Московский авиационный институт

(Национальный исследовательский университет)

Факультет №3:

*Системы управления, информатика и электроэнергетика*.

Кафедра 304.

Отчёт по лабораторной работе

По учебной дисциплине «Программирование»

На тему

«Формирование массивов экспериментальных данных»

Группа: М3О-219Бк-22

Выполнили:

Никулин Г. С.,

Евстифеев А. С.

Приняла:

*Дмитриева Е.А.*

Москва 2024 г.

**Оглавление**

[Постановка задача 3](#_30j0zll)

[1 этап работы 3](#_1fob9te)

[2 этап работы 3](#_3znysh7)

[Структурная схема алгоритма 5](#_2et92p0)

[Главная функция 5](#_tyjcwt)

[Код программы 7](#_4d34og8)

[Файл int.cpp 7](#_2s8eyo1)

[Файл double.cpp 7](#_2s8eyo1)

[Результат работы программы 13](#_1ksv4uv)

[Для целочисленных значений: 13](#_44sinio)

[Упорядоченные по возрастанию: 13](#_2jxsxqh)

[Упорядоченные по убыванию: 13](#_z337ya)

[Случайная последовательность: 14](#_3j2qqm3)

[Частично упорядоченная пилообразная последовательность: 15](#_1y810tw)

[Частично упорядоченная ступенчатая последовательность: 15](#_4i7ojhp)

[Частично упорядоченная синусоидальная последовательность: 18](#_2p2csry)

[Для вещественных значений: 16](#_3whwml4)

[Упорядоченные по возрастанию: 16](#_2bn6wsx)

[Упорядоченные по убыванию: 17](#_qsh70q)

[Случайная последовательность: 17](#_3as4poj)

[Частично упорядоченная пилообразная последовательность: 17](#_1pxezwc)

[Частично упорядоченная ступенчатая последовательность: 18](#_49x2ik5)

[Частично упорядоченная синусоидальная последовательность: 18](#_2p2csry)

[Графики зависимости времени формирования целочисленных последовательностей от количества элементов 20](#_23ckvvd)

[Для целочисленных значений: 20](#_ihv636)

[Вывод 34](#_xvir7l)

# Постановка задача

## 1 этап работы

Разработать функции, формирующие последовательности чисел, для целых чисел и чисел с плавающей запятой (должно быть два комплекта функций и соответственно результатов):

• упорядоченные по возрастанию,

• упорядоченные по убыванию,

• случайную последовательность,

• частично упорядоченные: «пилообразные», «синусоидальные», «ступенчатые».

для целых чисел и чисел с плавающей запятой (должно быть два комплекта функций и соответственно результатов).

При реализации функций считать, что выделение памяти под последовательности происходит вне этих функций, функции в качестве формальных параметров получают указатель на массив, его размер и, возможно, диапазон изменения величин и длину интервалов (для частично упорядоченных последовательностей).

Рекомендуется использовать указатели на функцию в качестве формального параметра или массив указателей на функции для автоматизации сбора статистической информации.

Для массивов размерностью от 150 до 200 элементов (размерность выбирается самостоятельно) предусмотреть вывод значений в файл.

Используя данные из файлов, построить графики полученных зависимостей (возможно с привлечением других программных средств) для доказательства правильности полученных функций.

## 2 этап работы

1. Оценить длительность формирования последовательностей всех типов для различных значений размеров последовательностей (5⋅105, 10⋅105, …, 50⋅105), и на основе полученных значений построить графики зависимостей длительностей формирования массивов от их размера (лучше в виде столбчатых диаграмм, можно в Microsoft Exel).

2. Составить отчет, в котором привести структурные схемы алгоритмов главной функции и какой-то одной из функций (например, реализующей частично упорядоченную последовательность), текст программы, результаты по временным интервалам для различных размерностей массивов, графики зависимостей (времени от размерностей) и выводы по полученным результатам.

# Структурная схема алгоритма

Main:



# Код программы

### Файл int.cpp

#include <iostream>

#include <chrono>

#include <fstream>

#include <cmath>

void generate\_int\_saw(int\* a, int size, int mod, int range);

void generate\_int\_sin(int\* a, int size, int mod, int range);

void generate\_int\_lad(int\* a, int size, int mod, int range);

void generate\_up(int\* arr, int arrLength, int mod, int range);

void generate\_down(int\* arr, int arrLength, int mod, int range);

void generate\_random(int\* arr, int arrlength, int mod, int range);

void write\_at\_file(int\* array, int size, std::string filename);

typedef void(\*FUNC)(int\*, int, int, int);

FUNC functions[6] = { &generate\_int\_saw, &generate\_int\_sin, &generate\_int\_lad, generate\_up, &generate\_down, generate\_random };

int main() {

int modu = 100000;

int ran = 10;

int data[10] = { 150,10,15,20,25,30,35,40,45,50 };

for (int i = 0; i < 10; ++i) {

//data[i] \*= 100000;

}

std::string\*\* files = new std::string \* [6];

for (int i = 0; i < 6; i++)

files[i] = new std::string[10];

files[0][0] = "saw\_generated1.txt";

files[0][1] = "saw\_generated2.txt";

files[0][2] = "saw\_generated3.txt";

files[0][3] = "saw\_generated4.txt";

files[0][4] = "saw\_generated5.txt";

files[0][5] = "saw\_generated6.txt";

files[0][6] = "saw\_generated7.txt";

files[0][7] = "saw\_generated8.txt";

files[0][8] = "saw\_generated9.txt";

files[0][9] = "saw\_generated10.txt";

files[1][0] = "sin\_generated1.txt";

files[1][1] = "sin\_generated2.txt";

files[1][2] = "sin\_generated3.txt";

files[1][3] = "sin\_generated4.txt";

files[1][4] = "sin\_generated5.txt";

files[1][5] = "sin\_generated6.txt";

files[1][6] = "sin\_generated7.txt";

files[1][7] = "sin\_generated8.txt";

files[1][8] = "sin\_generated9.txt";

files[1][9] = "sin\_generated10.txt";

files[2][0] = "lad\_generated1.txt";

files[2][1] = "lad\_generated2.txt";

files[2][2] = "lad\_generated3.txt";

files[2][3] = "lad\_generated4.txt";

files[2][4] = "lad\_generated5.txt";

files[2][5] = "lad\_generated6.txt";

files[2][6] = "lad\_generated7.txt";

files[2][7] = "lad\_generated8.txt";

files[2][8] = "lad\_generated9.txt";

files[2][9] = "lad\_generated10.txt";

files[3][0] = "up\_generated1.txt";

files[3][1] = "up\_generated2.txt";

files[3][2] = "up\_generated3.txt";

files[3][3] = "up\_generated4.txt";

files[3][4] = "up\_generated5.txt";

files[3][5] = "up\_generated6.txt";

files[3][6] = "up\_generated7.txt";

files[3][7] = "up\_generated8.txt";

files[3][8] = "up\_generated9.txt";

files[3][9] = "up\_generated10.txt";

files[4][0] = "down\_generated1.txt";

files[4][1] = "down\_generated2.txt";

files[4][2] = "down\_generated3.txt";

files[4][3] = "down\_generated4.txt";

files[4][4] = "down\_generated5.txt";

files[4][5] = "down\_generated6.txt";

files[4][6] = "down\_generated7.txt";

files[4][7] = "down\_generated8.txt";

files[4][8] = "down\_generated9.txt";

files[4][9] = "down\_generated10.txt";

files[5][0] = "random\_generated1.txt";

files[5][1] = "random\_generated2.txt";

files[5][2] = "random\_generated3.txt";

files[5][3] = "random\_generated4.txt";

files[5][4] = "random\_generated5.txt";

files[5][5] = "random\_generated6.txt";

files[5][6] = "random\_generated7.txt";

files[5][7] = "random\_generated8.txt";

files[5][8] = "random\_generated9.txt";

files[5][9] = "random\_generated10.txt";

const char\* data\_files[] = { "data\_files1.txt","data\_files2.txt","data\_files3.txt","data\_files4.txt","data\_files5.txt","data\_files6.txt", "data\_files7.txt", "data\_files8.txt", "data\_files9.txt", "data\_files10.txt" };

for (int i = 0; i < 10; ++i) {

int size = data[i];

int\* array = new int[size];

std::chrono::\_\_enable\_if\_is\_duration<std::chrono::duration<long int, std::ratio<1, 1000000> > > time[6];

for (int j = 0; j < 6; j++) {

auto begin = std::chrono::steady\_clock::now();

functions[j](array, size, modu, ran);

auto end = std::chrono::steady\_clock::now();

time[j] = std::chrono::duration\_cast<std::chrono::microseconds>(end - begin);

write\_at\_file(array, size, files[j][i]);

