Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное   
учреждение высшего образования

Национальный исследовательский Нижегородский государственный университет им. Н.И. Лобачевского

Институт информационных технологий, математики и механики

**Отчет по лабораторной работе**

**«Сортировки»**

**Выполнила**:

студентка группы 3822Б1-ПМ1

Гордеева Т.С.

**Проверил**:

преподаватель каф. МОСТ,

Волокитин В.Д.

Нижний Новгород

2022

**Содержание**

[Постановка задачи 3](#_Toc26962562)

[Метод решения 4](#_Toc26962563)

[Руководство пользователя 5](#_Toc26962564)

[Описание программной реализации 6](#_Toc26962565)

[Подтверждение корректности 7](#_Toc26962566)

[Результаты экспериментов 8](#_Toc26962567)

[Заключение 9](#_Toc26962568)

[Приложение 10](#_Toc26962569)

# Постановка задачи

Целью является изучение разных видов сортировок, работающих за O(n²), O(n) и O(n). Задача состоит в написании программы и пользовательского интерфейса на языке Си для сортировки массива вещественных чисел типа double различными видами алгоритмов:

* сортировкой bubble;
* сортировкой вставками;
* сортировкой Шелла;
* сортировкой слиняем.

Также необходимо посчитать время работы каждой сортировки, корректность их работы и сравнить скорость выполнения, а также проверить соответствие теоретической оценке выполнения.

# Метод решения

Алгоритмы сортировок:

1. сортировка bubble;
2. сортировка вставками;
3. сортировка Шелла;
4. сортировка слиняем.
5. Сортировка bubble.

Упорядоченный массив создается на том же месте, где находится исходная последовательность. Идея метода заключается в том, чтобы сравнивать попарно соседние элементы. Каждый проход начинается с начала последовательности; сравнивается первый элемент со вторым, и если порядок между ними нарушен, то элементы меняются местами. Затем сравниваются второй и третий элементы и так далее до предпоследнего элемента. После первого прохода максимальный элемент встанет на последнее место или первое место, если сортировка по убыванию), то есть он как будто «всплывет». Поэтому этот метод называется сортировка пузырьком (или bubble). Во втором проходе происходит то же самое, только не сравниваем с последним элементом, так как он уже стоит на месте. Другими словами, количество сравнений будет n-1-i, где n – кол-во элементов в массиве, i – номер порядка. Так мы идем до предпоследнего элемента, который сравниваем 1 раз (выше элементы все отсортированные), последний элемент стоит на своем месте, сортировка закончена. В сортировку можно добавить флаг, чтобы понимать, совершались при проходе перестановки, если нет – закончить сортировку, массив упорядочен.

Сложность сортировки: за каждый проход делается n сравнений, но рассматриваем мы i-ый элемент, значит, выполняется (n-1)\*(n-1) сравнений. Получается асимптотика O(n²).

1. Сортировка вставками.

Главная идея метода состоит в том, что при добавлении нового элемента в уже отсортированный массив его необходимо вставлять на нужное место. Массив разделяется на 2 части — отсортированную и неотсортированную. Из неотсортированной части берется элемент и ставится в отсортированную часть на место, не нарушая сортировку. Поиск подходящего места для элемента осуществляется сравнением с элементами, стоящими перед ним. Если элемент остается на своем месте, то сортировка завершена, либо они меняются местами, и сортировка продолжается. Элемент так встает на свое место, от чего отсортированная часть растет, а неотсортированная уменьшается.

Сложность сортировки: общее время работы алгоритма оценивается сверху как O(n2).

1. Сортировка Шелла.

Идея сортировки заключается в сравнении разделенных на группы элементов последовательности, находящихся друг от друга на определенном расстоянии. Сначала это расстояние равно d=n/2 (n—количество элементов в массиве). В первом проходе в каждой группе находится 2 элемента на расстоянии n/2, в которых они сравниваются и, при необходимости, меняются местами. На следующих проходах расстояние сокращается в 2 раза, то есть во втором проходе он будет d/2=n/4, в третьем n/8 и так до момента, когда d=1, проход буде последний. Количество групп, соответственно, будет уменьшаться.

