МИНИСТЕРСТВО науки и высшего ОБРАЗОВАНИЯ РОссИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ

УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ

(национальный исследовательский университет)»

Институт №3 «Системы управления, информатика и электроэнергетика»

Кафедра № 304 «Вычислительные машины, системы и сети»

Программирование

Отчет по лабораторной работе № 1

Формирование массивов экспериментальных данных

Выполнили студенты группы М3О-209Б-22

Глазов В.В., Абдурахманов М.Р.

Проверила доцент, к.т.н., Дмитриева Е.А.

Москва 2023 г.

**Оглавление**

[Задание 3](#_Toc148292691)

[Структурные схемы алгоритмов 4](#_Toc148292692)

[Функция main 4](#_Toc148292693)

[Код программы 7](#_Toc148292694)

[Тестирование программы 13](#_Toc148292695)

[Тестирование некорректных режимов 13](#_Toc148292696)

[Тестирование корректных режимов 15](#_Toc148292697)

[Графики зависимостей времени от размерностей 18](#_Toc148292698)

[Вывод 20](#_Toc148292699)

# Задание

**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».

.

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(времени) 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.

# Структурные схемы алгоритмов

## **Функция main**

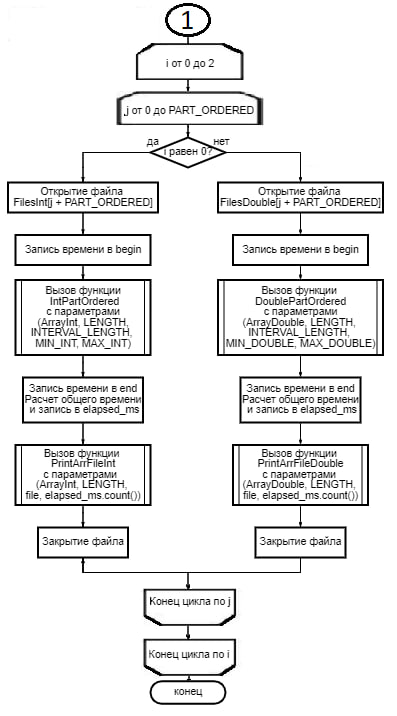
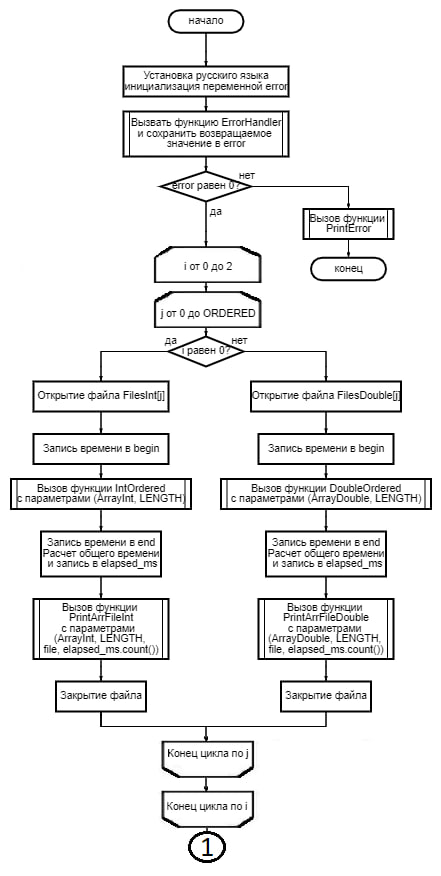


Рисунок 1. Структурная схема алгоритма функции main.**Функция SawSequenceInt**

1. Назначение:

Функция формирования «пилообразной»(частично-упорядоченной) последовательности

2. Прототип функции:

void SawSequenceInt(int Array[], int len, int intervalLen, int min, int max)

3. Обращение к функции:

IntPartOrdered[j](ArrayInt, LENGTH, INTERVAL\_LENGTH, MIN\_INT, MAX\_INT);

4. Блок-схема:



Рисунок 2. Структурная схема алгоритма функции random\_INT.

