# Задание №7 в рамках вычислительного практикума. Постановка замерного эксперимента

Студент: Краснов Леонид

Группа: ИУ7-21Б

Table of Contents

[Задание №7 в рамках вычислительного практикума. Постановка замерного эксперимента 1](#_Toc112348661)

[Цель 1](#_Toc112348662)

[Задание 1](#_Toc112348663)

[Подготовка программы к тестированию 1](#_Toc112348664)

[Исходный код программы 1](#_Toc112348665)

[1. Разные способы работы с элементами одномерного массива: 4](#_Toc112348666)

[a) использование операции индексации a[i]; 4](#_Toc112348667)

[b) формальная замена операции индексации на выражение \*(a + i); 4](#_Toc112348668)

[c) использование указателей для работы с массивом. 5](#_Toc112348669)

[2. Разные уровни оптимизации Os, O0, O1, O2, O3. 5](#_Toc112348670)

[1. build\_apps.sh, вызвав который, можно получить весь набор необходимых исполняемых файлов. 5](#_Toc112348671)

[2. update\_data.sh, вызвав который, можно добавить некоторые данные в датасет экспериментов. 6](#_Toc112348672)

[3. make\_preproc.sh|py, вызвав который, можно подготовить данные из набора, провести первичный анализ: посчитать среднее арифметическое, медианное, найти максимум и минимум, вычислить нижний и верхний квартили. 7](#_Toc112348673)

[4. make\_postproc.sh|py, вызвав который, можно получить указанные ниже графики. 9](#_Toc112348674)

[5. go.sh, вызвав который, можно получить данные эксперимента (скрипт вызывает по очереди предыдущие четыре). 15](#_Toc112348675)

[Графики 16](#_Toc112348676)

[1. Обычный кусочно-линейный график зависимости времени выполнения в любых единицах измерения времени от числа элементов массива для всех 15 вариантов программы. 16](#_Toc112348677)

[2. Кусочно-линейный график с ошибкой (среднее, максимум, минимум) для всех вариантов обработки массива при уровне оптимизации 02. 19](#_Toc112348678)

[3. График с усами (среднее, максимум, минимум; нижний, средний и верхний квартили) для варианта обработки «через квадратные скобки» при уровне оптимизации 03. 20](#_Toc112348679)

[Таблица для результатов обрабботки 21](#_Toc112348680)

[Ответы на вопросы 22](#_Toc112348681)

[1. Какой способ обработки быстрее и почему? 22](#_Toc112348682)

[2. В датасете появился статистический выброс, причины которого очевидны, например, эксперимент был поставлен на другой машине. Порядок экспериментов не был известен заранее. Можно ли вырезать данные со статистическим выбросом из датасета? 22](#_Toc112348683)

[3. В датасете появился статистический выброс. Известно, что эксперимент был поставлен вчера, существует резервная копия датасета за позавчера. Можно ли вырезать данные со статистическим выбросом из датасета? 22](#_Toc112348684)

[4. В датасете обнаружена серия экспериментов с одним результатом. Можно ли заменить её одним экспериментом? 22](#_Toc112348685)

[5. Если заполнение случайными числами массива (или любая другая инициализация) присутствует в каждом эксперименте, то почему Вы замеряете время только у целевого алгоритма? 23](#_Toc112348686)

# Цель

Провести сравнение производительности работы программы по двум плоскостям.

# Задание

## Подготовка программы к тестированию

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

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

При этом пользователь вводит кол-во элементов и сами элементы

## Исходный код программы

|  |
| --- |
| #include **<stdio.h>** #define **N** 10 #define **INPUT\_ERROR** 1 #define **NOELEMS\_ERROR** 2 #define **SUCCESS** 0   **int** count(**int** \*xbeg, **int** \*xend, **int** \*ybeg, **int** \*yend) {  **int** rez = 0;  **int** k1 = (xend - xbeg);  **int** k2 = (yend - ybeg);  **int** k;  **if** (k1 > k2)  {  k = k2;  }  **else** {  k = k1;  }  **while** (k > 0)  {  rez += \*xbeg \* \*ybeg;  xbeg++;  ybeg++;  k--;  }  **return** rez; }   **int** main(**void**) {  **int** a[**N**], x[**N**], y[**N**], n;  printf(**"Введите длину массива: "**);  **if** (scanf(**"%d"**, &n) != 1)  {  printf(**"Неверный ввод!\n"**);  **return INPUT\_ERROR**;  }  **if** (n > 10 || n < 1)  {  printf(**"Неверный ввод\n"**);  **return INPUT\_ERROR**;  }  **int** \*pabeg = a, \*paend = pabeg + n;  printf(**"\nВведите элементы массива через пробел: "**);  **while** (pabeg < paend)  {  **if** (scanf(**"%d"**, pabeg) != 1)  **return INPUT\_ERROR**;  pabeg++;  }  pabeg = a;  **int** \*pxbeg = x, p = 0;  **int** \*pybeg = y, q = 0;  **while** (pabeg < paend)  {  **if** (\*pabeg < 0)  {  \*pxbeg = \*pabeg;  p++;  pxbeg++;  }  **if** (\*pabeg > 0)  {  \*pybeg = \*pabeg;  q++;  pybeg++;  }  pabeg++;  }  pxbeg = x;  pybeg = y;  *// Реверсирование массива  //ukazatel1 < y + q / 2 && ukazatel2 > y + q / 2* **for** (**int** \*ukazatel1 = y, \*ukazatel2 = y + q - 1; ukazatel1 < y + q / 2; ukazatel1++, ukazatel2--)  {  **int** temp = \*ukazatel2;  \*ukazatel2 = \*ukazatel1;  \*ukazatel1 = temp;  }  **if** (p == 0 || q == 0)  {  printf(**"Нет отрицательных / положительных элементов!\n"**);  **return NOELEMS\_ERROR**;  }  **int** \*pxend = pxbeg + p;  **int** \*pyend = pybeg + q;  **int** rez = count(pxbeg, pxend, pybeg, pyend);  printf(**"%d"**, rez);  **return SUCCESS**; } |

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

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

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

Эта функция возвращает количество микросекунд с некоторого момента:

|  |
| --- |
| *// Получение времени в микросекундах* **unsigned long long** microseconds\_now(**void**) {  **struct** timeval val;  **if** (gettimeofday(&val, **NULL**))  {  **return** (**unsigned long long**) - 1;  }  **return** val.tv\_sec \* 1000000ULL + val.tv\_usec; *// Получение микросекунд из секунд и микросекунд* } |

