Задание №4 в рамках вычислительного практикума. Исследование характеристик программного обеспечения

Описание дерева исходного кода/скриптов/данных

├── apps – Содержит папки с исполняемыми файлами для разных видов замеров времени
│   ├── index_inside
│   ├── index_outside
│   ├── index_ticks_inside
│   ├── index_ticks_outside
│   ├── index_to_pointer_inside
│   ├── index_to_pointer_outside
│   ├── index_to_pointer_ticks_inside
│   ├── index_to_pointer_ticks_outside
│   ├── pointers_inside
│   ├── pointers_outside
│   ├── pointers_ticks_inside
│   └── pointers_ticks_outside
├── data Содержи т папки с необработанными данными о времени выполнения программ
│   ├── index_inside
│   ├── index_outside
│   ├── index_ticks_inside
│   ├── index_ticks_outside
│   ├── index_to_pointer_inside
│   ├── index_to_pointer_outside
│   ├── index_to_pointer_ticks_inside
│   ├── index_to_pointer_ticks_outside
│   ├── pointers_inside
│   ├── pointers_outside
│   ├── pointers_ticks_inside
│   └── pointers_ticks_outside
│

├── plots – папка для хранения векторных графиков со статистикой работы программ

├── prepdata Содержи т папки с обработанными данными о времени выполнения программ

│   ├── index_inside

│   ├── index_outside

│   ├── index_ticks_inside

│   ├── index_ticks_outside

│   ├── index_to_pointer_inside

│   ├── index_to_pointer_outside

│   ├── index_to_pointer_ticks_inside

│   ├── index_to_pointer_ticks_outside

│   ├── pointers_inside

│   ├── pointers_outside

│   ├── pointers_ticks_inside

│   └── pointers_ticks_outside

├── programs - папка с исходными кодами программ на языке C

└── scripts – Папка с исходными кодами программ на языке bash и python

    └── make_plots – Папка с bash скриптами, которые строят графики на основе обработанных данных

Список ПО, которое должно быть установлено:

- Интерпретатор Python3
- Компилятор gcc
- gnuplot
- bash

Инструкция по запуску:

Из любой папки запустить скрипт go.sh. Пример: ./scripts/go.sh

Листинги скриптов:

Скрипт **build_apps.sh** предназначен для автоматической компиляции C-программ с различными значениями макроса NMAX (количество элементов в массиве), обеспечивая создание отдельных исполняемых файлов для каждого значения.

```
#!/bin/bash

SCRIPT_DIR=$(dirname "$(readlink -f "$0")")

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

echo "Build process started..."

# Проверяю существование папки с программами
if [ ! -d "$SCRIPT_DIR/../programms" ]; then
   echo "Error: Programms directory not found."
   exit 1
fi

# Проверяю существование папки с исполняемыми файлами
if [ ! -d "$SCRIPT_DIR/../apps" ]; then
   mkdir "$SCRIPT_DIR/../apps"
fi

# Обходим каждую программу с расширением .c (разные сортировки)
for programm in "$SCRIPT_DIR"/../programms/*.c; do
   prog_name=$(basename "$programm")

   # Проверяю существование файла .c
   if [ ! -f "$programm" ]; then
      echo "Error: Source file $programm not found."
      exit 1
   fi

   if [ -d "$SCRIPT_DIR/../apps/${prog_name%.c}/" ]; then
      rm -f "$SCRIPT_DIR/../apps/${prog_name%.c}/"*.exe
   else
      mkdir "$SCRIPT_DIR/../apps/${prog_name%.c}/"
   fi

   # Обходим все знаечния NMAX (кол-во элементов в массиве)
   for size in $NMAX; do
      echo "Building app $prog_name, size = $size"
      gcc -std=gnu99 -Wall -Werror -Wextra -Wpedantic -o
"$SCRIPT_DIR/../apps/${prog_name%.c}/$size.exe" "$programm" -DNMAX="$size" -lm
   done
   echo
done

echo "Build process completed successfully."

exit 0
```

Листинг1

Скрипт **update_data.sh** автоматизирует процесс сбора и обновления данных для программ. Он запускает все исполняемые файлы, которые хранятся в папке apps и записывает вывод в соответствующую папку data до тех пор, пока rse собранных данных меньше 1% и количество замеров не более 1000

```
#!/bin/bash

SCRIPT_DIR=$(dirname "$(readlink -f "$0")")

echo
echo "Updating data..."
echo

# Обходим каждую программу с расширением .c (разные сортировки)
for programm in "$SCRIPT_DIR"/../programms/*.c; do
  prog_name=$(basename "$programm")
  if [ ! -d "$SCRIPT_DIR/../data/${prog_name%.c}/" ]; then
    mkdir "$SCRIPT_DIR/../data/${prog_name%.c}/"
  fi
  echo
  # Обходим каждый исполняемый файл
  for exe in "$SCRIPT_DIR/../apps/${prog_name%.c}"/*.exe; do
    exe_name=$(basename "$exe" .exe)
    data_file="$SCRIPT_DIR/../data/${prog_name%.c}/${exe_name}.txt"
    echo "Updating data for file: ${prog_name%.c}/$exe_name.txt"
    "$exe" >> "$data_file"

    # Проверка на наличие слова "inside" в имени файла
    if [[ "$data_file" == *"inside"* ]]; then
      continue
    fi
    # Инициализация счетчика итераций
    count=1
    max_iterations=1000
    min_iterations=10

    while (( count < max_iterations )); do
      # Увеличение счетчика итераций и запуск исполняемого файла с записью в
текстовый файл
      count=$((count + 1))
      "$exe" >> "$data_file"

      # Проверка на минимальное количество итераций и вычисление rse_value
      if (( count >= min_iterations )); then
        rse_value=$(python3 "$SCRIPT_DIR"/calc_rse.py "$data_file")
        #если rse < 1.0, прекращаем цикл записи данных
        if (( $(echo "$rse_value < 1.0" | bc -l) )); then
          break
        fi
      fi

      # Проверка на максимальное количество итераций
      if (( count >= max_iterations )); then
        break
      fi
    done
  done
```

```
done

echo
echo "Data was updated successfully"
echo

exit 0
```

Листинг 2

Скрипт **calc_rse.py** считает rse для файла, который передается аргументом командной строки.

```
import sys
import math

def calc_rse(data_file_path):
    with open(data_file_path, 'r') as file:
        data = [int(num) for num in file]
    avg = sum(data) / len(data)
    s_sq = sum((num - avg) ** 2 for num in data) / (len(data) - 1)
    s = math.sqrt(s_sq)
    std_err = s / math.sqrt(len(data))
    rse = (std_err / avg) * 100

    return rse

if __name__ == "__main__":
    data_file_path = sys.argv[1]
    rse_value = calc_rse(data_file_path)
    print(rse_value)
```

Листинг 3

Скрипт **make_preproc.sh** создает папку и файл data.txt, в который скрипт **preproc.py** запишет обработанные данные (Количество элементов массива, мин, макс, среднее, квартили)

```
#!/bin/bash

SCRIPT_DIR=$(dirname "$(readlink -f "$0")")

echo
echo "Preprocessing data..."

# Обходим каждую программу с расширением .c (разные сортировки)
for program in "$SCRIPT_DIR"/../programms/*.c; do
    prog_name=$(basename "$program" .c)
    # Создаю папку prepdata, если её ещё нет
    if [ ! -d "$SCRIPT_DIR/../prepdata/${prog_name}" ]; then
        mkdir "$SCRIPT_DIR/../prepdata/${prog_name}"
    fi

    # Удаляю файл data.txt, если он существует
    if [ -f "$SCRIPT_DIR/../prepdata/${prog_name}/data.txt" ]; then
        rm "$SCRIPT_DIR/../prepdata/${prog_name}/data.txt"
```

```
  fi

  # Создаю файл data.txt в каждой папке prepdata
  touch "$SCRIPT_DIR/../prepdata/${prog_name}/data.txt"

  # Запускаю python скрипт, которые обрабатывает данные из файла и записывает
статистику в папку prepdata в файл с соответствующим названием
  python3 "$SCRIPT_DIR"/preproc.py "$SCRIPT_DIR/../data/${prog_name}/"
"$SCRIPT_DIR/../prepdata/${prog_name}/data.txt"
done

echo
echo "Data was preprocessed successfully"
echo

exit 0
```

