Смирнов Иван ИУ7-22Б - 2023г.

Отчет

Задание №4

Исследование характеристик программного обеспечения

Целью работы является изучение расположения в памяти локальных переменных и представления структур.

Задание №1

По условию задания был проведен эксперимент замера среднего времени выполнения функции nanosleep для задержки в 1с, 100мс, 50мс, 10мс разными методами (gettimeofday, clock_gettime, clock, __rdtsc). Для этого было создано несколько программ с разными методами замера времени (см. папку /task_4/1):

Вариант 1 – **gettimeofday (main-1.c)**

```c
#include <stdio.h>
#include <time.h>
#include <sys/time.h>

#define OK 0

struct timespec
{
   time_t tv_sec;
   long tv_nsec;
};

int nanosleep(const struct timespec *req, struct timespec *rem);

// Замерительный метод - gettimeofday
int main (void)
{
   struct timespec tw = {0,10*1e+6};
   struct timespec tr;

   struct timeval current_time;
   unsigned long long beg, end;

   gettimeofday(&current_time, NULL);

   beg = current_time.tv_sec * 1000ULL + current_time.tv_usec / 1000ULL;
   nanosleep(&tw, &tr);
   gettimeofday(&current_time, NULL);
   end = current_time.tv_sec * 1000ULL + current_time.tv_usec / 1000ULL;
```

```
   printf("%llu\n", end-beg);

   return OK;
}
```

Вариант 2 – **clock_gettime (main-2.c)**

```
#define _POSIX_C_SOURCE 199309L

#include <time.h>
#include <stdio.h>
#include <unistd.h>

#define OK 0

// Замерительный метод - clock_gettime
int main (void)
{
   struct timespec tw = {0,10*1e+6};
   struct timespec tr;

   struct timespec start, ending;
   unsigned long long beg, end;

   clock_gettime( CLOCK_REALTIME, &start );

   beg = start.tv_sec * 1000ULL + start.tv_nsec / 1000ULL;
   nanosleep(&tw, &tr);
   clock_gettime( CLOCK_REALTIME, &ending );
   end = ending.tv_sec * 1000ULL + ending.tv_nsec / 1000ULL;

   printf("%llu\n", end-beg);

   return OK;
}
```

Вариант 3 – **clock (main-3.c)**

```
#include <time.h>
#include <stdio.h>
#include <unistd.h>

#define OK 0

struct timespec
{
   time_t tv_sec;
```

```
    long tv_nsec;
};

int nanosleep(const struct timespec *req, struct timespec *rem);

// Замерительный метод - clock()
int main (void)
{
    struct timespec tw = {0,10*1e+6};
    struct timespec tr;

    double time_spent = 0;

    clock_t begin = clock();
    nanosleep(&tw, &tr);
    clock_t end = clock();

    time_spent += (double)(end - begin) * 1000ULL / CLOCKS_PER_SEC;
    printf("%f\n", time_spent);

    return OK;
}
```

Вариант 4 – **__rdtsc (main-4.c)**

```
#include <time.h>
#include <stdio.h>
#include <unistd.h>
#include <x86gprintrin.h>

#define OK 0

struct timespec
{
    time_t tv_sec;
    long tv_nsec;
};

int nanosleep(const struct timespec *req, struct timespec *rem);

// Замерительный метод - __rdtsc()
int main (void)
{
    struct timespec tw = {0,10*1e+6};
    struct timespec tr;

    unsigned long long t1 = __rdtsc();
    nanosleep(&tw, &tr);
```

```
    unsigned long long t2 = __rdtsc();

    printf("%llu\n", (t2 - t1)/CLOCKS_PER_SEC);

    return OK;
}
```

Так же в рабочей папке присутствуют скрипты для удобства работы с измерениями.

build_release.sh ($key) – скрипт сборки программы с ключом (ключ указывает на номер собираемой программы, например, команда **./build_release.sh 3** соберет программу main-3.c).

```
#!/bin/bash

con="-"
if [ -z "$1" ]
then
    key=""
else
    key="${con}$1"
fi

gcc -std=c99 -Wall -Werror -Wpedantic -Wextra -Wfloat-equal -Wfloat-conversion -
Wvla -c main"$key".c
gcc main"$key".o -o app.exe -lm
```

dataset.sh ($key) – скрипт, который выполняет 20 тестовых измерений времени у программы (номер программы определяется ключом). Результаты измерений помещаются в текстовые файлы вида **./dataset/t_$key_$i.txt**, где i – номер измерения (теста).

```
#!/bin/bash

if [ -z "$1" ]
then
    echo "please, enter programm key!"
else
    ./build_release.sh "$1"
    for (( i=1; i <= 20; i++ ))
    do
        echo "test-'$i'"
        rm -f ./dataset/t_"$1"_"$i".txt
        ./app.exe >> ./dataset/t_"$1"_"$i".txt
    done
    echo "dataset done!"
fi
```

get_avg.sh ($key) **–** скрипт, который на основе данных, полученных из **dataset.sh**, считает среднее арифметическое измерений и записывает его в файл **./avg/t_$key.txt**.

```
#!/bin/bash

if [ -f "./dataset/t_'$1'_1.txt" ]
then
    summ=0
    n=0
    for (( i=1; i <= 20; i++ ))
    do
        while read -r n; do
            summ=$(( summ + n ))
        done < "./dataset/t_'$1'_'$i'.txt"
    done
    avg=$(( summ / 20 ))
    rm -f ./avg/t_"$1".txt
    echo "$avg" >> ./avg/t_"$1".txt
else
    echo "please, enter programm key!"
fi
```

Вспомогательные скрипты **check_scripts.sh** (проверка shellcheck всех скриптов), **chmod.sh** (выдача права на изменение для всех скриптов) и **clean.sh** (очистка временных файлов, а также текстовых файлов из каталогов ./dataset и ./avg) были взяты из лабораторных работ курса “Программирование на языке Си”. Данные 3 скрипта также будут использованы при решении “Задания 2”.

Ниже приведена таблица измерений (время измеряется в мс):

Реальное время	gettimeofday	clock_gettime	clock	__rdtsc
1000	1000	14060	0.8	2501
100	102	7377	0.05	255
50	53	1035	0.07	126
10	12	13625	0.02	31

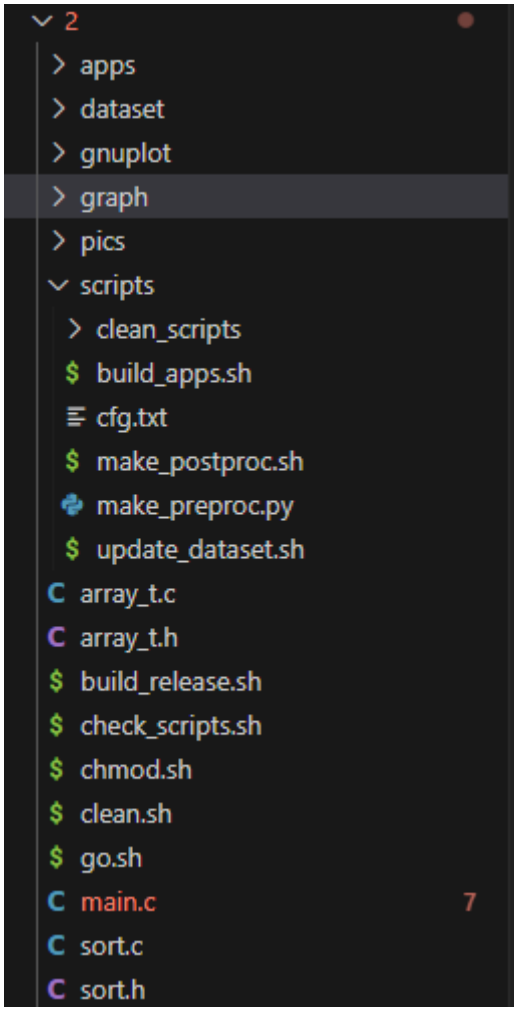
Как видно из таблицы наиболее точный метод – **gettimeofday.** Именно этот метод был использован при решении задания №2.

Метод __rdtsc имеет погрешность результата почти в 1.5 раза во всех рассматриваемых случаях. То есть присутствует закономерность, однако смещение в 1.5 раза - достаточно большое.

Методы clock_gettime и clock (из данных таблицы) являются самыми нестабильными. На разных машинах (Ubuntu 22/ Windows 11) функции выдают разные результаты. Clock() сообщает, сколько процессорного времени используется; процессорное время в свою очередь зависит от количества активных потоков (которое в разные состояния работы компьютера может быть разным). Из-за этого пользователь получает не тот результат, который хотел бы увидеть.

Задание №2

В данном задании необходимо проанализировать время выполнения трех разных реализаций сортировки выбором (selection sort) с разными типами оптимизации (O0 и O2) на случайно-отсортированном массиве, а также на отсортированном в прямом порядке.



(рис.1 – файловая структура задания №2)

Разные реализации сортировки основаны на разном обращении к элементам массива. Далее приведен код (на языке Си) каждой из реализаций.

1) Использование операции индексации a[i]

```c
// Реализация 1
// Функция находит минимум в диапазоне от strat_ind до end_ind
size_t find_min_elem1(const int a[], size_t start_ind, size_t end_ind)
{
    size_t min_elem = start_ind;
    for (size_t j = start_ind; j < end_ind; j++)
    {
        if (a[j] < a[min_elem])
        {
            min_elem = j;
        }
    }
    return min_elem;
}
// Функция сортировки выбором
void selection_sort1(int *a, size_t a_size)
{
    size_t min_elem = 0;
    int tmp_elem = 0;
    for (size_t i = 0; i < a_size; i++)
    {
        min_elem = find_min_elem1(a, i, a_size);
        tmp_elem = a[i];
        a[i] = a[min_elem];
        a[min_elem] = tmp_elem;
    }
}
```

2) Использование замены операции индексации на выражение *(a + i)

```c
// Реализация 2
size_t find_min_elem2(const int a[], size_t start_ind, size_t end_ind)
{
    size_t min_elem = start_ind;
    for (size_t j = start_ind; j < end_ind; j++)
    {
        if (*(a + j) < *(a + min_elem))
        {
            min_elem = j;
        }
    }
    return min_elem;
}

void selection_sort2(int *a, size_t a_size)
{
```

```
    size_t min_elem = 0;
    int tmp_elem = 0;
    for (size_t i = 0; i < a_size; i++)
    {
        min_elem = find_min_elem2(a, i, a_size);
        tmp_elem = *(a + i);
        *(a + i) = *(a + min_elem);
        *(a + min_elem) = tmp_elem;
    }
}
```

3) Использование указателей для работы с массивом

```
// Реализация 3
void selection_sort3(int *pb, int *pe)
{
    int min_elem = 0;
    int tmp_elem = *pb;
    for (int i = 0; i < (pe - pb); i++)
    {
        min_elem = i;
        for (int j = i; j < (pe - pb); j++)
        {
            //printf("l:%dr:%d\n", *(pb + j), min_elem);
            if (*(pb + j) < *(pb + min_elem))
            {
                min_elem = j;
            }
        }
        tmp_elem = *(*(&(pb)) + i);
        *(pb + i) = *(pb + min_elem);
        *(pb + min_elem) = tmp_elem;
    }
}
```

Все реализации находятся в файле *sort.c*. Описание каждой из реализаций можно прочитать в заголовочном файле *sort.h*.

