Лабораторная работа №4
«ГЕНЕРАЦИЯ ПСЕВДОСЛУЧАЙНЫХ ЧИСЕЛ»

по дисциплине "Методы программирования"

Выполнил Ковалев Даниил
СКБ171, вариант 12
МИЭМ НИУ ВШЭ

В данной лабораторной проведено сравнение следующих алгоритмов генерации псевдослучайных чисел:

- Банальный алгоритм **dummy**
- Собственный алгоритм **custom**
- Линейный конгруэнтный метод **std::minstd_rand0**
- Метод Мерсенна-Твистера **std::mt19937**

Алгоритмы применяются для генерации равномерного распределения целых чисел в диапазона от 0 до $2^{16}-1$ при помощи **std::uniform_int_distribution**. Проверка равномерности генерации происходит при помощи критерия хи-квадрат Пирсона. Стоит отметить, что он проверяет лишь равномерность распределения, но не то, насколько случайную выборку выдает алгоритм.

Выполнять нужно скрипт **python run.py**. Он запустит тестирование алгоритмов генерации псевдослучайных чисел на равномерность критерием хи-квадрат и на время генерации. Размеры массивов, на которых тестируется время генерации, берутся из **sizes.txt**, и являются равномерно распределенными на линейной оси от 10000 до 1000000.

```
1 ./4_prng --sizes=sizes.txt --output=timings.csv 2> test_prng.log
```

После этого средствами пакета **matplotlib** для **python** на основе **timings.csv** строится график зависимости времени генерации выборки от размера выборки при использовании разных алгоритмов.

Структура проекта:

🗁 **helpers** — см. ЛР1
🗁 **lab4**:
  🗎 **CMakeLists.txt**
  🗎 **dummy.h** — файл с описанием алгоритма ГПСЧ **dummy**
  🗎 **dummy.cpp** — файл с реализацией алгоритма ГПСЧ **dummy**
  🗎 **custom.h** — файл с описанием алгоритмов ГПСЧ **custom**
  🗎 **custom.cpp** — файл с реализацией алгоритмов ГПСЧ **custom**
  🗎 **chi_squared.h** — файл с реализацией проверки критерия хи-квадрат
  🗎 **stat_utils.h** — файл с реализацией подсчета статистических характеристик выборки
  🗎 **main.cpp** — файл с реализацией тестирования алгоритмов ГПСЧ
  🗎 **run.py** — скрипт запуска тестирования алгоритмов ГПСЧ
🗎 **CMakeLists.txt** — см. ЛР1

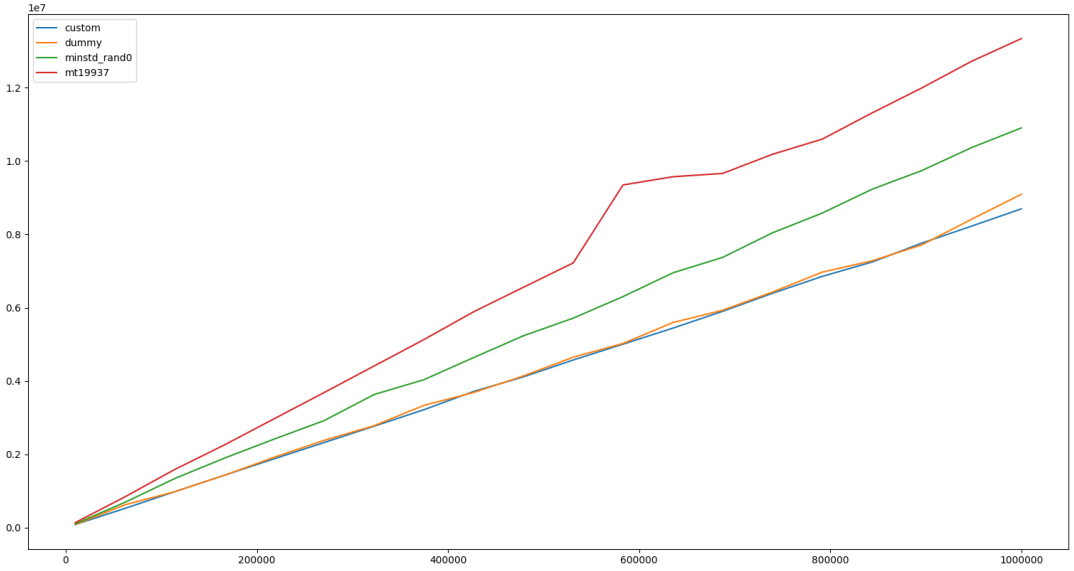
Приведем таблицу, построенную по файлу **timings.csv**. В ней записано время генерации выборки в наносекундах для каждого из 4 алгоритмов ГПСЧ на размерах выборок от 10000 до 1000000.