Листинг 4

Скрипт **preproc.py** обрабатывает данные из входного текстового файла и записывает статистику по этим данным в выходной файл

```
import sys
import statistics
import os

data_file_path = sys.argv[1]
target_file_path = sys.argv[2]

file_list = os.listdir(os.path.dirname(data_file_path))
file_list.sort(key=lambda x: int(x.split('.')[0]))

with open(target_file_path, 'a') as file:
  for txt_file in file_list:
    size = os.path.splitext(txt_file)[0]
    with open(os.path.join(os.path.dirname(data_file_path), txt_file), 'r') as f:
      content = f.read().splitlines()
    arr = [int(line) for line in content]
    file.write(f"{size} ")
    file.write(f"{min(arr)} ")
    file.write(f"{statistics.quantiles(arr, n=4)[0]} ")
    file.write(f"{statistics.median(arr)} ")
    file.write(f"{statistics.quantiles(arr, n=4)[-1]} ")
```

```
        file.write(f"{max(arr)} ")
        file.write(f"{statistics.mean(arr):.2f}\n")
```

Листинг 5

Скрипт **postproc.sh** запускает скрипты, которые строят различные графики зависимостей количества элементов и времени выполнения программы.

```
#!/bin/bash

echo
echo "Postprocessing data"
echo

# Получаем адрес текущей папки
SCRIPT_DIR=$(dirname "$(readlink -f "$0")")

# Переходим в папку с скриптами для построения графиков
cd "$SCRIPT_DIR/make_plots" || exit

# Запускаем все .sh файлы в этой папке
for script in *.sh; do
  ./"$script"
done

echo
echo "Data was postprocessed successfully"
echo
echo "Congratulations!!!!"
echo

exit 0
```

Листинг 6

Скрипт **go.sh** запускает все необходимые для проведения эксперимента скрипты

```
#!/bin/bash
SCRIPT_DIR=$(dirname "$(readlink -f "$0")")

bash "$SCRIPT_DIR"/build_apps.sh
bash "$SCRIPT_DIR"/update_data.sh
bash "$SCRIPT_DIR"/make_preproc.sh
bash "$SCRIPT_DIR"/make_postproc.sh
```

Листинг 7

Скрипт **clean_data.sh** очищает текстовые файлы из папок data и prepdata

```
#!/bin/bash

SCRIPT_DIR=$(dirname "$(readlink -f "$0")")

echo
echo "Deleting all texts files..."
echo

# Удаляю все txt файлы из папки с подготовленными данными о времени сортировок
if [ -d "$SCRIPT_DIR/../prepdata" ]; then
   for app_dir in "$SCRIPT_DIR/../prepdata"/*/; do
      rm -f "$app_dir"/*.txt
   done
fi

# Удаляю все txt файлы из папки с неподготовленными данными о времени сортировок
for app_dir in "$SCRIPT_DIR/../data"/*/; do
   rm -f "$app_dir"/*.txt
done

# Удаляю тектсовые файлы, которые были нужны для замеров времени внутри
rm -f "$SCRIPT_DIR"/*.txt

echo
```

```
echo "All texts files were deleted successfully."
echo


exit 0
```

Листинг 8

Скрипт **linear_time_inside.sh** строит кусочно-линейный график зависимости времени выполнения от количества элементов массива

```
#!/usr/bin/gnuplot -persist

reset
NO_ANIMATION = 1

set terminal svg size 1920, 1080
set output '../../plots/linear_time_inside.svg'

set xlabel "Количество элементов, шт"
set ylabel "Время, мкс"
set grid
set title "Линейный график для измерений времени внутри"

plot "../../prepdata/index_inside/data.txt" using 1:7 with linespoints pt 9 lt rgb "red" title "Indix inside",\
    "../../prepdata/index_to_pointer_inside/data.txt" using 1:7 with linespoints pt 8 lt rgb "blue" title "Indix to pointer inside",\
    "../../prepdata/pointers_inside/data.txt" using 1:7 with linespoints pt 7 lt rgb "green" title "Pointers Inside"
```

Листинг 9

Скрипт **moustashe_index_inside.sh строит график с усами** (среднее, максимум, минимум; нижний, средний и верхний квартили). Зависимость времени выполнения от количества элементов массива

```
#!/usr/bin/gnuplot -persist

reset
NO_ANIMATION = 1

set terminal svg size 1920, 1080
set output '../../plots/moustashe_index_inside.svg'

set xlabel "Количество элементов, шт"
set ylabel "Время, мкс"
set grid
set title "График с усами для измерений времени внутри. Индексы"

plot "../../prepdata/index_inside/data.txt" using 1:3:2:6:5 with candlesticks notitle whiskerbars,\
    "../../prepdata/index_inside/data.txt" using 1:7 with linespoints title "Index inside",\
    "../../prepdata/index_inside/data.txt" using 1:7:7:7:7 with candlesticks lt -1 notitle
```

Листинг 10

Скрипт **errors_bar_inside.sh** строит кусочно-линейный график с ошибкой (среднее, максимум, минимум).

```
#!/usr/bin/gnuplot -persist

reset
NO_ANIMATION = 1

set terminal svg size 1920, 1080 enhanced
set output '../../plots/error_bar_inside.svg'

set xlabel "Количество элементов, шт"
set ylabel "Время, мкс"
set grid
set title "График error-bar измерения времени внутри программы"

plot "../../prepdata/index_inside/data.txt" using 1:7:6:2 with yerrorbars notitle,\
    "../../prepdata/index_inside/data.txt" using 1:7 with lines lt rgb "red" title "Index Inside",\
    "../../prepdata/index_to_pointer_inside/data.txt" using 1:7:6:2 with yerrorbars notitle,\
    "../../prepdata/index_to_pointer_inside/data.txt" using 1:7 with lines lt rgb "blue" title
"Index to Pointer Inside",\
    "../../prepdata/pointers_inside/data.txt" using 1:7:6:2 with yerrorbars notitle,\
    "../../prepdata/pointers_inside/data.txt" using 1:7 with lines lt rgb "green" title "Pointers
Inside"
```

Листинг 11

Описание исходных программ на языке C:

Индексация вида a[i]:

1) Программа index_inside.c выполняет сортировку выбором и собирает данные о времени работы программы до тех пор, пока rse > 1.0 или количество замеров меньше 1000.

2) Программа index_outside.c выполняет сортировку выбором и выводит время её выполнения в консоль.

3) Программа index_ticks_inside.c выполняет сортировку выбором и собирает данные о количестве совершенных тиков процессора при выполнении программы до тех пор, пока rse > 1.0 или количество замеров меньше 1000.

4) Программа index_ticks_outside.c выполняет сортировку выбором и выводит количество совершенных тиков процессора в консоль

Замена индексации a[i] на *(a + i):

1) Программа index_to_pointer_inside.c выполняет сортировку выбором и собирает данные о времени работы программы до тех пор, пока rse > 1.0 или количество замеров меньше 1000.