Также были реализованы функции генерации списка случайно-отсортированных чисел (*init*), а также уже отсортированного (*init sorted*). Описание функций находится в заголовочном файле *array_t.h*; сами функции в *array_t.c*.

```
void init(int *a, int size)
{
    srand(time(NULL));
    for (int i = 0; i < size; i++)
        a[i] = (rand() % (size * 3));
}
```

```
void init_sorted(int *a, int size)
{
    for (int i = 0; i < size; i++)
        a[i] = i;
}
```

Листинг основной программы (***main.c***):

```
#include "array_t.h"
#include "sort.h"

#define OK 0
#define ERR_IO 1

#ifndef NMAX
#error NMAX IS NOT DEFINED
#endif
#ifndef SORT_TYPE
#error TYPE IS NOT DEFINED
#endif
#ifndef ARR_TYPE
#error ARR_TYPE IS NOT DEFINED
#endif

typedef int array_t[NMAX];

struct timespec
{
   time_t tv_sec;
   long tv_nsec;
};

int nanosleep(const struct timespec *req, struct timespec *rem);

// Замерительный метод - gettimeofday
int main(int argc, char **argv)
{
    if (argc != 2)
        return ERR_IO;
    array_t a;
    int sort_type = SORT_TYPE;
    int size = atoi(argv[1]);
    int array_type = ARR_TYPE;

    switch (array_type)
    {
        case 1:
            init(a, size);
```

```
            break;
        case 2:
            init_sorted(a, size);
            break;
    }

    struct timeval current_time;
    unsigned long long beg, end;

    // Замер начала
    gettimeofday(&current_time, NULL);
    beg = current_time.tv_sec * 1000ULL + current_time.tv_usec / 1000ULL;
    // Действия
    switch(sort_type)
    {
        case 1:
            selection_sort1(a, size);
            break;
        case 2:
            selection_sort2(a, size);
            break;
        case 3:
            selection_sort3(a, (a + size));
            break;
        default:
            return ERR_IO;
    }
    // Замер конца
    gettimeofday(&current_time, NULL);
    end = current_time.tv_sec * 1000ULL + current_time.tv_usec / 1000ULL;

    printf("%llu\n", (end - beg));

    return OK;
}
```

Программу необходимо запустить с одним аргументом, который отвечает за размер массива (в программе это переменная size). Такие переменные как максимальный размер статического массива (NMAX), тип сортировки (SORT_TYPE), тип массива (ARR_TYPE) определяются на этапе сборки. Программа выводит на экран время работы выбранной сортировки в миллисекундах (0.001 с.).

Чтобы проанализировать время для разного размера массива, разных типов оптимизации и разных реализаций сортировок, была написана целая анализирующая система. Необходимо запустить скрипт ***go.sh***.

```
#!/bin/bash

./scripts/build_apps.sh
./scripts/update_dataset.sh
./scripts/clean_scripts/clean_graph.sh
python3 ./scripts/make_preproc.py
./scripts/make_postproc.sh
```

Проанализируем работу скрипта. **(1)** Сначала из папки scripts запускается скрипт ***build_apps.sh***, который собирает все возможные версии программы (с двумя типами компиляции, двумя типами списков и тремя типами сортировок, то есть собирается в сумме 2*2*3=12 различных файлов).

build_apps.sh:

```
#!/bin/bash

cd ./scripts || exit
./clean_scripts/clean_apps.sh
i=0
while read -r line; do
    if [ $i -eq 0 ]; then
        opt=$line
    fi
    # if [ $i -eq 1 ]; then
    #     tests=$line
    # fi
    if [ $i -eq 2 ]; then
        sort_types=$line
    fi
    if [ $i -eq 3 ]; then
        nmax=$line
    fi
    if [ $i -eq 4 ]; then
        array_types="${line}"
    fi
    i=$((i + 1))
done < ./cfg.txt
# opt - O0; O2
# sort_type: 1 - индексация; 2 - *(a + i); 3 - указатели
# array_type: 1 - случайный; 2 - отсортированный
cd ..
for option in $opt; do
    for sort_type in $sort_types; do
        for array_type in $array_types; do
            ./build_release.sh "$option" "$sort_type" "$array_type" "$nmax"
        done
    done
```

```
done
```

Вся информация о сборке программы хранятся в файле ***./scripts/cfg.txt***

cfg.txt:

O0 O2

20

1 2 3

10000

1 2

1 10 50 100 250 500 1000 2000 3000 4000 5000 6000 7000 8000 9000 10000

В первой строке написаны типы оптимизации (O0 O2). Во второй строке написано количество проводимых измерений (тестов), то есть сколько раз запустится каждый из 12 исполняемых файлов. В третьей строке расположены разные реализации сортировки (описаны ранее). В четвертой строке написано максимальное количество элементов в массиве (по условию задачи оно рано 10000). В пятой строке написаны все возможные типы массива (1 – случайный; 2 - отсортированный). В шестой строке написаны все тестируемые размеры массива.
Далее все 12 исполняемых файлов собираются с помощью скрипта ***build_release.sh:***

```
#!/bin/bash

gcc -std=c99 -Wall -Werror -Wpedantic -Wextra -Wfloat-equal -Wfloat-conversion -
Wvla -"${1}" -DSORT_TYPE="${2}" -DARR_TYPE="${3}" -DNMAX="${4}" -c main.c
gcc -std=c99 -Wall -Werror -Wpedantic -Wextra -Wfloat-equal -Wfloat-conversion -
Wvla -c array_t.c
gcc -std=c99 -Wall -Werror -Wpedantic -Wextra -Wfloat-equal -Wfloat-conversion -
Wvla -c sort.c
gcc -o ./apps/app_"$1"_s"$2"_a"$3"_n"$4".exe main.o array_t.o sort.o -lm
```

Все исполняемые файлы помещаются в папку **apps**

(2) Далее запускается скрипт ***./scripts/update_data.sh***, который запускает tests раз (в данном случае 20) на разных размерах массива каждый из исполняемых файлов. Результаты (время в мс) помещаются в текстовые файлы в папку dataset.

```
#!/bin/bash

cd ./scripts || exit
./clean_scripts/clean_dataset.sh
```

```
i=0
while read -r line; do
    if [ $i -eq 0 ]; then
        opt=$line
    fi
    if [ $i -eq 1 ]; then
        tests=$line
    fi
    if [ $i -eq 2 ]; then
        sort_types=$line
    fi
    if [ $i -eq 3 ]; then
        nmax=$line
    fi
    if [ $i -eq 4 ]; then
        arr_types="${line}"
    fi
    if [ $i -eq 5 ]; then
        sizes=$line
    fi
    i=$((i + 1))
done < ./cfg.txt
cd ..

for option in $opt; do
    for sort_type in $sort_types; do
        for arr_type in $arr_types; do
            echo "app_'${option}'_s'${sort_type}'_a'${arr_type}'_n'${nmax}'.exe -
done"
            for size in $sizes; do
                for (( i = 0; i < "$tests"; i++ )); do
                    cmd="$(./apps/app_"${option}"_s"${sort_type}"_a"${arr_type}"_
n"${nmax}".exe "${size}")"
                    echo "$cmd" >>
./dataset/"${option}"_s"${sort_type}"_a"${arr_type}"_n"${size}".txt
                done
            done
        done
    done
done
```

(3) Далее запускается скрипт ***./scripts/make_preproc.py,*** который на основе данных из **dataset** формирует новые данные, необходимые для дальнейшего анализа (среднее, минимум и максимум, медианное значение, верхний и нижний квартили). Все полученные данные скрипт сохраняет в текстовых файлах в папку ***graph.*** Перед запуском скрипта удаляется старый “graph” (если существовал) с помощью вспомогательного скрипта **clean_graph.sh.**

```
# Считываем данные из конфига
with open("./scripts/cfg.txt", "r") as f:
    line = f.readline()
    i = 0
    while (len(line) != 0):
        line = line.strip()
        if (i == 0):
            options = list(line.split())
        elif (i == 1):
            tests = int(line)
        elif (i == 2):
            sort_types = list(line.split())
        elif (i == 4):
            arr_types = list(line.split())
        elif (i == 5):
            sizes = list(line.split())
        line = f.readline()
        i += 1

# Обрабатываем данные
def analyze(time):
    # time.sort()
    avg = sum(time) // len(time)
    minimum = min(time)
    low_quartile = time[len(time)//4]
    median = time[len(time)//2]
    high_quartile = time[len(time)//4*3]
    maximum = max(time)
    return str(avg), str(minimum), str(low_quartile), str(median),
str(high_quartile), str(maximum)


# Считываем и обрабатываем данные из датасета
for opt in options:
    for sort_type in sort_types:
        for arr_type in arr_types:
            for size in sizes:
                time = [0]*tests
                with
open(f"./dataset/{opt}_s{sort_type}_a{arr_type}_n{size}.txt", "r") as fin:
                    for i in range(tests):
                        time[i] = int(fin.readline().strip())
                time.sort()
                data = list(analyze(time))
                with open(f"./graph/{opt}_s{sort_type}_a{arr_type}.txt", 'a') as
fout:
                    for t in time:
                        fout.write(str(str(t) + " " + " ".join(data) + " " +
str(size) + "\n"))
```

Данные из graph выглядят примерно так:

104 84 50 64 94 100 122 8000

104 84 50 64 94 100 122 8000

108 84 50 64 94 100 122 8000

122 84 50 64 94 100 122 8000

71 98 71 82 99 111 140 9000

74 98 71 82 99 111 140 9000

75 98 71 82 99 111 140 9000

78 98 71 82 99 111 140 9000

79 98 71 82 99 111 140 9000

1-ый столбец – время из dataset

2-ой столбец – полученное среднее

3-ий столбец – минимум

4-ый столбец – нижний квартиль

5-ый столбец – медианное значение

6-ой столбец – верхний квартиль

7-ой столбец – максимум

8-ой столбец – размер массива

На основе этих данных можно построить графики зависимости времени сортировки от количества элементов в массиве.\