	10000	62105	114210	166315	218421	270526	322631
dummy	85734	624859	981281	1430809	1923260	2381536	2779241
custom	85373	526759	982457	1431955	1880827	2322528	2766701
ЛКМ	106158	692948	1341493	1898453	2418327	2920665	3629828
М–Т	138966	842204	1588972	2259488	2973562	3686302	4407291

374736	426842	478947	531052	583157	635263	687368
3334316	3685913	4137509	4648276	5022609	5592524	5927491
3215129	3714744	4112157	4571726	5003857	5439775	5898509
4033652	4638283	5232090	5712718	6300734	6947276	7369443
5127464	5886921	6554775	7218697	9344470	9568235	9658414

739473	791578	843684	895789	947894	1000000
6423607	6966725	7275459	7710057	8409338	9089586
6389848	6850997	7242403	7758477	8221618	8693846
8038211	8576899	9227042	9736266	10362999	10899665
10180160	10593060	11307454	11989598	12715871	13335107

График, построенный по этой таблице



Из построенных по этим данным графиков можно сделать следующие выводы:

- Все алгоритмы показывают хорошую равномерность распределения, но это не говорит о том, что они все выдают выборки с одинаковой энтропией
- Собственные реализации алгоритмов используют простейшие математические операции и работают быстрее линейного конгруэнтного метода, который, в свою очередь, работает быстрее метода Мерсенна-Твистера
- Время генерации выборки при использовании любого алгоритма ГПСЧ линейно зависит от размера выборки

Листинги с исходным кодом всех файлов расположены на следующих страницах отчета.

Листинг 1: lab4/CMakeLists.txt

```
set(PROJECT_NAME 4_prng)

project(${PROJECT_NAME} LANGUAGES CXX)

set(SOURCES main.cpp dummy.cpp custom.cpp)
set(HEADERS chi_squared.h stat_utils.h dummy.h custom.h)

add_executable(${PROJECT_NAME} ${SOURCES} ${HEADERS})

target_include_directories(${PROJECT_NAME} PRIVATE ${Helpers_INCLUDE_DIR})
target_link_libraries(${PROJECT_NAME} PRIVATE helpers)
target_link_libraries(${PROJECT_NAME} PRIVATE Boost::program_options)
target_link_libraries(${PROJECT_NAME} PRIVATE Boost::boost)

file(COPY run.py DESTINATION ${PROJECT_BINARY_DIR})
```

Листинг 2: lab4/dummy.h

```
/**
 * @file
 * @brief Заголовочный файл, содержащий описание простейшего алгоритма
 * генерации псевдослучайных чисел
 * @date Март 2020
*/
#ifndef DUMMY_H
#define DUMMY_H

#include <cstdint>

namespace my
{

class DummyPRNG
{
public:
    using result_type = std::uint32_t;

    DummyPRNG() = delete;
    DummyPRNG(result_type seed);

    static result_type min();
    static result_type max();

    result_type operator()();

private:
    result_type m_value;
};

} // namespace my

#endif // DUMMY_H
```

Листинг 3: lab4/dummy.cpp

```
#include "dummy.h"
#include <limits>

namespace my
{

DummyPRNG::DummyPRNG(DummyPRNG::result_type seed)
    : m_value(seed)
{
}
```

```
11
12 DummyPRNG::result_type DummyPRNG::operator()()
13 {
14     m_value *= m_value + 1;
15     return m_value;
16 }
17
18 DummyPRNG::result_type DummyPRNG::min()
19 {
20     return std::numeric_limits<DummyPRNG::result_type>::min();
21 }
22
23 DummyPRNG::result_type DummyPRNG::max()
24 {
25     return std::numeric_limits<DummyPRNG::result_type>::max();
26 }
27
28 } // namespace my
```