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

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

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

|  |
| --- |
| **int** main(**int** argc, **char** \*argv[]) |

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

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

## Разные способы работы с элементами одномерного массива:

### использование операции индексации a[i];

Целевая функция count с использованием операции индексации

Программа program\_1.c

|  |
| --- |
| **int** count(**const int** x\_arr[**N**], **int** k1, **const int** y\_arr[**N**], **int** k2) {  **int** rez = 0;  **int** k;  **if** (k1 > k2)  {  k = k2;  }  **else** {  k = k1;  }  **for** (**int** i =0; i < k; i++)  {  rez += x\_arr[i] \* y\_arr[i];  }  **return** rez; } |

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

Целевая функция count с формальной заменой операции индексации

Программа: program\_02.c

|  |
| --- |
| **int** count(**const int** x\_arr[**N**], **int** k1, **const int** y\_arr[**N**], **int** k2) {  **int** rez = 0;  **int** k;  **if** (k1 > k2)  {  k = k2;  }  **else** {  k = k1;  }  **for** (**int** i = 0; i < k; i++)  {  rez += \*(x\_arr + i) \* \*(y\_arr + i);  }  **return** rez; } |

### использование указателей для работы с массивом.

Целевая функция count с использованием указателей

Программа program\_3.c

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

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

### build\_apps.sh, вызвав который, можно получить весь набор необходимых исполняемых файлов.

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

Код скрипта

|  |
| --- |
| **#!/bin/bash** opts=**"O1 O2 O3 O0 Os"** programms=**"programm\_1 programm\_2 programm\_3" for** i **in** $programms; **do  for** opt **in** $opts; **do** gcc -std=c99 -Wall -Werror -Wpedantic -Wextra \  -**"$**{opt}**"** \  **"**$i**"**.c -o ./apps/**"$**{i}**"**\_**"$**{opt}**"**.exe  echo **"$**{i}**"**\_**"$**{opt}**"**.exe готово  **done done** |

### update\_data.sh, вызвав который, можно добавить некоторые данные в датасет экспериментов.

Имеется 15 исполняемых файлов вида: programm\_[номер\_версии\_программы]\_[уровень\_оптимизации].exe, при этом каждую программу нужно запустить с различным размером массива. Скрипт update\_data.sh запускает все исполняемые файлы с заданными размерами массива, записывая или дополняя соответствующие файлы data.

Код скрипта

|  |
| --- |
| **#!/bin/bash** programms=**"programm\_1 programm\_2 programm\_3"** sizes=**""** opts=**"O1 O2 O3 O0 Os"** count=100  i=300000  **while** [ $i -le 600000 ] **do** sizes=**"**$sizes $i**"** ((i+=15000)) **done  if** [ **!** -z $1 ]; **then** count=$1 **fi if** [ **!** -z $2 ]; **then** sizes=$2 **fi if** [ **!** -z $3 ]; **then** programms=$3 **fi if** [ **!** -z $4 ]; **then** opts=$4 **fi   for** prog **in** $programms; **do  for** co **in $**(seq **"**$count**"**); **do  for** opt **in** $opts; **do  for** i **in** $sizes; **do** echo -n -e **"$**{prog}**\_""$**{opt}**\_""$**{i}**\t** $co**/**$count **\r"** ./apps/**"$**{prog}**"**\_**"$**{opt}**"**.exe **"$**{i}**"**>>./data/**"$**{prog}**"**\_**"$**{opt}**"**\_**"$**{i}**"**.txt  **done  done  done done** |

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

|  |
| --- |
| bash update\_data.sh [Кол-во\_тестов] [кол-во\_элементов] [имя\_программы] [оптимизация] |

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

Код программы

|  |
| --- |
| **import** os **import** shutil   *# Функция принимает имя файла и возвращает массив целых чисел из этого файла* **def** read(str):  **with** open(str, **"r"**) **as** f:  mass = []  **for** line **in** f:  a = int(line)  mass.append(a)  **return** mass   **def** find\_sizes(folder):  sizes = []  files = os.listdir(folder)  **for** i **in** range(len(files)):  file = files[i]  file = file[14:]  file = file.replace(**".txt"**, **""**, 1)  sizes.append(int(file))  sizes = list(set(sizes))  sizes.sort()  **for** i **in** range(len(sizes)):  sizes[i] = str(sizes[i])  *# print(sizes)* **return** sizes   *# Функция принимает массив и возвращает среднеарифметическое* **def** average(arr):  sum = 0  **for** i **in** range(len(arr)):  sum += arr[i]  **return** sum/len(arr)   *# Функция принимает массив и возвращает его медианное значение* **def** find\_median(arr):  arr.sort()  *# print(arr)* **if** len(arr) == 1:  **return** arr[0]  **if** len(arr) % 2 == 0:  **return** arr[len(arr)//2]  **return** (int(arr[int(len(arr)/2) - 1]) + int(arr[int(len(arr)/2)])) / 2   *# Находит нижний квартиль* **def** finde\_lower\_quartile(arr):  a = len(arr)  *# граница 25%* a //= 4  **return** arr[a]   *# Находит верхний квартиль* **def** find\_upper\_quartile(arr):  a = len(arr)  *# Граница 75%* a //= 4  a \*= 3  **return** arr[a]   *# Функция удаляет подготовленные данные с прошлых эксперементов* **def** del\_old(folder):  shutil.rmtree(folder)  os.mkdir(folder)   *# Записывает полученные данные в файл* **def** save\_prep\_data(stri, avg, med, mini, maxi, up\_quart, low\_quart):  folder = **'./prep\_data/'** file = folder + stri  **with** open(file, **"w"**) **as** f:  f.write(str(avg) + **'\n'**)  f.write(str(med) + **'\n'**)  f.write(str(mini) + **'\n'**)  f.write(str(maxi) + **'\n'**)  f.write(str(low\_quart) + **'\n'**)  f.write(str(up\_quart))  file = **"\r"** + file    folder = **'./data/'** folder\_out = **'./prep\_data'** *# Удаляет предыдущие данные* del\_old(folder\_out)  programm = [**'programm\_1'**, **'programm\_2'**, **'programm\_3'**] option = [**'O0'**, **'O1'**, **'O2'**, **'O3'**, **'Os'**] **for** i **in** range(len(option)):  option[i] = **"\_"** + option[i]  size = find\_sizes(folder) **for** i **in** range(len(size)):  size[i] = **"\_"** + size[i]  ex = **'.txt'** n = 0 print(**"Подготовка данных"**) **for** prog **in** range(len(programm)):  **for** opt **in** range(len(option)):  **for** siz **in** range(len(size)):  stri = programm[prog] + option[opt] + size[siz] + ex  arr = read(folder + stri)  avg = average(arr)  med = find\_median(arr)  low\_quart = finde\_lower\_quartile(arr)  up\_quart = find\_upper\_quartile(arr)  mini = arr[0]  maxi = arr[len(arr) - 1]  save\_prep\_data(stri, avg, med, mini, maxi, up\_quart, low\_quart)  n += 1  progress = str(n) + **' / '** + str(len(programm) \* len(option) \* len(size)) + **'\r'** print(progress, end=**""**) print() |

Программа выполняет первичный анализ и записывает результаты в соответствующие файлы в папку prep\_data.

