e+07	1000	0.24
4000	2.8e+07	1000	0.2
4500	3.6e+07	1000	0.16
5000	4.4e+07	1000	0.12
5500	5.4e+07	1000	0.13
6000	6.4e+07	1000	0.11
6500	7.5e+07	1000	0.1
7000	8.6e+07	1000	0.15
7500	9.9e+07	1000	0.11
8000	1.1e+08	1000	0.1
8500	1.3e+08	1000	0.077
9000	1.4e+08	1000	0.078
9500	1.6e+08	1000	0.1
10000	1.8e+08	1000	0.084

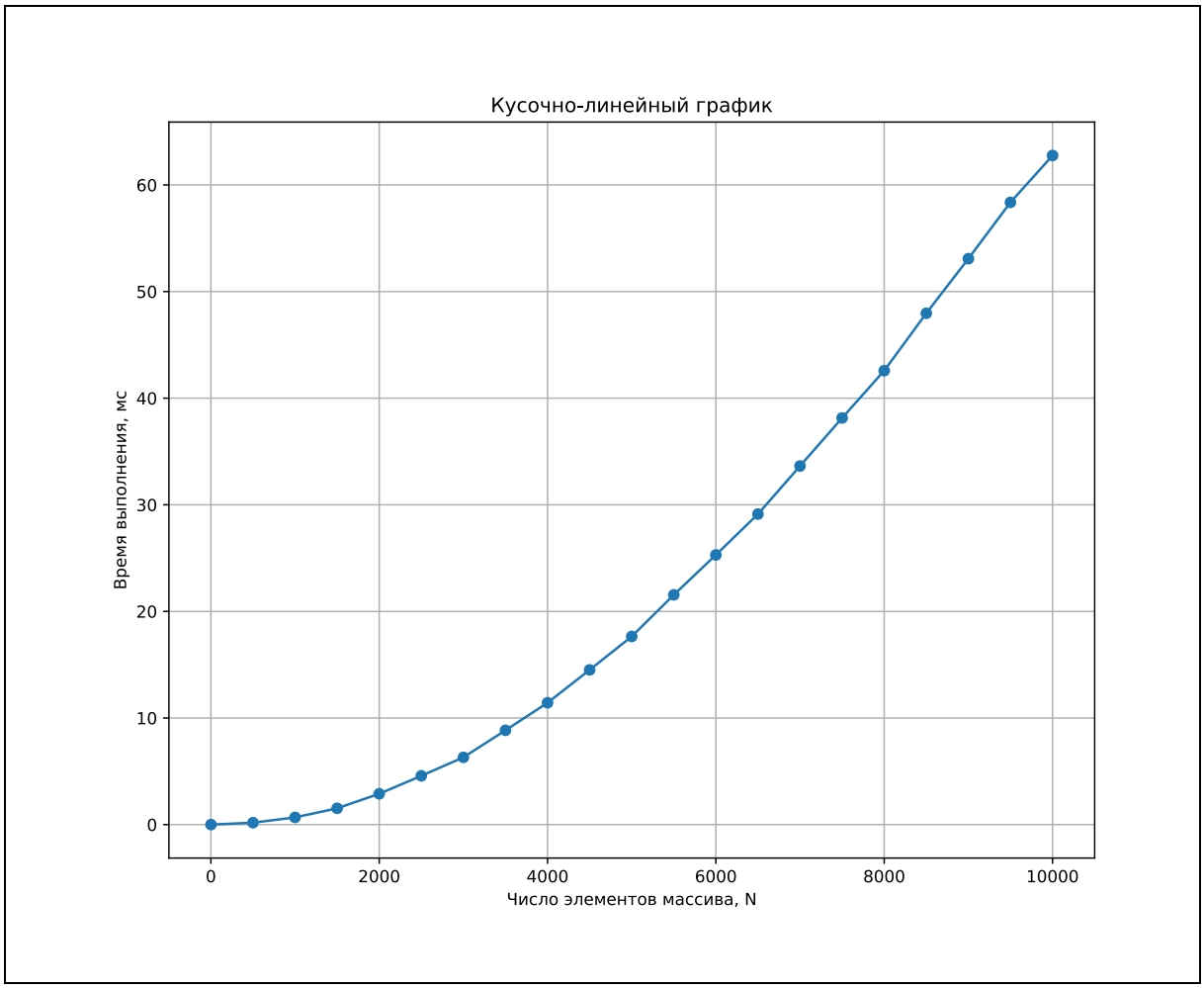

Кусочно-линейный график
1e8
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
Время выполнения, число тактов
0
2000
4000
6000
8000
10000
Число элементов массива, N
Кусочно-линейный график с ошибкой
1e8
2.0
1.5
1.0
0.5
0.0
Время выполнения, число тактов
0
2000
4000
6000
8000
10000
Число элементов массива, N

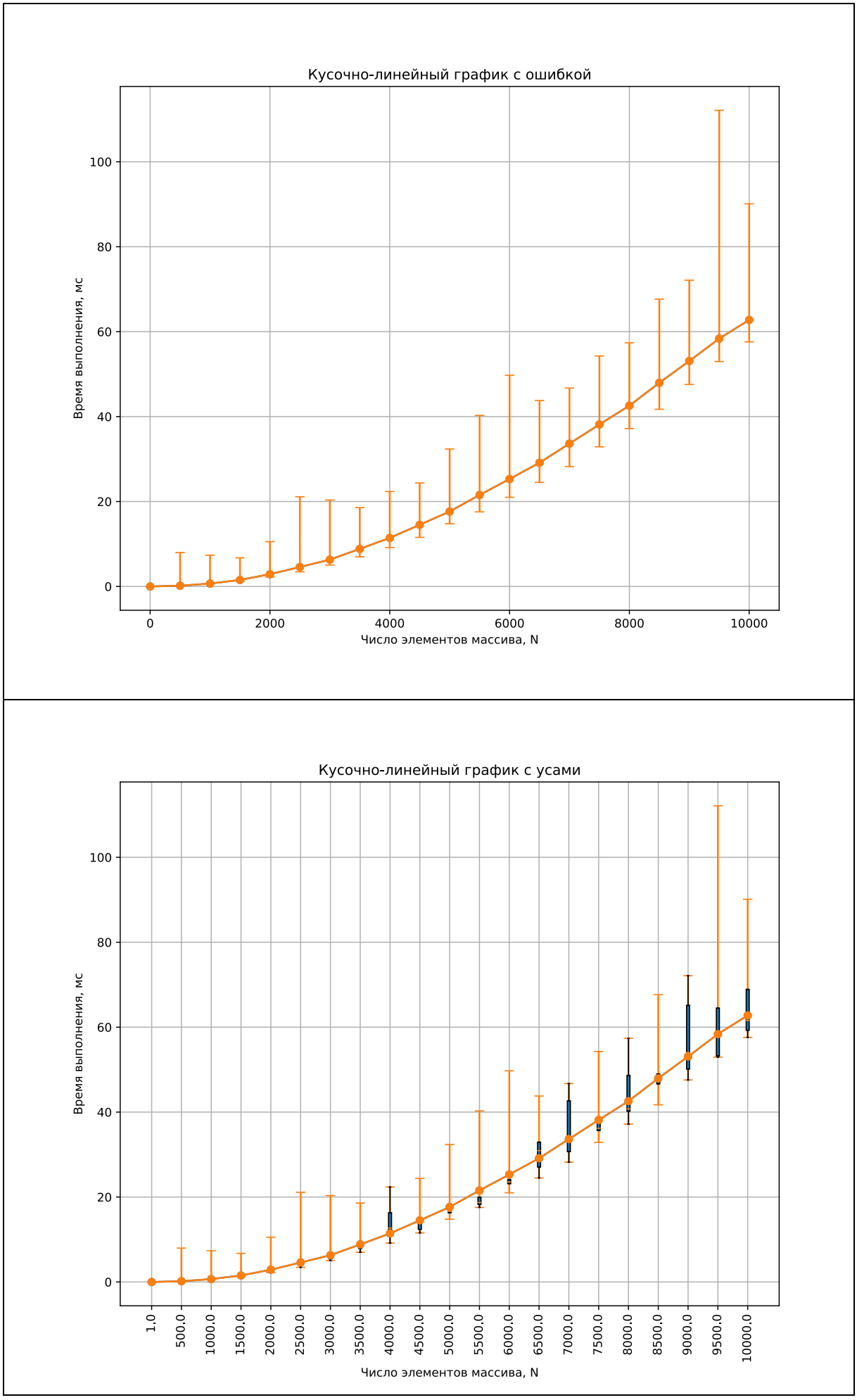


Инфраструктура измерения времени выполнения функции вне программы

Размер, N	t, мс	Кол-во повторов	RSE, %
1	0.00011	2100	0.99
500	0.18	10900	0.99
1000	0.68	2600	1
1500	1.5	1300	0.97
2000	2.9	1000	0.96
2500	4.6	1000	0.85
3000	6.3	1000	0.63
3500	8.8	1000	0.57
4000	11	1000	0.49
4500	15	1000	0.45
5000	18	1000	0.38
5500	22	1000	0.41
6000	25	1000	0.38
6500	29	1000	0.32
7000	34	1000	0.3

7500	38	1000	0.28
8000	43	1000	0.22
8500	48	1000	0.21
9000	53	1000	0.18
9500	58	1000	0.19
10000	63	1000	0.15




Кусочно-линейный график с ошибкой
Время выполнения, мс
Число элементов массива, N
0
20
40
60
80
100
0
2000
4000
6000
8000
10000
Кусочно-линейный график с усами
Время выполнения, мс
Число элементов массива, N
1.0
500.0
1000.0
1500.0
2000.0
2500.0
3000.0
3500.0
4000.0
4500.0
5000.0
5500.0
6000.0
6500.0
7000.0
7500.0
8000.0
8500.0
9000.0
9500.0
10000.0