}

FILE\* file;

const char\* df = data\_files[i];

file = fopen(df, "w+");

fprintf(file, "time saw:\t%d\n time sin:\t%d\n time lad:\t%d\n time up:\t%d\n time down:\t%d\n time random:\t%d", time[0], time[1], time[2], time[3], time[4], time[5]);

fclose(file);

}

return 0;

}

void generate\_up(int\* arr, int arrLength, int mod, int range) {

for (int i = 0; i < arrLength; ++i)

arr[i] = i \* double(mod) / double(arrLength);

}

void generate\_down(int\* arr, int arrLength, int mod, int range) {

for (int i = 0; i < arrLength; ++i)

arr[i] = (arrLength - i) \* double(mod) / double(arrLength);

}

void generate\_random(int\* arr, int arrLength, int mod, int range) {

for (int i = 0; i < arrLength; ++i)

arr[i] = rand() % mod;

}

void generate\_int\_saw(int\* a, int size, int mod, int range) {//saw

int c = 0;

while (c < size) {

int cur = 0;

while (cur < range && c < size) {

a[c] = cur \* int(mod / range);

c++;

cur++;

}

}

}

void generate\_int\_sin(int\* a, int size, int mod, int range) {//sinus

for (int i = 0; i < size; ++i)

a[i] = sin(2 \* i \* 3.14 \* range) \* mod;

}

void generate\_int\_lad(int\* a, int size, int mod, int range) {

int c = 0;

int line = 0;

while (c < size) {

int cur = 0;

while (c < size && cur < range) {

a[c] = line + (rand() % (mod / 20));

cur++;

c++;

}

line += mod;

}

}

void write\_at\_file(int\* array, int size, std::string filename) {

std::ofstream f;

f.open(filename);

for (int i = 0; i < size; ++i)

f << i << '\t' << array[i] << '\n';

f.close();

}

### Файл double.cpp

#include <iostream>

#include <chrono>

#include <fstream>

#include <cmath>

void generate\_saw(double\* a, int size, double mod, double range);

void generate\_sin(double\* a, int size, double mod, double range);

void generate\_lad(double\* a, int size, double mod, double range);

void generate\_up(double\* arr, int arrLength, double mod, double range);

void generate\_down(double\* arr, int arrLength, double mod, double range);

void generate\_random(double\* arr, int arrlength, double mod, double range);

void write\_at\_file(double\* array, int size, std::string filename);

typedef void(\*FUNC)(double\*, int, double, double);

FUNC functions[6] = { &generate\_saw, &generate\_sin, &generate\_lad, generate\_up, &generate\_down, &generate\_random };

int main() {

printf("567");

int modu = 10;

int ran = 10;

int data[10] = { 150,10,15,20,25,30,35,40,45,50 };

for (int i = 0; i < 10; ++i) {

//data[i] \*= 100000;

}

std::string\*\* files = new std::string \* [6];

for (int i = 0; i < 6; i++)

files[i] = new std::string[10];

files[0][0] = "saw\_generated1.txt";

files[0][1] = "saw\_generated2.txt";

files[0][2] = "saw\_generated3.txt";

files[0][3] = "saw\_generated4.txt";

files[0][4] = "saw\_generated5.txt";

files[0][5] = "saw\_generated6.txt";

files[0][6] = "saw\_generated7.txt";

files[0][7] = "saw\_generated8.txt";

files[0][8] = "saw\_generated9.txt";

files[0][9] = "saw\_generated10.txt";

files[1][0] = "sin\_generated1.txt";

files[1][1] = "sin\_generated2.txt";

files[1][2] = "sin\_generated3.txt";

files[1][3] = "sin\_generated4.txt";

files[1][4] = "sin\_generated5.txt";

files[1][5] = "sin\_generated6.txt";

files[1][6] = "sin\_generated7.txt";

files[1][7] = "sin\_generated8.txt";

files[1][8] = "sin\_generated9.txt";

files[1][9] = "sin\_generated10.txt";

files[2][0] = "lad\_generated1.txt";

files[2][1] = "lad\_generated2.txt";

files[2][2] = "lad\_generated3.txt";

files[2][3] = "lad\_generated4.txt";

files[2][4] = "lad\_generated5.txt";

files[2][5] = "lad\_generated6.txt";

files[2][6] = "lad\_generated7.txt";

files[2][7] = "lad\_generated8.txt";

files[2][8] = "lad\_generated9.txt";

files[2][9] = "lad\_generated10.txt";

files[3][0] = "up\_generated1.txt";

files[3][1] = "up\_generated2.txt";

files[3][2] = "up\_generated3.txt";

files[3][3] = "up\_generated4.txt";

files[3][4] = "up\_generated5.txt";

files[3][5] = "up\_generated6.txt";

files[3][6] = "up\_generated7.txt";

files[3][7] = "up\_generated8.txt";

files[3][8] = "up\_generated9.txt";

files[3][9] = "up\_generated10.txt";

files[4][0] = "down\_generated1.txt";

files[4][1] = "down\_generated2.txt";

files[4][2] = "down\_generated3.txt";

files[4][3] = "down\_generated4.txt";

files[4][4] = "down\_generated5.txt";

files[4][5] = "down\_generated6.txt";

files[4][6] = "down\_generated7.txt";

files[4][7] = "down\_generated8.txt";

files[4][8] = "down\_generated9.txt";

files[4][9] = "down\_generated10.txt";

files[5][0] = "random\_generated1.txt";

files[5][1] = "random\_generated2.txt";

files[5][2] = "random\_generated3.txt";

files[5][3] = "random\_generated4.txt";

files[5][4] = "random\_generated5.txt";

files[5][5] = "random\_generated6.txt";

files[5][6] = "random\_generated7.txt";

files[5][7] = "random\_generated8.txt";

files[5][8] = "random\_generated9.txt";

files[5][9] = "random\_generated10.txt";

const char\* data\_files[] = { "data\_files1.txt","data\_files2.txt","data\_files3.txt","data\_files4.txt","data\_files5.txt","data\_files6.txt", "data\_files7.txt", "data\_files8.txt", "data\_files9.txt", "data\_files10.txt" };

for (int i = 0; i < 10; ++i) {

int size = data[i];

printf("%d", size);

double\* array = new double[size];

std::chrono::\_\_enable\_if\_is\_duration<std::chrono::duration<long int, std::ratio<1, 1000000> > > time[6];

for (int j = 0; j < 6; j++) {

auto begin = std::chrono::steady\_clock::now();

functions[j](array, size, modu, ran);

auto end = std::chrono::steady\_clock::now();

time[j] = std::chrono::duration\_cast<std::chrono::microseconds>(end - begin);

write\_at\_file(array, size, files[j][i]);

}

FILE\* file;

const char\* df = data\_files[i];

file = fopen(df, "w+");

fprintf(file, "time saw:\t%d\n time sin:\t%d\n time lad:\t%d\n time up:\t%d\n time down:\t%d\n time random:\t%d", time[0], time[1], time[2], time[3], time[4], time[5]);

fclose(file);