Файлы записываются построчно на одной строке - 1 параметр

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

### make\_postproc.sh|py, вызвав который, можно получить указанные ниже графики.

Код программы

|  |
| --- |
| **import** os **import** shutil  **import** matplotlib.pyplot **as** plt  *# Получение информации из переданной строки о том, что содержится в файле* **def** whats\_in\_file(file):  sp\_file = file.split(**"\_"**)  prog\_num = sp\_file[1]  opt\_lvl = sp\_file[2]  col\_elems = sp\_file[3]  col\_elems = col\_elems.partition(**'.'**)[0]  file = **'post\_prep\_data\_'** + prog\_num + **'\_'** + opt\_lvl + **'.txt'   return** prog\_num, opt\_lvl, col\_elems, file   *# Плучение информации из файла* **def** params\_from\_file(folder\_input, file):  val = []  **with** open(folder\_input + file) **as** f:  **for** line **in** f:  val.append(float(line))  **return** val   *# Функция подготавливает данные для дальнейшего построения графика* **def** linal\_graph\_data(files\_input, folder\_input, folder\_out):  print(**"\_\_\_\_Таблица данных для кусочного графика\_\_\_\_"**)  print(**"col\_elems\t time"**)  files\_input.sort()  **for** file **in** files\_input:  avg = params\_from\_file(folder\_input, file)[0]  prog\_num, opt\_lvl, col\_elems, file = whats\_in\_file(file)  **with** open(folder\_out + file, **'a'**) **as** f:  f.write(str(col\_elems) + **' '** + str(avg) + **'\n'**)   files\_table = os.listdir(folder\_out)  files\_table.sort()  **for** file\_tab **in** files\_table:  print(**"\_\_tab data"**, file\_tab, **"\_\_"**)  **with** open(folder\_out + file\_tab) **as** f:  **for** line **in** f:  print(line, end=**""**)   *# Сортировка массивов X и Y по X* **def** sort\_2(arr1, arr2):  **for** i **in** range(len(arr1)):  **for** j **in** range(len(arr1) - 1):  **if** arr1[j] > arr1[j + 1]:  s = arr1[j]  arr1[j] = arr1[j + 1]  arr1[j + 1] = s  s = arr2[j]  arr2[j] = arr2[j + 1]  arr2[j + 1] = s   *# Запись подготовленных данных в массивы по X и по Y для линейного графика* **def** linal\_graph\_to\_mass(folder):  x\_array = []  y\_array = []  files\_input = os.listdir(folder)  files\_input.sort()  **for** file **in** files\_input:  file = folder + file  x\_arg = []  y\_arg = []  **with** open(file, **"r"**) **as** f:  **for** line **in** f:  args = line.split(**' '**)  x\_arg.append(int(args[0]))  y\_arg.append(float(args[1]))  sort\_2(x\_arg, y\_arg)  x\_array.append(x\_arg)  y\_array.append(y\_arg)  **return** x\_array, y\_array, files\_input   *# Строит линейный график* **def** plot\_linal\_graph(x\_array, y\_array, labels\_array):  *# Удаление лишнего из лэйблов* **for** i **in** range(len(labels\_array)):  labels\_array[i] = labels\_array[i].replace(**".txt"**, **""**)  labels\_array[i] = labels\_array[i].replace(**"post\_prep\_"**, **""**, 1)  labels\_array[i] = labels\_array[i].replace(**"data"**, **"prog"**)  fig, ax = plt.subplots()  *# Набор параметров для отображения графиков* linestyle\_array = [**'-'**, **'--'**, **'-.'**, **':'**, (0, (3, 1, 1, 1, 1, 1))]  markers = [**'^'**, **'s'**, **'X'**]  *# Генерация 15 видов линий* difference = []  **for** j **in** range(len(markers)):  **for** i **in** range(len(linestyle\_array)):  typ = [linestyle\_array[i], markers[j]]  difference.append(typ)  *# Построение графиков* **for** i **in** range(len(x\_array)):  ax.plot(x\_array[i], y\_array[i], label=labels\_array[i], linestyle=difference[i][0], marker=difference[i][1])   *# Добавляем подписи к осям:* ax.set\_xlabel(**'Кол-во элементов'**)  ax.set\_ylabel(**'Время (микросекунды)'**)   *# Добавление легенды, масштабной сетки и вывод графиков на поле* ax.legend()  ax.grid()   *# Сохранение графика* fig.savefig(**'line\_plot.svg'**)   *# Функция удаляет подготовленные данные с прошлых эксперементов* **def** del\_old(folder):  shutil.rmtree(folder)  os.mkdir(folder)   *# Функция подготавливает данные для графика с ошибкой* **def** error\_graph\_data(files\_input, folder\_input, folder\_out):  print(**"\_\_\_Таблица график с ошибкой\_\_\_"**)  print(**"col\_elems\t avg\t max\t min\t"**)  files\_input.sort()  **for** file **in** files\_input:  params = params\_from\_file(folder\_input, file)  avg = params[0]  maxi = params[3]  mini = params[2]  prog\_num, opt\_lvl, col\_elems, file = whats\_in\_file(file)  **if** opt\_lvl == **'O2'**:  **with** open(folder\_out + file, **'a'**) **as** f:  f.write(str(col\_elems) + **' '** + str(avg) + **' '** + str(maxi - avg) + **' '** + str(avg - mini) + **'\n'**)  *# print(str(col\_elems) + ' ' + str(avg) + ' ' + str(maxi) + ' ' + str(mini))* files\_table = os.listdir(folder\_out)  files\_table.sort()  **for** file\_tab **in** files\_table:  print(**"\_\_tab data"**, file\_tab, **"\_\_"**)  **with** open(folder\_out + file\_tab) **as** f:  **for** line **in** f:  line = line.split()  print(line[0], line[1], int(float(line[2]) + float(line[1])), int(float(line[3]) + float(line[1])))  *# Сортирует 3 массива по первому* **def** sort3(arr1, arr2, arr3):  **for** i **in** range(len(arr1)):  **for** j **in** range(len(arr1) - 1):  **if** arr1[j] > arr1[j + 1]:  s = arr1[j]  arr1[j] = arr1[j + 1]  arr1[j + 1] = s   s = arr2[j]  arr2[j] = arr2[j + 1]  arr2[j + 1] = s   s = arr3[0][j]  arr3[0][j] = arr3[0][j + 1]  arr3[0][j + 1] = s   s = arr3[1][j]  arr3[1][j] = arr3[1][j + 1]  arr3[1][j + 1] = s   *# Записывает данные из файлов по массивам* **def** error\_graph\_to\_mass(folder):  x\_array = []  y\_array = []  errors\_array = []  files\_data = os.listdir(folder)  files\_data.sort()  **for** file **in** files\_data:  file = folder + file  x\_arg = []  y\_arg = []  errors\_args = []  err\_max = []  err\_min = []  **with** open(file, **"r"**) **as** f:  **for** line **in** f:  args = line.split(**' '**)  x\_arg.append(int(args[0]))  y\_arg.append(float(args[1]))  err\_max.append(float(args[2]))  err\_min.append(float(args[3]))  errors\_args.append(err\_min.copy())  errors\_args.append(err\_max.copy())  sort3(x\_arg, y\_arg, errors\_args)  x\_array.append(x\_arg)  y\_array.append(y\_arg)  errors\_array.append(errors\_args)  **return** x\_array, y\_array, errors\_array, files\_data   *# Строит график ошибок* **def** plot\_error\_graph(x\_array, y\_array, errors\_array, labels\_array):  *# Удаление лишнего из лэйблов* **for** i **in** range(len(labels\_array)):  labels\_array[i] = labels\_array[i].replace(**".txt"**, **""**)  labels\_array[i] = labels\_array[i].replace(**"post\_prep\_"**, **""**, 1)  labels\_array[i] = labels\_array[i].replace(**"data"**, **"prog"**)  fig, ax = plt.subplots()  *# Набор параметров для отображения графиков* markers = [**'^'**, **'s'**, **'X'**]  linestyle\_array = [**'-'**, **'--'**, **'-.'**]  *# Построение графиков* **for** i **in** range(len(x\_array)):  ax.errorbar(x\_array[i], y\_array[i], yerr=errors\_array[i], label=labels\_array[i], marker=markers[i],  linestyle=linestyle\_array[i])  *# Добавляем подписи к осям:* ax.set\_xlabel(**'Кол-во элементов'**)  ax.set\_ylabel(**'Время (микросекунды)'**)  *# Добавление легенды, масштабной сетки и вывод графиков на поле* ax.legend()  ax.grid()   *# Сохранение графика* fig.savefig(**'error\_plot.svg'**)   *# График с усами (среднее, максимум, минимум; нижний, средний и верхний квартили) для варианта обработки # «через квадратные скобки» при уровне оптимизации O3  # Записывает данные из data в массив* **def** boxplot\_prep\_data(folder\_prep\_data, folder\_for\_table):  print(**"\_\_Таблица данных для графика с усами\_\_"**)  print(**"col\_elems\tavg\t med\t min\t max\t up\_quart\t low\_quart"**)  files\_table = os.listdir(folder\_for\_table)  files\_table.sort()  **for** file **in** files\_table:  prog\_num, opt\_lvl, col\_elems, file1 = whats\_in\_file(file)  params = params\_from\_file(folder\_prep\_data, file)  avg = params[0]  median = params[1]  maxi = params[3]  mini = params[2]  up\_quart = params[4]  low\_quart = params[5]  **if** opt\_lvl == **"O2" and** prog\_num == **'1'**:  print(col\_elems, avg, median, mini, maxi, up\_quart, low\_quart, sep=**'\t'**)  data\_array = []  elements\_data = []  times = []  **for** file **in** files:  time\_data = []  time\_data.clear()  *# Сбор данных* prog\_num, opt\_lvl, col\_elems, file1 = whats\_in\_file(file)  *# Запись подготовленных данных* **if** opt\_lvl == **'O3' and** prog\_num == **'1'**:  **with** open(folder\_prep\_data + file, **"r"**) **as** f:  **for** line **in** f:  time\_data.append(float(line))  times.append(time\_data)  elements\_data.append(col\_elems)   data\_array.append(times.copy())  data\_array.append(elements\_data.copy())  sort\_2(data\_array[0], data\_array[1])  **return** data\_array   **def** plot\_boxplot(arr):  fig, ax = plt.subplots()  *# Набор параметров для отображения графиков  # Построение графиков   # Creating plot* ax.boxplot(arr[0])   *# Добавляем подписи к осям:  # ax.set\_xlabel('Кол-во элементов')* ax.set\_ylabel(**'Время (микросекунды)'**)   *# Добавление легенды, масштабной сетки и вывод графиков на поле* ax.grid()   *# Сохранение графика* fig.savefig(**'boxplot.svg'**)   *# Заданные папки* folder\_in = **"prep\_data/"** folder\_lingraph = **"post\_prep\_lingraph\_data/"** folder\_errorgraph = **"errorgraph\_data/"** folder\_boxplot = **"post\_prep\_box\_plot\_data/"** folder\_data = **'data/'** *# Массив в котором хранятся имена файлов из папки folder\_in* files = os.listdir(folder\_in)  *# Линейный график  # Удаление старых данных* del\_old(folder\_lingraph)  *# Подготовка данных для построения графика* linal\_graph\_data(files, folder\_in, folder\_lingraph)  *# Построение графика по подготовленным данным* x, y, labels = linal\_graph\_to\_mass(folder\_lingraph) plot\_linal\_graph(x, y, labels)  *# График с ошибкой  # Удаление старых данных* del\_old(folder\_errorgraph)  *# Подготовка данных для построения графика* error\_graph\_data(files, folder\_in, folder\_errorgraph)  *# Построение графика по подготовленным данным* x, y, errors, labels = error\_graph\_to\_mass(folder\_errorgraph) plot\_error\_graph(x, y, errors, labels)  *# Ящик с усами   # Подготовка данных для построения графика* data\_arr = boxplot\_prep\_data(folder\_data, folder\_in)  plot\_boxplot(data\_arr)   *# Показать все построенное* plt.show() |