Листинг 4: lab4/custom.h

```
1  /**
2   * @file
3   * @brief Заголовочный файл, содержащий описание собственного алгоритма
4   * генерации псевдослучайных чисел
5   * @date Март 2020
6  */
7  #ifndef CUSTOM_H
8  #define CUSTOM_H
9
10 #include <cstdint>
11
12 namespace my
13 {
14
15 class CustomPRNG
16 {
17 public:
18     using result_type = std::uint32_t;
19
20     CustomPRNG() = delete;
21     CustomPRNG(result_type seed);
22
23     static result_type min();
24     static result_type max();
25
26     result_type operator()();
27
28 private:
29     result_type m_value;
30 };
31
32 } // namespace my
33
34 #endif // CUSTOM_H
```

Листинг 5: lab4/custom.cpp

```
1 #include "custom.h"
2 #include <limits>
3
4 namespace my
5 {
6
7 CustomPRNG::CustomPRNG(CustomPRNG::result_type seed)
8     : m_value(seed)
9 {
```

```
}

CustomPRNG::result_type CustomPRNG::operator()()
{
    std::uint32_t old = m_value;
    m_value <<= 7;
    m_value += old ^ 0x9908b0df;
    return m_value;
}

CustomPRNG::result_type CustomPRNG::min()
{
    return std::numeric_limits<CustomPRNG::result_type>::min();
}

CustomPRNG::result_type CustomPRNG::max()
{
    return std::numeric_limits<CustomPRNG::result_type>::max();
}

} // namespace my
```

Листинг 6: lab4/stat_utils.h

```
/**
 * @file
 * @brief Заголовочный файл, содержащий реализацию статистических функций
 * @date Март 2020
*/
#ifndef STAT_UTILS_H
#define STAT_UTILS_H

#include <iterator>
#include <numeric>
#include <cmath>

namespace my
{

namespace stat
{

/**
 * Вычисляет выборочное среднее
 * @tparam Sample тип выборки (например, `std::vector<int>`)
 * @param[in] sample выборка
 * @return выборочное среднее
 */
template <typename Sample>
double mean(const Sample& sample)
{
    return static_cast<double>(std::accumulate(std::begin(sample), std::end(sample),
    0.0)) / std::size(sample);
}

/**
 * Вычисляет среднеквадратичное отклонение выборки
 * @tparam Sample тип выборки (например, `std::vector<int>`)
 * @param[in] sample выборка
 * @return среднеквадратичное отклонение
 */
template <typename Sample>
double variance(const Sample& sample)
{
    double average = mean(sample);
    double sum = 0;
    for (const auto& value : sample)
        sum += (value - average) * (value - average);
    sum /= std::size(sample);
```

```
45      return std::sqrt(sum);
46  }
47
48  /**
49   * Вычисляет коэффициент вариации выборки
50   * @tparam Sample тип выборки (например, `std::vector<int>`)
51   * @param[in] sample выборка
52   * @return коэффициент вариации
53   */
54  template <typename Sample>
55  double variability(const Sample& sample)
56  {
57      return variance(sample) / mean(sample);
58  }
59
60  } // namespace stat
61
62  } // namespace my
63
64  #endif // STAT_UTILS_H
```