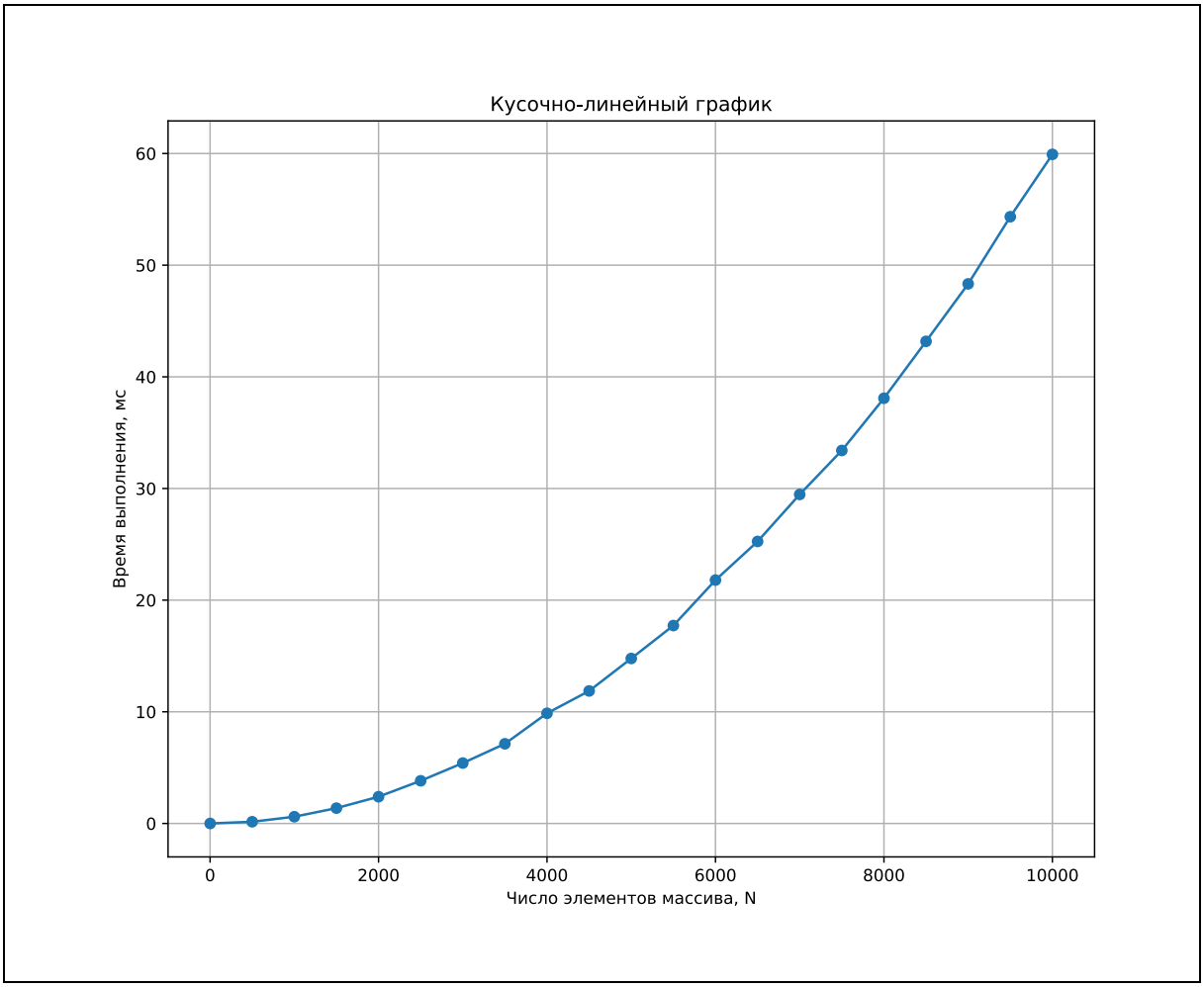
Сортировка одномерного массива с использованием формальной замены операции индексации на выражение *(a + i)

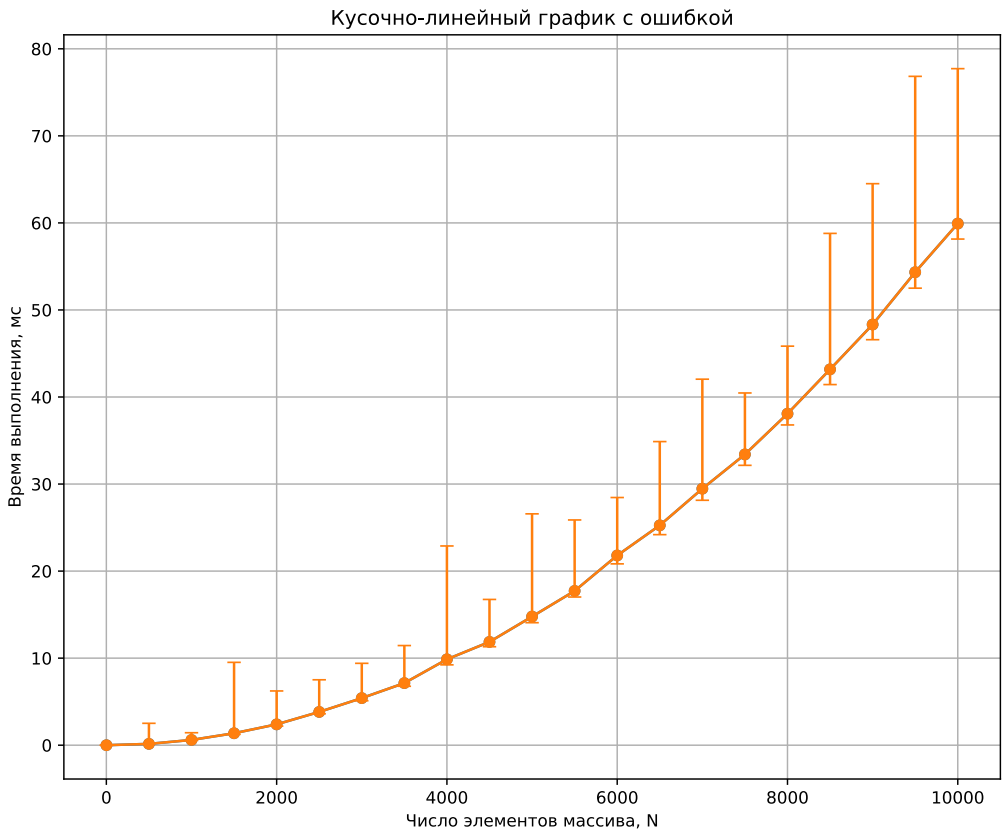
```
…
void insertion_sort(int *a, size_t n)
{
   for (size_t i = 1; i < n; i++)
   {
      int item = *(a + i);
      int j = i;
      while ((j > 0) && (item < *(a + (j - 1))))
      {
         *(a + j) = *(a + (j - 1));
         j--;
      }
      *(a + j) = item;
   }
}
…
```

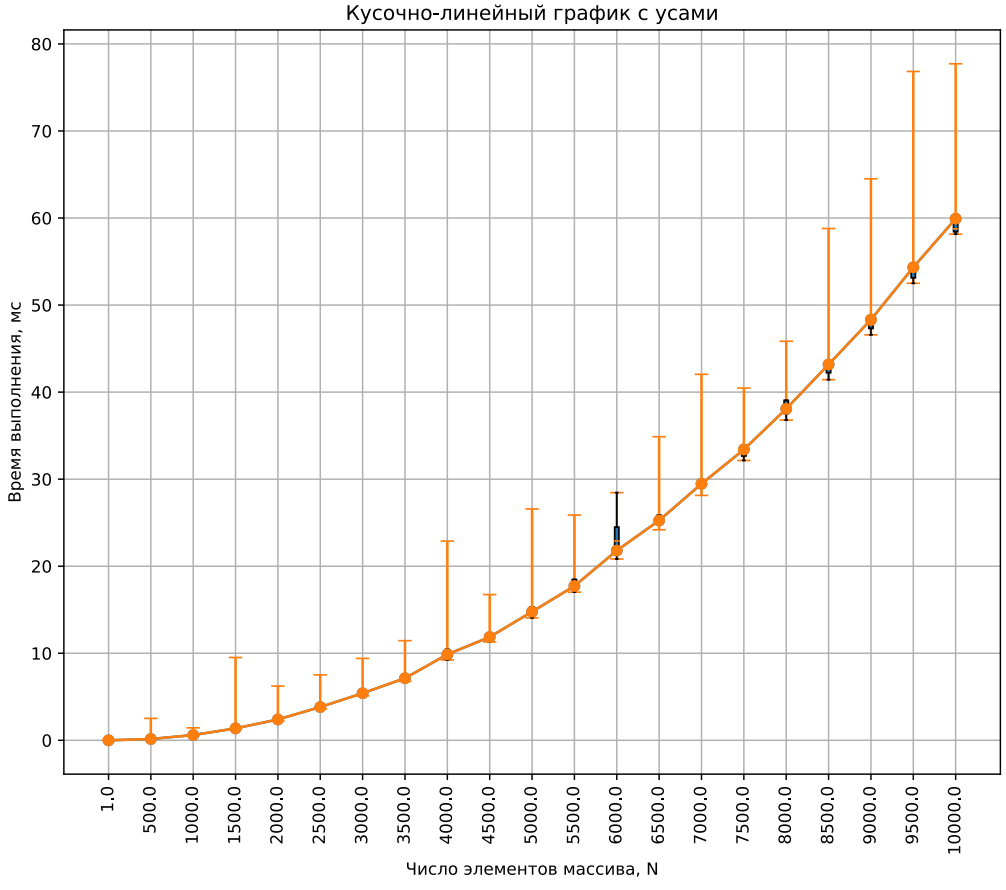
Инфраструктура измерения времени выполнения функции в самой программе

Размер, N	t, мс	Кол-во повторов	RSE, %
1	3.7e-05	1000	0.28
500	0.15	1749	1
1000	0.6	1000	0.35
1500	1.4	1000	0.65
2000	2.4	1000	0.29
2500	3.8	1000	0.19
3000	5.4	1000	0.17
3500	7.1	1000	0.15
4000	9.9	1000	0.23
4500	12	1000	0.12
5000	15	1000	0.14
5500	18	1000	0.11