5. Описание параметров:

|  |  |  |  |
| --- | --- | --- | --- |
| Имя параметра | Тип параметра | Назначение | Вход или выход |
| Array | int | Массив данных | Вход |
| len | int | Длина массива | Вход |
| intervalLen | int | Длина интервала | Вход |
| min | int | Минимальное значение | Вход |
| max | int | Максимальное значение | Вход |

# Код программы

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Программирование \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*Project type :Win64 Console Application \*

\*Project name :lab1\_arrays.sln \*

\*File name :lab1\_arrays.cpp \*

\*Language :CPP, MSVS 2022 \*

\*Programmers : Абдурахманов М.Р., Глазов В.В., М3О-209Б-22 \*

\*Modified By : \*

\*Created: 03.10.2023 \*

\*Last revision:14.10.2023 \*

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <iostream>

#include <cmath>

#include <cstring>

#include <windows.h>

#include <chrono>

using namespace std;

//CONSTANTS

const int   SEQUENCES\_QUANTITY = 6,

            ORDERED = 3,

            PART\_ORDERED = 3,

            MIN\_INT = -1000,

            MAX\_INT = 1000,

            LENGTH = 3500000,

            INTERVAL\_LENGTH = 5000;

const double    MIN\_DOUBLE = -1000,

                MAX\_DOUBLE = 1000;

//VARIABLES

int ArrayInt[LENGTH];

double ArrayDouble[LENGTH];

//GENERATING INTEGER SEQUENCES

void AscendingSequenceInt(int[], int);

void DescendingSequenceInt(int[], int);

void RandomSequenceInt(int[], int);

void SawSequenceInt(int[], int, int, int, int);

void SinSequenceInt(int[], int, int, int, int);

void StepSequenceInt(int[], int, int, int, int);

//GENERATING DOUBLE SEQUENCES

void AscendingSequenceDouble(double[], int);

void DescendingSequenceDouble(double[], int);

void RandomSequenceDouble(double[], int);

void SawSequenceDouble(double[], int, int, double, double);

void SinSequenceDouble(double[], int, int, double, double);

void StepSequenceDouble(double[], int, int, double, double);

//PRINTING SEQUENCES

void PrintArrConsoleInt(int[], int);

void PrintArrFileInt(int[], int, FILE\*, int64\_t);

void PrintArrConsoleDouble(double[], int);

void PrintArrFileDouble(double[], int, FILE\*, int64\_t);

//POINTERS INTEGER FUNCTIONS

void (\*funcInt1)(int[], int) = AscendingSequenceInt;

void (\*funcInt2)(int[], int) = DescendingSequenceInt;

void (\*funcInt3)(int[], int) = RandomSequenceInt;

void (\*funcInt4)(int[], int, int, int, int) = SawSequenceInt;

void (\*funcInt5)(int[], int, int, int, int) = SinSequenceInt;

void (\*funcInt6)(int[], int, int, int, int) = StepSequenceInt;

void(\*IntOrdered[ORDERED])(int[], int) ={funcInt1, funcInt2, funcInt3};

void(\*IntPartOrdered[PART\_ORDERED])(int[], int, int, int, int) ={funcInt4, funcInt5, funcInt6};

//POINTERS DOUBLE FUNCTIONS

void (\*funcDouble1)(double[], int) = AscendingSequenceDouble;

void (\*funcDouble2)(double[], int) = DescendingSequenceDouble;

void (\*funcDouble3)(double[], int) = RandomSequenceDouble;

void (\*funcDouble4)(double[], int, int, double, double) = SawSequenceDouble;

void (\*funcDouble5)(double[], int, int, double, double) = SinSequenceDouble;

void (\*funcDouble6)(double[], int, int, double, double) = StepSequenceDouble;

void(\*DoubleOrdered[ORDERED])(double[], int) ={funcDouble1, funcDouble2, funcDouble3};

void(\*DoublePartOrdered[PART\_ORDERED])(double[], int, int, double, double) ={funcDouble4, funcDouble5, funcDouble6};

//ERROR HANDLERS

int ErrorHandler(int, int, double, double, int, int);

void PrintError(int);

