БУ ВО Ханты-Мансийского автономного округа – Югры

«Сургутский государственный университет»

Политехнический институт

Кафедра автоматики и компьютерных систем

ОТЧЁТ

по лабораторной работе №1

по дисциплине: «Алгоритмы и структуры данных»

Выполнил: студент группы №609-22,

Бельтюков Михаил Олегович

Принял: старший преподаватель кафедры АиКС

Назаров Евгений Владимирович

Сургут

2024г.

**Цель работы**

Освоить принципы формирования монотонных (упорядоченных и упорядоченных в обратном порядке), частично упорядоченных и случайных последовательностей данных; изучить функции, позволяющие производить оценку длительности времени выполнения алгоритмов

**Задание**

1. Разработать функции, формирующие упорядоченные (в том числе и в обратном порядке), частично упорядоченные и случайные последовательности целых чисел и чисел с плавающей запятой. При реализации функций считать, что выделение памяти под последовательности происходит вне этих функций, а в качестве формальных параметров функции получают указатель на массив, его размер и, возможно, диапазон изменения величин и длину интервалов (для частично упорядоченных последовательностей). Рекомендуется продумать единый прототип для функций, формирующих последовательности, определить его и использовать массив указателей на функции для автоматизации сбора статистической информации.
2. Оценить длительность формирования последовательностей всех типов для нескольких значений размеров последовательностей (5⋅105, 10⋅105, …, 50⋅105), и на основе полученных значений построить графики зависимостей длительностей формирования от размера последовательностей.
3. Составить отчет, в котором привести блок-схемы функций (возможно, на примере одной функции), реализующих формирование частично упорядоченных последовательностей, и главной функции, полученные графики зависимостей и выводы по полученным результатам.

**Ход работы**

Для удобства применения в будущих заданиях, функции генерации разработаны как методы классов “Массив целых чисел” и “Массив дробных чисел”.

void swap\_arr(void \* arr, int i, int j, int doub\_or\_int);

class double\_array {

private:

int size;

double \* arr;

void inversion\_gen(int start);

public:

void output();

void gen\_rand(double min\_val, double max\_val);

void gen\_up(double min\_val, double max\_val, double step);

void gen\_down(double min\_val, double max\_val, double step);

void step\_gen(double min\_val, double max\_val);

void sin\_gen(double min\_val, double max\_val);

void sawtooth\_gen(double min\_val, double max\_val);

void kvazi\_gen(double min\_val, double max\_val);

double\_array(int n) {

arr = new double[n];

size = n;

}

};

class int\_array {

private:

int \* arr;

int size;

void inversion\_gen(int start);

public:

void output();

void gen\_rand(int min\_val, int max\_val);

void gen\_up(int min\_val, int max\_val, int step);

void gen\_down(int min\_val, int max\_val, int step);

void sin\_gen(int min\_val, int max\_val);

void sawtooth\_gen(int min\_val, int max\_val);

void step\_gen(int min\_val, int max\_val);

void kvazi\_gen(int min\_val, int max\_val);

int\_array(int n) {

arr = new int[n];

for (int i=0;i<n;i++) arr[i] = 0;

size = n;

}

};

Листинг 1 – Определение классов

Для генерации последовательностей по возрастанию и убыванию запрашиваются пороговые значения и шаг движения числа. В случае если было превышено пороговое значение, значение всех последующих и текущего элементов массива становятся равными пороговому.

void int\_array::gen\_up(int min\_val, int max\_val, int step) {

int val = min\_val;

int i = 0;

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

arr[i] = val;

if (val+step <= max\_val)

val += step;

else

val = max\_val;

}

}

void int\_array::gen\_down(int min\_val, int max\_val, int step) {

int val = max\_val;

int i = 0;

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

arr[i] = val;

if (val-step >= min\_val)

val -= step;

else

val = min\_val;

}

}

Листинг 2 – Генерация убывающей и возрастающей последовательностей

void int\_array::gen\_rand(int min\_val, int max\_val) {

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

arr[i] = rand() % (max\_val-min\_val+1) + min\_val;

}

}

Листинг 3 – генерация случайной последовательности

\*\*График с калькофиса\*\*

Все частично упорядоченные последовательности принимают пороговые значения, шаг рассчитывается внутри методов.