}

return 0;

}

void generate\_up(double\* arr, int arrLength, double mod, double range) {

for (int i = 0; i < arrLength; ++i)

arr[i] = i \* double(mod) / double(arrLength);

}

void generate\_down(double\* arr, int arrLength, double mod, double range) {

for (int i = 0; i < arrLength; ++i)

arr[i] = (arrLength - i) \* double(mod) / double(arrLength);

}

void generate\_random(double\* arr, int arrLength, double mod, double range) {

for (int i = 0; i < arrLength; ++i)

arr[i] = (double)rand() / RAND\_MAX \* (mod);

}

void generate\_saw(double\* a, int size, double mod, double range) {//saw

int c = 0;

while (c < size) {

int cur = 0;

while (cur < range && c < size) {

a[c] = int(cur \* mod / range);

c++;

cur++;

}

}

}

void generate\_sin(double\* a, int size, double mod, double range) {//sinus

for (int i = 0; i < size; ++i)

a[i] = sin(2 \* i \* 3.14 \* range) \* mod;

}

void generate\_lad(double\* a, int size, double mod, double range) {

int c = 0;

int line = 0;

while (c < size) {

int cur = 0;

while (c < size && cur < range) {

a[c] = line + ((double)rand() / RAND\_MAX \* (mod / 20));

cur++;

c++;

}

line += mod;

}

}

void write\_at\_file(double\* array, int size, std::string filename) {

std::ofstream f;

f.open(filename);

for (int i = 0; i < size; ++i)

f << i << '\t' << array[i] << '\n';

f.close();

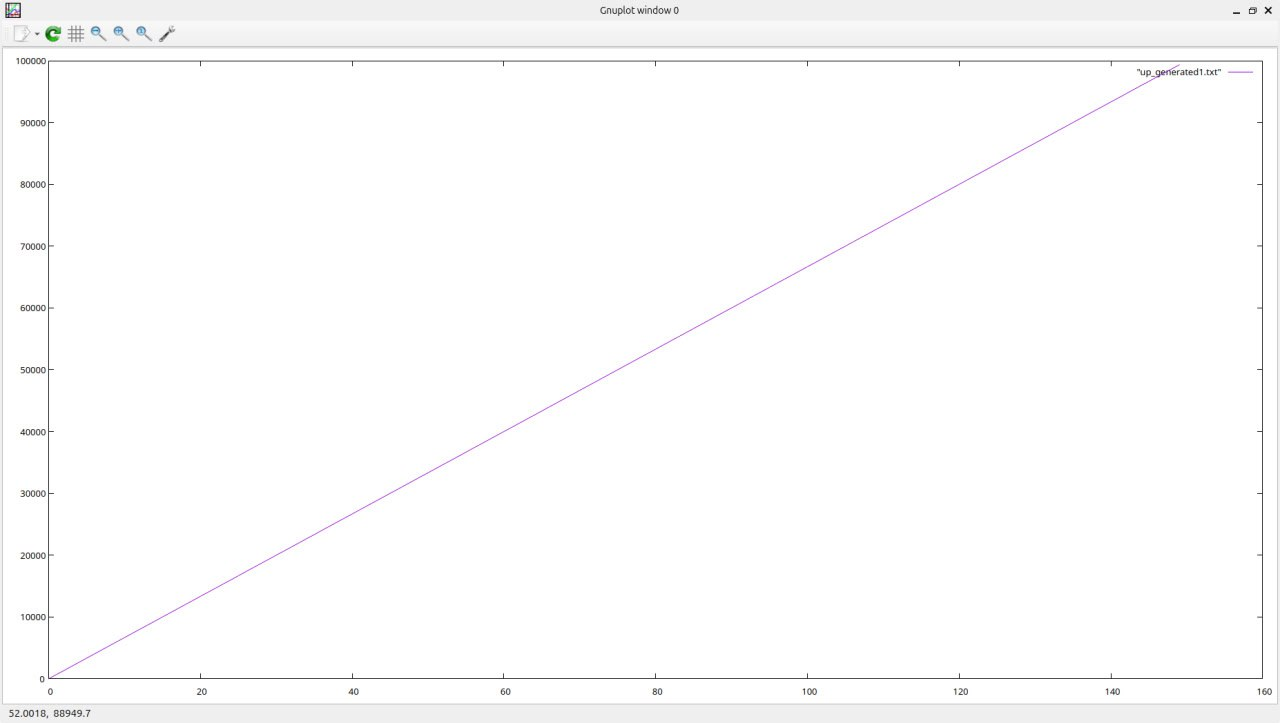
}

# Результат работы программы

Результатом работы программы является 60 файлов, расширения .txt. По построенным программой последовательностям чисел, для проверки, были созданы графики, показывающие соответствие их поставленной задаче:

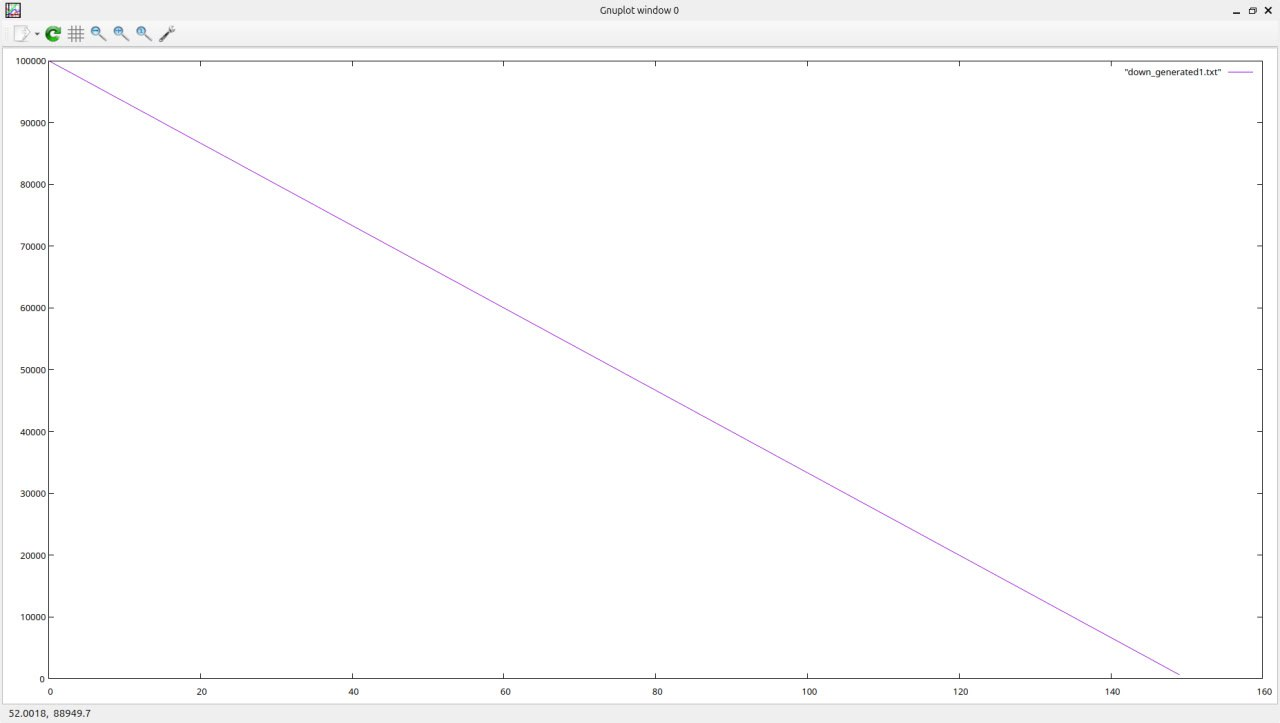
## Для целочисленных значений:

### Упорядоченные по возрастанию:

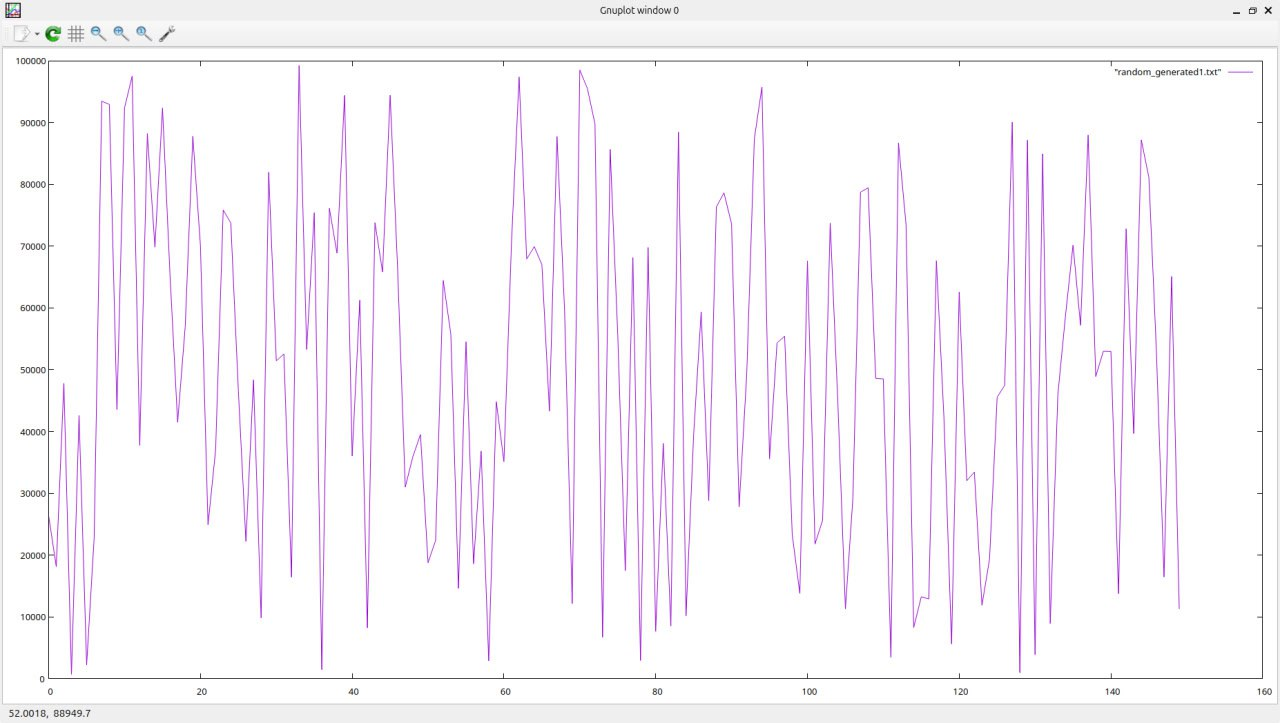


### 

### Упорядоченные по убыванию:

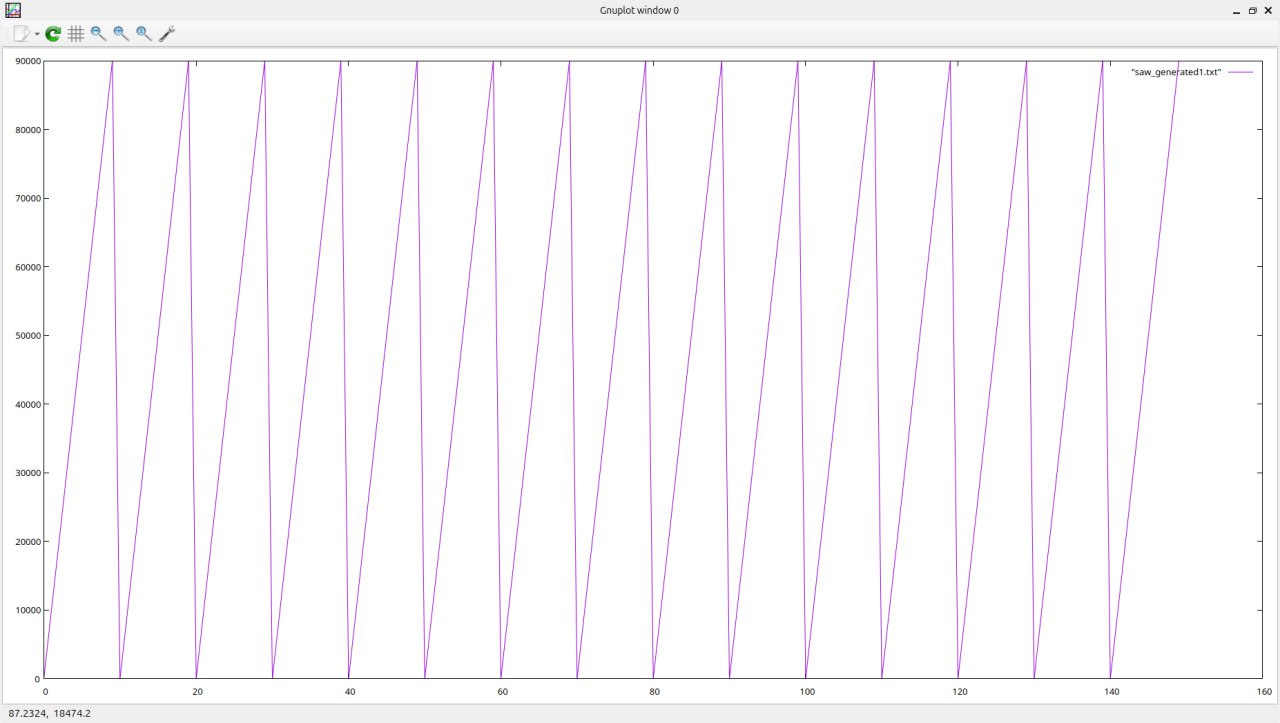


### Случайная последовательность:



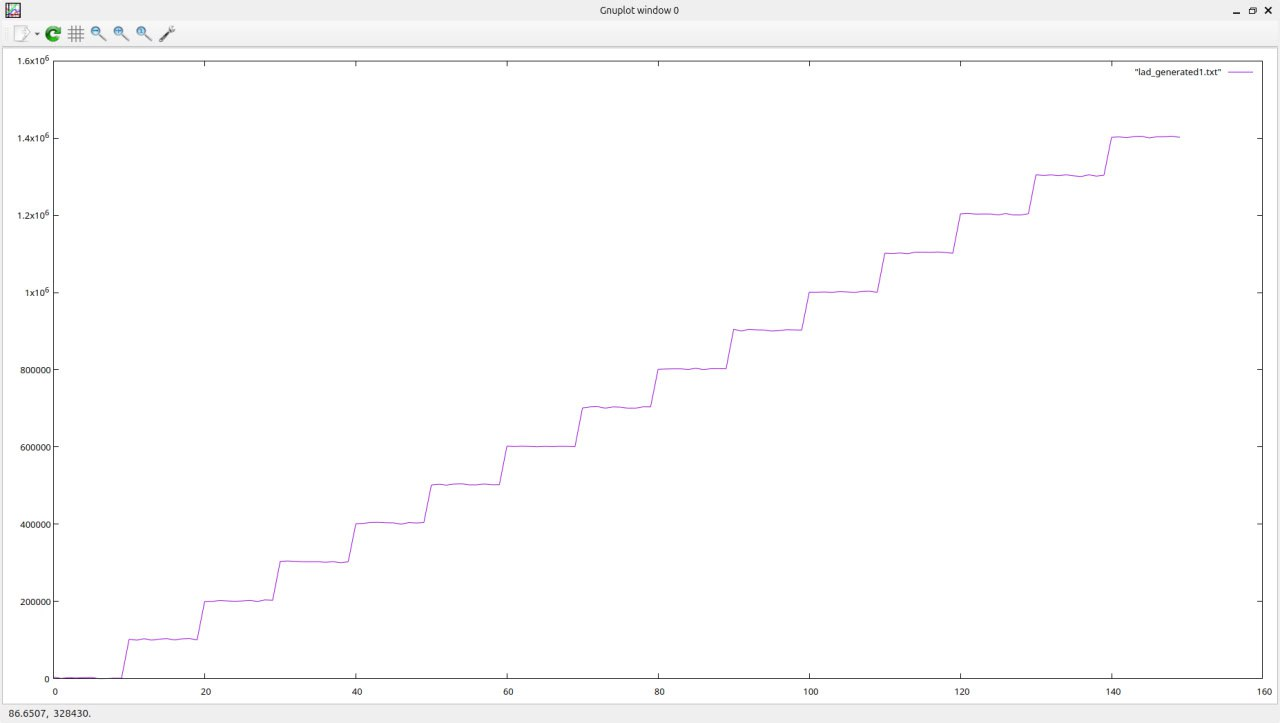