2) Программа index_to_pointer_outside.c выполняет сортировку выбором и выводит время её выполнения в консоль.

3) Программа index_to_pointer_ticks_inside.c выполняет сортировку выбором и собирает данные о количестве совершенных тиков процессора при выполнении программы до тех пор, пока rse > 1.0 или количество замеров меньше 1000.

4) Программа index_to_pointer_ticks_outside.c выполняет сортировку выбором и выводит количество совершенных тиков процессора в консоль

Использование указателей:

1) Программа pointers_inside.c выполняет сортировку выбором и собирает данные о времени работы программы до тех пор, пока rse > 1.0 или количество замеров меньше 1000.

2) Программа pointers_outside.c выполняет сортировку выбором и выводит время её выполнения в консоль.

3) Программа pointers_ticks_inside.c выполняет сортировку выбором и собирает данные о количестве совершенных тиков процессора при выполнении программы до тех пор, пока rse > 1.0 или количество замеров меньше 1000.

4) Программа pointers_ticks_outside.c выполняет сортировку выбором и выводит количество совершенных тиков процессора в консоль

Таблицы с результатами:

Использование операции индексации a[i]

Измерения времени выполнения функции сортировки выбором вне программы.

Размер	t, мкс	Кол-во повторов	RSE, %
1000	375	797	0.99
2000	1514	328	0.99
3000	3348	137	0.99
4000	5950	37	0.99
5000	9290	15	0.99
6000	13130	10	0.74
7000	18198	10	0.73
8000	23466	12	0.94
9000	30111	10	0.43
10000	36567	11	0.99

Измерения выполнения функции сортировки выбором внутри программы.

Размер	t, мкс	Кол-во повторов	RSE, %
1000	350	1000	1.19
2000	1267	455	0.99
3000	2853	153	0.99
4000	5080	84	0.99
5000	7957	18	0.99
6000	11280	37	0.99
7000	15340	10	0.62
8000	20142	10	0.49
9000	25799	10	0.44
10000	31746	10	0.7

Измерения количества тиков процессора при выполнении функции сортировки выбором внутри программы.

Размер	Тики, tcs	Кол-во повторов	RSE, %
1000	811001	33	0.97
2000	2865146	730	0.99
3000	6727579	255	0.99
4000	11857864	143	0.99
5000	18683815	63	0.99
6000	27927496	35	0.99
7000	36863850	31	0.99
8000	48931893	16	0.96
9000	62816315	10	0.92
10000	77904215	10	0.63

Измерения количества тиков процессора при выполнении функции сортировки выбором снаружи программы.

Размер	Тики, tcs	Кол-во повторов	RSE, %
1000	957522	924	0.99
2000	3838375	10	0.91
3000	8511158	10	0.41
4000	15380649	21	0.98
5000	23485878	10	0.95
6000	32506901	10	0.98
7000	45533778	10	0.99
8000	57876773	10	0.82
9000	73787108	10	0.89
10000	91149583	10	0.67

Использование операции индексации замены индексации *(a + i)

Измерения времени выполнения функции сортировки выбором вне программы.

Размер	t, мкс	Кол-во повторов	RSE, %
1000	377	1000	0.99
2000	1482	406	0.99
3000	3390	95	0.99
4000	5850	41	0.99
5000	9264	20	0.99
6000	13263	14	0.95
7000	18032	14	0.94
8000	23318	10	0.69
9000	29532	10	0.95
10000	36605	10	0.96

Измерения выполнения функции сортировки выбором внутри программы.

Размер	t, мкс	Кол-во повторов	RSE, %
1000	337	14	0.99
2000	1303	469	0.99
3000	2790	154	0.99
4000	5012	91	0.99
5000	7843	39	0.99
6000	11583	22	0.99
7000	15427	26	0.98
8000	20592	10	0.85
9000	25628	10	0.35
10000	31446	10	0.96

Измерения количества тиков процессора при выполнении функции сортировки выбором внутри программы.

Размер	t, мкс	Кол-во повторов	RSE, %
1000	790164	1000	1.24
2000	2755459	419	0.99
3000	6359525	368	0.99
4000	11869357	152	0.99
5000	18922293	60	0.98
6000	27357808	15	0.94
7000	36595087	18	0.95
8000	48432729	18	0.99
9000	61514283	10	0.73
10000	74379632	10	0.95

Измерения количества тиков процессора при выполнении функции сортировки выбором снаружи программы.

Размер	t, мкс	Кол-во повторов	RSE, %
1000	949354	1000	1.14
2000	3724652	467	0.99
3000	8340518	128	0.99
4000	14852269	45	0.99
5000	22772186	25	0.97
6000	33532659	26	0.97
7000	44659845	12	0.99
8000	58481177	10	0.94
9000	75490850	10	0.96
10000	91604125	10	0.83

Использование указателей

Измерения времени выполнения функции сортировки выбором вне программы.

Размер	t, мкс	Кол-во повторов	RSE, %
1000	420	1000	1.10
2000	1631	418	0.99
3000	3589	80	0.98
4000	6168	44	0.98
5000	9657	25	0.97
6000	13802	12	0.99
7000	18592	10	0.87
8000	24675	10	0.95
9000	30622	10	0.52
10000	37802	10	0.64

Измерения выполнения функции сортировки выбором внутри программы.

Размер	t, мкс	Кол-во повторов	RSE, %
1000	390	1000	1.15
2000	1468	510	0.99
3000	3280	135	0.99
4000	6229	35	0.99
5000	9700	11	0.98
6000	13835	34	0.98
7000	18808	25	0.97
8000	24889	11	0.97
9000	31020	10	0.40
10000	38852	10	0.39

Измерения количества тиков процессора при выполнении функции сортировки выбором внутри программы.

Размер	t, мкс	Кол-во повторов	RSE, %
1000	992888	17	0.97
2000	3511178	625	0.99
3000	7812468	17	0.99
4000	14482362	15	0.99
5000	22497955	49	0.98
6000	32828595	13	0.97
7000	45619856	14	0.95
8000	60071329	16	0.98
9000	77195763	10	0.68
10000	95771688	10	0.92