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

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

Код скрипта

|  |
| --- |
| **#!/bin/bash** mkdir **"apps"** bash build\_apps.sh mkdir **"data"** bash update\_data.sh mkdir **"prep\_data"** python3 make\_preproc.py mkdir **"errorgraph\_data"** mkdir **"post\_prep\_lingraph\_data"** python3 make\_postproc.py |

## Графики

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

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

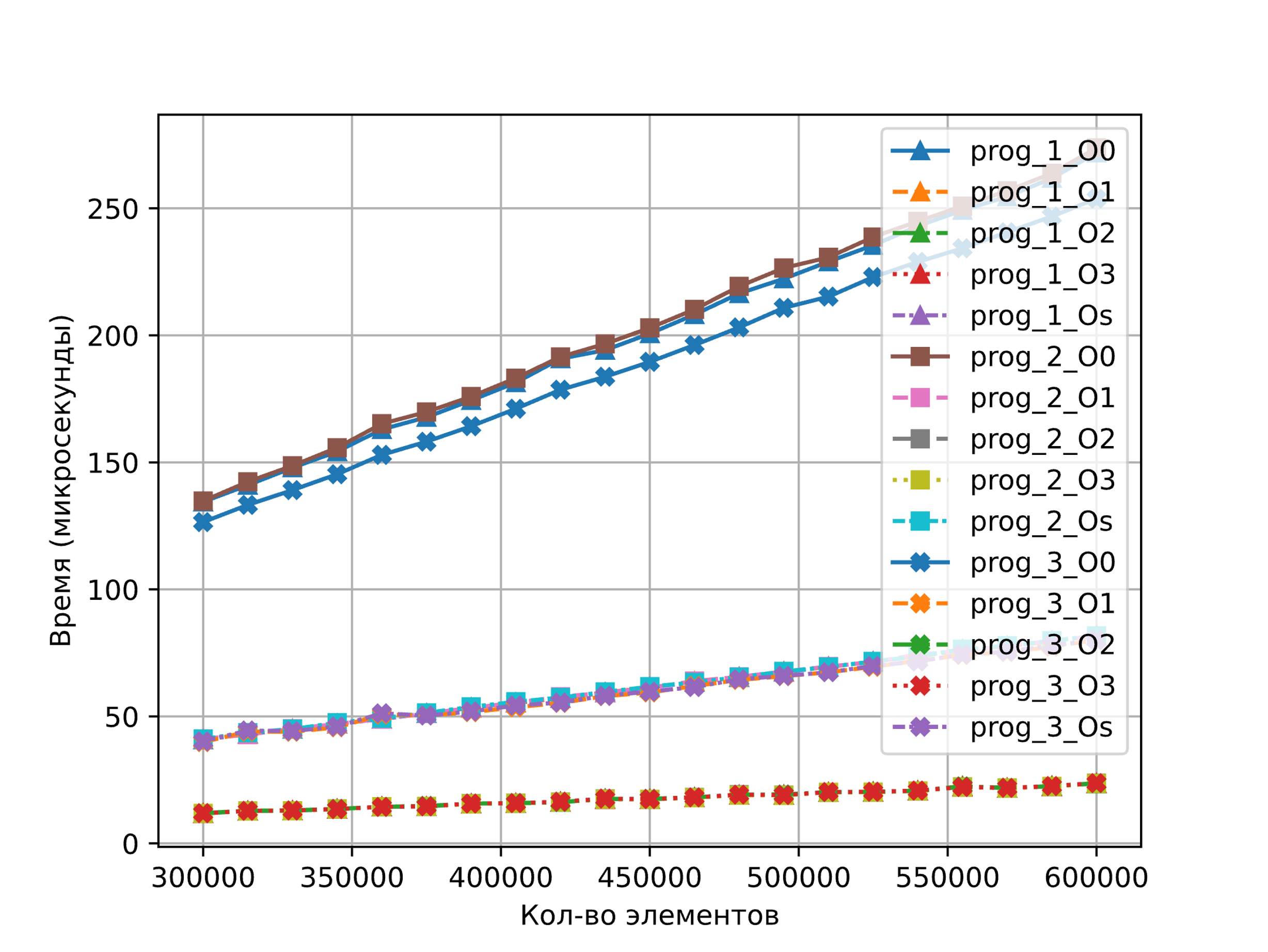


Таблица к кусочно-линейному графику

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| prog\_1\_O0 | | prog\_1\_O1 | | prog\_1\_O2 | | prog\_1\_O3 | | prog\_1\_Os | |
| col\_elems | time | col\_elems | time | col\_elems | col\_elems | time | col\_elems | time | col\_elems |
| 300000 | 134.46 | 300000 | 40.89 | 300000 | 11.74 | 300000 | 11.78 | 300000 | 40.84 |
| 315000 | 141.03 | 315000 | 43.18 | 315000 | 12.85 | 315000 | 12.77 | 315000 | 43.03 |
| 330000 | 147.9 | 330000 | 44.97 | 330000 | 12.86 | 330000 | 12.96 | 330000 | 44.95 |
| 345000 | 154.26 | 345000 | 47.04 | 345000 | 13.53 | 345000 | 13.47 | 345000 | 47.0 |
| 360000 | 162.96 | 360000 | 49.04 | 360000 | 14.44 | 360000 | 14.41 | 360000 | 48.95 |
| 375000 | 167.77 | 375000 | 51.14 | 375000 | 14.52 | 375000 | 14.51 | 375000 | 51.18 |
| 390000 | 174.41 | 390000 | 53.29 | 390000 | 15.67 | 390000 | 15.57 | 390000 | 53.43 |
| 405000 | 181.41 | 405000 | 55.37 | 405000 | 15.76 | 405000 | 15.81 | 405000 | 55.33 |
| 420000 | 190.8 | 420000 | 57.19 | 420000 | 16.22 | 420000 | 16.32 | 420000 | 57.1 |
| 435000 | 194.18 | 435000 | 59.45 | 435000 | 17.48 | 435000 | 17.33 | 435000 | 59.32 |
| 450000 | 200.68 | 450000 | 61.34 | 450000 | 17.42 | 450000 | 17.32 | 450000 | 61.42 |
| 465000 | 208.19 | 465000 | 63.76 | 465000 | 18.19 | 465000 | 17.99 | 465000 | 63.44 |
| 480000 | 216.47 | 480000 | 65.32 | 480000 | 19.1 | 480000 | 19.02 | 480000 | 65.44 |
| 495000 | 222.27 | 495000 | 67.65 | 495000 | 19.07 | 495000 | 19.0 | 495000 | 67.47 |
| 510000 | 229.0 | 510000 | 69.47 | 510000 | 20.17 | 510000 | 20.16 | 510000 | 69.45 |
| 525000 | 235.46 | 525000 | 71.52 | 525000 | 20.28 | 525000 | 20.18 | 525000 | 71.56 |
| 540000 | 242.99 | 540000 | 73.76 | 540000 | 20.78 | 540000 | 20.66 | 540000 | 73.53 |
| 555000 | 249.27 | 555000 | 75.77 | 555000 | 22.52 | 555000 | 22.18 | 555000 | 76.48 |
| 570000 | 254.57 | 570000 | 77.63 | 570000 | 21.76 | 570000 | 21.79 | 570000 | 77.7 |
| 585000 | 261.87 | 585000 | 79.46 | 585000 | 22.62 | 585000 | 22.35 | 585000 | 79.7 |
| 600000 | 271.83 | 600000 | 81.85 | 600000 | 23.51 | 600000 | 23.66 | 600000 | 81.76 |
| prog\_2\_O0 | | prog\_2\_O1 | | prog\_2\_O2 | | prog\_2\_O3 | | prog\_2\_Os | |
| col\_elems | time | col\_elems | time | col\_elems | col\_elems | time | col\_elems | time | col\_elems |
| 300000 | 134.77 | 300000 | 41.13 | 300000 | 11.81 | 300000 | 11.83 | 300000 | 41.11 |
| 315000 | 142.35 | 315000 | 42.82 | 315000 | 12.77 | 315000 | 12.72 | 315000 | 43.6 |
| 330000 | 148.66 | 330000 | 45.11 | 330000 | 12.83 | 330000 | 12.85 | 330000 | 45.03 |
| 345000 | 155.77 | 345000 | 47.01 | 345000 | 13.6 | 345000 | 13.55 | 345000 | 47.48 |
| 360000 | 165.19 | 360000 | 49.05 | 360000 | 14.39 | 360000 | 14.47 | 360000 | 49.31 |
| 375000 | 169.82 | 375000 | 51.21 | 375000 | 14.55 | 375000 | 14.58 | 375000 | 51.39 |