6000	22	1000	0.12
6500	25	1000	0.12
7000	29	1000	0.13
7500	33	1000	0.093
8000	38	1000	0.078
8500	43	1000	0.12
9000	48	1000	0.1
9500	54	1000	0.095
10000	60	1000	0.087

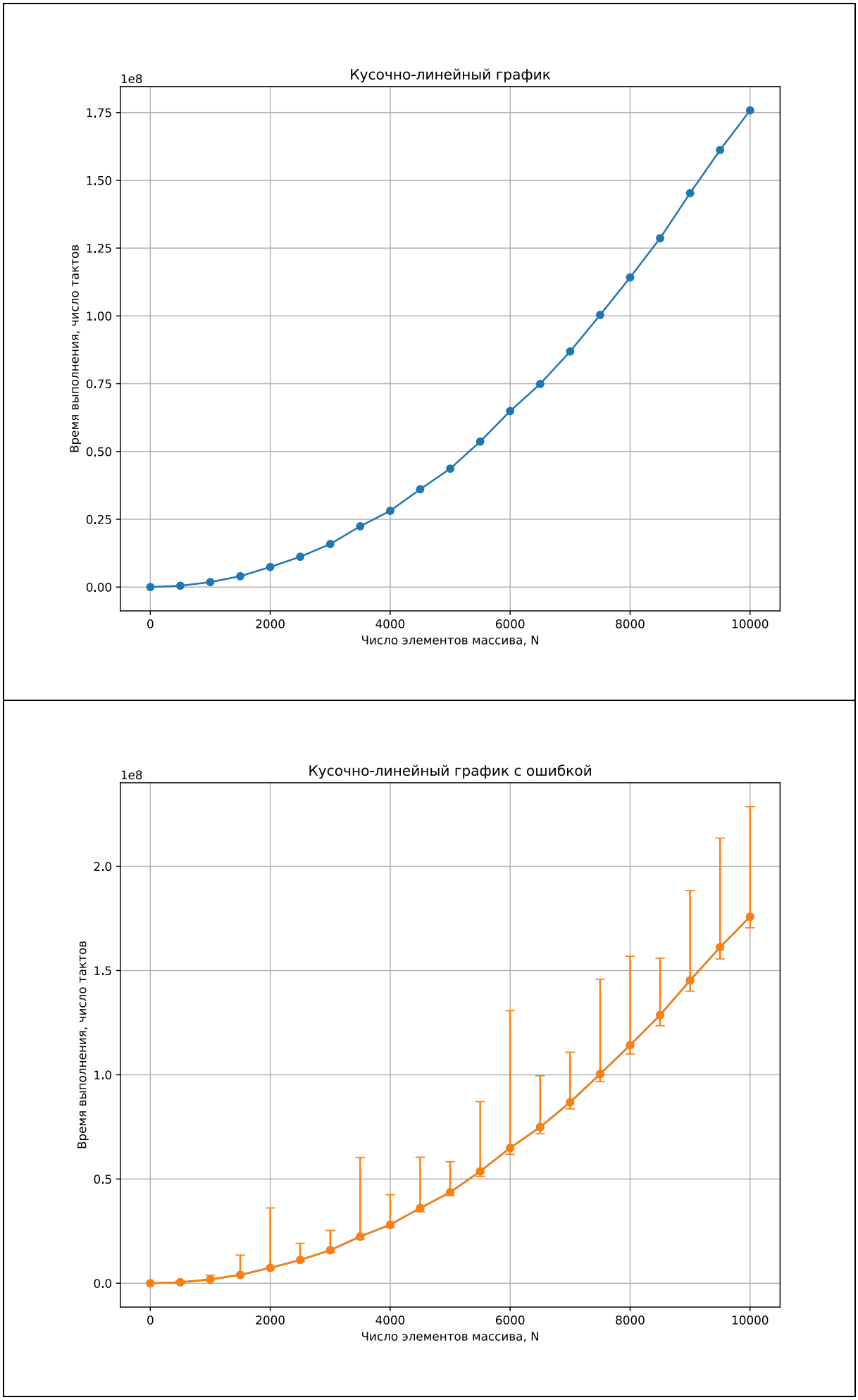


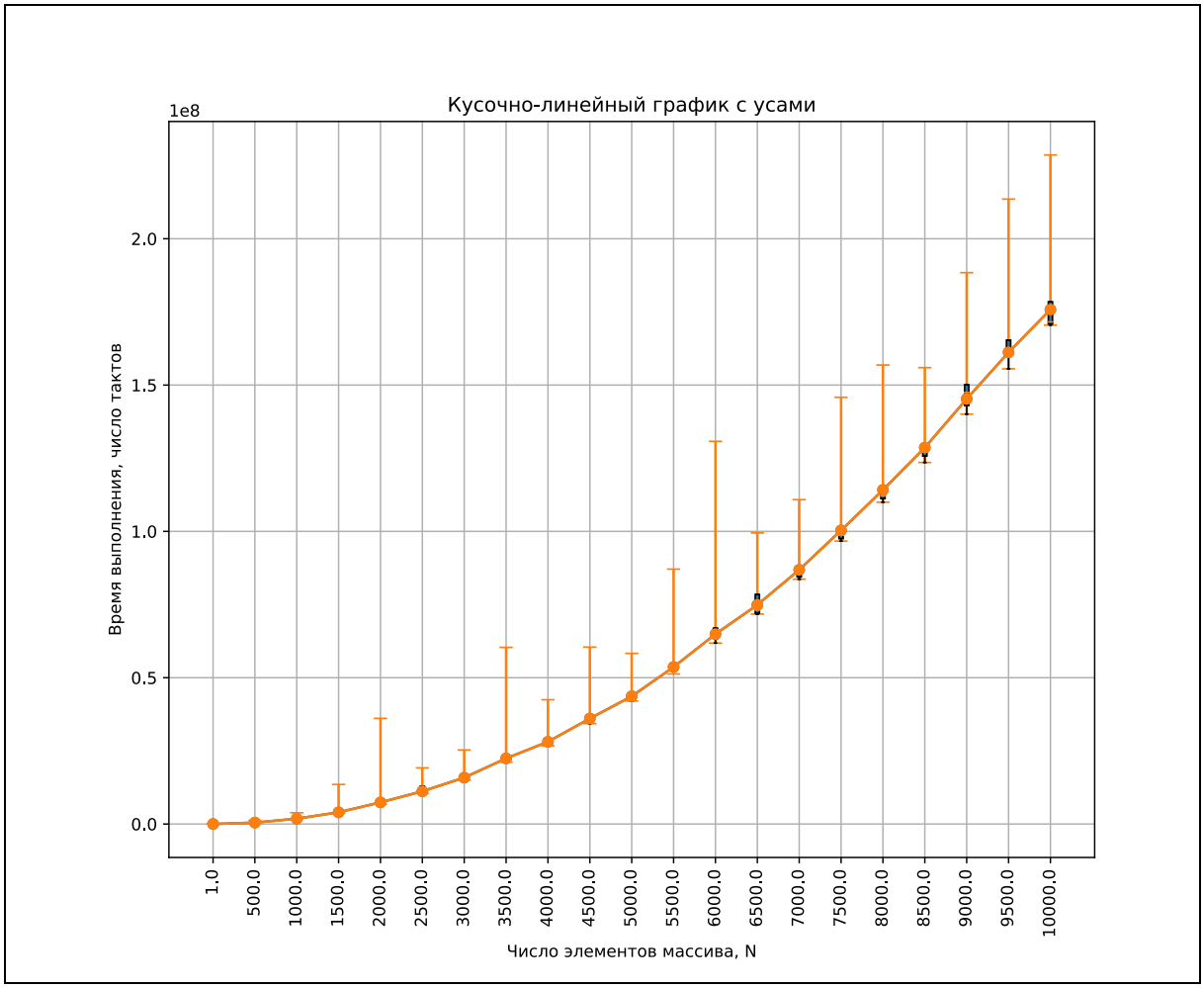

Кусочно-линейный график с ошибкой
Время выполнения, мс
Число элементов массива, N
0
10
20
30
40
50
60
70
80
0
2000
4000
6000
8000
10000


Кусочно-линейный график с усами
Время выполнения, мс
Число элементов массива, N
0
10
20
30
40
50
60
70
80
1.0
500.0
1000.0
1500.0
2000.0
2500.0
3000.0
3500.0
4000.0
4500.0
5000.0
5500.0
6000.0
6500.0
7000.0
7500.0
8000.0
8500.0
9000.0
9500.0
10000.0

Инфраструктура измерения времени выполнения функции в самой программе с использованием Time Stamp Counter

Размер, N	t, число тактов	Кол-во повторов	RSE, %
1	40	1001	1
500	4.8e+05	1000	0.57
1000	1.8e+06	1000	0.36
1500	4e+06	1000	0.4
2000	7.4e+06	1000	0.65
2500	1.1e+07	1000	0.2
3000	1.6e+07	1000	0.19
3500	2.2e+07	1000	0.24
4000	2.8e+07	1000	0.14
4500	3.6e+07	1000	0.15
5000	4.4e+07	1000	0.089
5500	5.4e+07	1000	0.15
6000	6.5e+07	1000	0.2
6500	7.5e+07	1000	0.12
7000	8.7e+07	1000	0.1
7500	1e+08	1000	0.097
8000	1.1e+08	1000	0.095
8500	1.3e+08	1000	0.1
9000	1.5e+08	1000	0.1
9500	1.6e+08	1000	0.094
10000	1.8e+08	1000	0.087