### 

### Частично упорядоченная пилообразная последовательность:

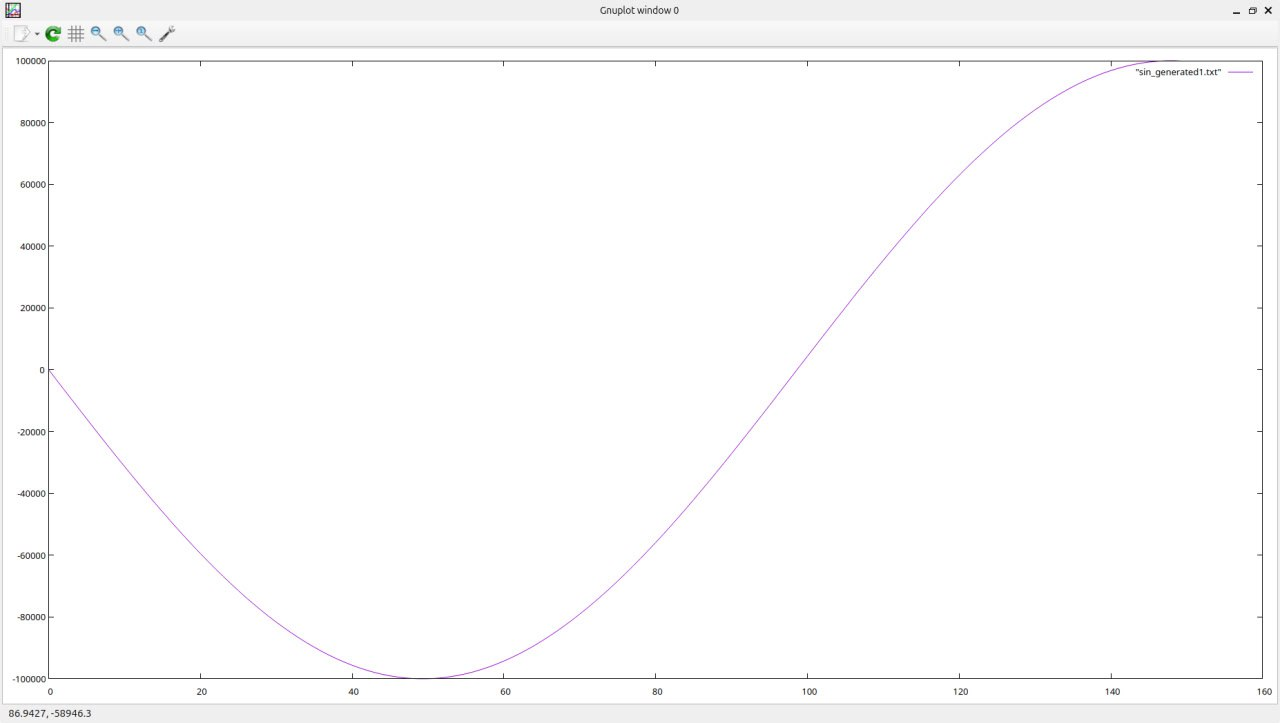


### 

### Частично упорядоченная ступенчатая последовательность:

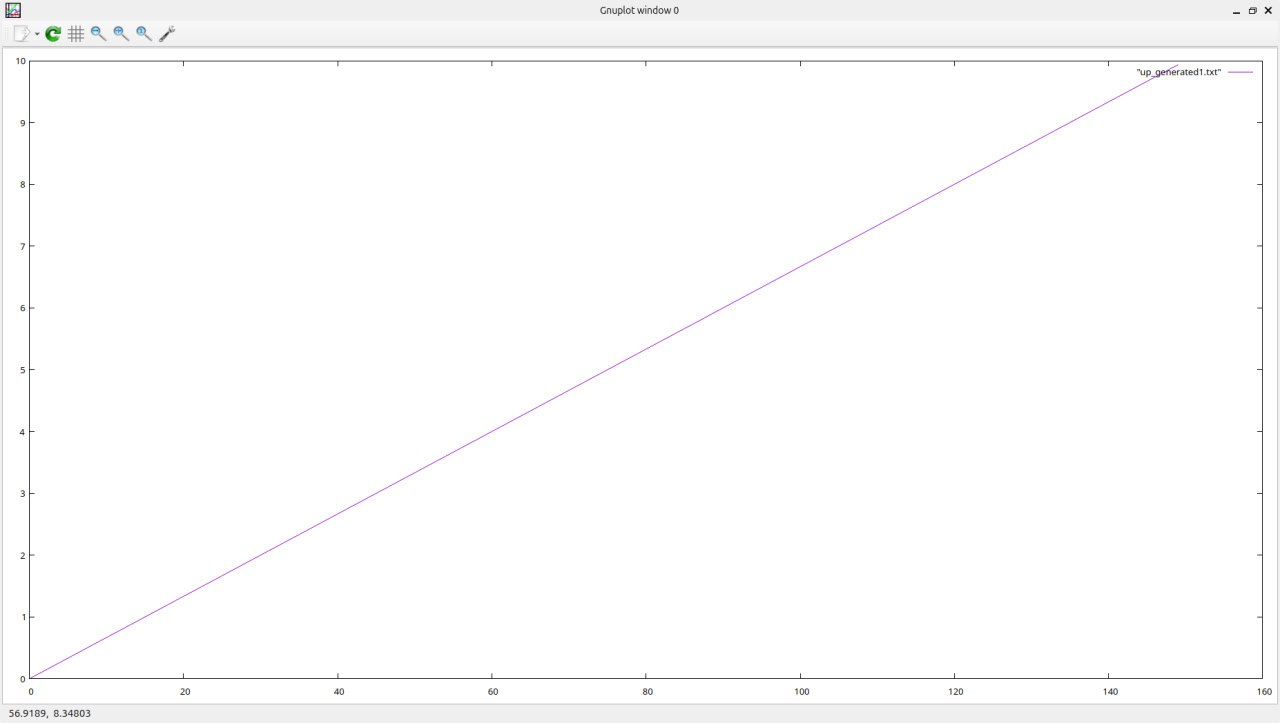