(4) Запускается скрипт ***./scripts/make_outproc.sh,*** который строит графики и сохраняет их в образцах (файлах .gpi) в папке gnuplot.

```
#!/bin/bash

TXT="./graph/*.txt"

PREFIX="./gnuplot/gnuplot_"

./scripts/clean_scripts/clean_gnuplot.sh

# Общий кусочно-линейный график
```

```
for file in $TXT; do
    basename=$(basename "${file}")
    name=$PREFIX$basename".gpi"
    {
        echo "set output \"./gnuplot/${basename}.svg\""
        cat "./gnuplot/tmpl.txt"
        echo "set terminal svg size 1080, 720"
        echo "plot \"$file\" using 8:2 with lines title \"$basename\""
    } >> "$name"
done

# График с ошибкой
for file in $TXT; do
    basename=$(basename "${file}")
    name=$PREFIX$basename"_error.gpi"
    {
        echo "set output \"./gnuplot/${basename}_Error.svg\""
        cat "./gnuplot/tmpl.txt"
        echo "set terminal svg size 1080, 720"
        echo "plot \"$file\" using 8:2:3:7 with yerror title \"$basename Error\",
\"$file\" using 8:2 with lines title \"$basename\""
    } >> "$name"
done

# График с усами
for file in $TXT; do
    basename=$(basename "${file}")
    name=$PREFIX$basename"_Moustache.gpi"
    {
        cat "./gnuplot/tmpl.txt"
        echo "set terminal svg size 1080, 720"
        echo "set output \"./gnuplot/${basename}Moustache.svg\""
        echo "plot \"$file\" using 8:4:3:7:6 with candlesticks title \"$basename
Moustache\", \"$file\" using 8:2 with lines title \"$basename\""
    } >> "$name"
done

# Графики из задания
cd ./scripts || exit
./clean_scripts/clean_dataset.sh
i=0
while read -r line; do
    if [ $i -eq 0 ]; then
        opt=$line
    fi
    if [ $i -eq 2 ]; then
        sort_types=$line
    fi
    if [ $i -eq 4 ]; then
```

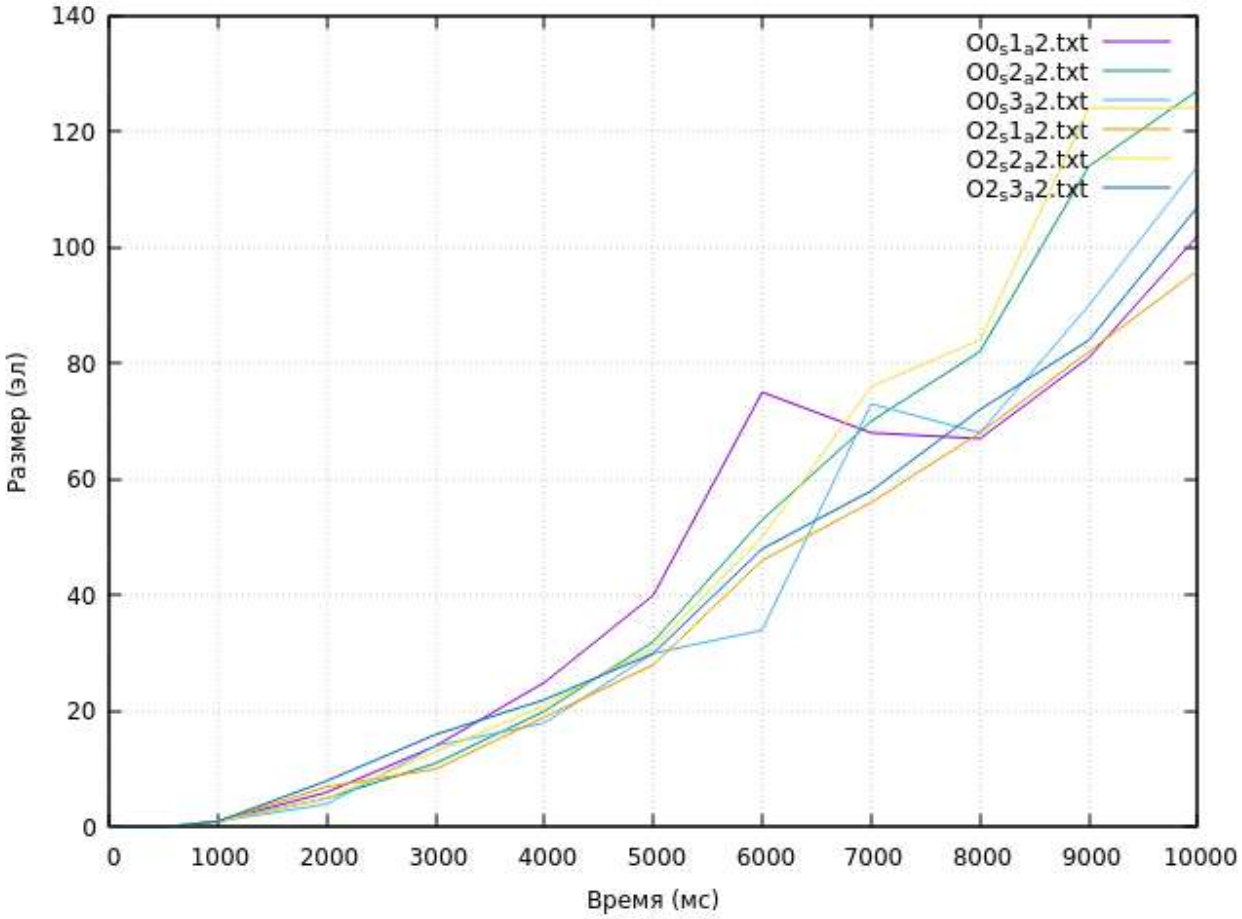
```
        arr_types="${line}"
    fi
    i=$((i + 1))
done < ./cfg.txt
cd ..

# График-1
name=$PREFIX"comp_1.gpi"
cat "./gnuplot/tmpl.txt" > "$name"
echo "set terminal svg size 1080, 720" >> "$name"
echo "set output \"./gnuplot/Comparative_Graph.svg\"" >> "$name"

count=0
for option in $opt; do
    for sort_type in $sort_types; do
        for arr_type in $arr_types; do
            file=./graph/"${option}"_s"${sort_type}"_a"${arr_type}".txt
            if [ "$arr_type" -eq 2 ]; then
                basename=$(basename "${file}")
                if [ $count -eq 0 ]; then
                    echo "plot \"$file\" using 8:2 with lines title
\"$basename\", \\" >> "$name"
                else
                    echo "\"$file\" using 8:2 with lines title \"$basename\", \\"
>> "$name"
                fi
                count=$((count + 1))
            fi
        done
    done
done

# График-2
name=$PREFIX"comp_2.gpi"
cat "./gnuplot/tmpl.txt" > "$name"
echo "set terminal svg size 1080, 720" >> "$name"
echo "set output \"./gnuplot/Comparative_Graph.svg\"" >> "$name"

count=0
for option in $opt; do
    for sort_type in $sort_types; do
        for arr_type in $arr_types; do
            file=./graph/"${option}"_s"${sort_type}"_a"${arr_type}".txt
            if [ "$arr_type" -eq 1 ]; then
                basename=$(basename "${file}")
                if [ $count -eq 0 ]; then
                    echo "plot \"$file\" using 8:2 with lines title
\"$basename\", \\" >> "$name"
                else
```

```
                    echo "\"$file\" using 8:2 with lines title \"$basename\", \\"
>> "$name"
                fi
                count=$((count + 1))
            fi
        done
    done
done

# График-3
name=$PREFIX"comp_3.gpi"
cat "./gnuplot/tmpl.txt" > "$name"
echo "set terminal svg size 1080, 720" >> "$name"
echo "set output \"./gnuplot/Comparative_Graph.svg\"" >> "$name"

count=0
for option in $opt; do
    for sort_type in $sort_types; do
        for arr_type in $arr_types; do
            file=./graph/"${option}"_s"${sort_type}"_a"${arr_type}".txt
            if [ "$option" == "O2"  ]; then
                basename=$(basename "${file}")
                if [ $count -eq 0 ]; then
                    echo "plot \"$file\" using 8:2:3:7 with yerror title
\"$basename Error\", \"$file\" using 8:2 with lines title \"$basename\"" >>
"$name"
                else
                    echo "\"$file\" using 8:2:3:7 with yerror title \"$basename
Error\", \"$file\" using 8:2 with lines title \"$basename\"" >> "$name"
                fi
                count=$((count + 1))
            fi
        done
    done
done

# График-4
name=$PREFIX"comp_4.gpi"
cat "./gnuplot/tmpl.txt" > "$name"
echo "set terminal svg size 1080, 720" >> "$name"
echo "set output \"./gnuplot/Comparative_Graph.svg\"" >> "$name"

count=0
for option in $opt; do
    for sort_type in $sort_types; do
        for arr_type in $arr_types; do
            file=./graph/"${option}"_s"${sort_type}"_a"${arr_type}".txt
            if [ "$option" == "O2"  ]; then
                basename=$(basename "${file}")
```

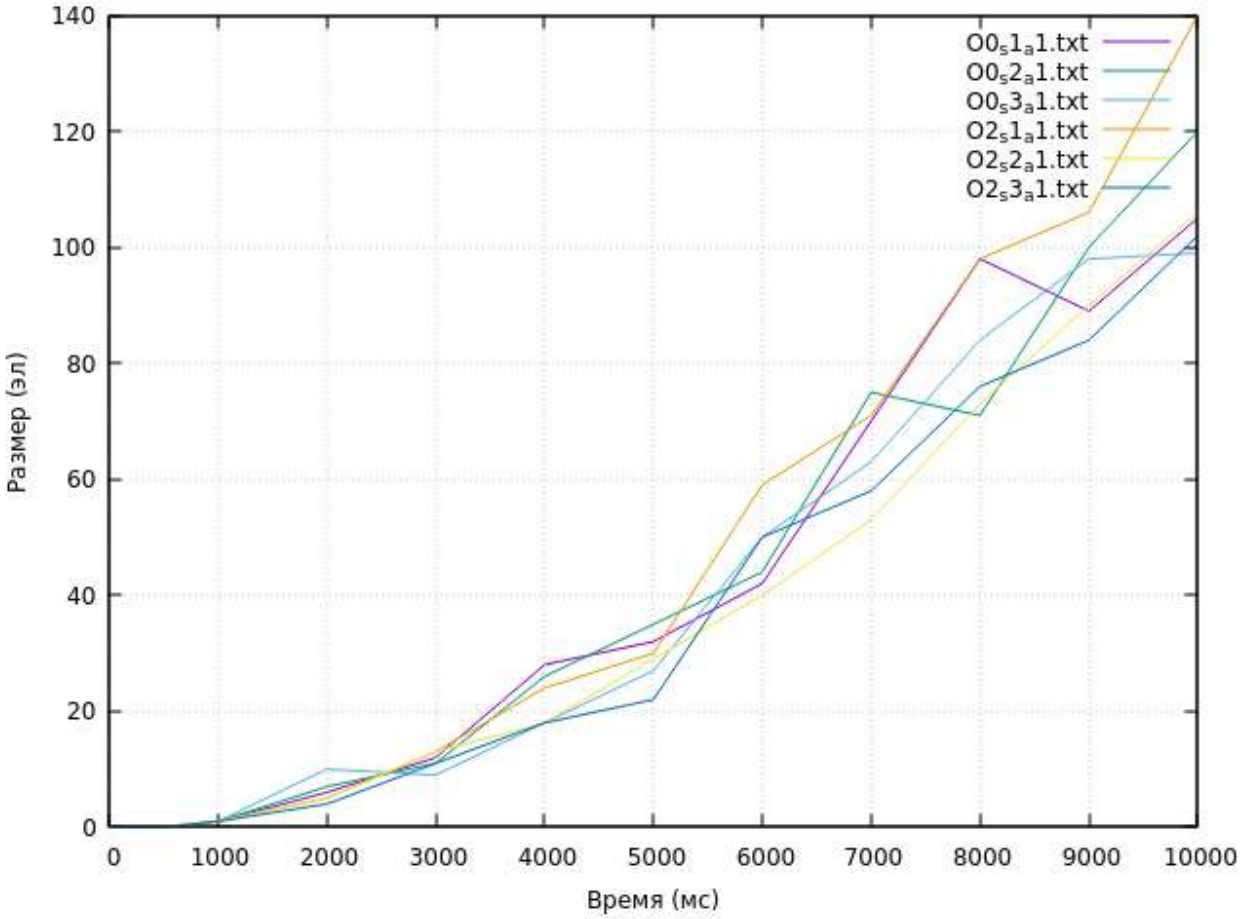
```
                if [ $count -eq 0 ]; then
                    echo "plot \"$file\" using 8:4:3:7:6 with candlesticks title
\"$basename Moustache\", \"$file\" using 8:2 with lines title \"$basename\"" >>
"$name"
                else
                    echo "\"$file\" using 8:4:3:7:6 with candlesticks title
\"$basename Moustache\", \"$file\" using 8:2 with lines title \"$basename\"" >>
"$name"
                fi
                count=$((count + 1))
            fi
        done
    done
done
# ЗАПУСТИТЬ ГРАФИК
#gnuplot $PATH$NAME".gpi" -persist
```