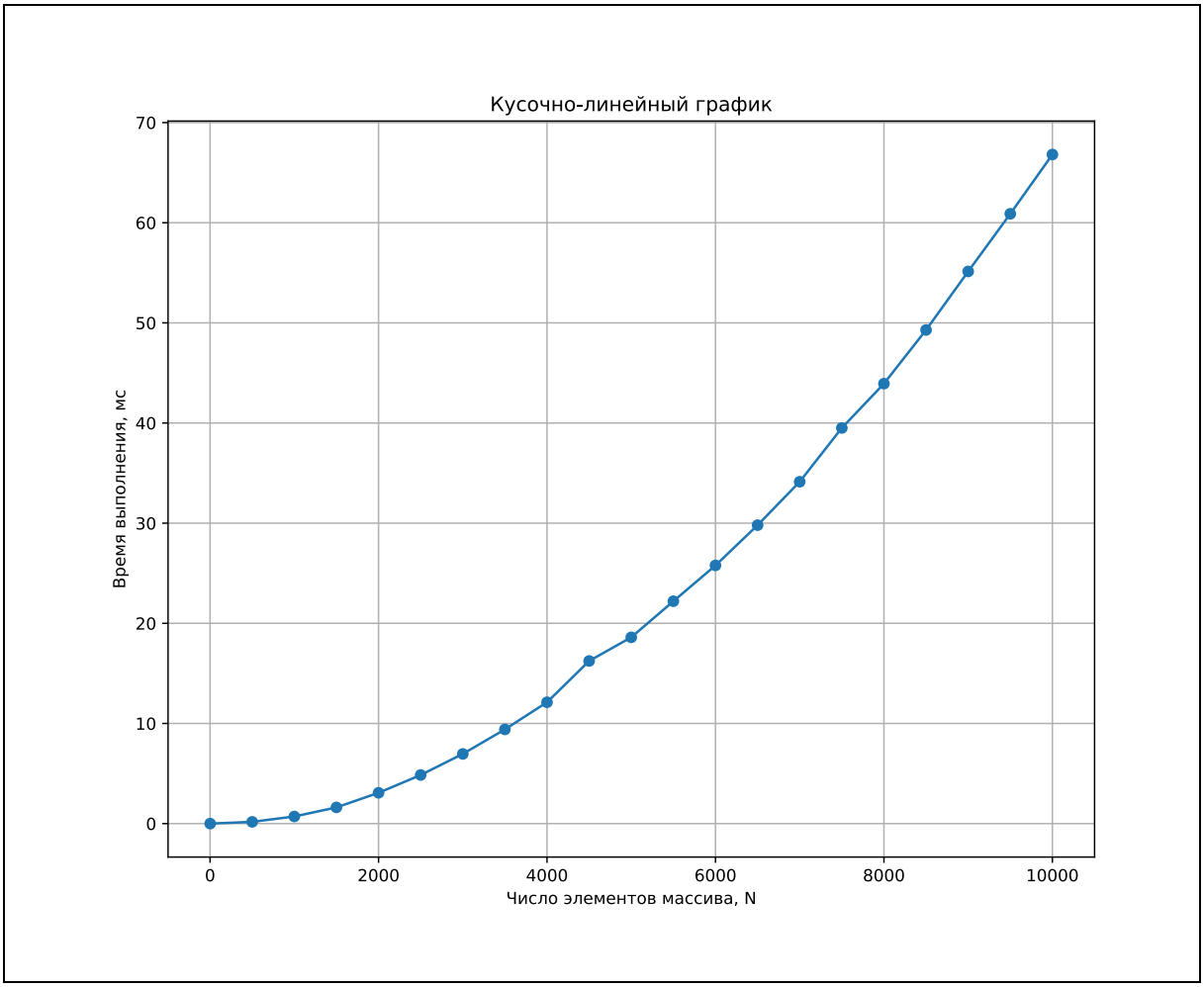

Кусочно-линейный график
1e8
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
Время выполнения, число тактов
0
2000
4000
6000
8000
10000
Число элементов массива, N
Кусочно-линейный график с ошибкой
1e8
2.0
1.5
1.0
0.5
0.0
Время выполнения, число тактов
0
2000
4000
6000
8000
10000
Число элементов массива, N

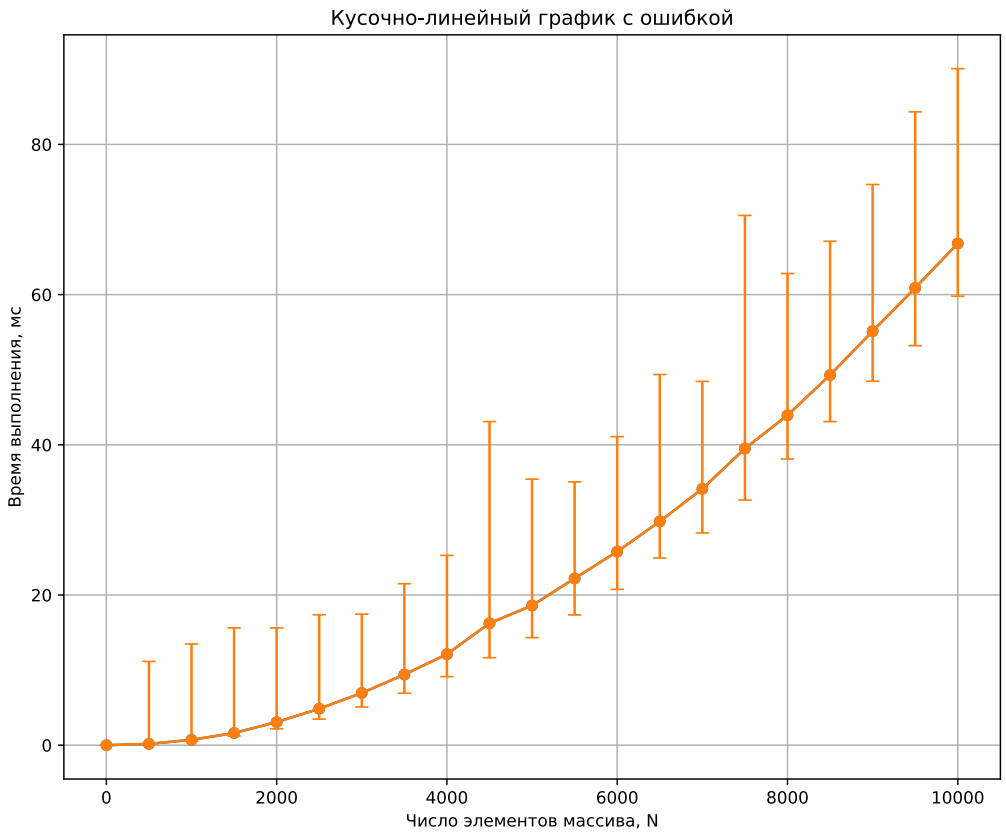


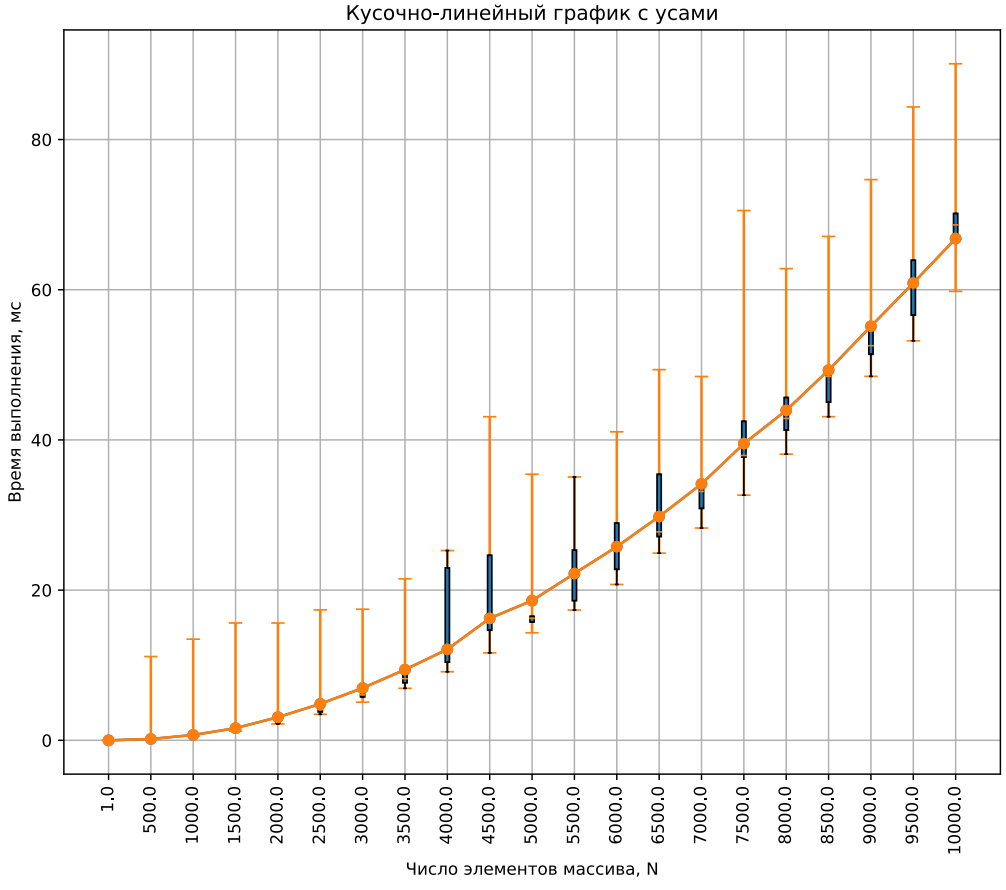
Инфраструктура измерения времени выполнения функции вне программы

Размер, N	t, мс	Кол-во повторов	RSE, %
1	0.00011	2600	0.99
500	0.18	23800	1
1000	0.71	7100	1
1500	1.6	3800	0.98
2000	3.1	2400	0.99
2500	4.9	1400	0.99
3000	7	1000	0.9
3500	9.4	1000	0.74
4000	12	1000	0.67
4500	16	1000	0.81
5000	19	1000	0.5
5500	22	1000	0.42
6000	26	1000	0.39
6500	30	1000	0.35
7000	34	1000	0.3

7500	40	1000	0.3
8000	44	1000	0.26
8500	49	1000	0.24
9000	55	1000	0.23
9500	61	1000	0.22
10000	67	1000	0.2