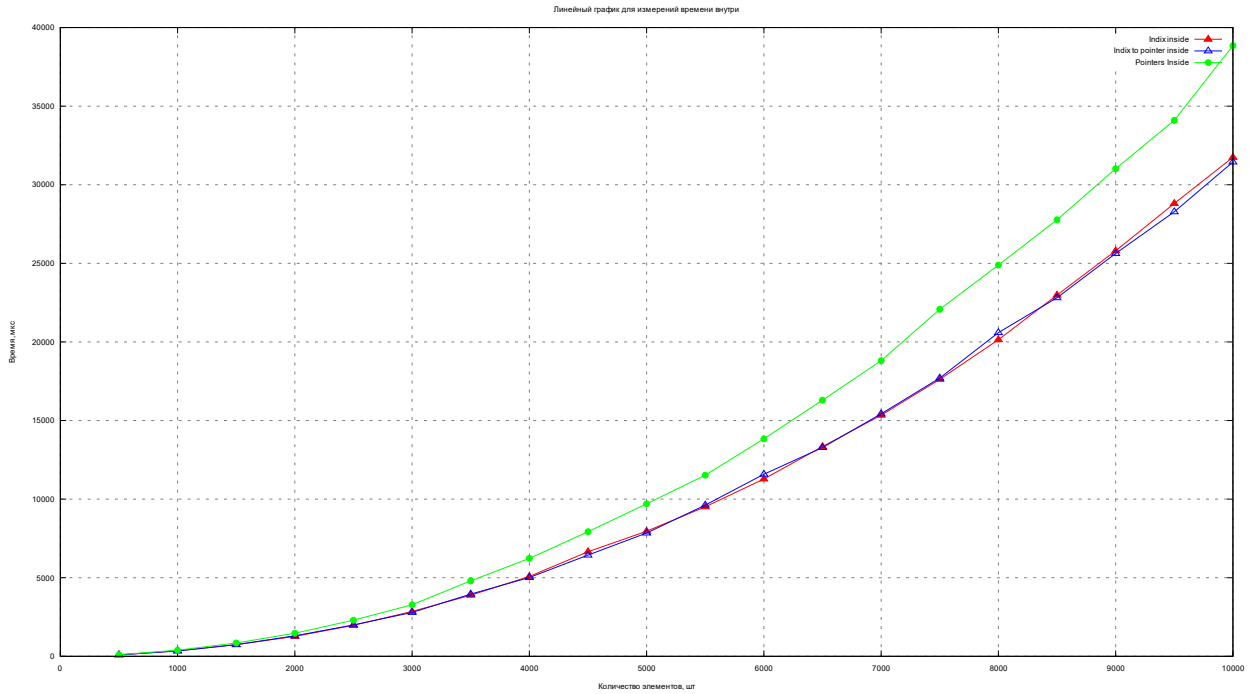
Измерения количества тиков процессора при выполнении функции сортировки выбором снаружи программы.

Размер	t, мкс	Кол-во повторов	RSE, %
1000	933411	1000	1.08
2000	3867334	419	0.99
3000	8415319	137	0.00
4000	14923777	69	0.08
5000	23556675	12	0.94
6000	33894159	10	0.90
7000	45532341	14	0.98
8000	60171778	14	0.98
9000	75339540	10	0.66
10000	93077787	10	0.78

Кусочно-линейные графики

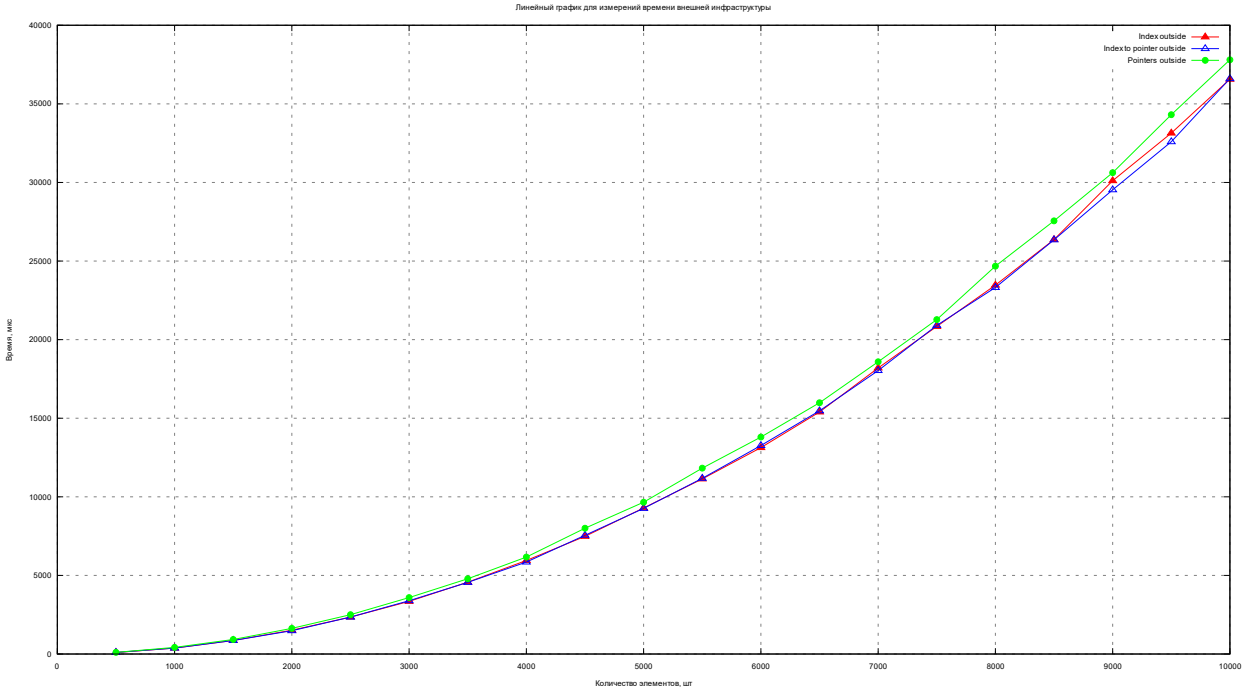
Измерения внутри:

Кусочно-линейный график зависимости времени выполнения в микросекундах от числа элементов массива.



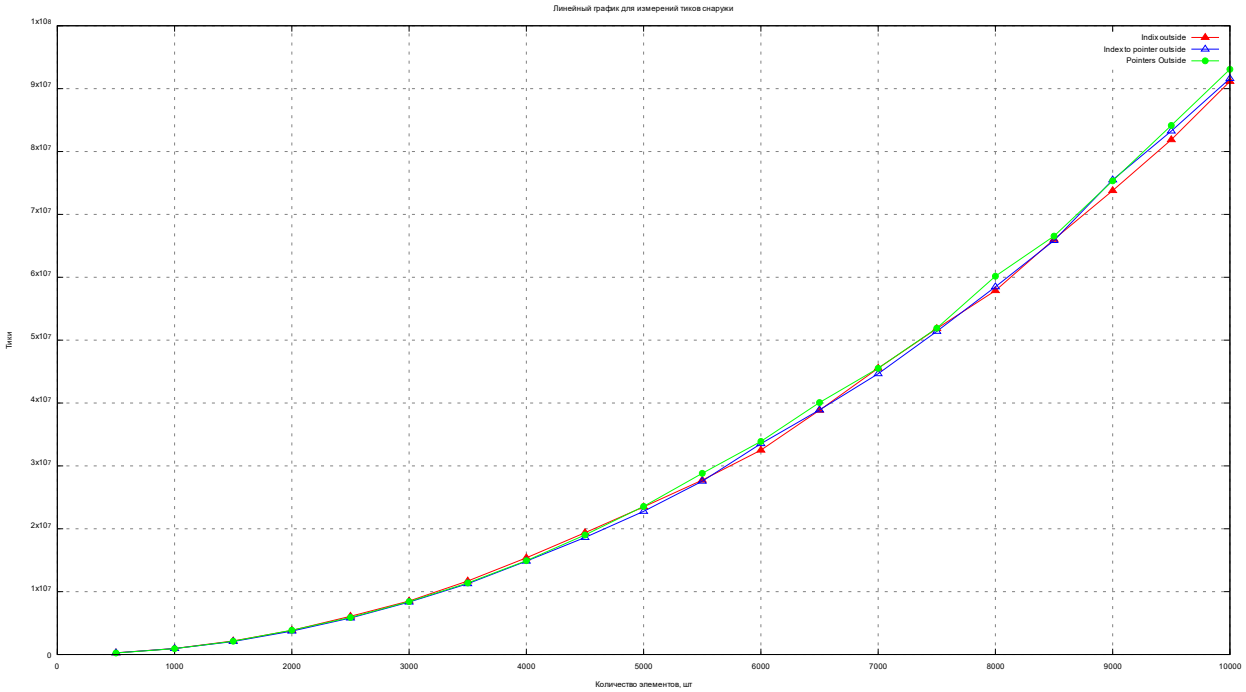
Измерения снаружи:

Кусочно-линейный график зависимости времени выполнения в микросекундах от числа элементов массива.