//FILES

//INT

const char AscendingInt[] = "output/AscendingInt.txt";

const char DescendingInt[] = "output/DescendingInt.txt";

const char RandomInt[] = "output/RandomInt.txt";

const char SawInt[] = "output/SawInt.txt";

const char SinInt[] = "output/SinInt.txt";

const char StepInt[] = "output/StepInt.txt";

//DOUBLE

const char AscendingDouble[] = "output/AscendingDouble.txt";

const char DescendingDouble[] = "output/DescendingDouble.txt";

const char RandomDouble[] = "output/RandomDouble.txt";

const char SawDouble[] = "output/SawDouble.txt";

const char SinDouble[] = "output/SinDouble.txt";

const char StepDouble[] = "output/StepDouble.txt";

const char\* FilesInt[6] = {AscendingInt, DescendingInt, RandomInt, SawInt, SinInt, StepInt};

const char\* FilesDouble[6] = {AscendingDouble, DescendingDouble, RandomDouble, SawDouble, SinDouble, StepDouble};

int main(){

    setlocale(LC\_ALL, "ru");

    int error = ErrorHandler(MIN\_INT, MAX\_INT, MIN\_DOUBLE, MAX\_DOUBLE, LENGTH, INTERVAL\_LENGTH);

    if (error != 0){

        PrintError(error);

        return error;

    }

    for (int i = 0; i < 2; i++){

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

            if (i == 0){

                FILE \*file = fopen(FilesInt[j], "w");

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

                IntOrdered[j](ArrayInt, LENGTH);

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

                auto elapsed\_ms = chrono::duration\_cast<chrono::microseconds>(end - begin);

                PrintArrFileInt(ArrayInt, LENGTH, file, elapsed\_ms.count());

                fclose(file);

            } else{

                FILE \*file = fopen(FilesDouble[j], "w");

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

                DoubleOrdered[j](ArrayDouble, LENGTH);

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

                auto elapsed\_ms = chrono::duration\_cast<chrono::microseconds>(end - begin);

                PrintArrFileDouble(ArrayDouble, LENGTH, file, elapsed\_ms.count());

                fclose(file);

            }

        }

    }

    for (int i = 0; i < 2; i++){

        for (int j = 0; j < PART\_ORDERED; j++){

            if (i == 0){

                FILE \*file = fopen(FilesInt[j + PART\_ORDERED], "w");

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

                IntPartOrdered[j](ArrayInt, LENGTH, INTERVAL\_LENGTH, MIN\_INT, MAX\_INT);

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

                auto elapsed\_ms = chrono::duration\_cast<chrono::microseconds>(end - begin);

                PrintArrFileInt(ArrayInt, LENGTH, file, elapsed\_ms.count());

                fclose(file);

            } else{

                FILE \*file = fopen(FilesDouble[j + PART\_ORDERED], "w");

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

                DoublePartOrdered[j](ArrayDouble, LENGTH, INTERVAL\_LENGTH, MIN\_DOUBLE, MAX\_DOUBLE);

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

                auto elapsed\_ms = chrono::duration\_cast<chrono::microseconds>(end - begin);

                PrintArrFileDouble(ArrayDouble, LENGTH, file, elapsed\_ms.count());

                fclose(file);

            }

        }

    }

}

//error handler

int ErrorHandler(int minInt, int maxInt, double minDouble, double maxDouble, int length, int intervalLength){

    if (maxInt < minInt) return 1;

    if (maxDouble < minDouble) return 2;

    if (length < 1) return 3;

    if (length < intervalLength) return 4;

    if (intervalLength < 1) return 5;

    return 0;

}

//print an error

void PrintError(int error){

    printf("ERROR: ");

    switch (error){

        case 1:

            printf("MIN\_INT must be less then MAX\_INT. error code: %d\n", error);

            break;

        case 2:

            printf("MIN\_DOUBLE must be less then MAX\_DOUBLE. error code: %d\n", error);

            break;

        case 3:

            printf("LENGTH must be greater then 0. error code: %d\n", error);

            break;

        case 4:

            printf("INTERVAL\_LENGTH must be less then LENGTH. error code: %d\n", error);

            break;

        case 5:

            printf("INTERVAL\_LENGTH must be greater then 0. error code: %d\n", error);

            break;

    }

}

//print int array to console

void PrintArrConsoleInt(int Array[], int len){

    if (len >= 100000){

        return;