В конце скрипта написана команда, с помощью которой можно увидеть отображения графика в программе gnuplot (*gnuplot $PATH$NAME".gpi" -persist*).
Графики строятся следующим образом: по оси x – количество элементов (столбец 8) в массиве, по оси y – среднее замеренное время для данного количества элементов (столбец 2)Ъ.

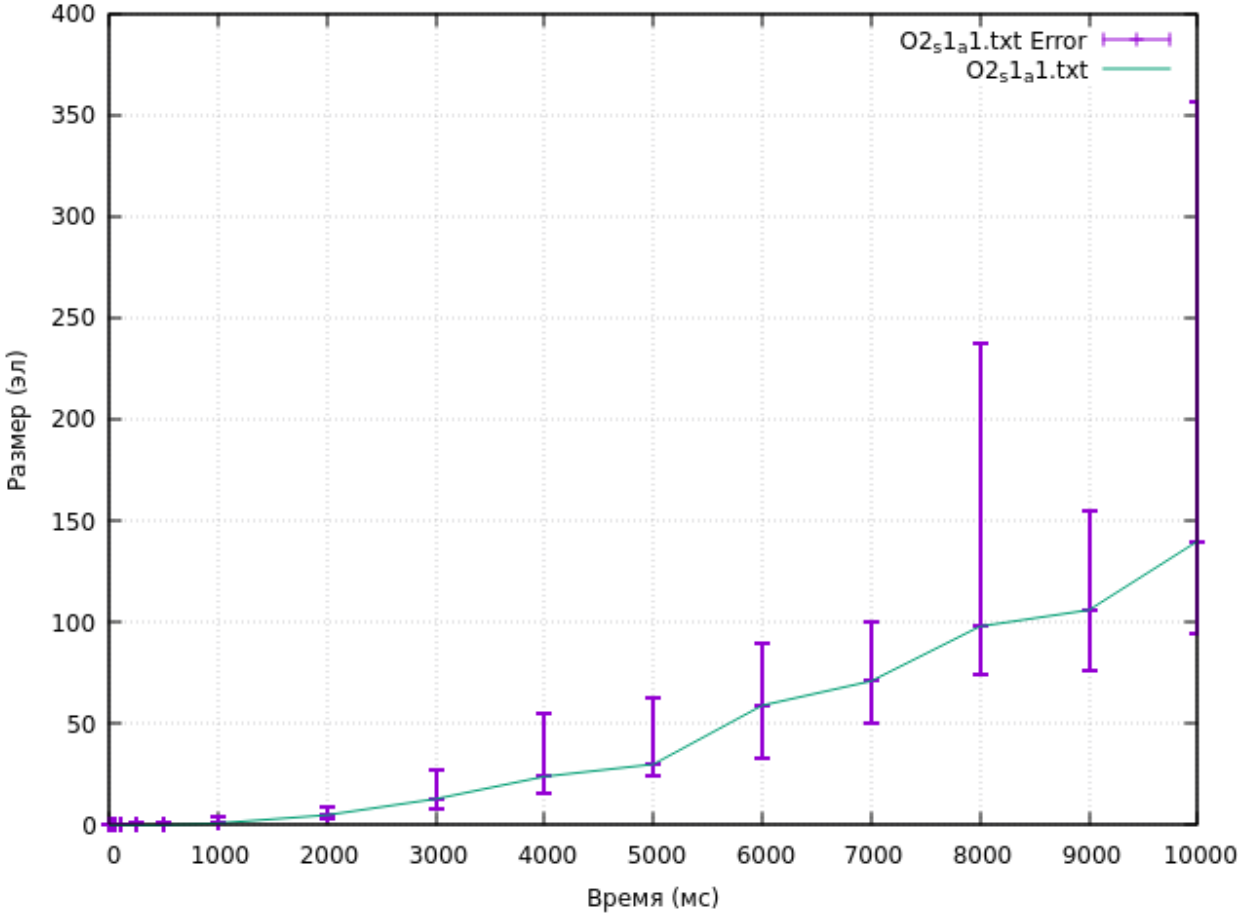
Главные графики:

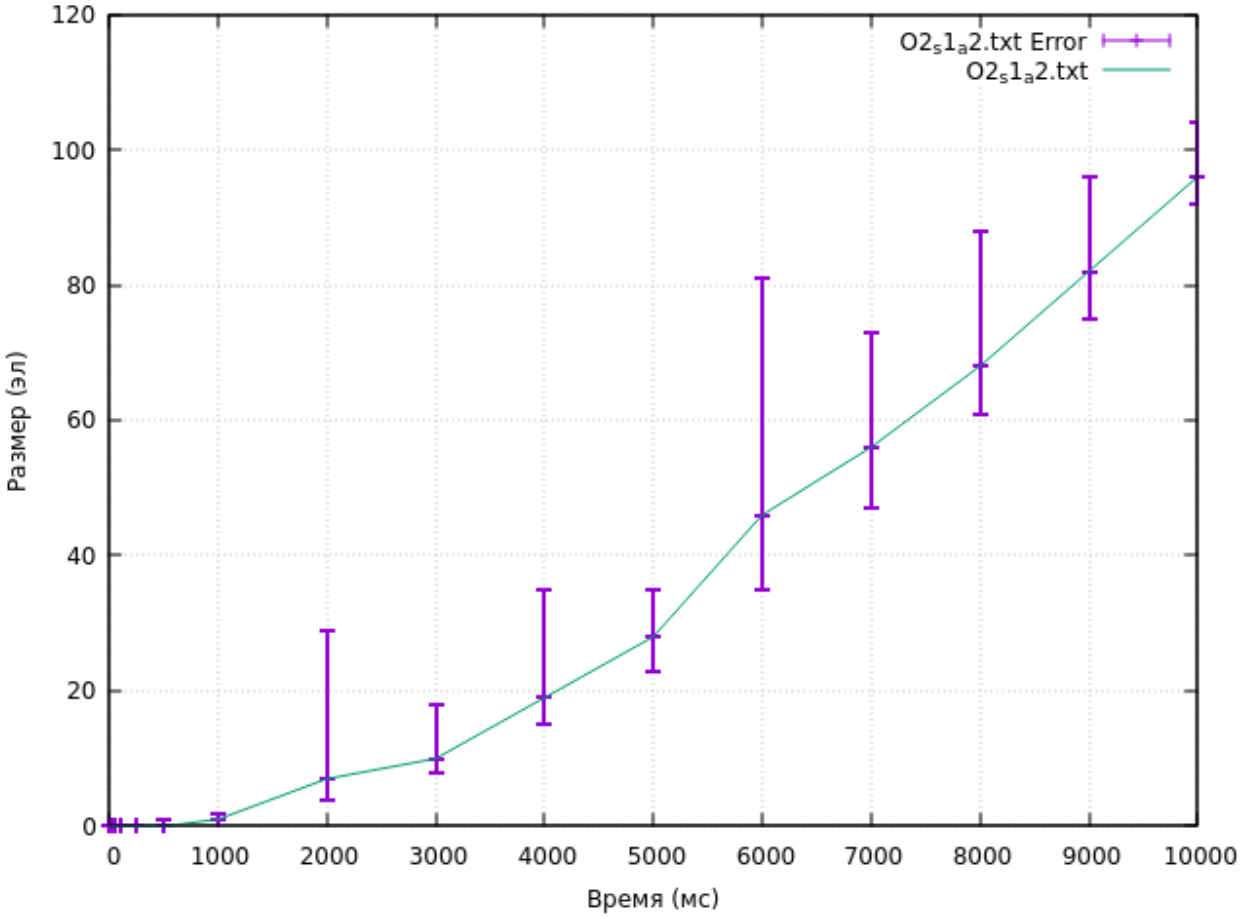


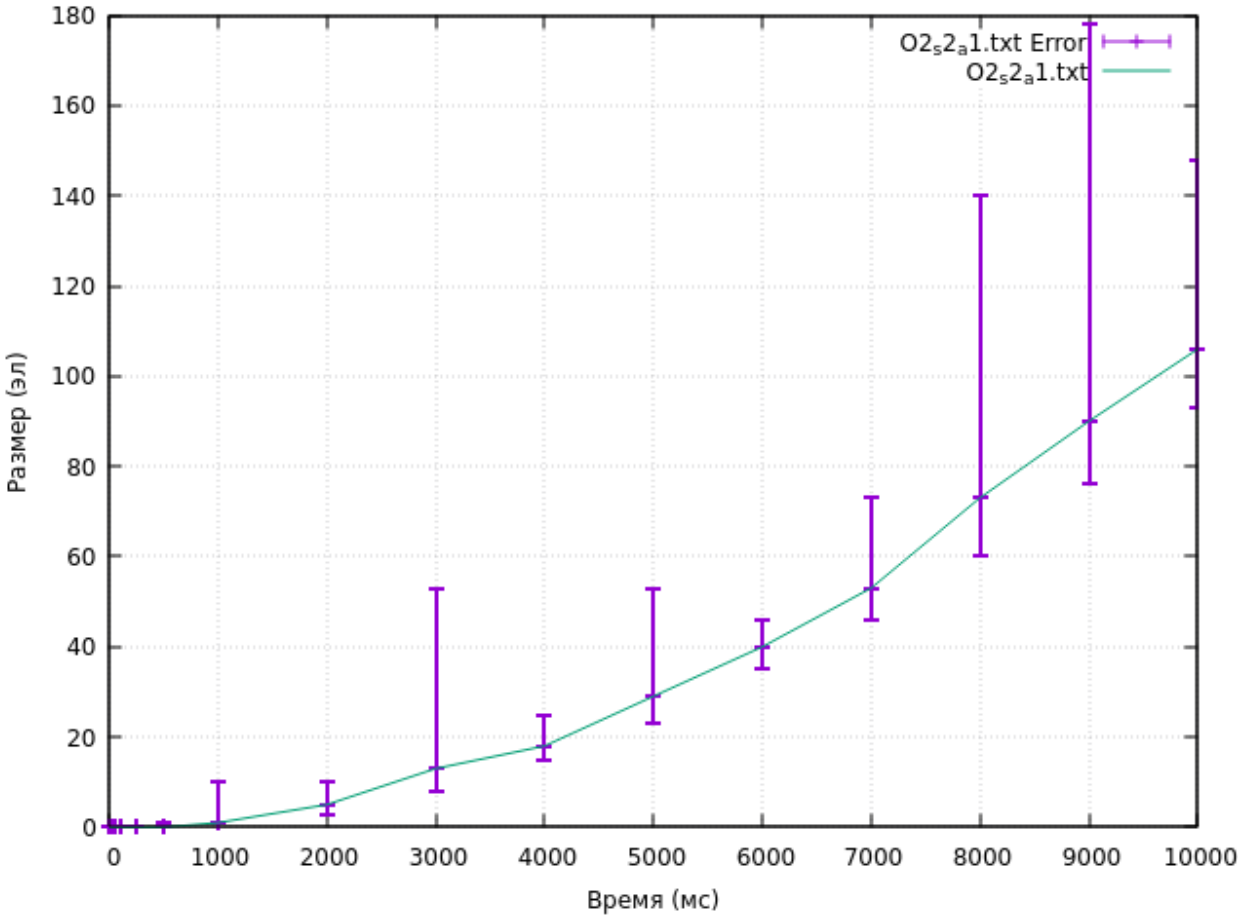
(Обычный кусочно-линейный график зависимости времени выполнения в любых единицах измерения времени от числа элементов массива для всех 6 вариантов программы, обрабатывающей наилучший случай.)

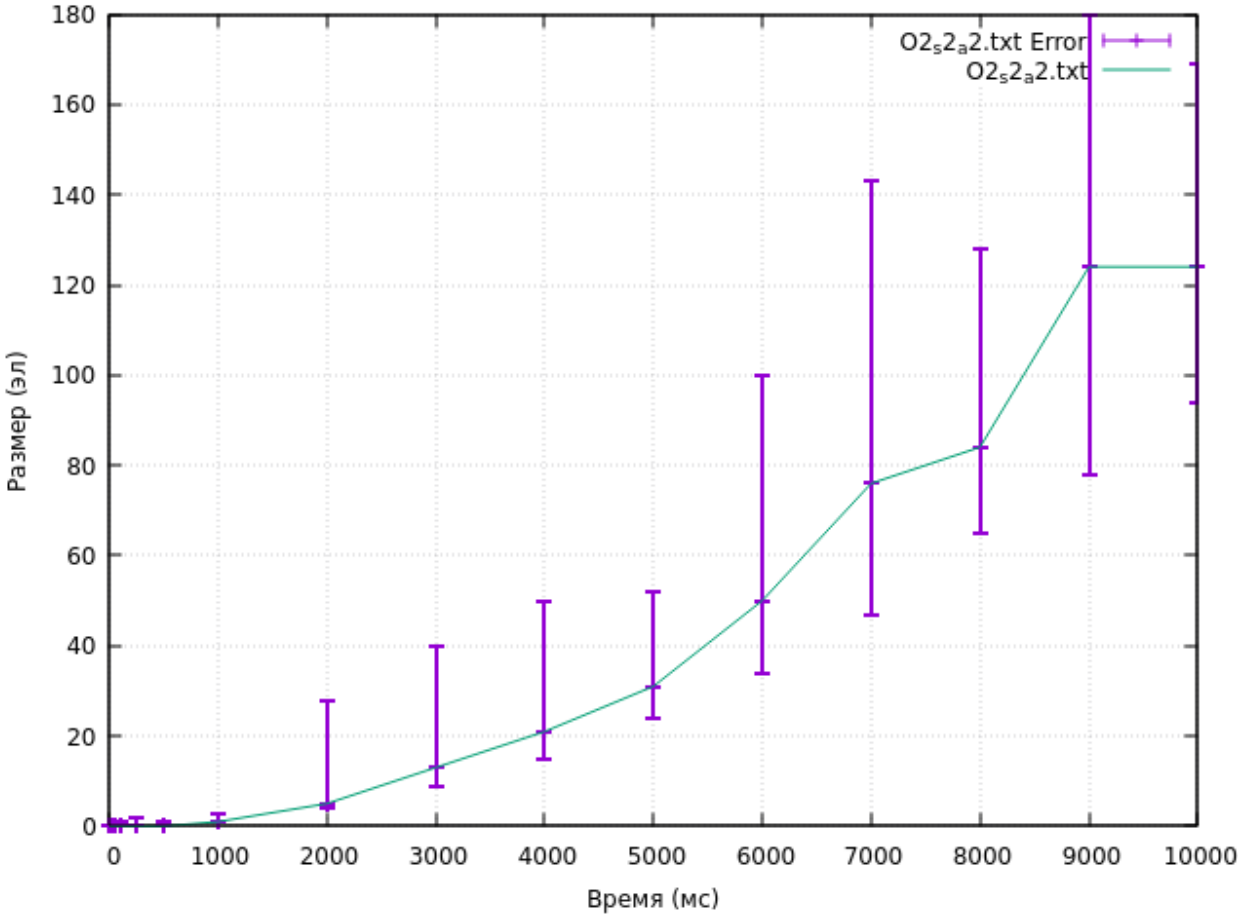


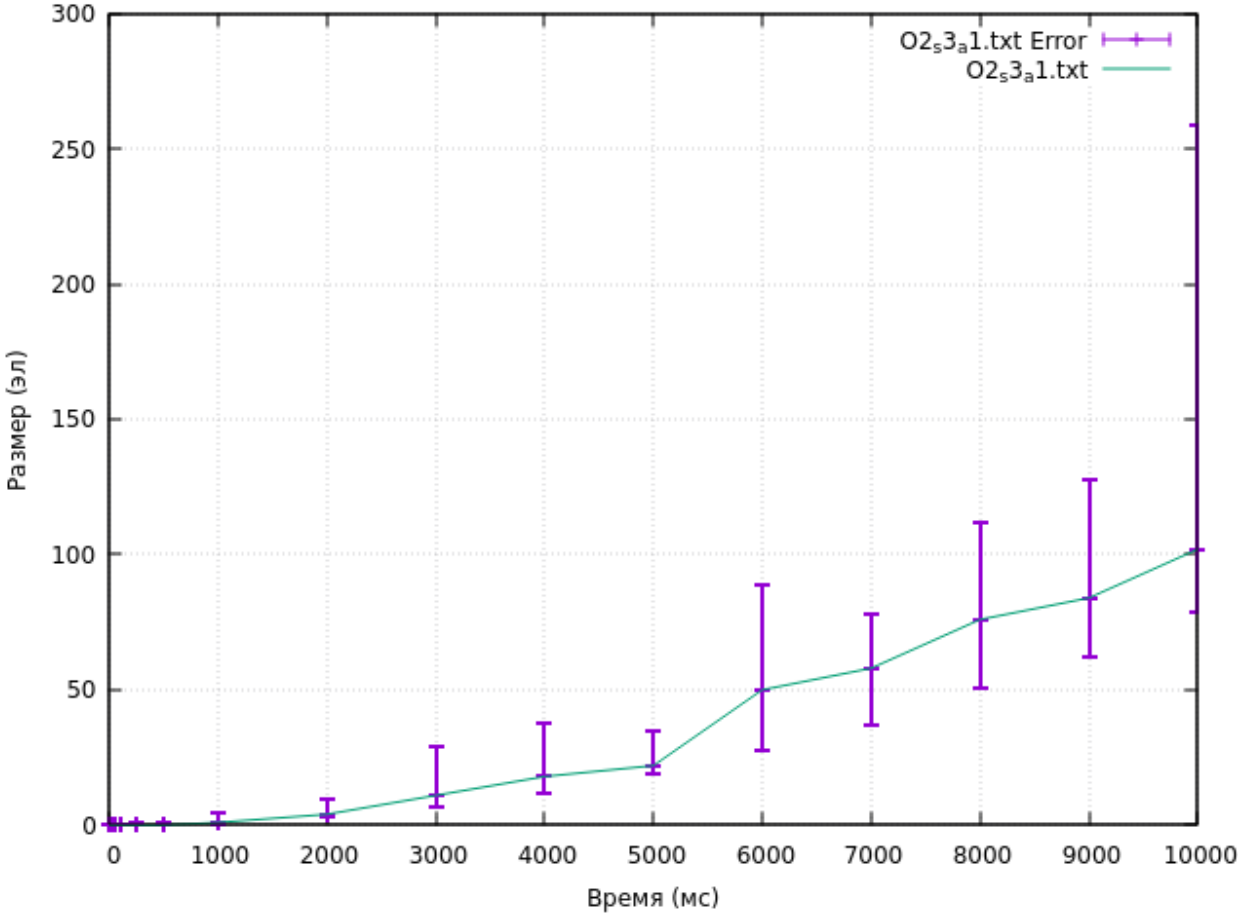
(Обычный кусочно-линейный график зависимости времени выполнения в любых единицах измерения времени от числа элементов массива для всех 6 вариантов программы, обрабатывающей массив общего вида.)