| 390000 | 175.9 | 390000 | 53.3 | 390000 | 15.66 | 390000 | 15.54 | 390000 | 53.75 |
| 405000 | 183.14 | 405000 | 55.44 | 405000 | 15.83 | 405000 | 15.79 | 405000 | 55.75 |
| 420000 | 191.52 | 420000 | 57.71 | 420000 | 16.3 | 420000 | 16.25 | 420000 | 57.41 |
| 435000 | 196.65 | 435000 | 59.34 | 435000 | 17.5 | 435000 | 17.42 | 435000 | 59.62 |
| 450000 | 202.92 | 450000 | 61.24 | 450000 | 17.33 | 450000 | 17.34 | 450000 | 61.7 |
| 465000 | 210.21 | 465000 | 63.92 | 465000 | 18.09 | 465000 | 18.0 | 465000 | 63.4 |
| 480000 | 219.28 | 480000 | 65.57 | 480000 | 19.18 | 480000 | 18.97 | 480000 | 65.45 |
| 495000 | 226.52 | 495000 | 67.68 | 495000 | 19.0 | 495000 | 18.95 | 495000 | 67.69 |
| 510000 | 230.7 | 510000 | 69.62 | 510000 | 20.09 | 510000 | 20.11 | 510000 | 69.43 |
| 525000 | 238.7 | 525000 | 71.39 | 525000 | 20.16 | 525000 | 20.17 | 525000 | 71.65 |
| 540000 | 244.88 | 540000 | 74.35 | 540000 | 20.6 | 540000 | 20.63 | 540000 | 73.66 |
| 555000 | 250.85 | 555000 | 75.85 | 555000 | 22.23 | 555000 | 22.21 | 555000 | 76.44 |
| 570000 | 256.87 | 570000 | 77.73 | 570000 | 21.81 | 570000 | 21.84 | 570000 | 77.8 |
| 585000 | 263.76 | 585000 | 79.75 | 585000 | 22.46 | 585000 | 22.41 | 585000 | 79.79 |
| 600000 | 273.78 | 600000 | 81.6 | 600000 | 23.69 | 600000 | 23.6 | 600000 | 81.78 |
| prog\_3\_O0 | | prog\_3\_O1 | | prog\_3\_O2 | | prog\_3\_O3 | | prog\_3\_Os | |
| col\_elems | time | col\_elems | time | col\_elems | col\_elems | time | col\_elems | time | col\_elems |
| 300000 | 126.48 | 300000 | 39.89 | 300000 | 11.97 | 300000 | 11.85 | 300000 | 40.2 |
| 315000 | 133.26 | 315000 | 44.03 | 315000 | 12.86 | 315000 | 12.82 | 315000 | 44.45 |
| 330000 | 139.09 | 330000 | 43.95 | 330000 | 13.02 | 330000 | 12.96 | 330000 | 44.22 |
| 345000 | 145.33 | 345000 | 45.69 | 345000 | 13.59 | 345000 | 13.5 | 345000 | 46.03 |
| 360000 | 152.96 | 360000 | 50.71 | 360000 | 14.42 | 360000 | 14.38 | 360000 | 51.21 |
| 375000 | 158.16 | 375000 | 50.38 | 375000 | 14.77 | 375000 | 14.6 | 375000 | 50.31 |
| 390000 | 164.22 | 390000 | 51.64 | 390000 | 15.67 | 390000 | 15.69 | 390000 | 51.94 |
| 405000 | 171.11 | 405000 | 53.61 | 405000 | 15.92 | 405000 | 15.84 | 405000 | 54.28 |
| 420000 | 178.59 | 420000 | 55.3 | 420000 | 16.25 | 420000 | 16.44 | 420000 | 55.45 |
| 435000 | 183.69 | 435000 | 57.98 | 435000 | 17.57 | 435000 | 17.63 | 435000 | 58.11 |
| 450000 | 189.56 | 450000 | 59.41 | 450000 | 17.58 | 450000 | 17.45 | 450000 | 59.8 |
| 465000 | 196.19 | 465000 | 61.97 | 465000 | 18.15 | 465000 | 18.11 | 465000 | 61.52 |
| 480000 | 203.12 | 480000 | 64.35 | 480000 | 19.21 | 480000 | 19.2 | 480000 | 64.65 |
| 495000 | 210.83 | 495000 | 65.83 | 495000 | 19.28 | 495000 | 19.03 | 495000 | 65.96 |
| 510000 | 215.23 | 510000 | 67.37 | 510000 | 20.2 | 510000 | 20.23 | 510000 | 67.36 |
| 525000 | 222.91 | 525000 | 69.52 | 525000 | 20.29 | 525000 | 20.35 | 525000 | 69.87 |
| 540000 | 228.9 | 540000 | 71.94 | 540000 | 20.74 | 540000 | 20.8 | 540000 | 71.64 |
| 555000 | 234.29 | 555000 | 74.13 | 555000 | 22.48 | 555000 | 22.19 | 555000 | 74.37 |
| 570000 | 240.6 | 570000 | 75.33 | 570000 | 21.91 | 570000 | 21.85 | 570000 | 75.54 |
| 585000 | 246.79 | 585000 | 77.39 | 585000 | 22.54 | 585000 | 22.61 | 585000 | 77.91 |
| 600000 | 253.85 | 600000 | 80.15 | 600000 | 23.73 | 600000 | 23.77 | 600000 | 80.06 |