void int\_array::sin\_gen(int min\_val, int max\_val) {

int mid, i = 0, n = size, j = 0, val = max\_val, step=(max\_val-min\_val)/interval;

mid = interval / 2;

while (n - interval >= 0) {

for (i=0+j;i<mid+j;i++) {

arr[i] = val;

if (val-step >= min\_val)

val -= step;

else

val = min\_val;

}

for (i=mid+j;i<interval+j;i++) {

arr[i] = val;

if (val+step <= max\_val)

val += step;

else

val = max\_val;

}

n -= interval;

j += interval;

val = max\_val;

}

}

Листинг 4 – Генерация синусоидальной последовательности

void int\_array::sawtooth\_gen(int min\_val, int max\_val) {

int i=0, j=0, mn = min\_val, mx = max\_val, l=0, n=size, step=(max\_val-min\_val)/interval;

while (n - interval >= 0) {

for (i=0+j;i<interval+j;i++) {

if (i % 2 == l) {

arr[i] = mn;

mn+=step;

}

else {

arr[i] = mx;

mx-=step;

}

}

l = (l+1)%2;

n -= interval;

j += interval;

mx = max\_val;

mn = min\_val;

}

}

Листинг 5 – генерация пилообразной последовательности

void int\_array::step\_gen(int min\_val, int max\_val) {

int range = max\_val - min\_val, i=0, j=0, n=size, step = range / (n / interval) / 2, mn=step+min\_val, mx = 2\*step+min\_val;

while (n - interval >= 0) {

for (i=0+j;i<interval+j;i++) {

arr[i] = rand() % (mx-mn+1) + mn;

}

n -= interval;

j += interval;

mn += 2 \* step;

mx += 2 \* step;

}

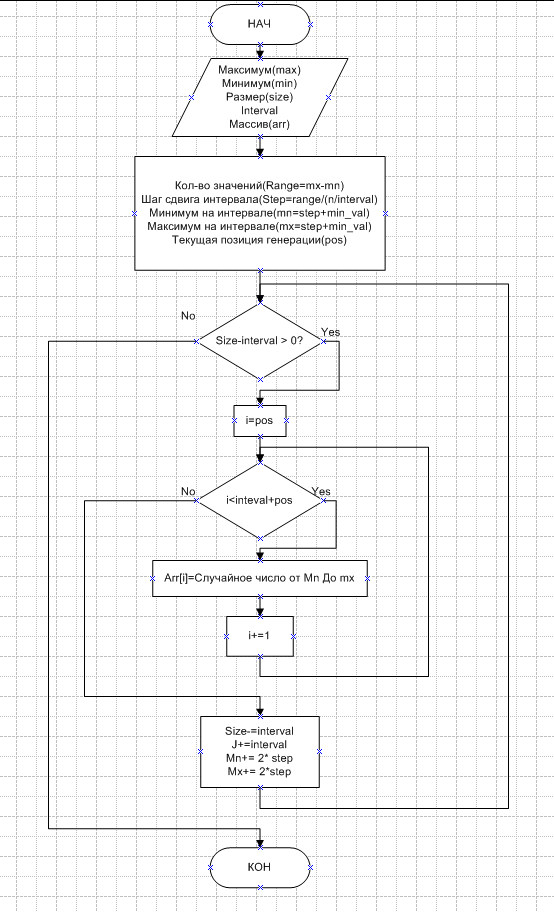
}

Листинг 6 – генерация ступенчатой последовательности

Генерация квази-последовательности представлена в приложении А

\*\*график\*\*

Пример реализации генерации частично упорядоченной последовательности представлен как блок схема ступенчатой генерации.