Кусочно-линейный график с ошибкой
Время выполнения, мс
0
20
40
60
80
0
2000
4000
6000
8000
10000
Число элементов массива, N


Кусочно-линейный график с усами
Время выполнения, мс
0
20
40
60
80
1.0
500.0
1000.0
1500.0
2000.0
2500.0
3000.0
3500.0
4000.0
4500.0
5000.0
5500.0
6000.0
6500.0
7000.0
7500.0
8000.0
8500.0
9000.0
9500.0
10000.0
Число элементов массива, N

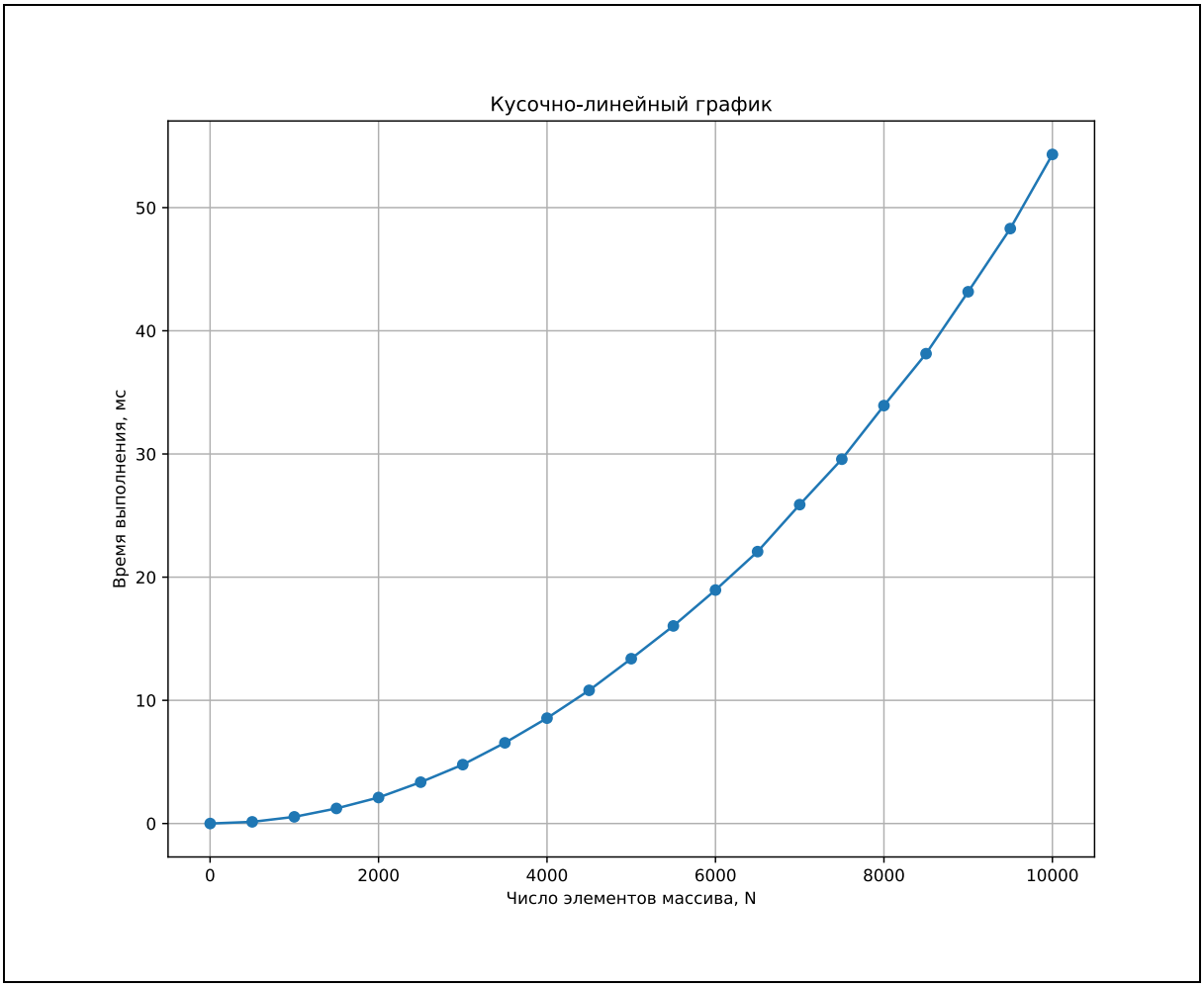
Сортировка одномерного массива с использованием указателей для работы с массивом

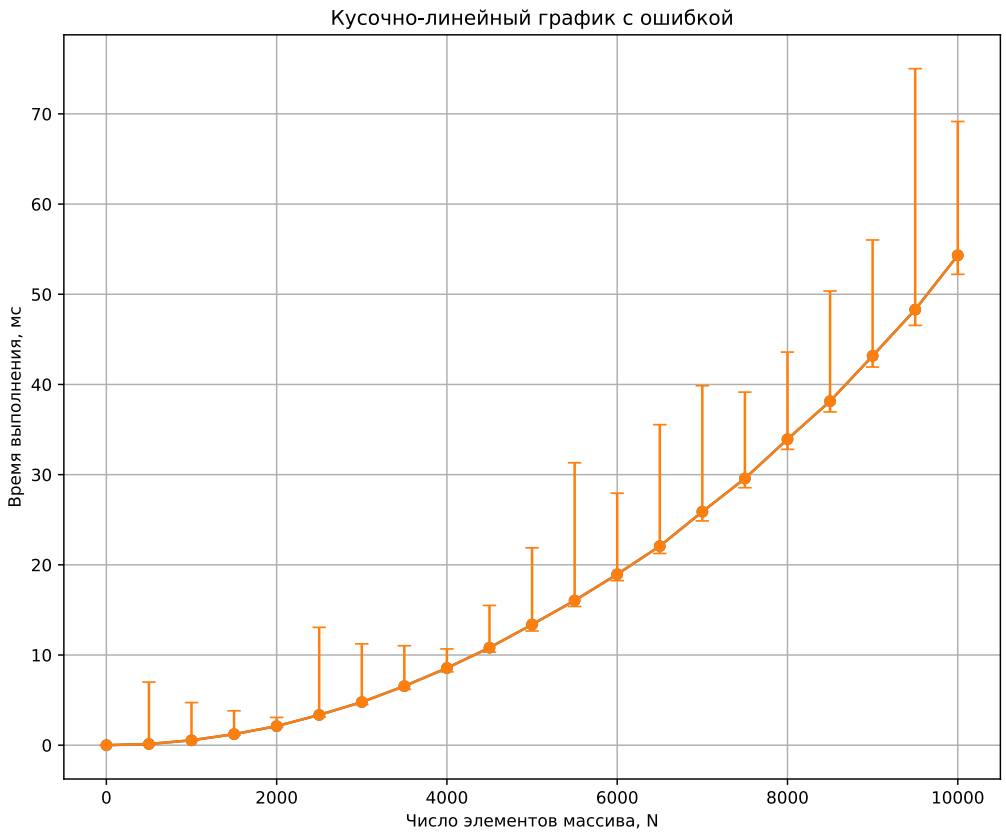
```
…
void insertion_sort(int *arr, size_t n)
{
   for (int *i = arr + 1; i < arr + n; i++)
   {
      int item = *i;
      int *j = i;
      while ((j > arr) && (item < *(j - 1)))
      {
         *j = *(j - 1);
         j--;
      }
      *j = item;
   }
}
…
```

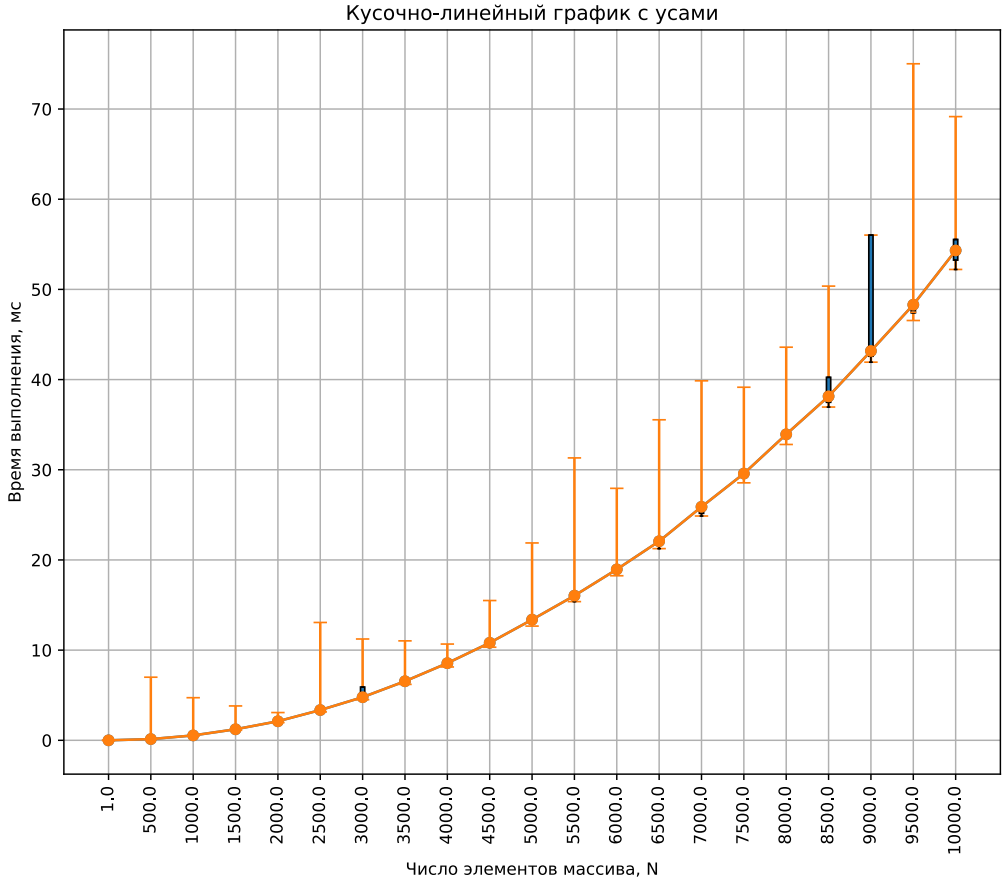
Инфраструктура измерения времени выполнения функции в самой программе