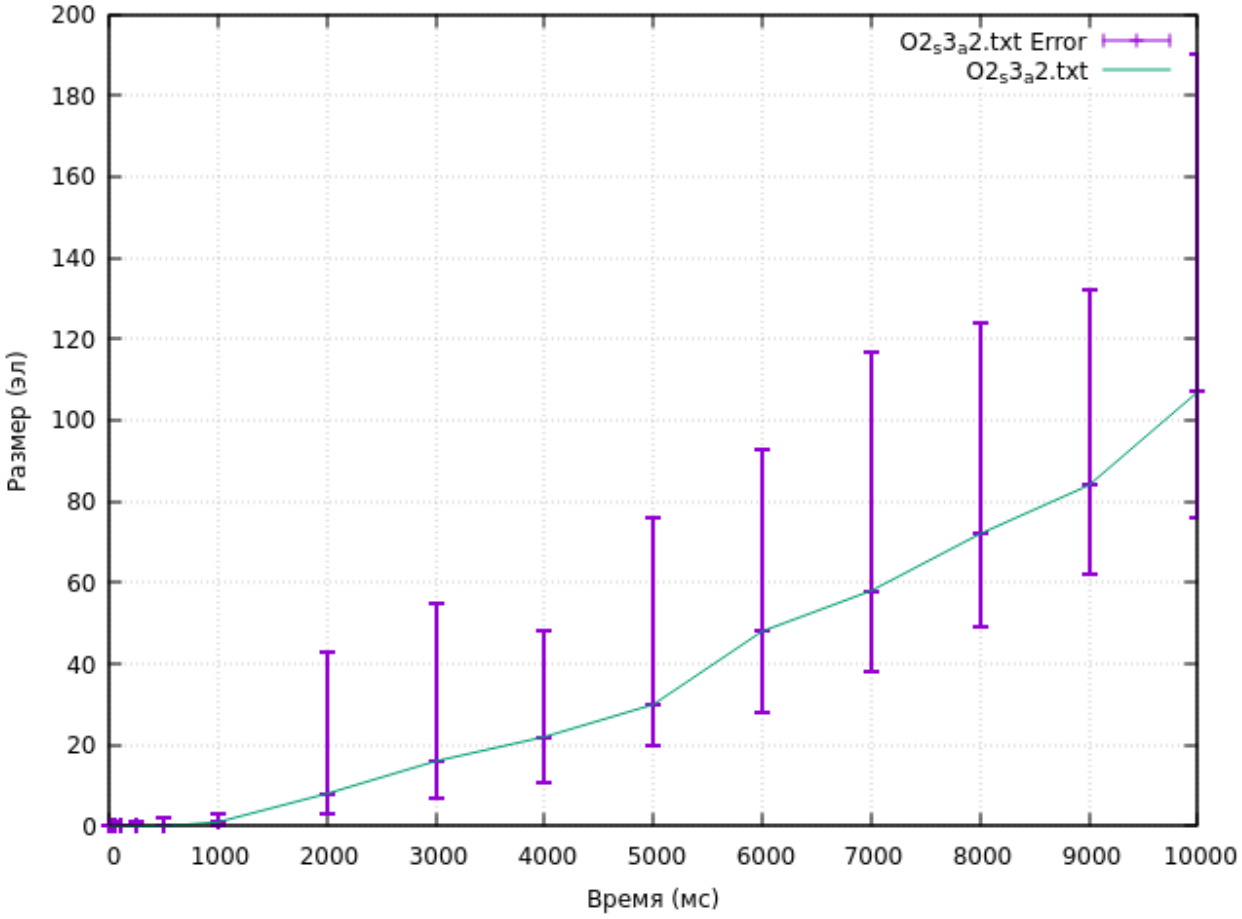

O2s1a1.txt Error
O2s1a1.txt
Размер (эл)
Время (мс)
0
50
100
150
200
250
300
350
400
0
1000
2000
3000
4000
5000
6000
7000
8000
9000
10000


O2s1a2.txt Error
O2s1a2.txt
Размер (эл)
Время (мс)
0
20
40
60
80
100
120
0
1000
2000
3000
4000
5000
6000
7000
8000
9000
10000

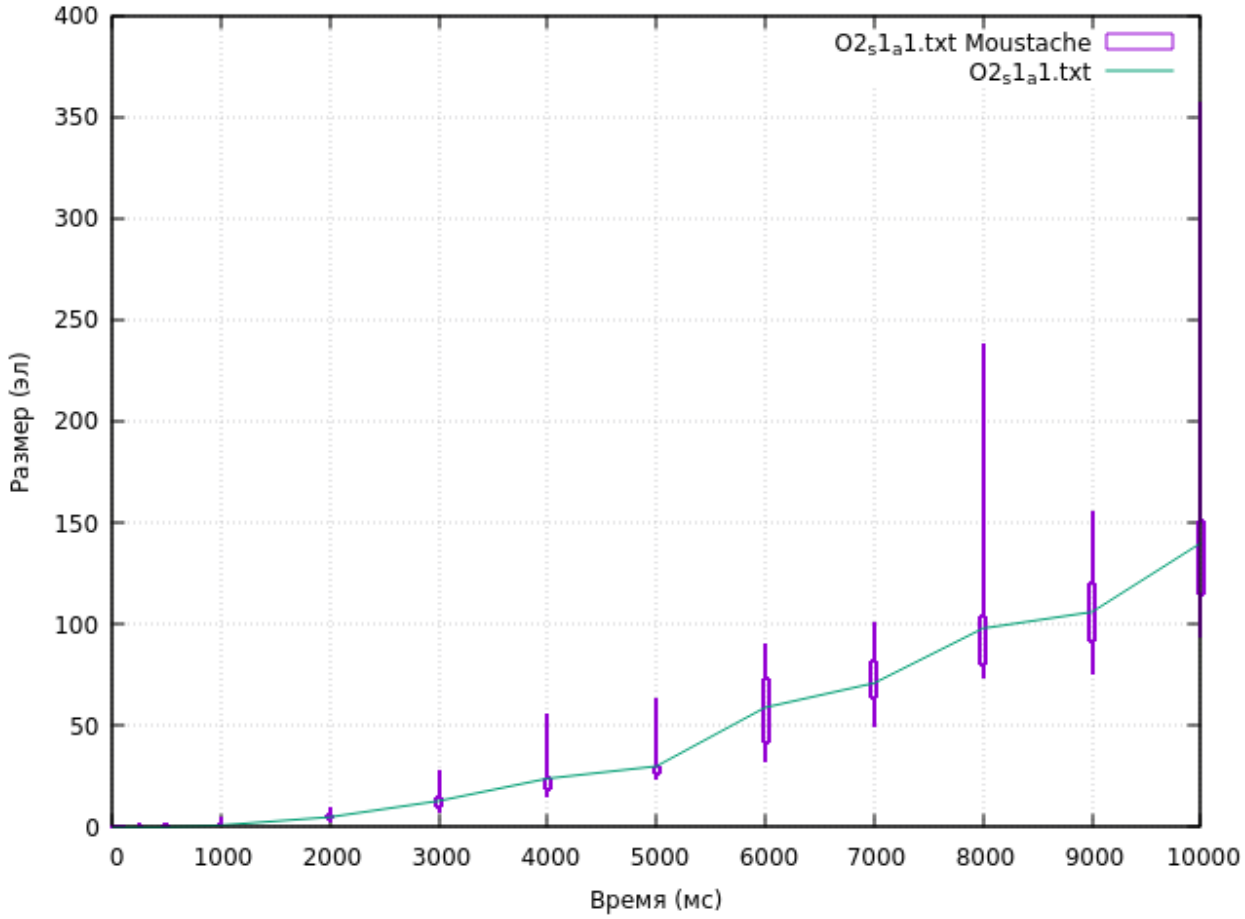

O2s2a1.txt Error
O2s2a1.txt
Размер (эл)
Время (мс)
0
20
40
60
80
100
120
140
160
180
0
1000
2000
3000
4000
5000
6000
7000
8000
9000
10000

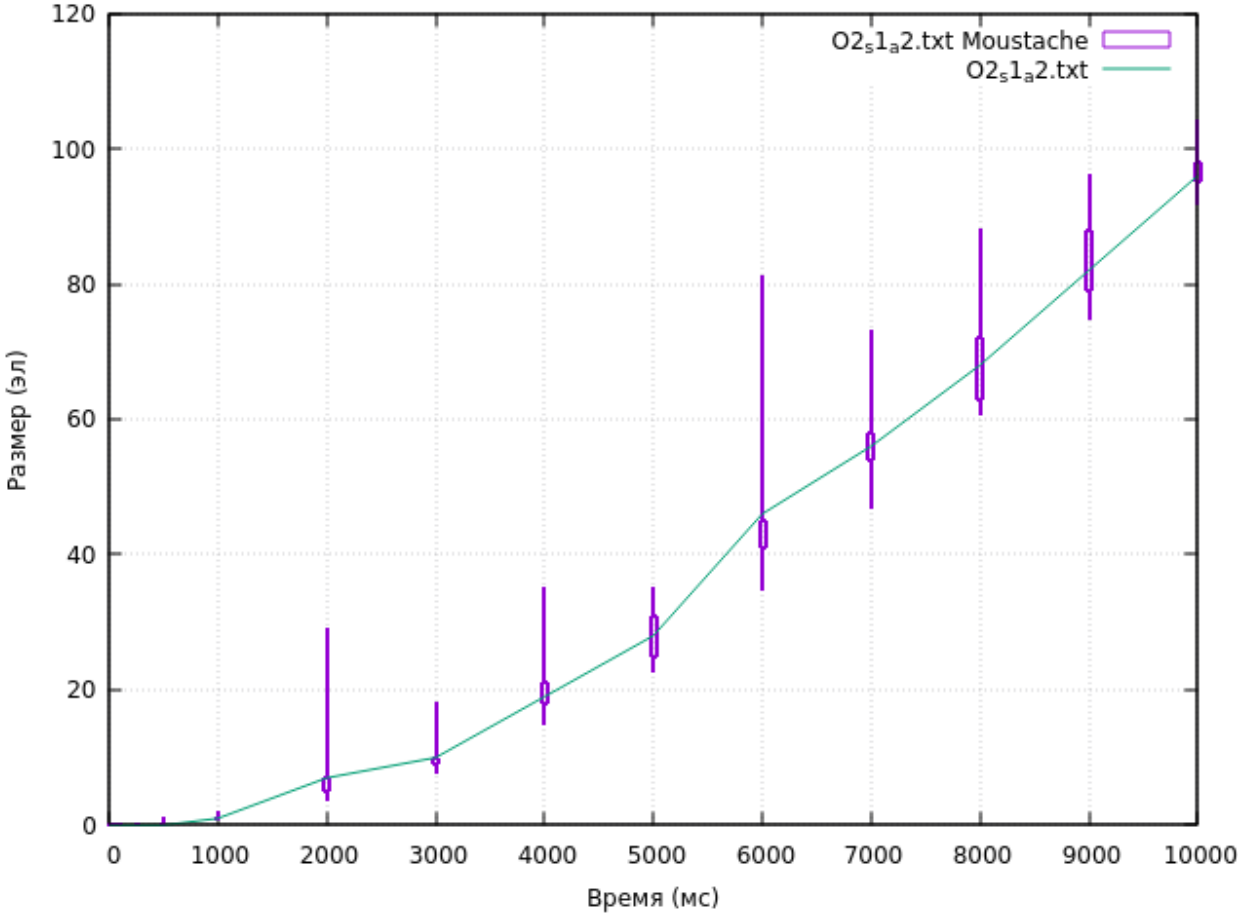

O2s2a2.txt Error
O2s2a2.txt
Размер (эл)
Время (мс)
0
20
40
60
80
100
120
140
160
180
0
1000
2000
3000
4000
5000
6000
7000
8000
9000
10000