Листинг 7: lab4/chi_squared.h

```
1   /**
2    * @file
3    * @brief Заголовочный файл, содержащий реализацию проверки критерия
4    * хи-квадрат Пирсона на принадлежность равномерному распределению
5    * @date Март 2020
6   */
7   #ifndef CHI_SQUARED_H
8   #define CHI_SQUARED_H
9
10  #include <boost/math/distributions.hpp>
11  #include <vector>
12  #include <stdexcept>
13  #include <cmath>
14  #include <algorithm>
15  #include <utility>
16  #include <sstream>
17
18  namespace my
19  {
20
21  /// Допустимые типы распределений
22  enum class DistributionType
23  {
24      UNIFORM_INT,  ///< равномерное целочисленное
25      UNIFORM_REAL, ///< равномерное непрерывное
26  };
27
28  /**
29   * Вычисляет значение хи-квадрат для данной выборки
30   * @tparam Number тип элемента выборки
31   * @param[in] sample выборка
32   * @param[in] min минимальное значение, которое могло быть получено при генерации выборки
33   * @param[in] max максимальное значение, которое могло быть получено при генерации выборки
34   * @param[in] type тип распределения
35   * @return `std::pair`, первый элемент которой – посчитанное значение хи-квадрат,
36   * второй – количество степеней свободы
37   */
38  template <typename Number>
39  std::pair<double, std::size_t> chi_squared(const std::vector<Number>& sample, Number min,
        Number max, DistributionType type)
40  {
41      std::ostringstream exception_message;
42      if (min >= max)
43      {
44          exception_message << "Range minimum " << min << " is not less than range maximum " <<
        max;
```

```
45          throw std::invalid_argument(exception_message.str());
46      }
47      if (sample.size() < 2)
48      {
49          exception_message << "Sample size " << sample.size() << " is too small";
50          throw std::invalid_argument(exception_message.str());
51      }
52
53      int add = (type == DistributionType::UNIFORM_INT ? 1 : 0);
54      double range_size = static_cast<double>(max) - min + add;
55
56      std::size_t interval_count = static_cast<std::size_t>(1 + std::log2(sample.size()));
57      if (type == DistributionType::UNIFORM_INT && static_cast<double>(interval_count) >
        range_size)
58          interval_count = static_cast<std::size_t>(range_size);
59      double interval_size = range_size / interval_count;
60
61      std::vector<Number> interval_edges(interval_count + 1);
62      interval_edges.front() = min;
63      for (std::size_t i = 1; i + 1 < interval_edges.size(); ++i)
64          interval_edges[i] = static_cast<Number>(min + interval_size * i);
65      interval_edges.back() = max;
66
67      std::vector<double> interval_probabilities(interval_count);
68      for (std::size_t i = 0; i + 1 < interval_probabilities.size(); ++i)
69          interval_probabilities[i] = (interval_edges[i + 1] - interval_edges[i]) / range_size;
70      interval_probabilities.back() = (interval_edges[interval_count] - interval_edges[
        interval_count - 1] + add) / range_size;
71
72      std::vector<double> emperical_probabilities(interval_count, 0);
73      for (const Number& value : sample)
74      {
75          std::size_t index = std::upper_bound(interval_edges.begin(), interval_edges.end(),
        value) - interval_edges.begin() - 1;
76          if (index == interval_count && type == DistributionType::UNIFORM_INT && value == max)
77          {
78              ++emperical_probabilities.back();
79              continue;
80          }
81          if (index == interval_count)
82          {
83              exception_message << "Sample value " << value << " is not in range [" << min << "
        ; " << max
84                                << (type == DistributionType::UNIFORM_INT ? "]" : ")");
85              throw std::invalid_argument(exception_message.str());
86          }
87          ++emperical_probabilities[index];
88      }
89      std::for_each(emperical_probabilities.begin(), emperical_probabilities.end(), [&sample](
        double& value){ value /= sample.size(); });
90
91      double result = 0;
92      for (std::size_t i = 0; i < interval_count; ++i)
93      {
94          double numerator = emperical_probabilities[i] - interval_probabilities[i];
95          result += numerator * numerator / interval_probabilities[i];
96      }
97      result *= sample.size();
98
99      return { result, interval_count - 1 };
100 }
101
102 /**
103  * Проверяет статистический критерий хи-квадрат Пирсона для заданной выборки
104  * @tparam Number тип элемента выборки
105  * @param[in] sample выборка
106  * @param[in] min минимальное значение, которое могло быть получено при генерации выборки
107  * @param[in] max максимальное значение, которое могло быть получено при генерации выборки
108  * @param[in] significance_level уровень значимости (вероятность отклонить истинную гипотезу)
109  * @param[in] type тип распределения
```

```
110  * @return `true`, если выборка удовлетворяет критерию хи-квадрат, `false` в противном случае
111  */
112 template <typename Number>
113 bool check_chi_squared_criteria(const std::vector<Number>& sample, Number min, Number max,
        double significance_level, DistributionType type)
114 {
115     if (significance_level <= 0 || significance_level >= 1)
116     {
117         std::ostringstream exception_message;
118         exception_message << "Significance level " << significance_level << " is not in (0;
        1) interval";
119         throw std::invalid_argument(exception_message.str());
120     }
121     namespace bm = boost::math;
122     auto [chi_squared_value, degrees_of_freedom] = chi_squared<Number>(sample, min, max, type
        );
123     return (chi_squared_value <= bm::quantile(bm::chi_squared_distribution<double>(
        degrees_of_freedom), 1 - significance_level));
124 }
125
126 } // namespace my
127
128 #endif // CHI_SQUARED_H
```