Блок схема 1 – генерация ступенчатой последовательности

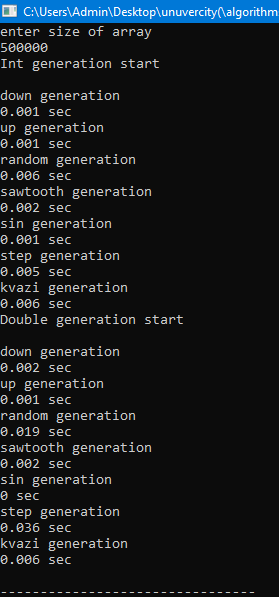


Рисунок 1 – Время генерации 5\*105 элементов

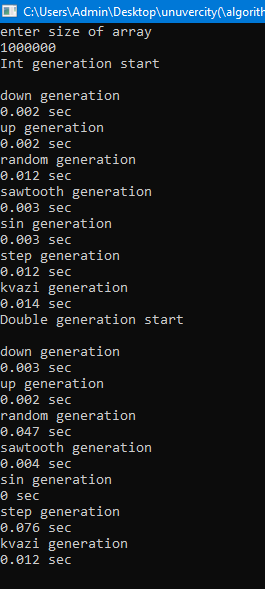


Рисунок 2 – Время генерации 10\*105 элементов

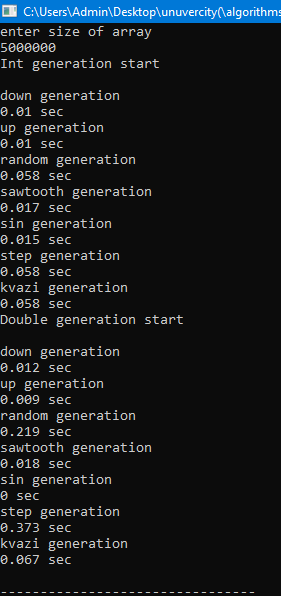


Рисунок 3 – Время генерации 50\*105 элементов

**Вывод**

Освоены принципы формирования монотонных (упорядоченных и упорядоченных в обратном порядке), частично упорядоченных и случайных последовательностей данных. Изучены функции, позволяющие производить оценку длительности времени выполнения алгоритмов.

**Приложения**

void int\_array::kvazi\_gen(int min\_val, int max\_val) {

int i=0, j=0, n=size, val = min\_val;

int step = (max\_val - min\_val) / (n / interval) / 2, mn=step+min\_val, mx = 2\*step+min\_val;

while (n - interval >= 0) {

for (i=0+j;i<interval+j;i++) {

arr[i] = val;

if (val+step <= mx)

val += (step/interval);

else

val = mx;

}

i--;

inversion\_gen(j);

n -= interval;

j += interval;

mn += 2 \* step;

mx += 2 \* step;

}

}

void int\_array::inversion\_gen(int start) {

int inversions = 0, swapped, temp;

int mx\_index, mn\_index;

inversions = rand() % (interval / 2 + 1);

switch (inversions) {

case 1:

mn\_index = 0;

mx\_index = interval-1;

swapped = rand() % (mx\_index+1) + mn\_index;

// СЃРІР°Рї РІР»РµРІРѕ РёР»Рё РІРїСЂР°РІРѕ

if ((rand() % 2 && swapped != mn\_index) || swapped == mx\_index) {

swap\_arr(arr, start+swapped, start+swapped - 1, 1);

}

else {

swap\_arr(arr, start+swapped, start+swapped + 1, 1);

}

break;

case 2:

mn\_index = 0;

mx\_index = (interval/2)-1;

swapped = rand() % (mx\_index+1) + mn\_index;

if ((rand() % 2 && swapped != mn\_index) || swapped == mx\_index)

swap\_arr(arr, start+swapped, start+swapped - 1, 1);

else

swap\_arr(arr, start+swapped, start+swapped + 1, 1);

mn\_index = interval/2;

mx\_index = interval/2;

swapped = rand() % (mx\_index) + mn\_index;

if ((rand() % 2 && swapped != mn\_index) || swapped == mn\_index+mx\_index-1)

swap\_arr(arr, start+swapped, start+swapped - 1, 1);

else

swap\_arr(arr, start+swapped, start+swapped + 1, 1);

break;

case 3:

mn\_index = 0;

mx\_index = interval/3;

swapped = rand() % mx\_index;

if ((rand() % 2 && swapped != mn\_index) || swapped == mx\_index - 1)

swap\_arr(arr, start+swapped, start+swapped - 1, 1);

else

swap\_arr(arr, start+swapped, start+swapped + 1, 1);

mn\_index = interval/3;

swapped = rand() % (mx\_index) + mn\_index;

if ((rand() % 2 && swapped != mn\_index) || swapped == mx\_index + mn\_index - 1)

swap\_arr(arr, start+swapped, start+swapped - 1, 1);

else

swap\_arr(arr, start+swapped, start+swapped + 1, 1);

mn\_index = interval/3 \* 2;