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

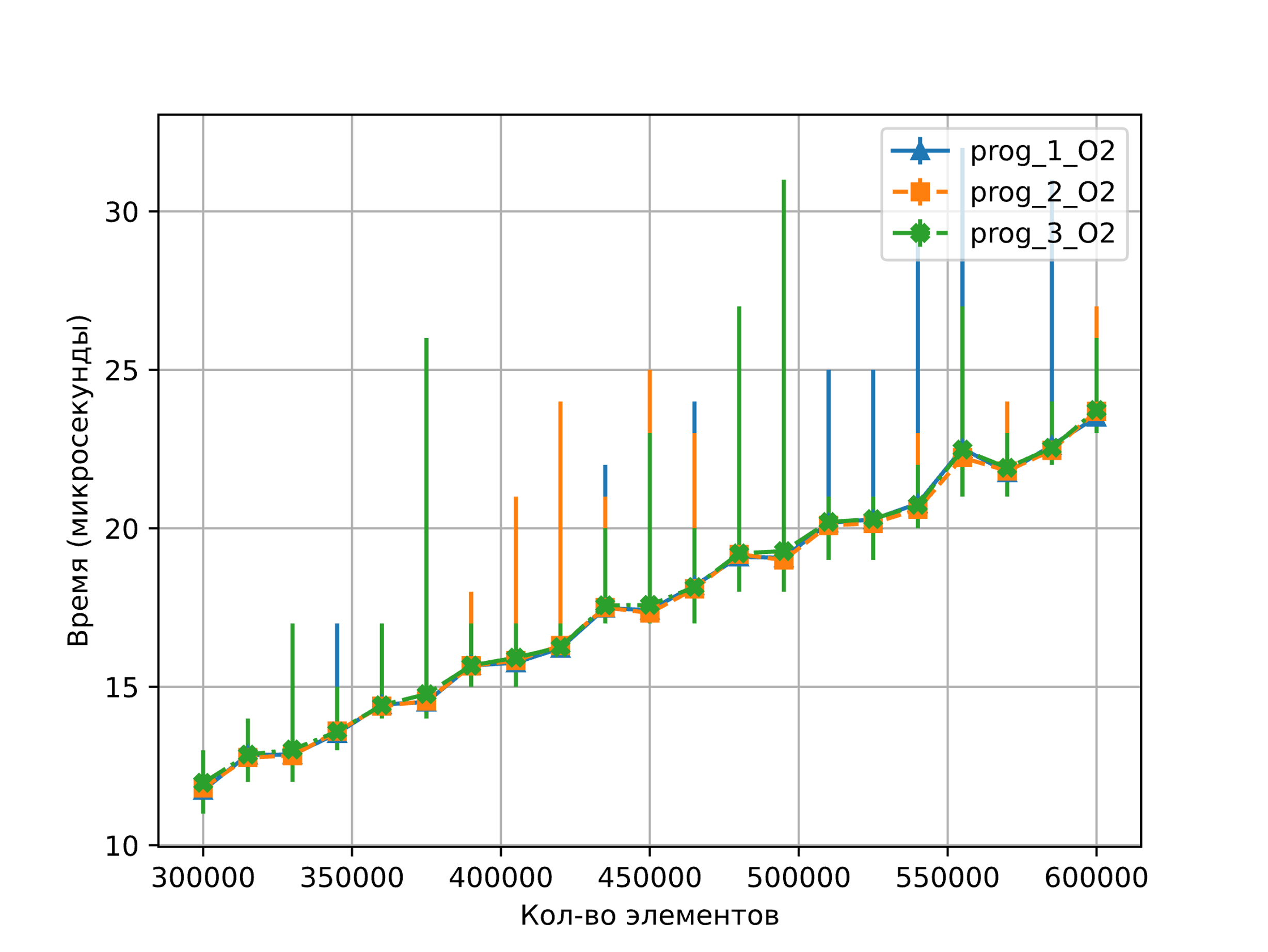


Таблица к кусочно-линейному графику ошибок

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Prog\_1\_O2 | | | | Prog\_2\_O2 | | | | Prog\_3\_O2 | | | |
| col\_elems | avg | max | min | col\_elems | avg | max | min | col\_elems | avg | max | min |
| 300000 | 11.74 | 13 | 11 | 300000 | 11.81 | 13 | 11 | 300000 | 11.97 | 13 | 11 |
| 315000 | 12.85 | 14 | 12 | 315000 | 12.77 | 14 | 12 | 315000 | 12.86 | 14 | 12 |
| 330000 | 12.86 | 14 | 12 | 330000 | 12.83 | 14 | 12 | 330000 | 13.02 | 17 | 12 |
| 345000 | 13.53 | 17 | 13 | 345000 | 13.6 | 15 | 13 | 345000 | 13.59 | 15 | 13 |
| 360000 | 14.44 | 16 | 14 | 360000 | 14.39 | 16 | 14 | 360000 | 14.42 | 17 | 14 |
| 375000 | 14.52 | 16 | 14 | 375000 | 14.55 | 16 | 14 | 375000 | 14.77 | 26 | 14 |
| 390000 | 15.67 | 17 | 15 | 390000 | 15.66 | 18 | 15 | 390000 | 15.67 | 17 | 15 |
| 405000 | 15.76 | 18 | 15 | 405000 | 15.83 | 21 | 15 | 405000 | 15.92 | 17 | 15 |
| 420000 | 16.22 | 18 | 16 | 420000 | 16.3 | 24 | 16 | 420000 | 16.25 | 17 | 16 |
| 435000 | 17.48 | 22 | 17 | 435000 | 17.5 | 21 | 17 | 435000 | 17.57 | 20 | 17 |
| 450000 | 17.42 | 19 | 17 | 450000 | 17.33 | 25 | 17 | 450000 | 17.58 | 23 | 17 |
| 465000 | 18.19 | 24 | 17 | 465000 | 18.09 | 23 | 17 | 465000 | 18.15 | 20 | 17 |
| 480000 | 19.1 | 21 | 18 | 480000 | 19.18 | 23 | 18 | 480000 | 19.21 | 27 | 18 |
| 495000 | 19.07 | 21 | 18 | 495000 | 19 | 22 | 18 | 495000 | 19.28 | 31 | 18 |
| 510000 | 20.17 | 25 | 19 | 510000 | 20.09 | 21 | 19 | 510000 | 20.2 | 21 | 19 |
| 525000 | 20.28 | 25 | 20 | 525000 | 20.16 | 21 | 19 | 525000 | 20.29 | 21 | 19 |
| 540000 | 20.78 | 29 | 20 | 540000 | 20.6 | 23 | 20 | 540000 | 20.74 | 22 | 20 |
| 555000 | 22.52 | 32 | 21 | 555000 | 22.23 | 25 | 21 | 555000 | 22.48 | 27 | 21 |
| 570000 | 21.76 | 23 | 21 | 570000 | 21.81 | 24 | 21 | 570000 | 21.91 | 23 | 21 |
| 585000 | 22.62 | 31 | 22 | 585000 | 22.46 | 24 | 22 | 585000 | 22.54 | 24 | 22 |
| 600000 | 23.51 | 25 | 23 | 600000 | 23.69 | 27 | 23 | 600000 | 23.73 | 26 | 23 |