Листинг 8: lab4/main.cpp

```
1  #include "io_operations.h"
2  #include "chi_squared.h"
3  #include "stat_utils.h"
4  #include "dummy.h"
5  #include "custom.h"
6  #include <boost/program_options.hpp>
7  #include <boost/range/adaptor/indexed.hpp>
8  #include <chrono>
9  #include <fstream>
10 #include <iostream>
11 #include <iomanip>
12 #include <string>
13 #include <vector>
14 #include <random>
15 #include <limits>
16
17 using ArraySize = std::size_t;
18 using PRNGName = std::string;
19 using Time = std::int64_t;
20 using SizeToTime = std::map<ArraySize, Time>;
21 using TimingResults = std::map<PRNGName, SizeToTime>;
22 using IntegerType = std::uint16_t;
23 using IntegerSample = std::vector<IntegerType>;
24
25 template <typename Distribution, typename Number, typename Generator>
26 std::vector<Number> generate_sample(std::size_t size, Distribution& distribution,
       Generator& generator)
27 {
28     std::vector<Number> sample(size);
29     for (Number& value : sample)
30         value = distribution(generator);
31     return sample;
32 }
33
34 template <typename Integer, typename Generator>
35 std::vector<Integer> generate_integer_sample(std::size_t size, Integer min, Integer max,
       Generator& generator)
36 {
37     std::uniform_int_distribution<Integer> dist(min, max);
38     return generate_sample<std::uniform_int_distribution<Integer>, Integer, Generator>(
       size, dist, generator);
39 }
40
```

```
41 void test_samples(const std::vector<IntegerSample>& samples, IntegerType min,
       IntegerType max, double significance_level)
42 {
43     std::size_t accepted_count = 0;
44     for (auto sample : samples | boost::adaptors::indexed(0))
45     {
46         std::cout << std::setw(4) << sample.index() << ' ';
47         std::cout << "Chi squared: ";
48         if (my::check_chi_squared_criteria(sample.value(), min, max, significance_level,
        my::DistributionType::UNIFORM_INT))
49         {
50             std::cout << "accept.  ";
51             ++accepted_count;
52         }
53         else
54         {
55             std::cout << "decline. ";
56         }
57         std::cout << std::internal << std::setprecision(2) << "Sample: "
58                   << "mean = "        << std::fixed << std::setw(8) << my::stat::mean(
       sample.value()) << ", "
59                   << "variance = "    << std::fixed << std::setw(8) << my::stat::
       variance(sample.value()) << ", "
60                   << "variability = " << std::fixed << std::setw(4) << my::stat::
       variability(sample.value()) << '\n';
61     }
62     std::cout << "TOTAL: accepted " << accepted_count << " out of " << samples.size() <<
        " samples (" << 100.0 * accepted_count / samples.size() << "%)\n";
63 }
64
65 template <typename Generator>
66 void test_generator(const std::string& name, std::size_t sample_count, std::size_t
       sample_size,
67                     IntegerType min, IntegerType max, double significance_level)
68 {
69     std::cout << "Testing " << name << '\n';
70
71     Generator prng(std::random_device{}());
72     std::vector<IntegerSample> samples(sample_count);
73     for (IntegerSample& sample : samples)
74         sample = generate_integer_sample<IntegerType, Generator>(sample_size, min, max,
       prng);
75
76     test_samples(samples, min, max, significance_level);
77 }
78
79 void test_all_generators(std::size_t sample_count, std::size_t sample_size,
80                          IntegerType min, IntegerType max, double significance_level)
81 {
82     test_generator<std::mt19937>     ("mt19937",      sample_count, sample_size, min,
       max, significance_level);
83     test_generator<std::minstd_rand0>("minstd_rand0", sample_count, sample_size, min,
       max, significance_level);
84     test_generator<my::DummyPRNG>    ("dummy",        sample_count, sample_size, min,
       max, significance_level);
85     test_generator<my::CustomPRNG>   ("custom",       sample_count, sample_size, min,
       max, significance_level);
86 }
87
88 template <typename Generator>
89 SizeToTime test_prng_timing(const std::vector<ArraySize>& sample_sizes, IntegerType min,
        IntegerType max)
90 {
91     Generator prng(std::random_device{}());
92     SizeToTime answer;
93     for (ArraySize sample_size : sample_sizes)
94     {
95         if (answer.contains(sample_size))
96             continue;
97         using namespace std::chrono;
```

```
98          time_point<high_resolution_clock> begin = high_resolution_clock::now();
99          IntegerSample sample = generate_integer_sample<IntegerType, Generator>(
        sample_size, min, max, prng);
100         time_point<high_resolution_clock> end = high_resolution_clock::now();