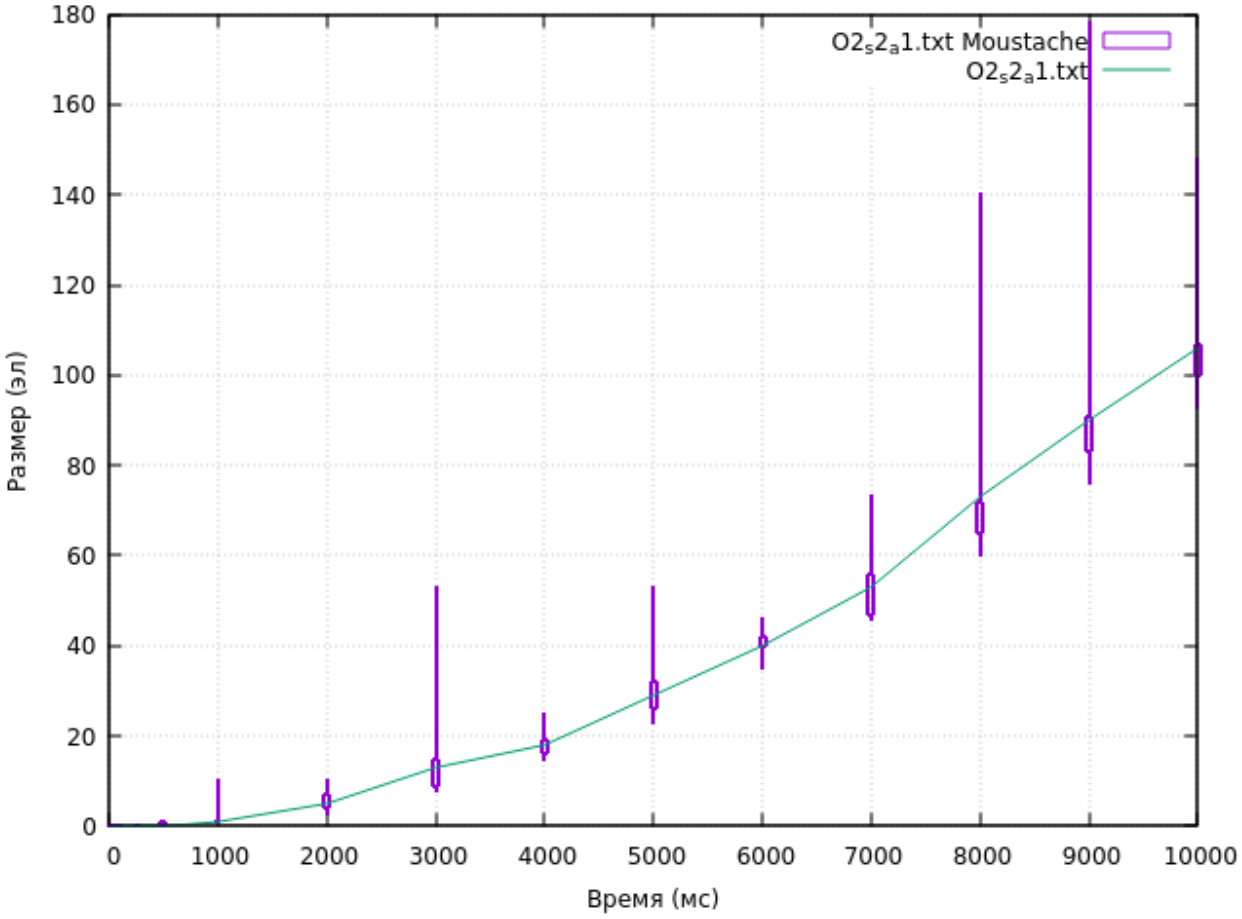

O2s3a1.txt Error
O2s3a1.txt
Размер (эл)
Время (мс)
0
50
100
150
200
250
300
0
1000
2000
3000
4000
5000
6000
7000
8000
9000
10000

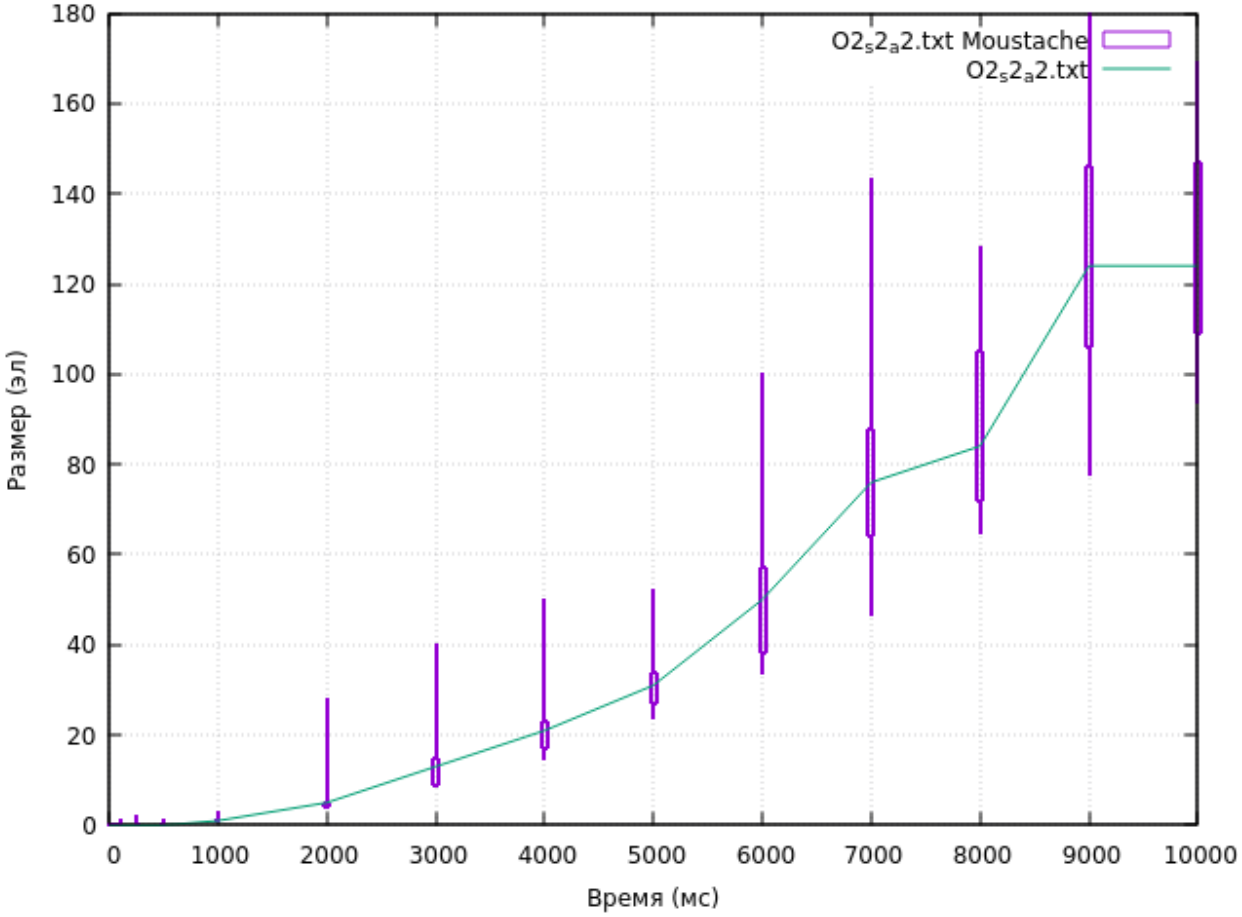

O2s3a2.txt Error
O2s3a2.txt
Размер (эл)
Время (мс)
0
20
40
60
80
100
120
140
160
180
200
0
1000
2000
3000
4000
5000
6000
7000
8000
9000
10000

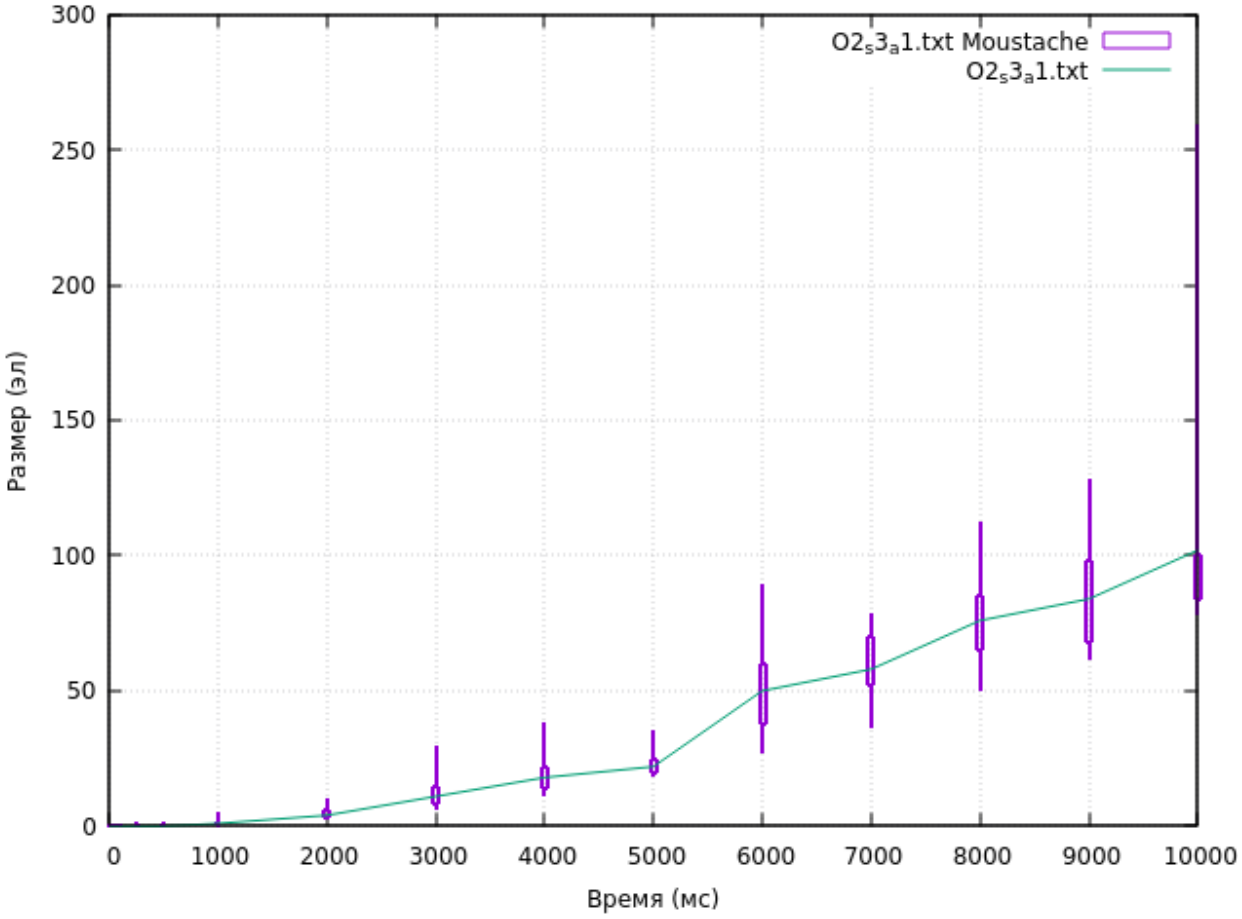
(Кусочно-линейный график (6 штук) с ошибкой (среднее, максимум, минимум) для всех вариантов обработки массива при уровне оптимизации O2.)