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

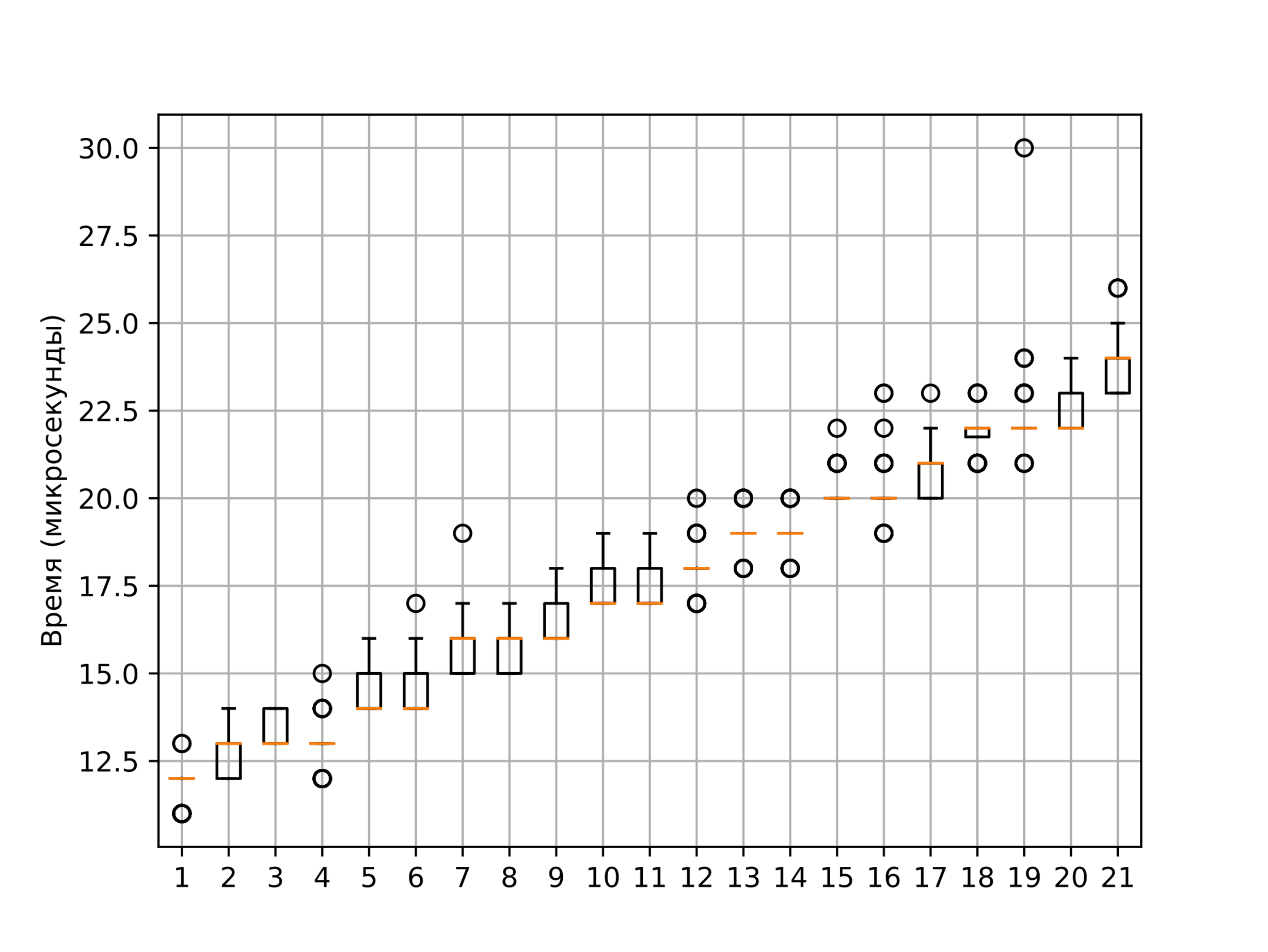


Таблица для графика с усами

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| data 1 O3 | | | | | | |
| col\_elems | avg | med | min | max | up\_quart | low\_quart |
| 300000 | 12 | 13 | 12 | 12 | 12 | 12 |
| 315000 | 12 | 13 | 13 | 13 | 13 | 13 |
| 330000 | 13 | 14 | 13 | 13 | 13 | 13 |
| 345000 | 13 | 13 | 13 | 13 | 13 | 13 |
| 360000 | 14 | 14 | 14 | 14 | 15 | 14 |
| 375000 | 15 | 14 | 16 | 14 | 15 | 14 |
| 390000 | 16 | 16 | 16 | 16 | 16 | 15 |
| 405000 | 16 | 16 | 16 | 15 | 16 | 16 |
| 420000 | 17 | 17 | 16 | 16 | 16 | 16 |
| 435000 | 18 | 17 | 18 | 17 | 18 | 17 |
| 450000 | 18 | 17 | 17 | 17 | 18 | 17 |
| 465000 | 18 | 18 | 18 | 17 | 19 | 17 |
| 480000 | 19 | 20 | 19 | 19 | 19 | 20 |
| 495000 | 19 | 19 | 19 | 19 | 19 | 19 |
| 510000 | 21 | 21 | 20 | 20 | 21 | 19 |
| 525000 | 20 | 21 | 20 | 20 | 20 | 20 |
| 540000 | 21 | 21 | 20 | 21 | 21 | 20 |
| 555000 | 22 | 23 | 22 | 22 | 22 | 22 |
| 570000 | 22 | 22 | 22 | 21 | 22 | 22 |
| 585000 | 23 | 24 | 22 | 22 | 23 | 23 |

## Таблица для результатов обрабботки

Таблица для результатов обработки «через квадратные скобки» с уровнем оптимизации O2 со столбцами: длина массива n, время выполнения tn, величина для всех строк, кроме последней.

|  |  |  |
| --- | --- | --- |
| N | T | Заданная величина |
| 300000 | 11.00 | 3 |
| 315000 | 13.00 | 0 |
| 330000 | 13.00 | 0 |
| 345000 | 13.00 | 2 |
| 360000 | 14.00 | 0 |
| 375000 | 14.00 | 3 |
| 390000 | 16.00 | -2 |
| 405000 | 15.00 | 2 |
| 420000 | 16.00 | 2 |
| 435000 | 17.00 | 0 |
| 450000 | 17.00 | 2 |
| 465000 | 18.00 | 5 |
| 480000 | 21.00 | -2 |
| 495000 | 20.00 | 3 |
| 510000 | 22.00 | 0 |
| 525000 | 22.00 | -2 |
| 540000 | 21.00 | 2 |
| 555000 | 22.00 | 0 |
| 570000 | 22.00 | 2 |
| 585000 | 23.00 | 2 |
| 600000 | 24.00 |  |

## Ответы на вопросы

### Какой способ обработки быстрее и почему?

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

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

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

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

Так как известна причина выброса, то можно.

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

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

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

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

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

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