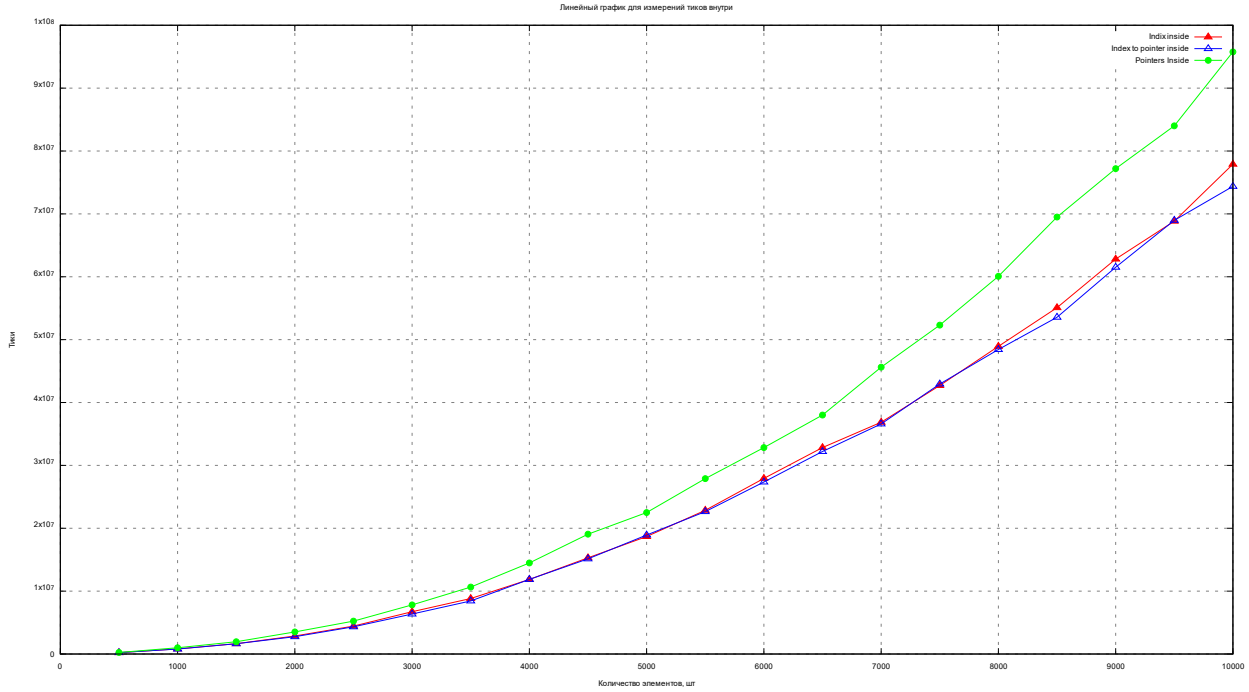
Измерения снаружи:

Кусочно-линейный график зависимости количества тиков процессор от числа элементов массива



Измерения внутри:

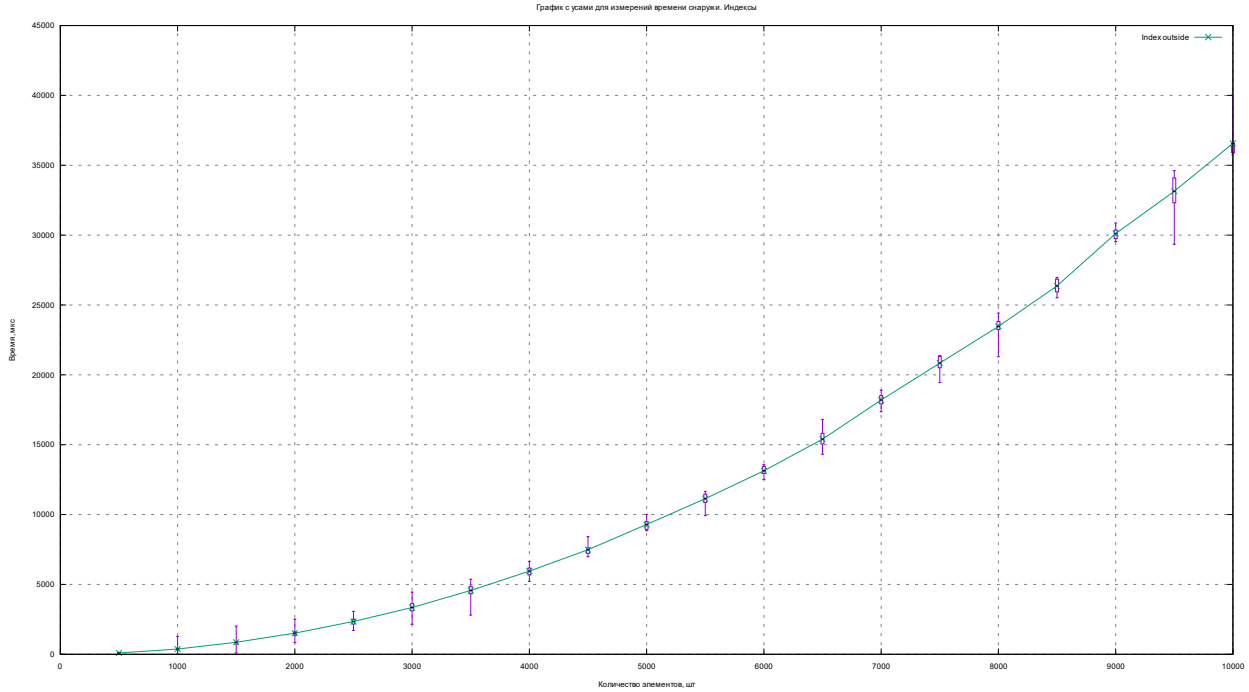
Кусочно-линейный график зависимости количества тиков процессор от числа элементов массива



Графики с усами.

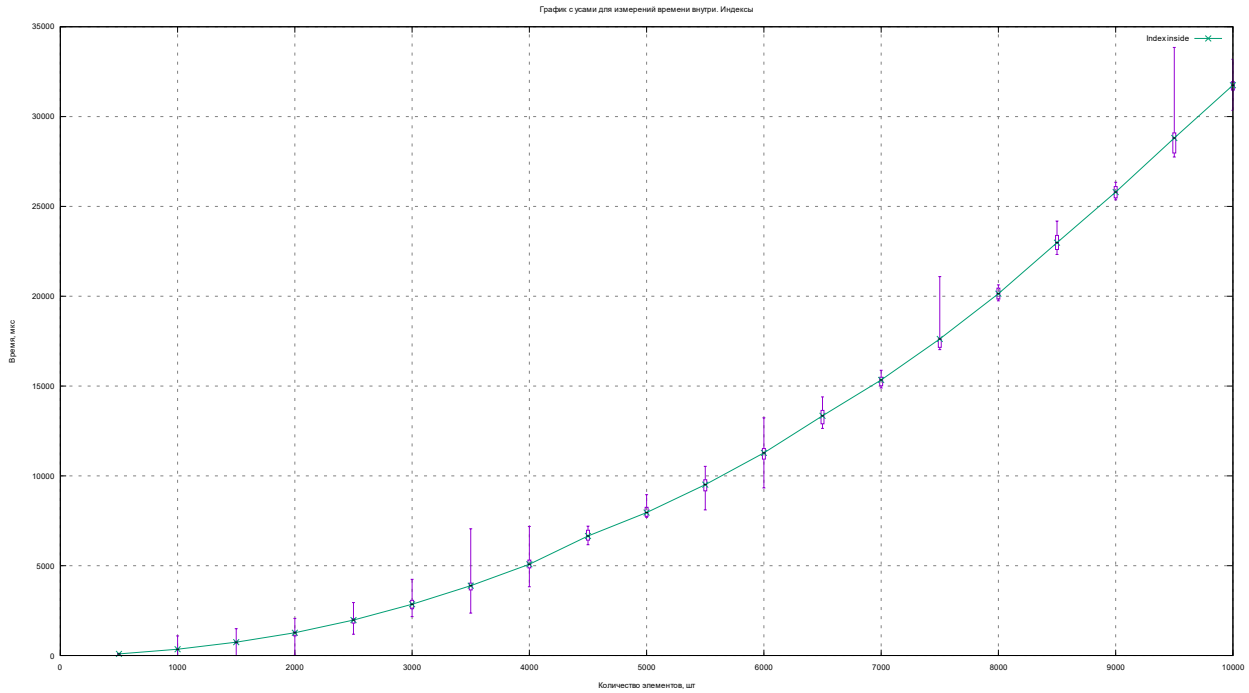
Измерения снаружи. Индексация a[i]:

График с усами (среднее, максимум, минимум; нижний, средний и верхний квартили).
Зависимость времени выполнения от количества элементов массива



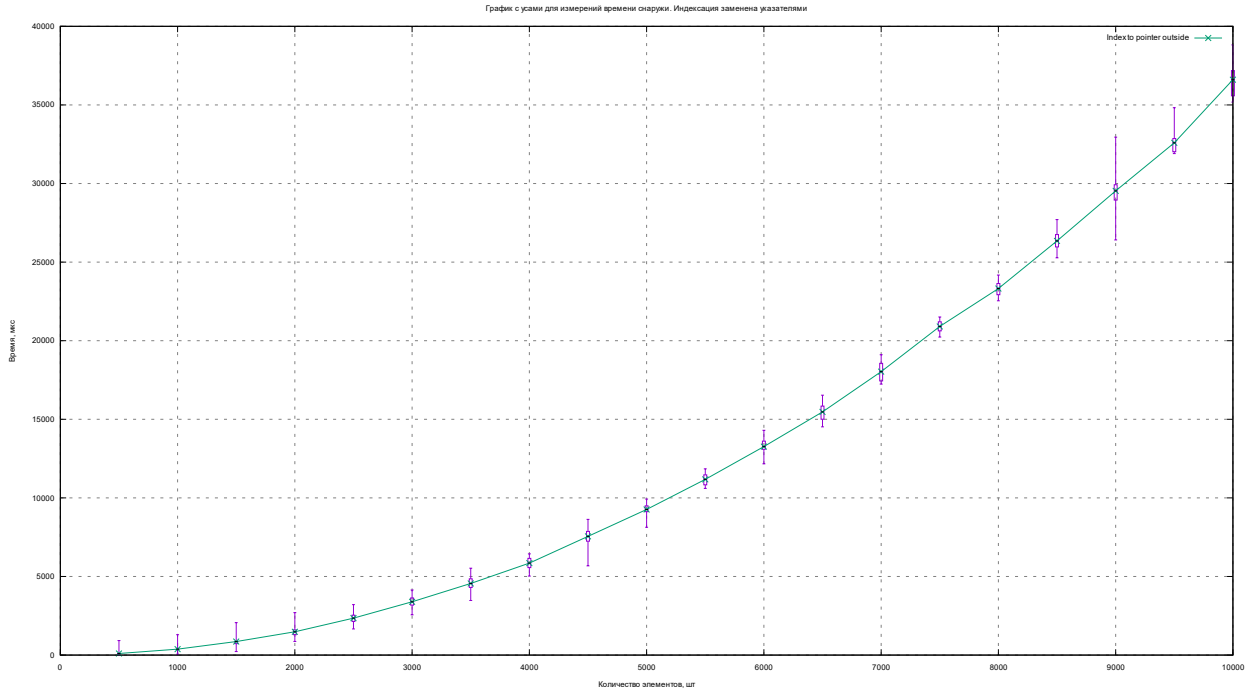
Измерения внутри. Индексация a[i]:

График с усами (среднее, максимум, минимум; нижний, средний и верхний квартили).
Зависимость времени выполнения от количества элементов массива



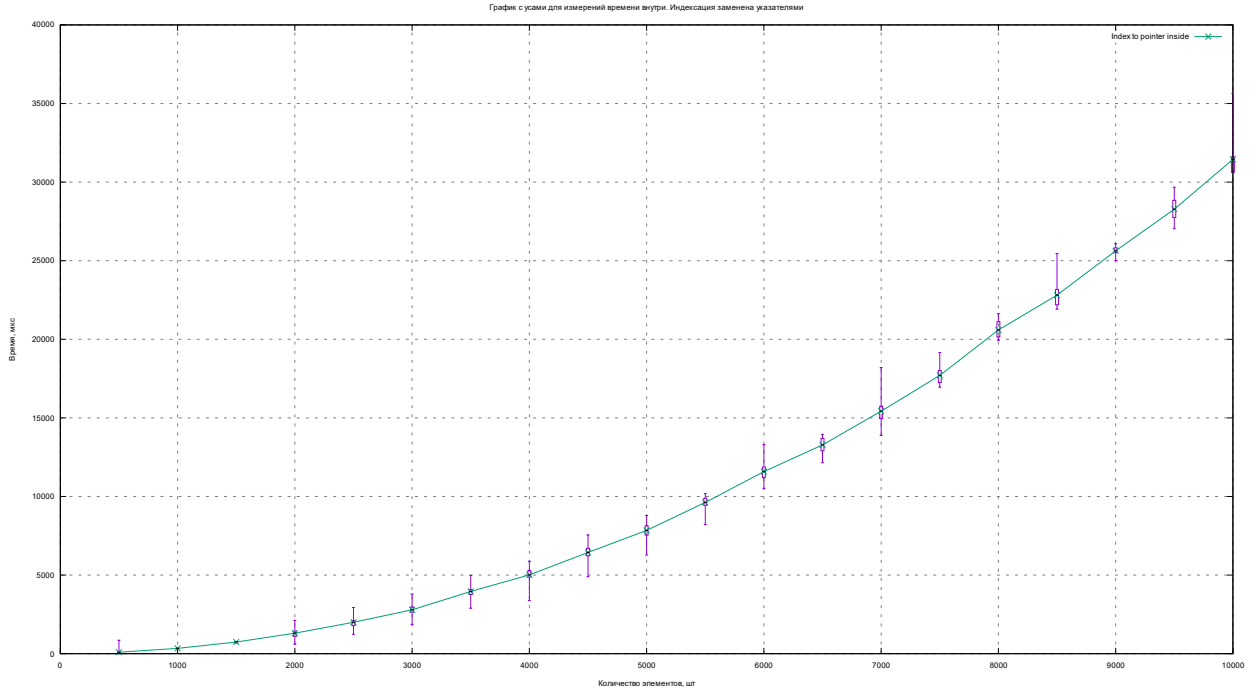
Измерения снаружи. Замена индексации *(a + i):

График с усами (среднее, максимум, минимум; нижний, средний и верхний квартили). Зависимость времени выполнения от количества элементов массива