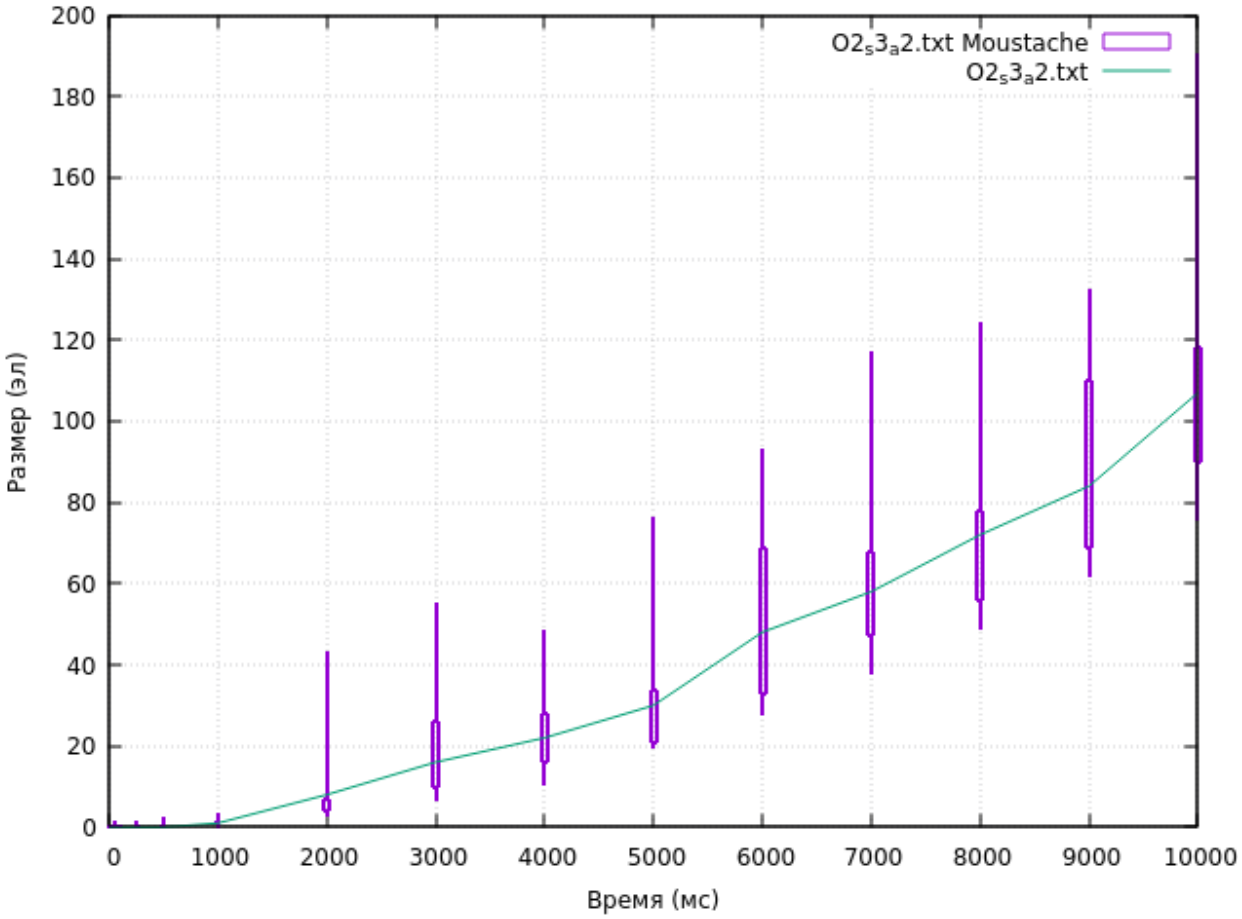



O2s1a2.txt Moustache
O2s1a2.txt
Размер (эл)
Время (мс)
0
20
40
60
80
100
120
0
1000
2000
3000
4000
5000
6000
7000
8000
9000
10000


O2s2a1.txt Moustache
O2s2a1.txt
Размер (эл)
Время (мс)
0
20
40
60
80
100
120
140
160
180
0
1000
2000
3000
4000
5000
6000
7000
8000
9000
10000


O2s2a2.txt Moustache
O2s2a2.txt
Размер (эл)
Время (мс)
0
20
40
60
80
100
120
140
160
180
0
1000
2000
3000
4000
5000
6000
7000
8000
9000
10000


O2s3a1.txt Moustache
O2s3a1.txt
Размер (эл)
Время (мс)
0
50
100
150
200
250
300
0
1000
2000
3000
4000
5000
6000
7000
8000
9000
10000



(График с усами (среднее, максимум, минимум; нижний, средний и верхний квартили) для варианта обработки «через квадратные скобки» при уровне оптимизации O2.)

Как видно из графиков (1) и (2) время сортировки выбором для заранее отсортированного массива и для случайного примерно одинаково, так как сам алгоритм подразумевает нахождение следующего минимума после предыдущего (то есть в отсортированном массиве алгоритм все равно будет искать минимум, так как он изначально не знает (и не может проверить!), что минимум стоит в начале списка).

Из графиков (3) и (4) видно, что чем больше элементов в массиве, тем больше значение ошибки (а значит более нестабильный замер времени, по крайней мере в миллисекундах).

Таблицы:

1)

O0_s1_a1:

Размер массива (n)	Время (T, мс) (среднее)
1	0

10	0
50	0
100	0
250	0
500	0
1000	1
2000	6
3000	12
4000	28
5000	32
6000	42
7000	70
8000	98
9000	89
10000	105

O0_s1_a2:

Размер массива (n)	Время (T, мс) (среднее)
1	0
10	0
50	0
100	0
250	0
500	0
1000	1
2000	6
3000	14
4000	25
5000	40
6000	75
7000	68
8000	67
9000	81
10000	102

O0_s2_a1:

Размер массива (n)	Время (T, мс) (среднее)
1	0
10	0
50	0
100	0
250	0

500	0
1000	1
2000	7
3000	11
4000	26
5000	35
6000	44
7000	75
8000	71
9000	100
10000	120

O0_s2_a2:

Размер массива (n)	Время (T, мс) (среднее)
1	0
10	0
50	0
100	0
250	0
500	0
1000	1
2000	5
3000	11
4000	20
5000	32
6000	53
7000	70
8000	82
9000	114
10000	127

O0_s3_a1:

Размер массива (n)	Время (T, мс) (среднее)
1	0
10	0
50	0
100	0
250	0
500	0
1000	1
2000	10
3000	9

4000	18
5000	27
6000	50
7000	63
8000	84
9000	98
10000	99

O0_s3_a2:

Размер массива (n)	Время (T, мс) (среднее)
1	0
10	0
50	0
100	0
250	0
500	0
1000	1
2000	4
3000	14
4000	18
5000	30
6000	34
7000	73
8000	68
9000	90
10000	114

2)

O2_s1_a1:

Размер массива (n)	Время (T, мс) (среднее)	$\frac{ln\ t_{i+1} - ln\ t_i}{ln\ n_{i+1} - ln\ n_i}$
1	0	-
10	0	-
50	0	-
100	0	-
250	0	-
500	0	-
1000	1	2.58
2000	6	1.70
3000	12	2.95

4000	28	0.60
5000	32	1.49
6000	42	3.31
7000	70	2.51
8000	98	-0.82
9000	89	1.57
10000	105	-

O2_s1_a2:

Размер массива (n)	Время (T, мс) (среднее)	$\frac{ln\ t_{i+1} - ln\ t_i}{ln\ n_{i+1} - ln\ n_i}$
1	0	-
10	0	-
50	0	-
100	0	-
250	0	-
500	0	-
1000	1	2.81
2000	7	0.88
3000	10	2.23
4000	19	1.74
5000	28	2.72
6000	46	1.28
7000	56	1.45
8000	68	1.59
9000	82	1.50
10000	96	-

O2_s2_a1:

Размер массива (n)	Время (T, мс) (среднее)	$\frac{ln\ t_{i+1} - ln\ t_i}{ln\ n_{i+1} - ln\ n_i}$
1	0	-
10	0	-
50	0	-
100	0	-
250	0	-
500	0	-
1000	1	2.32
2000	5	2.36
3000	13	1.13
4000	18	2.14

5000	29	1.76
6000	40	1.83
7000	53	2.40
8000	73	1.78
9000	90	1.55
10000	106	-

O2_s2_a2:

Размер массива (n)	Время (T, мс) (среднее)	$\frac{ln\ t_{i+1} - ln\ t_i}{ln\ n_{i+1} - ln\ n_i}$
1	0	-
10	0	-
50	0	-
100	0	-
250	0	-
500	0	-
1000	1	2.32
2000	5	2.37
3000	13	1.67
4000	21	1.75
5000	31	2.62
6000	50	2.72
7000	76	0.75
8000	84	3.31
9000	124	0
10000	124	-

O2_s3_a1:

Размер массива (n)	Время (T, мс) (среднее)	$\frac{ln\ t_{i+1} - ln\ t_i}{ln\ n_{i+1} - ln\ n_i}$
1	0	-
10	0	-
50	0	-
100	0	-
250	0	-
500	0	-
1000	1	2
2000	4	2.49
3000	11	1.71
4000	18	0.90
5000	22	4.50

6000	50	0.96
7000	58	2.02
8000	76	0.85
9000	84	1.84
10000	102	-

O2_s3_a2:

Размер массива (n)	Время (T, мс) (среднее)	$\frac{ln\ t_{i+1} - ln\ t_i}{ln\ n_{i+1} - ln\ n_i}$
1	0	-
10	0	-
50	0	-
100	0	-
250	0	-
500	0	-
1000	1	3
2000	8	1.71
3000	16	1.11
4000	22	1.39
5000	30	4.50
6000	48	2.58
7000	58	1.40
8000	72	1.31
9000	84	2.30
10000	107	-

Ответы на вопросы:
1) Быстрее всех работает __rdtsc(), так как он сразу считает тики процессора и в отличии от clock() – без задержки.

2) Если в ходе эксперимента в датасете обнаружены несколько одинаковых измерений, то их нельзя заменить на одно, потому что в следствии нельзя определить, какое реальное значение должно принимать время. По распределению Гаусса, чем больше проведено измерений, тем больше вероятность того, что измеренное значение ближе к реальному.

3) В ходе эксперимента замеряется только время целевого алгоритма, то есть сортировки, так как инициализация массива так же занимает время у процессора. На разных платформах также разные операции могут занимать разное время.