swapped = rand() % (mx\_index) + mn\_index;

if ((rand() % 2 && swapped != mn\_index) || swapped == mx\_index + mn\_index - 1)

swap\_arr(arr, start+swapped, start+swapped - 1, 1);

else

swap\_arr(arr, start+swapped, start+swapped + 1, 1);

break;

case 4:

temp = rand() % 3; // РІ РєР°РєРѕРј РјРµСЃС‚Рµ РІСЃС‚РІР°РёРј РЅР° 2 РїРѕР·РёС†РёРё

mn\_index = 0;

mx\_index = interval/3;

swapped = rand() % mx\_index;

if (temp != 0 && ((rand() % 2 && swapped != mn\_index) || swapped == mx\_index - 1))

swap\_arr(arr, start+swapped, start+swapped - 1, 1);

else if (temp == 0) {

// mx elem

swap\_arr(arr, mn\_index, mn\_index+mx\_index-1, 1);

swap\_arr(arr, mn\_index+1, mn\_index+mx\_index-1, 1);

}

else

swap\_arr(arr, start+swapped, start+swapped + 1, 1);

mn\_index = interval/3 \* 2;

swapped = rand() % (mx\_index) + mn\_index;

if (temp != 0 && ((rand() % 2 && swapped != mn\_index) || swapped == mx\_index - 1))

swap\_arr(arr, start+swapped, start+swapped - 1, 1);

else if (temp == 1) {

// mx elem

swap\_arr(arr, mn\_index, mn\_index+mx\_index-1, 1);

swap\_arr(arr, mn\_index+1, mn\_index+mx\_index-1, 1);

}

else

swap\_arr(arr, start+swapped, start+swapped + 1, 1);

mn\_index = 0;

mx\_index = interval/3;

swapped = rand() % mx\_index;

if (temp != 0 && ((rand() % 2 && swapped != mn\_index) || swapped == mx\_index - 1))

swap\_arr(arr, start+swapped, start+swapped - 1, 1);

else if (temp == 2) {

// mx elem

swap\_arr(arr, mn\_index, mn\_index+mx\_index-1, 1);

swap\_arr(arr, mn\_index+1, mn\_index+mx\_index-1, 1);

}

else

swap\_arr(arr, start+swapped, start+swapped + 1, 1);

break;

case 5:

temp = rand() % 3; // РІ РєР°РєРѕРј РјРµСЃС‚Рµ РЅРµ РІСЃС‚РІР°РёРј РЅР° 2 РїРѕР·РёС†РёРё

mn\_index = 0;

mx\_index = interval/3;

swapped = rand() % mx\_index;

if (temp != 0 && ((rand() % 2 && swapped != mn\_index) || swapped == mx\_index - 1))

swap\_arr(arr, start+swapped, start+swapped - 1, 1);

else if (temp != 0) {

// mx elem

swap\_arr(arr, mn\_index, mn\_index+mx\_index-1, 1);

swap\_arr(arr, mn\_index+1, mn\_index+mx\_index-1, 1);

}

else

swap\_arr(arr, start+swapped, start+swapped + 1, 1);

mn\_index = interval/3 \* 2;

swapped = rand() % (mx\_index) + mn\_index;

if (temp != 0 && ((rand() % 2 && swapped != mn\_index) || swapped == mx\_index - 1))

swap\_arr(arr, start+swapped, start+swapped - 1, 1);

else if (temp != 1) {

// mx elem

swap\_arr(arr, mn\_index, mn\_index+mx\_index-1, 1);

swap\_arr(arr, mn\_index+1, mn\_index+mx\_index-1, 1);

}

else

swap\_arr(arr, start+swapped, start+swapped + 1, 1);

mn\_index = 0;

mx\_index = interval/3;

swapped = rand() % mx\_index;

if (temp != 0 && ((rand() % 2 && swapped != mn\_index) || swapped == mx\_index - 1))

swap\_arr(arr, start+swapped, start+swapped - 1, 1);

else if (temp != 2) {

// mx elem

swap\_arr(arr, mn\_index, mn\_index+mx\_index-1, 1);

swap\_arr(arr, mn\_index+1, mn\_index+mx\_index-1, 1);

}

else

swap\_arr(arr, start+swapped, start+swapped + 1, 1);

break;

}

}

Приложение А – генерация квази-последовательности