### Частично упорядоченная синусоидальная последовательность:

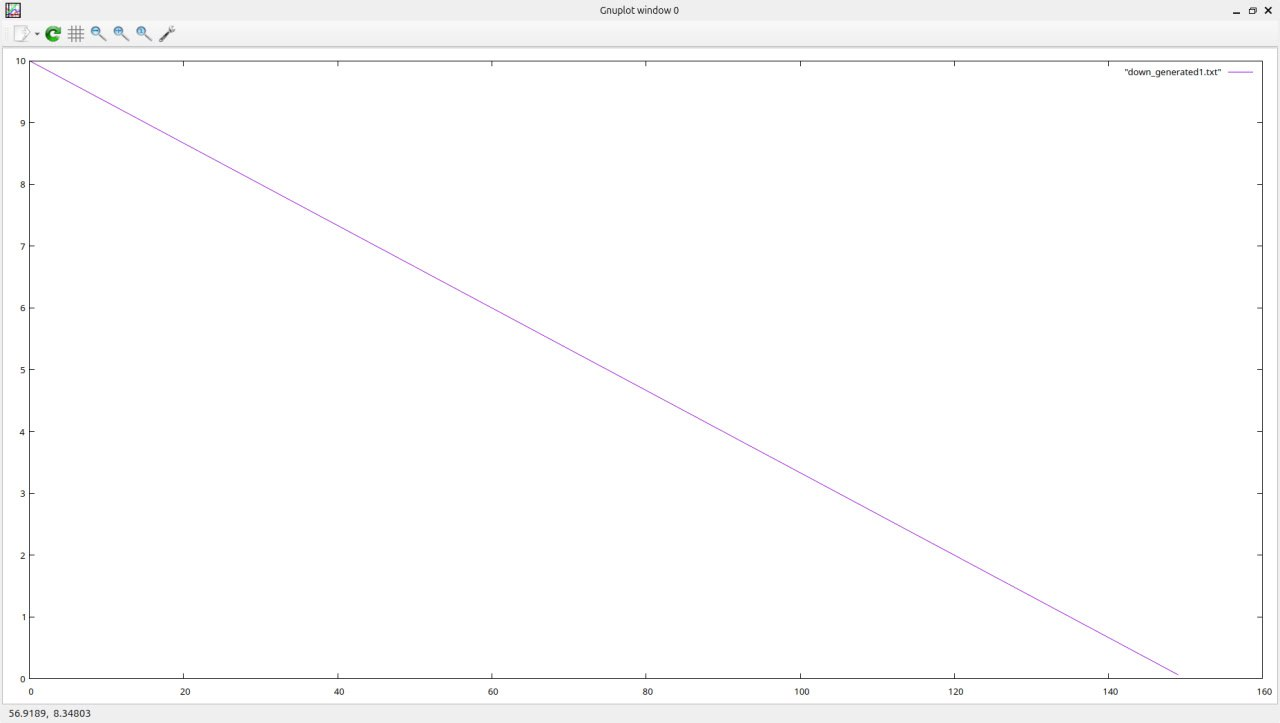


## Для вещественных значений:

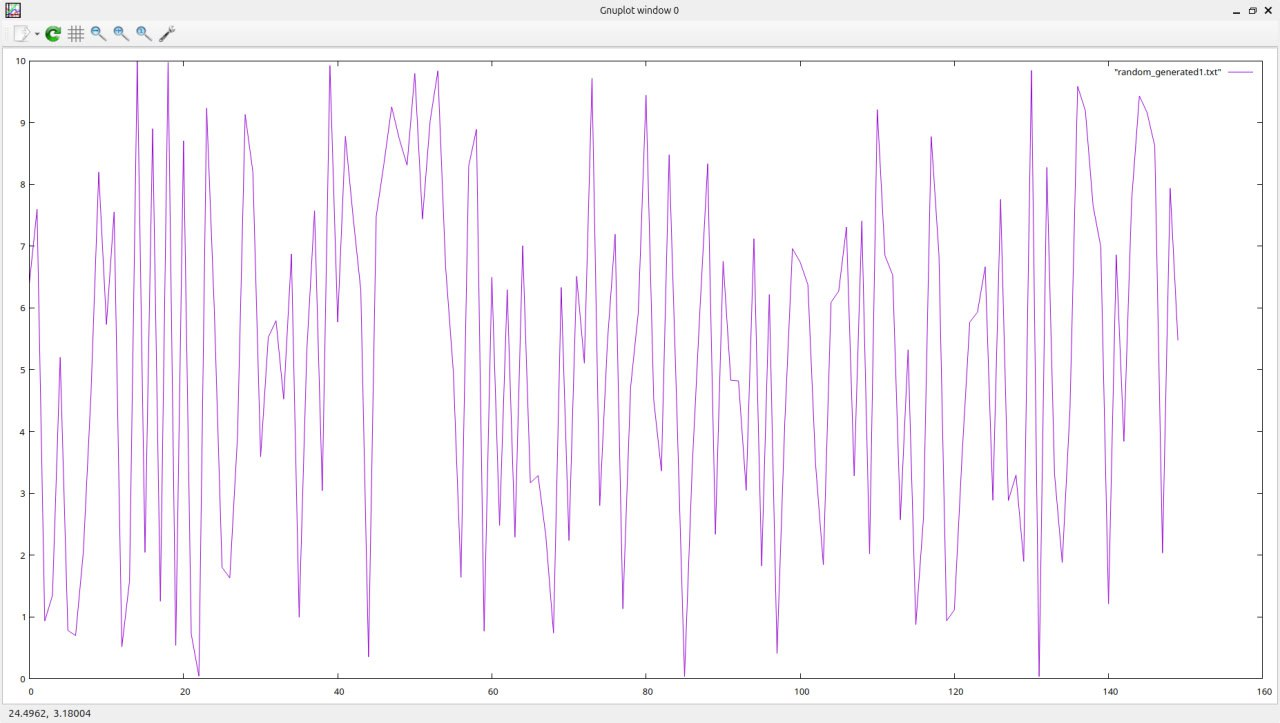
### Упорядоченные по возрастанию:



### Упорядоченные по убыванию:

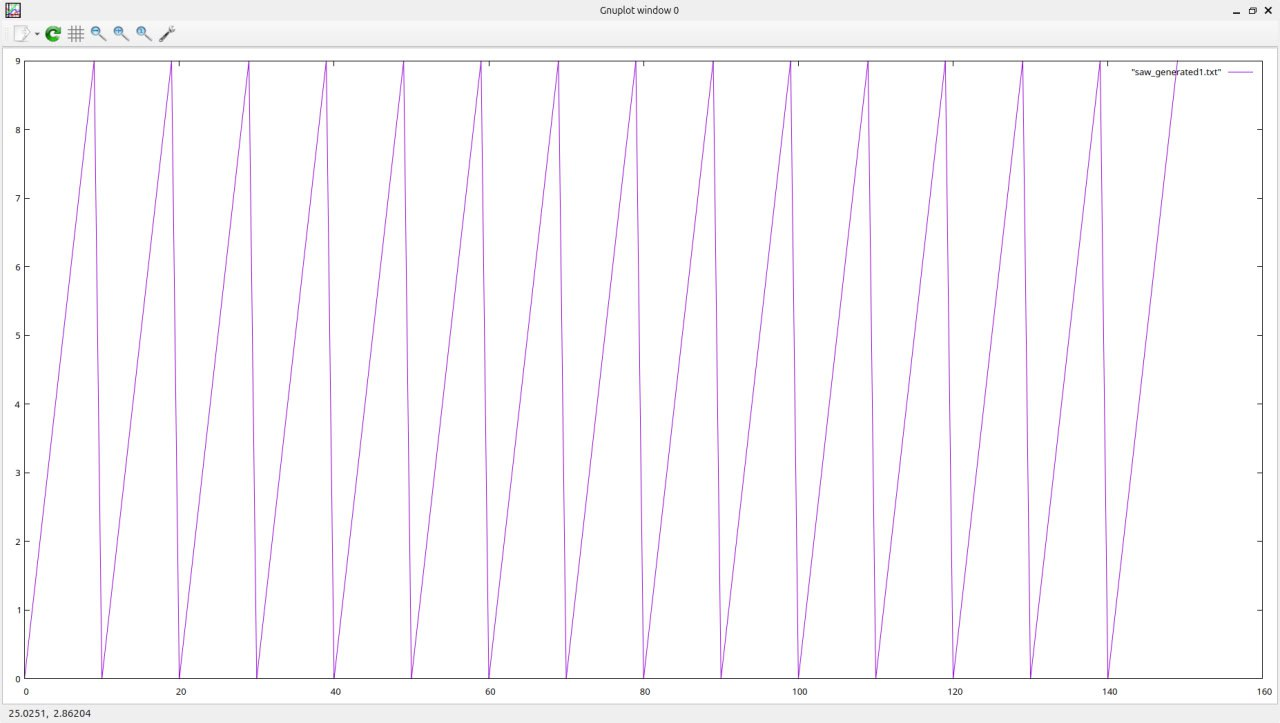


### Случайная последовательность:

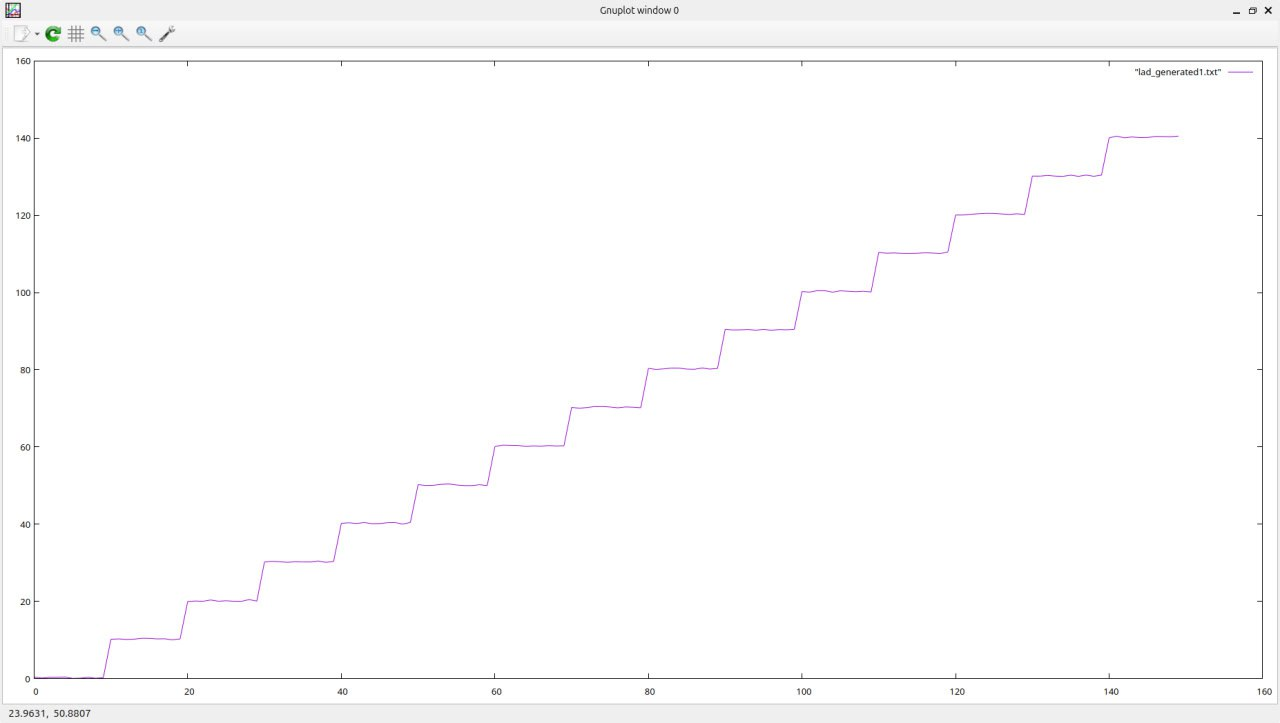


### 

### Частично упорядоченная пилообразная последовательность:

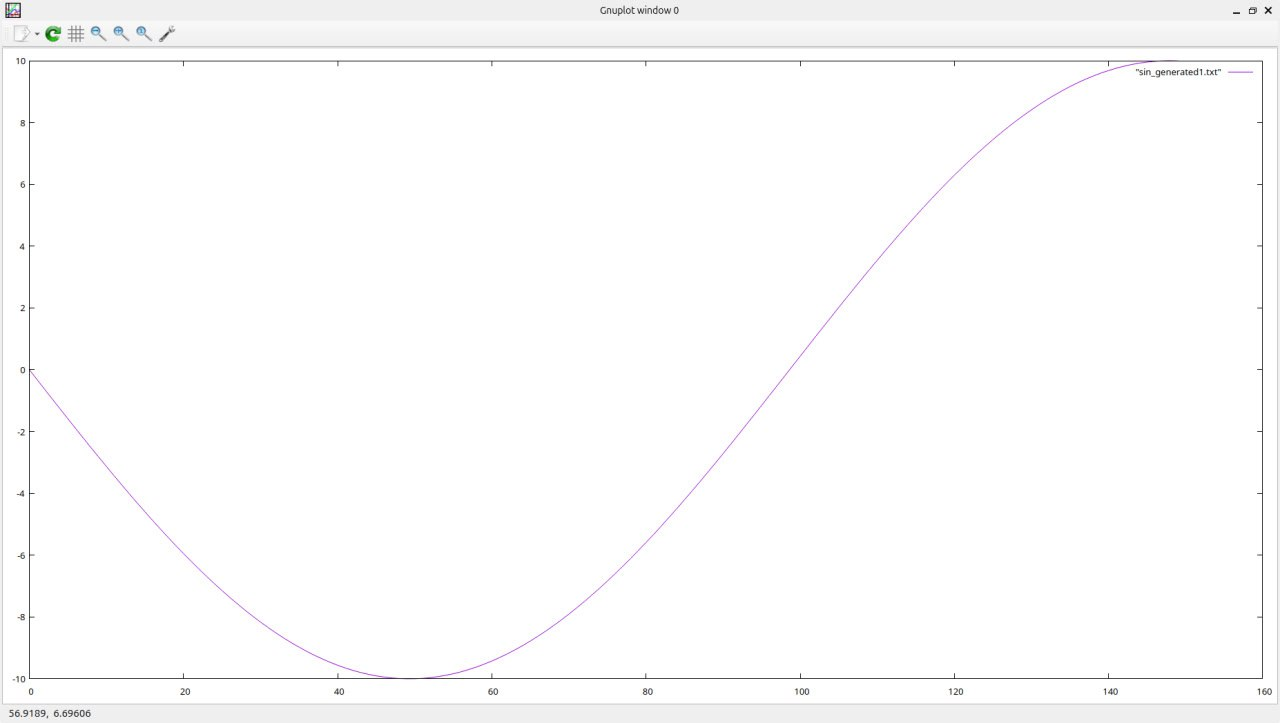


### Частично упорядоченная ступенчатая последовательность:



### 

### Частично упорядоченная синусоидальная последовательность:



## Графики зависимости времени формирования целочисленных последовательностей от количества элементов

## 

## Для вещественных значений:

### Упорядоченные по возрастанию:

|  |  |
| --- | --- |
| Размерность массива | Среднее время работы программы, (с\*10^-6) |
| 5\*10^5 | time saw: 7927  time sin: 19051  time lad: 17567  time up: 4627  time down: 4556  time random: 16906 |
| 10\*10^5 | time saw: 15397  time sin: 37964  time lad: 35060  time up: 9194  time down: 8662  time random: 23521 |
| 15\*10^5 | time saw: 12391  time sin: 22466  time lad: 17358  time up: 4544  time down: 4557  time random: 16574 |
| 20\*10^5 | time saw: 11692  time sin: 35798  time lad: 24331  time up: 6063  time down: 6043  time random: 21931 |
| 25\*10^5 | time saw: 13426  time sin: 57677  time lad: 28684  time up: 7483  time down: 7512  time random: 27554 |
| 30\*10^5 | time saw: 16467  time sin: 80351  time lad: 37547  time up: 9294  time down: 9216  time random: 32879 |
| 35\*10^5 | time saw: 18701  time sin: 103138  time lad: 40290  time up: 10415  time down: 10691  time random: 38714 |
| 40\*10^5 | time saw: 21549  time sin: 127489  time lad: 46015  time up: 12398  time down: 11767  time random: 44261 |
| 45\*10^5 | time saw: 24414  time sin: 148958  time lad: 51999  time up: 12895  time down: 14066  time random: 48716 |
| 50\*10^5 | time saw: 25671  time sin: 173000  time lad: 56409  time up: 14443  time down: 15267  time random: 53757 |