    }

    for (int i = 0; i < len; i++){

        char end[3];

        if (i != len - 1){

            strcpy(end, ", ");

        } else{

            strcpy(end, "]\n");

        }

        printf("%d%s", Array[i], end);

    }

}

//print int array to file

void PrintArrFileInt(int Array[], int len, FILE \*file, int64\_t time){

    fprintf(file, "Время работы алгоритма по формированию последовательности: %d(мкС)\n", time);

    if (len >= 100000){

        return;

    }

    for (int i = 0; i < len; i++){

        fprintf(file, "%d\n", Array[i]);

    }

    fprintf(file, "\n");

}

//generates an ascending integer sequence

void AscendingSequenceInt(int Array[], int len){

    double k = ((MAX\_INT - MIN\_INT) \* 1.0) / (len \* 1.0), b = MIN\_INT;

    for (int i = 0; i < len; i++){

        Array[i] = k \* i + b;

    }

}

//generates a descending integer sequence

void DescendingSequenceInt(int Array[], int len){

    double k = ((MAX\_INT - MIN\_INT) \* -1.0) / (len \* 1.0), b = MAX\_INT;

    for (int i = 0; i < len; i++){

        Array[i] = k \* i + b;

    }

}

//generates a random integer sequence

void RandomSequenceInt(int Array[], int len){

    srand(time(NULL));

    for (int i = 0; i < len; i++){

        Array[i] = MIN\_INT + rand() % (MAX\_INT - MIN\_INT + 1);

    }

}

//generates a saw integer sequence

void SawSequenceInt(int Array[], int len, int intervalLen, int min, int max){

    double k = ((max - min) \* 1.0) / (intervalLen \* 1.0);

    for (int i = 0; i < len; i++){

        Array[i] = (int)(k \* (i % intervalLen)) + min;

    }

}

//generates a sinusoidal integer sequence

void SinSequenceInt(int Array[], int len, int intervalLen, int min, int max){

    double x = 2 \* M\_PI / (intervalLen \* 1.0);

    int k = (max - min) / 2;

    for (int i = 0; i < len; i++){

        Array[i] = k \* sin(i \* x) + min + k;

    }

}

//generates a step integer sequence

void StepSequenceInt(int Array[], int len, int intervalLen, int min, int max){

    int intMin, intMax, intCount, stepHigh;

    if (len % intervalLen == 0){

        intCount = len / intervalLen;

    } else{

        intCount = len / intervalLen + 1;

    }

    stepHigh = (max - min) / intCount;

    intMin = min;

    intMax = min + stepHigh;

    srand(time(NULL));

    for (int i = 0; i < len; i++){

        if (i > 0 && i < len - 1 && i % intervalLen == 0 &&  (i + 1) / intervalLen > 0){

            intMin = intMax;

            intMax += stepHigh;

        }

        Array[i] = rand() % (intMax - intMin + 1) + intMin;

    }

}

//print double array to console

void PrintArrConsoleDouble(double Array[], int len){

    if (len >= 100000){

        return;

    }

    for (int i = 0; i < len; i++){

        char end[3];

        if (i != len - 1){

            // end = ", ";

            strcpy(end, ", ");

        } else{

            strcpy(end, "]\n");

        }

        printf("%.3f%s", Array[i], end);

    }

}

//print double array to file

void PrintArrFileDouble(double Array[], int len, FILE \*file, int64\_t time){

    fprintf(file, "Время работы алгоритма по формированию последовательности: %d(мкС)\n", time);

    if (len >= 100000){

        return;

    }

    for (int i = 0; i < len; i++){

        fprintf(file, "%.3f\n", Array[i]);

    }

    fprintf(file, "\n");

}

//generates an ascending double sequence

void AscendingSequenceDouble(double Array[], int len){

    double k = ((double)(MAX\_DOUBLE - MIN\_DOUBLE) - 0.125) / (double)(len), b = MIN\_DOUBLE + 0.875;

    for (int i = 0; i < len; i++){

        Array[i] = k \* i + b;