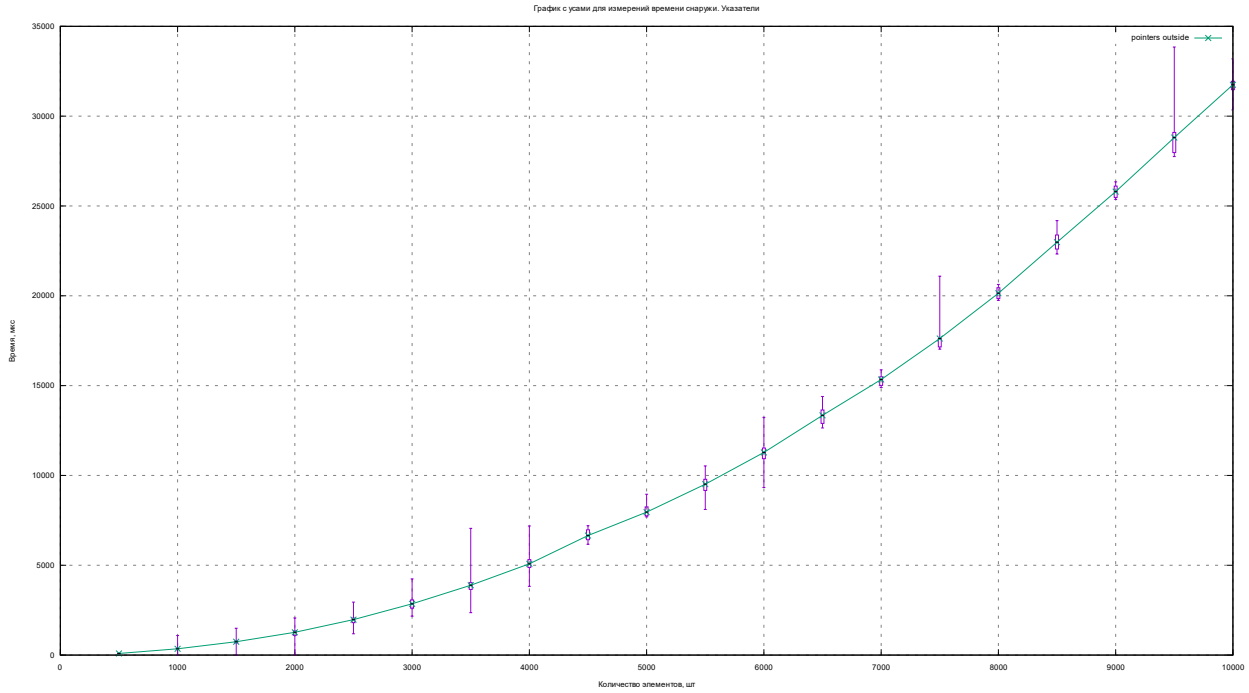
Измерения внутри. Замена индексации *(a + i):

График с усами (среднее, максимум, минимум; нижний, средний и верхний квартили). Зависимость времени выполнения от количества элементов массива



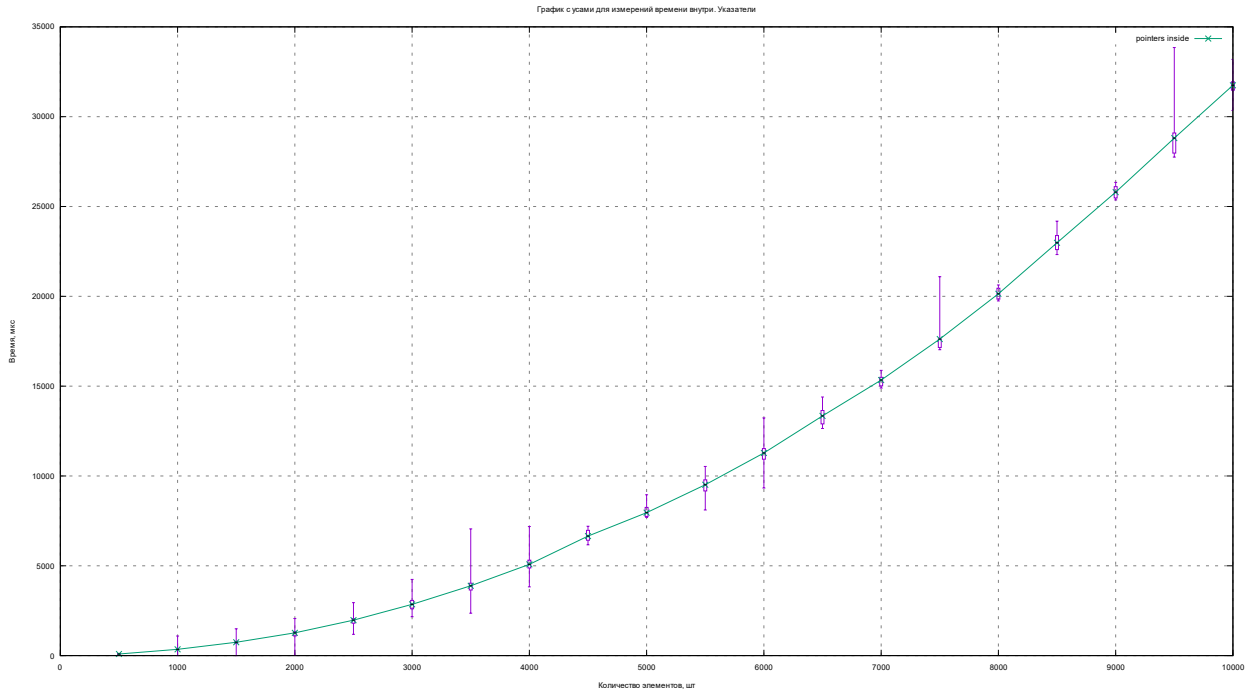
Измерения снаружи. Указатели:

График с усами (среднее, максимум, минимум; нижний, средний и верхний квартили). Зависимость времени выполнения от количества элементов массива

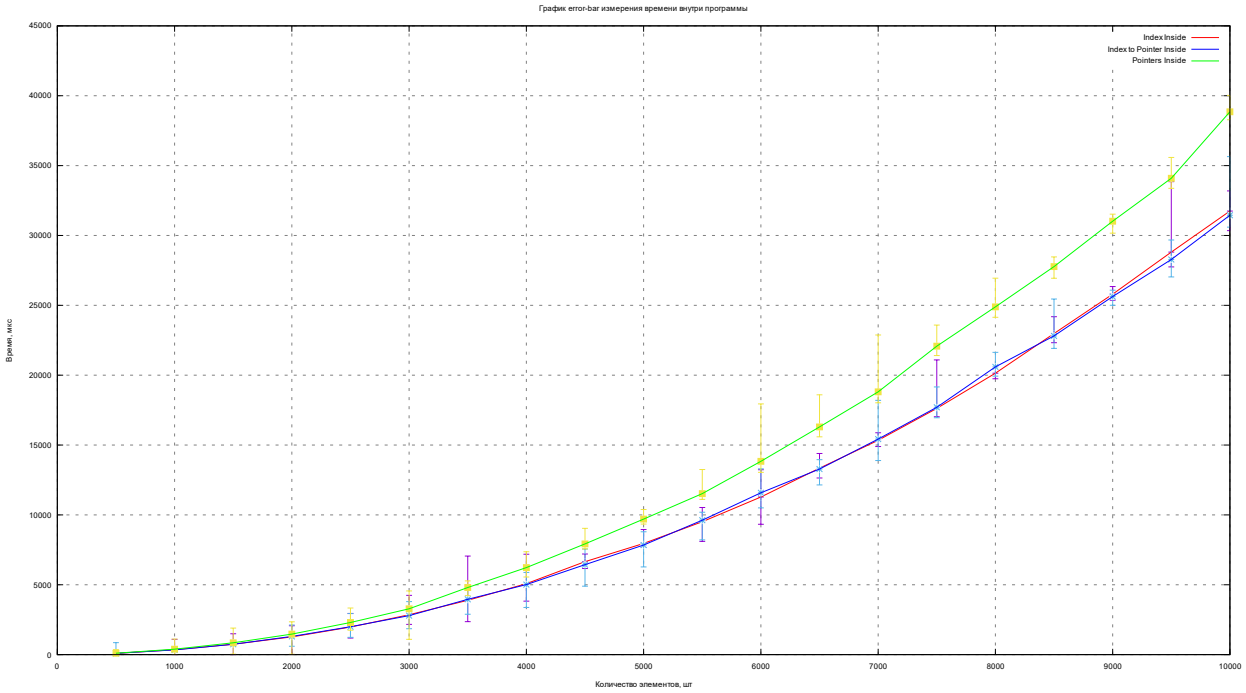


Измерения внутри. Указатели:

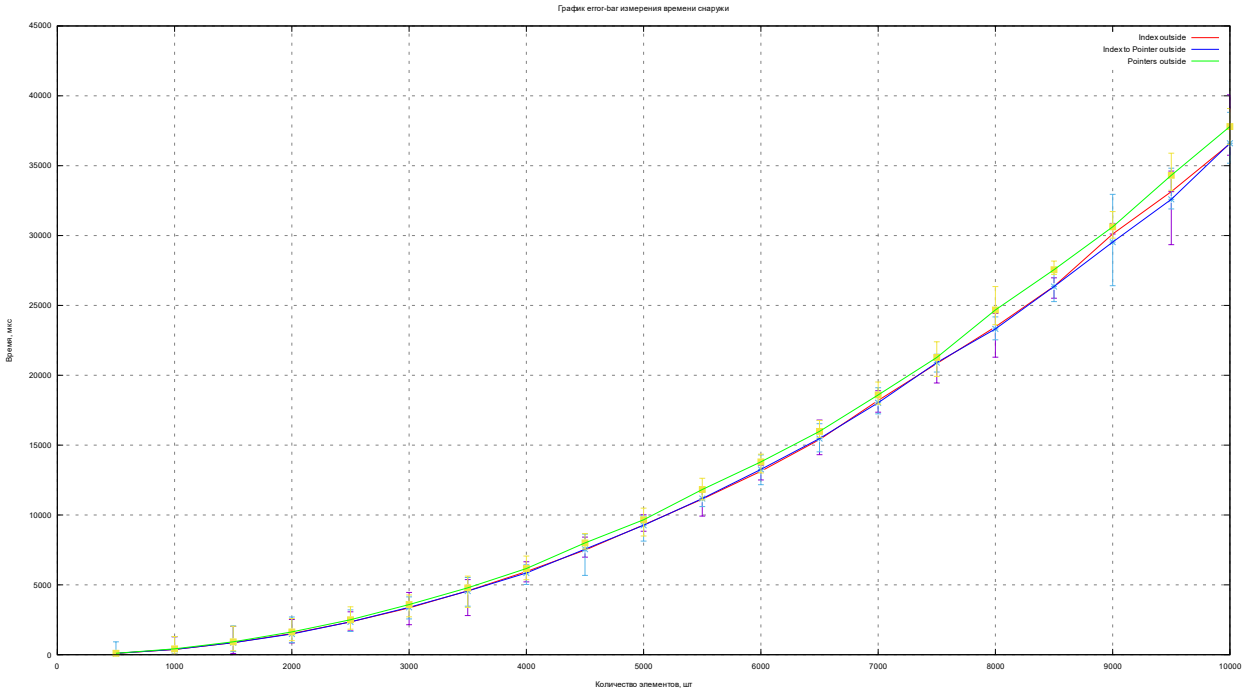
График с усами (среднее, максимум, минимум; нижний, средний и верхний квартили). Зависимость времени выполнения от количества элементов массива



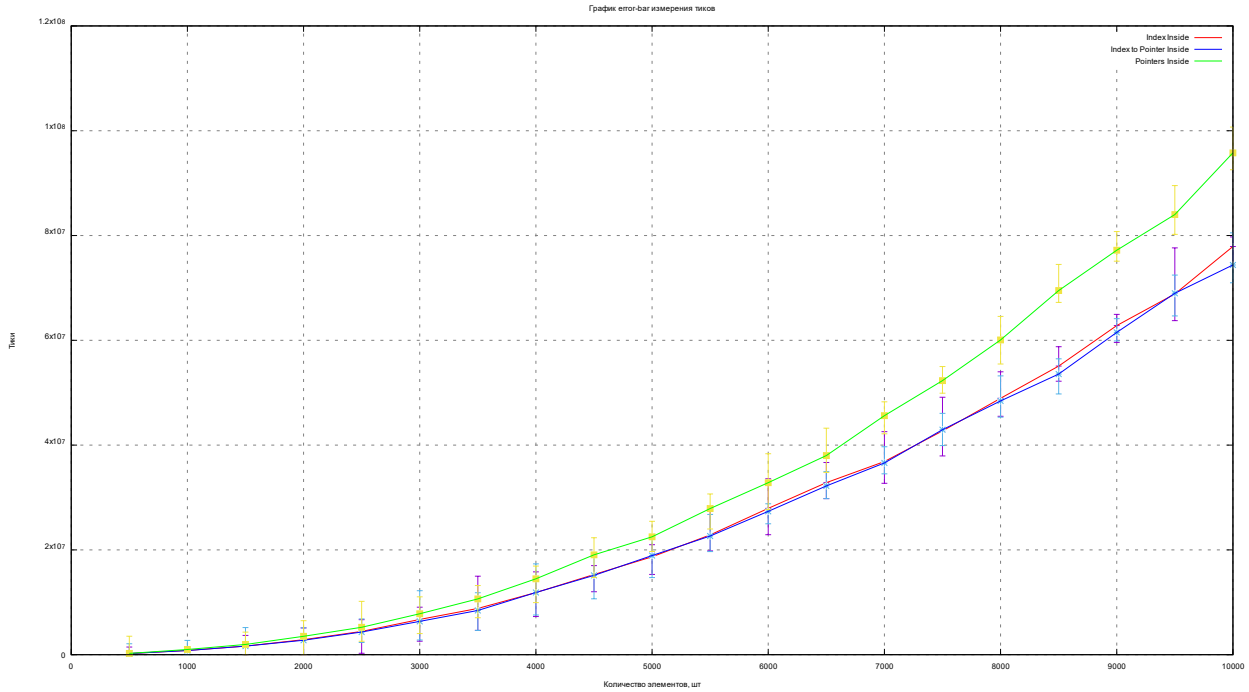
Кусочно-линейный график с ошибкой (среднее, максимум, минимум). Измерения времени внутри:



Кусочно-линейный график с ошибкой (среднее, максимум, минимум). Измерения времени снаружи:



Кусочно-линейный график с ошибкой (среднее, максимум, минимум). Измерения тиков:



Выводы:

После проведения замеров, сбора и обработки данных о времени выполнения программ делаю выводы, что конкретно на моей машине при замерах внутри использование указателей менее эффективно по времени, чем использование индексация или замена индексации. Индексация и замена индексации имеют почти одинаковую эффективность времени выполнения при разных размерах массивов. А при замерах вне программы, использование указателей лишь чуть-чуть хуже по времени, чем использование индексации или замены индексации. Индексация и замена индексации имеют практически одинаковую эффективность. Время выполнения сортировки при замерах вне программы немного больше, чем при замерах внутри программы