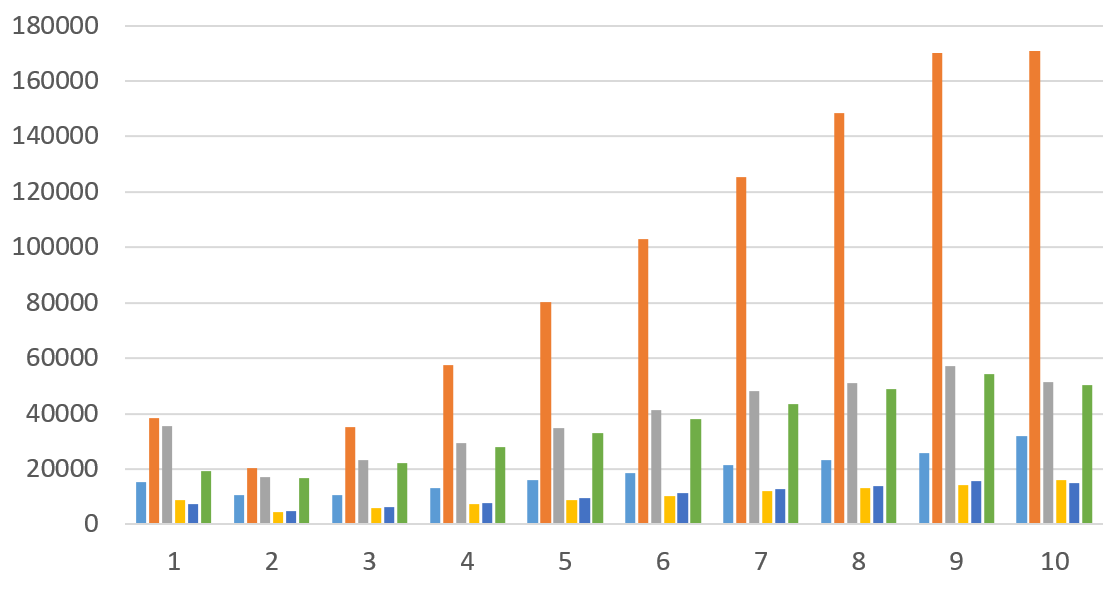
## Для целочисленных значений:

|  |  |
| --- | --- |
| Размерность массива | Среднее время работы программы, (с\*10^-6) |
| 5\*10^5 | time saw: 9933  time sin: 19785  time lad: 16132  time up: 4772  time down: 4592  time random: 15798 |
| 10\*10^5 | time saw: 19606  time sin: 37973  time lad: 32195  time up: 9728  time down: 9065  time random: 31204 |
| 15\*10^5 | time saw: 28820  time sin: 56848  time lad: 48613  time up: 14463  time down: 13273  time random: 46598 |
| 20\*10^5 | time saw: 39179  time sin: 107738  time lad: 53295  time up: 13277  time down: 10177  time random: 28989 0 |
| 25\*10^5 | time saw: 20110  time sin: 62083  time lad: 26429  time up: 7921  time down: 7514  time random: 25576 |
| 30\*10^5 | time saw: 19518  time sin: 80477  time lad: 30866  time up: 9546  time down: 8769  time random: 30579 |
| 35\*10^5 | time saw: 22771  time sin: 103195  time lad: 36864  time up: 10770  time down: 10539  time random: 36036 |
| 40\*10^5 | time saw: 25914  time sin: 125744  time lad: 41847  time up: 12924  time down: 12040  time random: 41109 0 |
| 45\*10^5 | time saw: 29135  time sin: 148638  time lad: 47398  time up: 14233  time down: 13465  time random: 46135 |
| 50\*10^5 | time saw: 31992  time sin: 170955  time lad: 51523  time up: 15916  time down: 14898  time random: 50357 |

### 

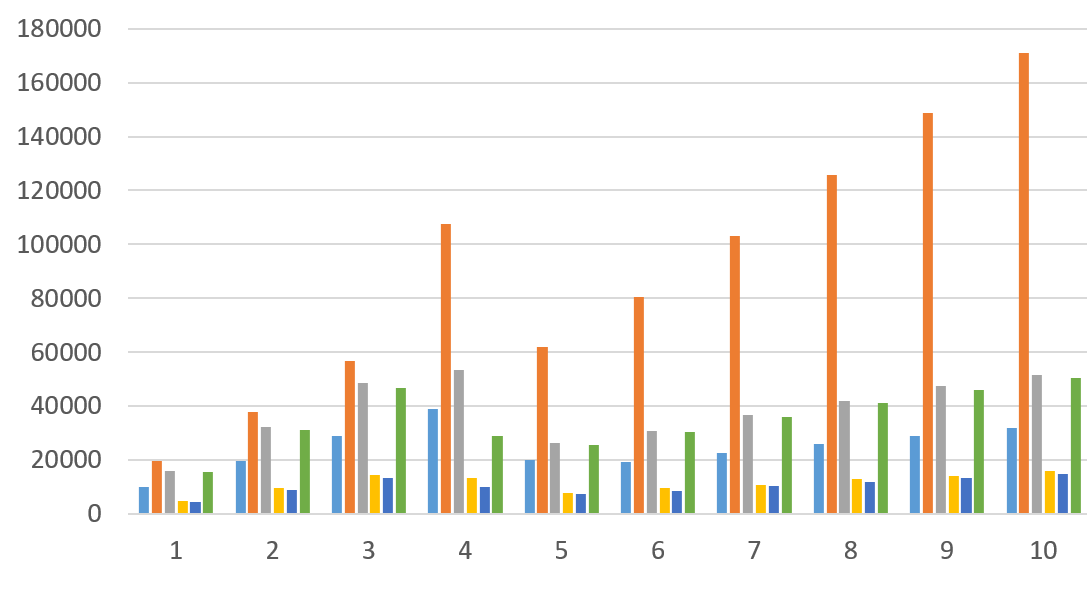
### 

**Время работы программы для вещественных значений на едином графике:**



1 (light blue) - time saw, 2 (orange)- time sin, 3 (gray) - time lad, 4 (yellow) - time up, 5 (dark blue) - time down, 6 (green) - time random.

**Время работы программы для целочисленных значений на едином графике:**



1 (light blue) - time saw, 2 (orange)- time sin, 3 (gray) - time lad, 4 (yellow) - time up, 5 (dark blue) - time down, 6 (green) - time random.

# Вывод

По итогу выполнения лабораторной работы можно сделать вывод по поводу зависимости времени выполнения создания числовой последовательности от её длинны и сложности. Для каждой числовой последовательность обнаружена линейная зависимость времени создания от количества элементов — чем больше элементов, тем больше время создания. Последовательности, упорядоченные по возрастанию и убыванию, для одинакового количества элементов создаются за одинаковое, самое минимальное время. Следующей по времени создания идёт пилообразная последовательность. Она создается медленнее, чем линейная последовательность, потому что применяется большее количество операций, в том числе взятие по модулю. Следующими по времени создания являются функции случайной и лестничной последовательности. В них используется функция rand() и некоторые другие операции.