Размер, N	t, мс	Кол-во повторов	RSE, %
1	3.7e-05	1000	0.25
500	0.13	7710	1
1000	0.54	1000	0.91
1500	1.2	1000	0.37
2000	2.1	1000	0.19
2500	3.4	1000	0.37
3000	4.8	1000	0.24
3500	6.6	1000	0.14
4000	8.6	1000	0.11
4500	11	1000	0.11
5000	13	1000	0.17
5500	16	1000	0.16
6000	19	1000	0.11

6500	22	1000	0.12
7000	26	1000	0.12
7500	30	1000	0.087
8000	34	1000	0.083
8500	38	1000	0.093
9000	43	1000	0.078
9500	48	1000	0.11
10000	54	1000	0.11

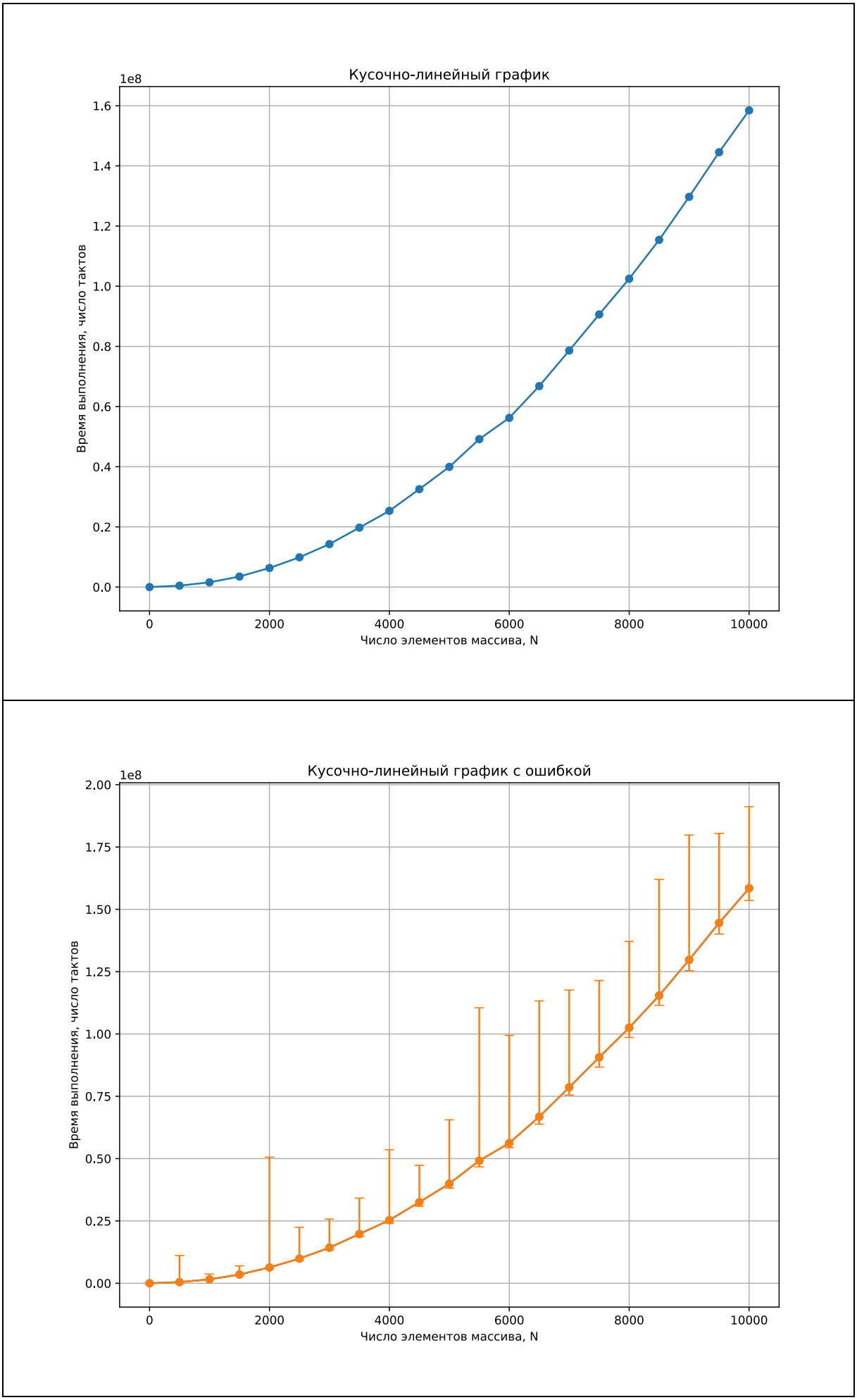


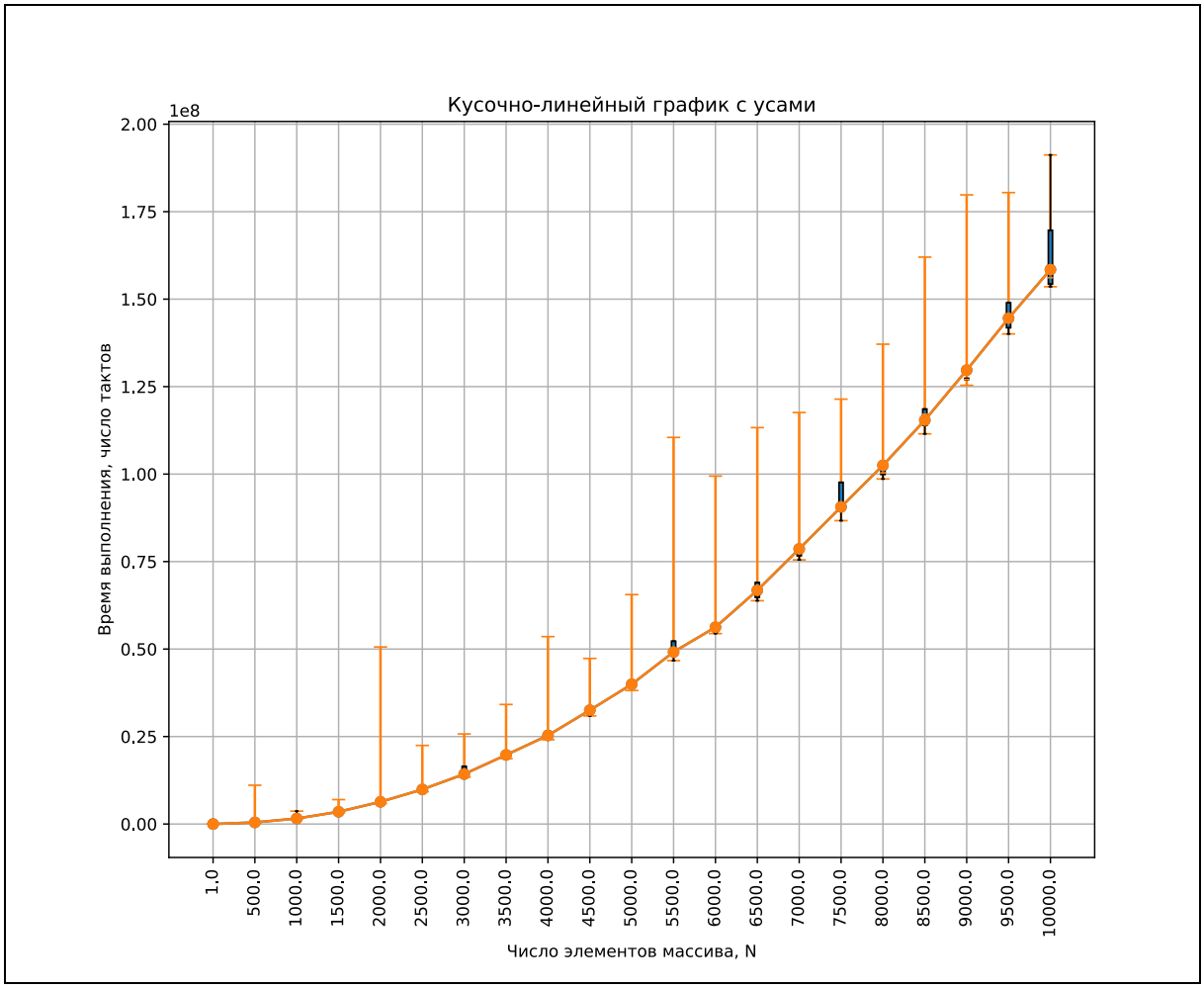

Кусочно-линейный график с ошибкой
Время выполнения, мс
0
10
20
30
40
50
60
70
0
2000
4000
6000
8000
10000
Число элементов массива, N


Кусочно-линейный график с усами
Время выполнения, мс
0
10
20
30
40
50
60
70
1.0
500.0
1000.0
1500.0
2000.0
2500.0
3000.0
3500.0
4000.0
4500.0
5000.0
5500.0
6000.0
6500.0
7000.0
7500.0
8000.0
8500.0
9000.0
9500.0
10000.0
Число элементов массива, N

Инфраструктура измерения времени выполнения функции в самой программе с использованием Time Stamp Counter

Размер, N	t, число тактов	Кол-во повторов	RSE, %
1	42	1000	0.96
500	4.6e+05	2456	1
1000	1.6e+06	1000	0.39
1500	3.5e+06	1000	0.25
2000	6.3e+06	1000	0.75
2500	9.9e+06	1000	0.2
3000	1.4e+07	1000	0.21
3500	2e+07	1000	0.15
4000	2.5e+07	1000	0.16
4500	3.3e+07	1000	0.12
5000	4e+07	1000	0.13
5500	4.9e+07	1000	0.18
6000	5.6e+07	1000	0.11
6500	6.7e+07	1000	0.15
7000	7.9e+07	1000	0.12
7500	9.1e+07	1000	0.098
8000	1e+08	1000	0.099
8500	1.2e+08	1000	0.1
9000	1.3e+08	1000	0.11
9500	1.4e+08	1000	0.077
10000	1.6e+08	1000	0.08