    }

}

//generates a descending double sequence

void DescendingSequenceDouble(double Array[], int len){

    double k = ((MAX\_DOUBLE - MIN\_DOUBLE - 0.125) \* -1.0) / (len \* 1.0), b = MAX\_DOUBLE - 0.125;

    for (int i = 0; i < len; i++){

        Array[i] = k \* i + b;

    }

}

//generates a random double sequence

void RandomSequenceDouble(double Array[], int len){

    srand(time(NULL));

    for (int i = 0; i < len; i++){

        Array[i] =  (double)(rand())/RAND\_MAX \* (MAX\_DOUBLE - MIN\_DOUBLE) + MIN\_DOUBLE;

    }

}

//generates a saw double sequence

void SawSequenceDouble(double Array[], int len, int intervalLen, double min, double max){

    double k = (max - min) / (double)(intervalLen);

    for (int i = 0; i < len; i++){

        Array[i] = (double)(k \* (i % intervalLen)) + min;

    }

}

//generates a sinusoidal double sequence

void SinSequenceDouble(double Array[], int len, int intervalLen, double min, double max){

    double x = 2 \* M\_PI / (intervalLen \* 1.0);

    double k = (max - min) / (double)(2);

    for (int i = 0; i < len; i++){

        Array[i] = k \* sin(i \* x) + min + k;

    }

}

//generates a step double sequence

void StepSequenceDouble(double Array[], int len, int intervalLen, double min, double max){

    int intMin, intMax, intCount;

    double stepHigh;

    if (len % intervalLen == 0){

        intCount = len / intervalLen;

    } else{

        intCount = len / intervalLen + 1;

    }

    stepHigh = (max - min) / (double)(intCount);

    intMin = min;

    intMax = min + stepHigh;

    srand(time(NULL));

    for (int i = 0; i < len; i++){

        if (i > 0 && i < len - 1 && i % intervalLen == 0 && (i + 1) / intervalLen > 0){

            intMin = intMax;

            intMax += stepHigh;

        }

        Array[i] =  (double)(rand())/RAND\_MAX \* (intMax - intMin) + intMin;

    }

}

# Тестирование программы

## Тестирование некорректных режимов

**Тест 1**

Цель: проверить работу программы при MIN\_INT >= MAX\_INT

Исходные данные:

SEQUENCES\_QUANTITY = 6,

ORDERED = 3,

PART\_ORDERED = 3,

**MIN\_INT = 2000,**

**MAX\_INT = 1000,**

LENGTH = 3500000,

INTERVAL\_LENGTH = 5000;

MIN\_DOUBLE = -1000,

MAX\_DOUBLE = 1000;

Ожидаемый результат: «ERROR: MIN\_INT must be less then MAX\_INT. error code: 1»

Полученный результат:



Вывод по тесту: полученный результат совпал с ожидаемым.

**Тест 2**

Цель: проверить работу программы при MIN\_DOUBLE >= MAX\_ DOUBLE

Исходные данные:

SEQUENCES\_QUANTITY = 6,

ORDERED = 3,

PART\_ORDERED = 3,

MIN\_INT = -1000,

MAX\_INT = 1000,

LENGTH = 3500000,

INTERVAL\_LENGTH = 5000;

**MIN\_DOUBLE = 2222.222,**

**MAX\_DOUBLE = 1000;**

Ожидаемый результат: «ERROR: MIN\_DOUBLE must be less then MAX\_DOUBLE. error code: 2»

Полученный результат:



Вывод по тесту: полученный результат совпал с ожидаемым.

**Тест 3**

Цель: проверить работу программы при LENGTH < 1

Исходные данные:

SEQUENCES\_QUANTITY = 6,

ORDERED = 3,

PART\_ORDERED = 3,

MIN\_INT = -1000,

MAX\_INT = 1000,

LENGTH = -200,

INTERVAL\_LENGTH = 5000;

MIN\_DOUBLE = -1000,

MAX\_DOUBLE = 1000;

Ожидаемый результат: «ERROR: LENGTH must be greater then 0. error code: 3».

Полученный результат:



Вывод по тесту: полученный результат совпал с ожидаемым.