101         answer[sample_size] = duration_cast<nanoseconds>(end - begin).count();
102     }
103     return answer;
104 }
105
106 TimingResults test_all_timings(const std::vector<ArraySize>& sample_sizes, IntegerType
        min, IntegerType max)
107 {
108     TimingResults answer;
109     answer["mt19937"]      = test_prng_timing<std::mt19937>     (sample_sizes, min, max)
        ;
110     answer["minstd_rand0"] = test_prng_timing<std::minstd_rand0>(sample_sizes, min, max)
        ;
111     answer["dummy"]        = test_prng_timing<my::DummyPRNG>    (sample_sizes, min, max)
        ;
112     answer["custom"]       = test_prng_timing<my::CustomPRNG>   (sample_sizes, min, max)
        ;
113     return answer;
114 }
115
116 int main(int argc, char* argv[]) try
117 {
118     std::ios::sync_with_stdio(false);
119     std::cin.tie(nullptr);
120
121     namespace po = boost::program_options;
122     po::options_description desc("Allowed options");
123     desc.add_options()
124         ("help,H", "Print this message")
125         ("sizes,S", po::value<std::string>()->required(), "Text file with data sizes to
        be testes in the following format:\n"
126                                                           "size_0 size_1 size_2 ...
        size_n")
127         ("output,O", po::value<std::string>()->required(), "csv file to write test
        timing results, the format is:\n"
128                                                            "algo_name;result_for_size_0
        ;...;result_for_size_n")
129         ;
130
131     po::variables_map vm;
132     try
133     {
134         po::store(parse_command_line(argc, argv, desc), vm);
135         if (vm.contains("help"))
136         {
137             std::cout << desc << "\n";
138             return 0;
139         }
140         po::notify(vm);
141     }
142     catch (const po::error& error)
143     {
144         std::cerr << "Error while parsing command-line arguments: "
145                   << error.what() << "\nPlease use --help to see help message\n";
146         return 1;
147     }
148
149     std::string sizes_filename = vm["sizes"].as<std::string>();
150     std::string output_time_filename = vm["output"].as<std::string>();
151
152     // test chi squared
153     {
154         std::size_t sample_count = 10;
155         std::size_t sample_size = 1'000;
156         IntegerType min = std::numeric_limits<IntegerType>::min(), max = std::
        numeric_limits<IntegerType>::max();
```

```
157         double significance_level = 0.05;
158         test_all_generators(sample_count, sample_size, min, max, significance_level);
159         std::cout.flush();
160     }
161 
162     // test timings
163     {
164         IntegerType min = std::numeric_limits<IntegerType>::min(), max = std::
    numeric_limits<IntegerType>::max();
165         std::vector<ArraySize> sizes = read_sizes(sizes_filename);
166 
167         // csv header
168         std::ofstream output(output_time_filename);
169         output << "name";
170         for (ArraySize size : sizes)
171             output << ';' << size;
172         output << '\n';
173 
174         TimingResults results = test_all_timings(sizes, min, max);
175         for (auto& [name, timings] : results)
176         {
177             std::cerr << std::endl << "Algorithm: " << name << std::endl;
178             for (auto [size, time] : timings)
179                 std::cerr << size << ": " << time << std::endl;
180             print_timings_csv_line(output, name, timings);
181         }
182     }
183 }
184 catch (const std::exception& e)
185 {
186     std::cerr << e.what() << '\n';
187     return 1;
188 }
```

Листинг 9: lab4/run.py

```
1  import subprocess
2  import os
3  import numpy as np
4  import pandas as pd
5  from matplotlib import pyplot as plt
6 
7  try:
8      if not os.path.isfile("sizes.txt"):
9          f = open("sizes.txt", "w")
10         f.write(" ".join(np.linspace(10000, 1000000, 20).astype(int).astype(str)) + "\n")
11         f.close()
12 
13     print("Running tests...")
14     if not os.path.isfile("timings.csv"):
15         subprocess.call("./4_prng --sizes=sizes.txt --output=timings.csv 2> test_prng.log",
       shell=True)
16     print("Done!")
17 
18     raw_time = pd.read_csv("timings.csv", sep=';')
19     data_time = dict()
20     columns_time = list(raw_time.columns[1:].astype(int))
21     for i in range(raw_time.shape[0]):
22         data_time[raw_time.iloc[i]["name"]] = list(raw_time.iloc[i][1:])
23     plt.figure(figsize=(12, 8))
24     for name, timings in data_time.items():
25         plt.plot(columns_time, timings, label=name)
26         plt.legend()
27     plt.show()
28 
29 except Exception as e:
30     print(e)
```