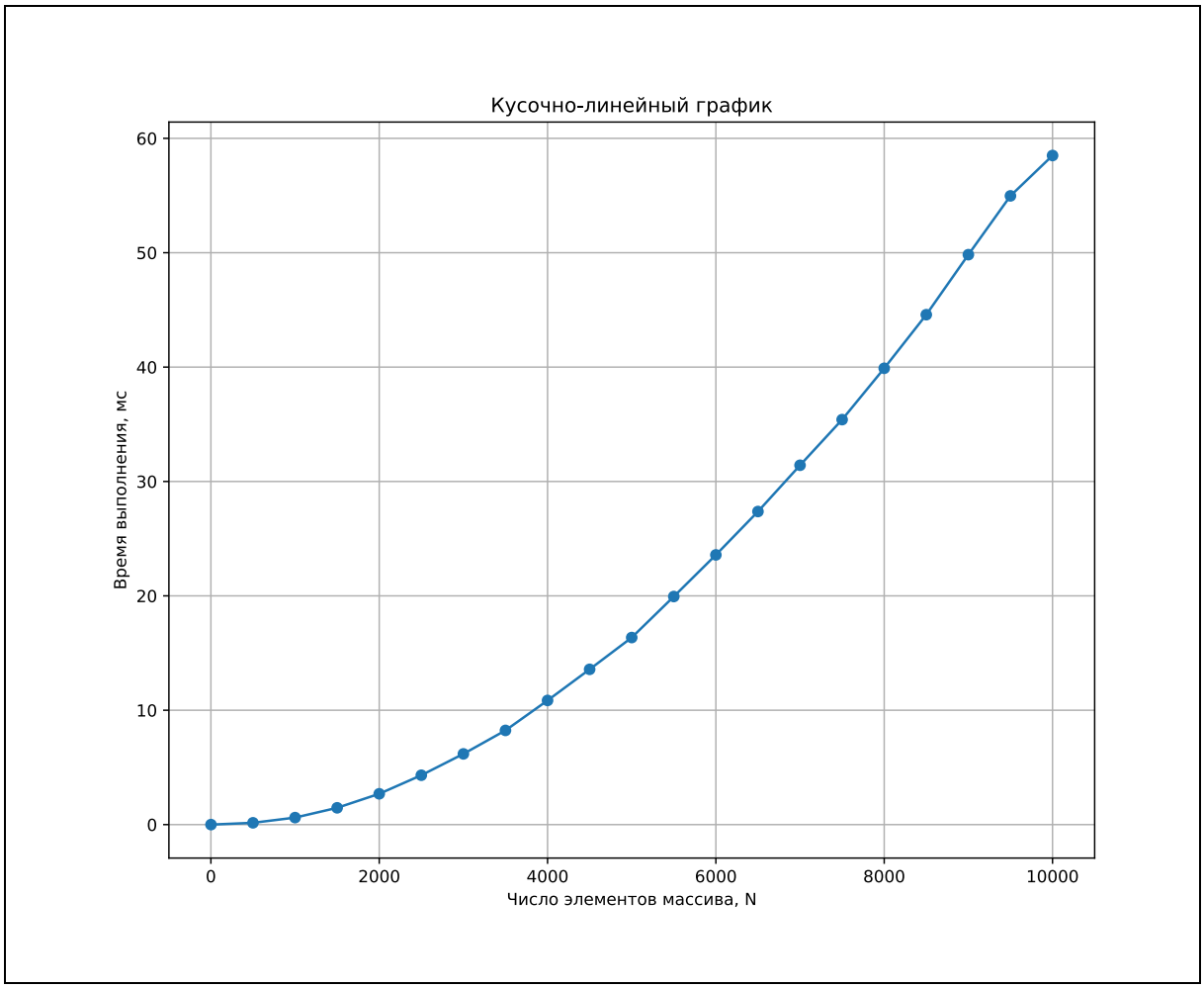

Кусочно-линейный график
1e8
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
Время выполнения, число тактов
0
2000
4000
6000
8000
10000
Число элементов массива, N
Кусочно-линейный график с ошибкой
1e8
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
Время выполнения, число тактов
0
2000
4000
6000
8000
10000
Число элементов массива, N

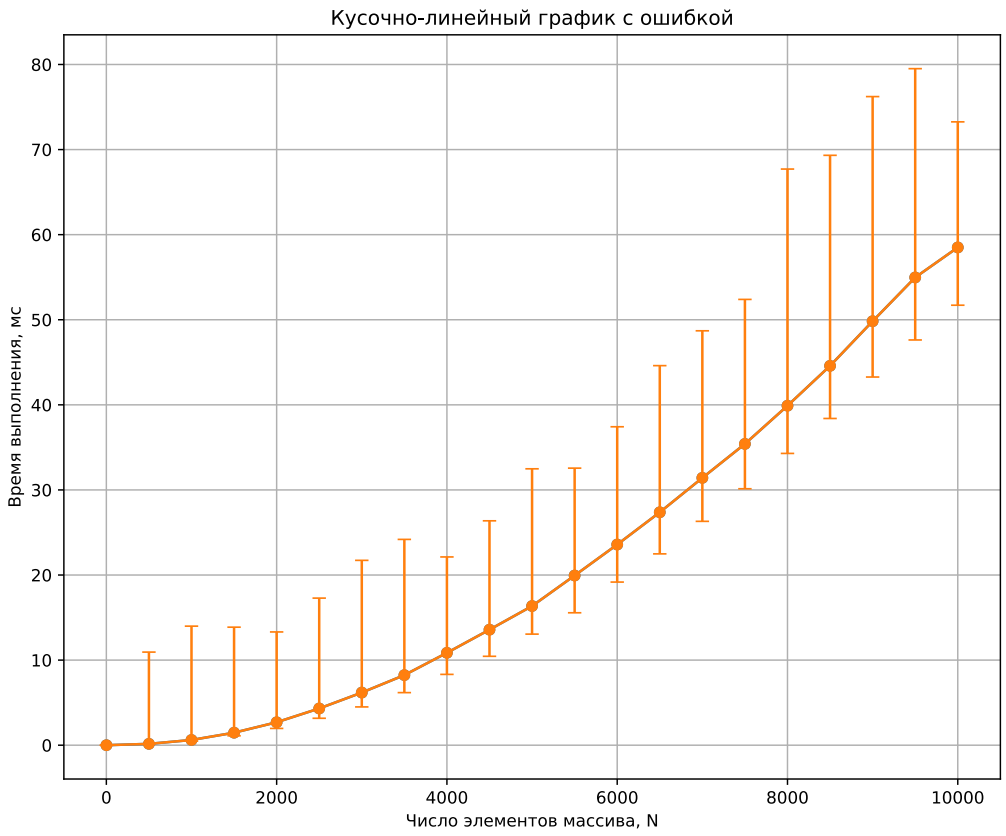


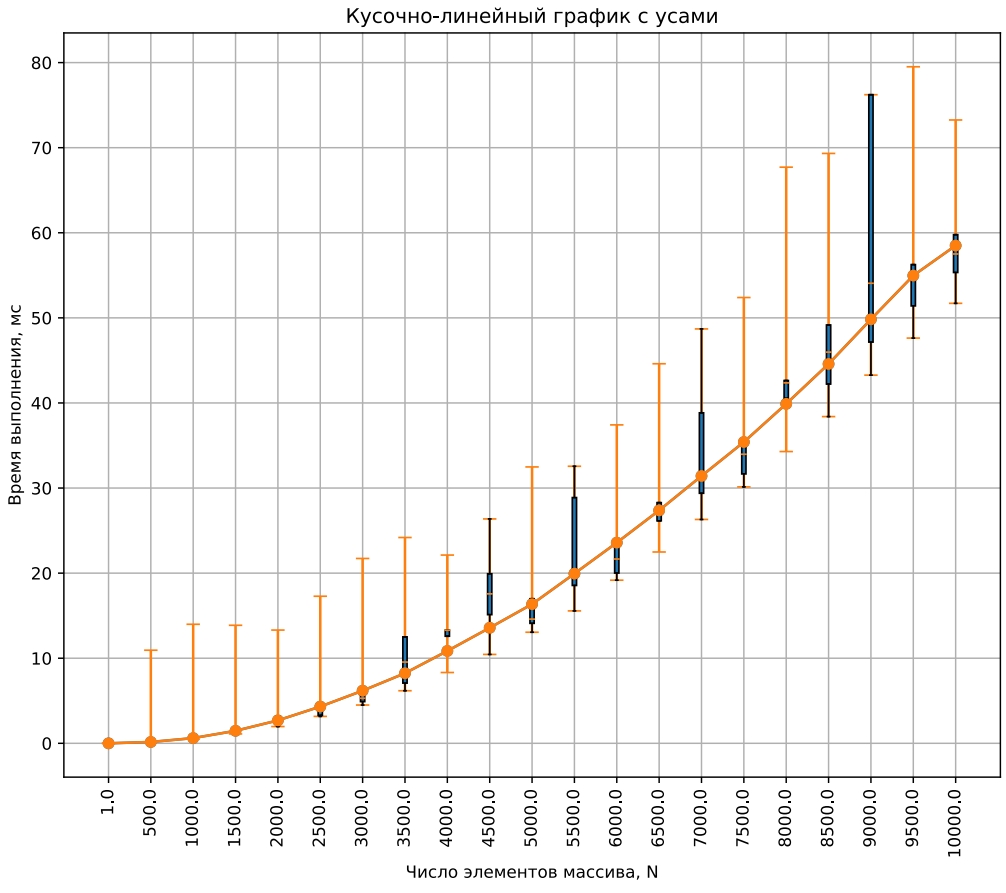
Инфраструктура измерения времени выполнения функции вне программы