**Тест 4**

Цель: проверить работу программы при INTERVAL\_LENGTH > LENGTH

Исходные данные:

SEQUENCES\_QUANTITY = 6,

ORDERED = 3,

PART\_ORDERED = 3,

MIN\_INT = -1000,

MAX\_INT = 1000,

LENGTH = 1000,

INTERVAL\_LENGTH = 5000;

MIN\_DOUBLE = -1000,

MAX\_DOUBLE = 1000;

Ожидаемый результат: «ERROR: INTERVAL\_LENGTH must be less then LENGTH. error code: 4»

Полученный результат:



Вывод по тесту: полученный результат совпал с ожидаемым.

**Тест 5**

Цель: проверить работу программы при INTERVAL\_LENGTH < 1

Исходные данные:

SEQUENCES\_QUANTITY = 6,

ORDERED = 3,

PART\_ORDERED = 3,

MIN\_INT = -1000,

MAX\_INT = 1000,

LENGTH = 1000,

INTERVAL\_LENGTH = -2;

MIN\_DOUBLE = -1000,

MAX\_DOUBLE = 1000;

Ожидаемый результат: « ERROR: INTERVAL\_LENGTH must be greater then 0. error code: 5

Полученный результат:



Вывод по тесту: полученный результат совпал с ожидаемым.

## Тестирование корректных режимов

**Тест 1**

Цель: проверить работу программы при длине массива = 200, построить графики полученных последовательностей и доказать корректность полученных результатов.

Исходные данные:

SEQUENCES\_QUANTITY = 6,

ORDERED = 3,

PART\_ORDERED = 3,

MIN\_INT = -100,

MAX\_INT = -100,

LENGTH = 200,

INTERVAL\_LENGTH = 90;

MIN\_DOUBLE = -100,

MAX\_DOUBLE = 100;

Полученный результат:

Вывод по тесту: полученный результат совпал с ожидаемым.

# Графики зависимостей времени от размерностей

Таблица зависимости времени (мкс) генерирования массива от размерности.

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Length** | **AscInt** | **AscDouble** | **DescInt** | **DescDouble** | **RandInt** | **RandDouble** | **SawInt** | **SawDouble** | **SinInt** | **SinDouble** | **StepInt** | **StepDouble** |
| **5\*10^5** | 1094 | 1811 | 579 | 662 | 4505 | 4327 | 950 | 783 | 20011 | 20076 | 4363 | 4830 |
| **10 \* 10^5** | 2077 | 2942 | 1070 | 1293 | 8932 | 8546 | 1545 | 1463 | 37458 | 36747 | 8023 | 9334 |
| **15 \* 10^5** | 3156 | 4882 | 1527 | 2099 | 13196 | 13129 | 2302 | 2243 | 55660 | 54992 | 12046 | 13953 |
| **20 \* 10^5** | 4401 | 5907 | 2116 | 2699 | 17302 | 15585 | 2890 | 3143 | 77288 | 80332 | 17500 | 19694 |
| **25 \* 10^5** | 5044 | 8276 | 2427 | 3310 | 21217 | 20301 | 3744 | 3677 | 92082 | 94542 | 20217 | 22856 |
| **30 \* 10^5** | 6256 | 8886 | 3447 | 4134 | 26389 | 25796 | 4817 | 4703 | 121388 | 119807 | 27534 | 27628 |
| **35 \* 10^5** | 8988 | 12219 | 3631 | 4359 | 35945 | 30706 | 5383 | 5485 | 136715 | 136767 | 31028 | 32088 |
| **40 \* 10^5** | 9623 | 12559 | 5204 | 5254 | 36660 | 31181 | 5753 | 6059 | 151310 | 155366 | 32963 | 36016 |
| **45 \* 10^5** | 10595 | 13672 | 4738 | 6267 | 36358 | 36494 | 6552 | 6575 | 168604 | 167571 | 37021 | 39710 |
| **50 \* 10^5** | 10804 | 15621 | 5459 | 6812 | 44068 | 41598 | 7796 | 7465 | 199909 | 189553 | 42052 | 47635 |

По x – размер массива, по y – время формирования(мкс).

Вывод

Разработка программы остановлена на том основании, что:

1. Полученные результаты совпали с ожидаемыми.
2. Считаем набор тестов полным.