Размер, N	t, мс	Кол-во повторов	RSE, %
1	0.00012	2800	1
500	0.16	21100	1
1000	0.61	4500	0.98
1500	1.5	3500	0.98
2000	2.7	2000	1
2500	4.3	1400	0.98
3000	6.2	1000	0.95
3500	8.2	1000	0.79
4000	11	1000	0.65
4500	14	1000	0.6
5000	16	1000	0.55
5500	20	1000	0.45
6000	24	1000	0.4
6500	27	1000	0.37
7000	31	1000	0.34

7500	35	1000	0.28
8000	40	1000	0.27
8500	45	1000	0.25
9000	50	1000	0.24
9500	55	1000	0.23
10000	58	1000	0.17




Кусочно-линейный график с ошибкой
Время выполнения, мс
Число элементов массива, N
0
10
20
30
40
50
60
70
80
0
2000
4000
6000
8000
10000


Кусочно-линейный график с усами
Время выполнения, мс
Число элементов массива, N
0
10
20
30
40
50
60
70
80
1.0
500.0
1000.0
1500.0
2000.0
2500.0
3000.0
3500.0
4000.0
4500.0
5000.0
5500.0
6000.0
6500.0
7000.0
7500.0
8000.0
8500.0
9000.0
9500.0
10000.0

Приложение

1. Листинг build_apps.sh

```
#!/bin/bash

SIZES="1 500 1000 1500 2000 2500 3000 3500 4000 4500 5000 5500 6000 6500 7000 7500
8000 8500 9000 9500 10000"

mkdir -p ./apps

c_files=$(ls ./c_files/*.c)

for c_file in $c_files; do
    filename=$(basename "$c_file" .c)
    mkdir -p "./apps/${filename}"
    for size in $SIZES; do
        gcc -std=c99 -O0 "$c_file" -o "./apps/${filename}/size_${size}.exe" -lm -
DNMAX="$size"
    done
done
```

2. Листинг update_data.sh

```
#!/bin/bash

COUNT=100

if [[ "$1" != "all" ]]; then
    if [ -n "$2" ]; then
        COUNT=$2
    fi

    filepath=$1
    size=$(echo "$filepath" | cut -d'_' -f2)
    app="./apps/${filepath}.exe"
```

```
    for ((i=0; i<COUNT; i+=1)); do
        echo "$size" | "$app" >> ./data/"${filepath}".txt
    done
else
    if [ -n "$2" ]; then
        COUNT=$2
    fi

    dirapps=$(ls ./apps/)
    for dirapp in $dirapps; do
        mkdir -p "./data/${dirapp}/"
        method=$(echo "$dirapp" | cut -d'_' -f1)
        echo "${dirapp} first updating in progress"
        if [[ "$method" == "internal" ]]; then
            for app in $(ls ./apps/"$dirapp"/); do
                nameapp=$(basename "$app" .exe)
                ./apps/"${dirapp}"/"${app}" >> ./data/"${dirapp}"/"${nameapp}".txt
            done
        else
            for app in $(ls ./apps/"$dirapp"/); do
                nameapp=$(basename "$app" .exe)
                for ((i=0; i<COUNT; i+=1)); do
                    ./apps/"${dirapp}"/"${app}" >> ./data/"${dirapp}"/"${nameapp}".txt
                done
            done
        fi
        echo -e "${dirapp} first updating in done\n"
    done
fi
```

3. Листинг make_preproc.py

```
import os

os.makedirs('./prepdata', exist_ok=True)
```

```
for dirs in os.listdir('./data'):
    os.makedirs(f'./prepdata/{dirs}', exist_ok=True)
    for file_name in os.listdir(f'data/{dirs}'):
        with open(f'./data/{dirs}/{file_name}', 'r') as file:
            data = [float(line.strip()) for line in file]

        minimum = data[0]
        maximum = data[0]
        mean = 0
        count = 0
        for i in range(len(data)):
            minimum = min(minimum, data[i])
            maximum = max(maximum, data[i])
            mean += data[i]
            count += 1
        mean /= count
        first_quart = data[count//4]
        second_quart = data[count//2]
        third_quart = data[(3 * count)//4]

        with open(f'./prepdata/{dirs}/{file_name}', 'w') as file:
            file.write(f'Среднее арифметическое:{mean:.6f}\n')
            file.write(f'Минимум:{minimum}\n')
            file.write(f'Максимум:{maximum}\n')
            file.write(f'Первый квартиль:{first_quart}\n')
            file.write(f'Второй квартиль:{second_quart}\n')
            file.write(f'Третий квартиль:{third_quart}\n')
            file.write(f'Количество итераций:{count}\n')
```

4. Листинг make_postproc.py

```
import os
import matplotlib.pyplot as pyplot

os.makedirs("graphs", exist_ok=True)
for dirs in os.listdir('prepdata'):
```

```
    os.makedirs(f'graphs/{dirs}', exist_ok=True)
    time = []
    size = []
    maximum = []
    minimum = []
    first_quart = []
    second_quart = []
    third_quart = []
    for file_name in os.listdir(f'prepdata/{dirs}'):
        with open(f'prepdata/{dirs}/{file_name}', "r") as file:
            cur_size = os.path.basename(file_name).split(".")[0].split("_")[-1]
            size.append(float(cur_size))
            time_line = file.readline().rstrip("\n")
            time.append(float(time_line.split(":")[1]))
            minimum_line = file.readline().rstrip("\n")
            minimum.append(float(minimum_line.split(":")[1]))
            maximum_line = file.readline().rstrip("\n")
            maximum.append(float(maximum_line.split(":")[1]))
            first_quart_line = file.readline().rstrip("\n")
            first_quart.append(float(first_quart_line.split(":")[1]))
            second_quart_line = file.readline().rstrip("\n")
            second_quart.append(float(second_quart_line.split(":")[1]))
            third_quart_line = file.readline().rstrip("\n")
            third_quart.append(float(third_quart_line.split(":")[1]))

    sorted_arrays = sorted(zip(size, time, minimum, maximum, first_quart, second_quart, third_quart))
    size = [size for size, _, _, _, _, _, _ in sorted_arrays]
    time = [time for _, time, _, _, _, _, _ in sorted_arrays]
    err = [[time - err_min for _, time, err_min, _, _, _, _ in sorted_arrays],
           [err_max - time for _, time, _, err_max, _, _, _ in sorted_arrays]]

    data = []
    for i in range(len(sorted_arrays)):
        data.append([sorted_arrays[i][2], sorted_arrays[i][4], sorted_arrays[i][5], sorted_arrays[i][6], sorted_arrays[i][3]])
```

```
    pyplot.subplots(figsize=(10, 8))
    pyplot.plot(size, time, marker='o')
    pyplot.xlabel('Число элементов массива, N')
    if "tsc" in dirs:
        pyplot.ylabel('Время выполнения, число тактов')
    else:
        pyplot.ylabel('Время выполнения, мс')
    pyplot.title('Кусочно-линейный график')
    pyplot.grid(True)
    pyplot.savefig(f'graphs/{dirs}/segmented.svg', format='svg')

    pyplot.title('Кусочно-линейный график с ошибкой')
    pyplot.errorbar(size, time, yerr=err, fmt='-o', capsize=4)
    pyplot.savefig(f'graphs/{dirs}/error.svg', format='svg')

    if (len(size) != 0):
        pyplot.boxplot(data, positions=size, patch_artist=True, showfliers=False, widths=50)
    pyplot.title('Кусочно-линейный график с усами')
    pyplot.xticks(rotation=90)
    pyplot.savefig(f'graphs/{dirs}/moustache.svg', format='svg')
    pyplot.cla()
```

5. Листинг go.sh

```
#!/bin/bash

MAX_ITER=100000
MIN_ITER=1000

COUNT=100

echo "build apps is creating"
bash ./build_apps.sh
echo -e "creating is done\n"
```

```
bash ./update_data.sh "all" $COUNT

dirs=$(ls ./data/)
for dir in $dirs; do
    files=$(ls ./data/"$dir"/)
    method=$(echo "$dir" | cut -d'_' -f1)
    if [[ "$method" != "internal" ]]; then
        echo "${dir} is updating"
        for file in $files; do
            echo "${file} is updating"
            n=$(basename "$file" | cut -d'_' -f2 | cut -d'.' -f1)
            rse=$(echo "./data/$dir/$file" | python3 get_rse.py)
            cnt=$COUNT
            while (( ($(echo "$rse > 1" | bc -l) || cnt < MIN_ITER) && cnt < MAX_ITER)); do
                bash ./update_data.sh "${dir}/size_${n}" $COUNT
                cnt=$((cnt+COUNT))
                rse=$(echo "./data/$dir/$file" | python3 get_rse.py)
            done
            echo -e "updating is done\n"
        done
    fi
done

echo "datas is preprocessing"
python3 make_preproc.py
echo -e "preprocessing is done\n"

echo "datas is postprocessing"
python3 make_postproc.py
echo -e "postprocessing is done\n"
```

6. Листинг get_rse.py

```
file_path = input()
with open(file_path, 'r') as file:
    times = [float(line.strip()) for line in file]
```

```
size = int(file_path.split("_")[-1].split(".")[0])
n = len(times)

avg = sum(times) / n

s_sq = 0
for i in range(n):
    s_sq += (times[i] - avg)**2
s_sq /= (n - 1)
s = s_sq ** 0.5

std_err = s / (n ** 0.5)
rse = std_err / avg * 100

print(rse)
```

7. Листинг table_rse.py

```
import os

directory_path = "./data"
for root, dirs, files in os.walk(directory_path):
    datas = []
    print('\n', root)
    for file_name in files:
        file_path = os.path.join(root, file_name)
        with open(file_path, 'r') as file:
            times = [float(line.strip()) for line in file]

        size = int(file_path.split("_")[-1].split(".")[0])
        n = len(times)

        avg = sum(times) / n

        s_sq = 0
```

```
    for i in range(n):
        s_sq += (times[i] - avg)**2
    s_sq /= (n - 1)
    s = s_sq ** 0.5

    std_err = s / (n ** 0.5)
    rse = std_err / avg * 100

    datas.append([size, f"{size:<15}{avg:<15.2g}{n:<15}{rse:<15.2g}"])

for _, line in sorted(datas):
    print(line)
```