Сложность сортировки: О(n\*log n), в худшем случае – О(n2).

1. Сортировка слиянием.

Алгоритм основывается на том, чтобы исходный массив делить пополам до тех пор, пока не останутся массивы из 1 элемента. Далее к каждой паре одноэлементных массивов применяется «слияние»: сравниваются первые два элемента каждых массивов, и наименьший элемент записывается во вспомогательный массив, а в том начальном подмассиве, откуда записали элемент, переходим к следующему. Когда один из массивов закончится, оставшиеся элементы дописываются во вспомогательный массив. «Слияние» применяется ко всем парам одноэлементных массивов, пока те не закончатся. Далее применяется к двухэлементным массивам, четырехэлементным и так до n-ой длины отсортированного массива. В конце отсортированный вспомогательный массив копируется в изначальный.

Сложность сортировки: во вспомогательный массив сливаются пары одноэлементных массивов, их n, затем двухэлементных массивов 2\*(n/2)=n, и так далее. Всего операций n+n+…+n= log2(n), так как массив каждый раз делится пополам. Среднее сложность O(n\*log(n)).

# Руководство пользователя

При запуске программа сначала просит ввести количество элементов в изначальном неотсортированном массиве. Затем программа просит ввести диапазон значений элементов. После ввода появляется меню с пунктами и соответствующими действиями (рис.1). Пользователь должен ввести число от 1 до 5 для выполнения соответствующего действия.

При некорректном вводе программа печатает «Неправильный ввод команды» и просит заново ввести номер команды (рис.2).

При корректном вводе программа выполнит заданную сортировку и выведет на экран отсортированный массив (при необходимости), время, за которое отсортировался массив и результат проверки массива на верность сортировки («массив отсортирован верно» или «ошибка сортировки») (рис.3). После этого можно выбрать другие виды сортировок, программа будет работать, пока не нажмут цифру 5. Под пятым пунктом находится команда «Конец», после срабатывания которой программа заканчивает свою работу.

Изображение выглядит как текст

Автоматически созданное описание

Рис.1

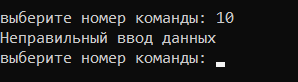


Рис.2

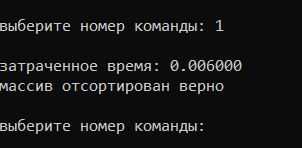


Рис.3

# Описание программной реализации

Программа состоит из одного файла (lab\_rab.cpp), в котором расположены все функции: вспомогательные, функции сортировок и главная (main).

**void swap(double\* a, double\* b)** – функция, меняющая указатели на элементы массива местами. Принимает указатель на один элемент, потом на второй.

**int compare(const void\* a, const void\* b)** – функция принимает на вход два указателя на сравниваемые числа и возвращает их взаимное отношение, больше(1)/равно(0)/меньше(-1). Эта функция является вспомогательной для *qsort()* из стандартной библиотеки Си.

**int check(double\* mas1, double\* mas, int na) –** функция проверки корректности работы сортировки (более подробно см. раздел «Подтверждение корректности»).

**void check\_res(double\* mas, double\* mas1, int na) –** функция выводит результат сортировки. Принимает два указателя на начала массивов (отсортированный и неотсортированный/начальный). С помощью check проверяет верно ли отсортирован массив. Если верно все, то выводит «массив отсортирован верно», иначе «Ошибка сортировки».

**void bubble(double \*mas, int na) –** функция сортировки bubble. Принимает указатель на начало исходного массива и его длину. Внешний цикл считает количество проходов по массиву, внутренний **–** проходит по массиву и меняет неупорядоченные элементы.

**int place(double\* mas, double r, int k) –** вспомогательная функция для сортировки «Вставками». Принимает указатель на начало исходного массива, элемент массива и количество упорядоченных элементов в массиве. Функция находит местоположение элемента среди упорядоченных элементов.

**void shift(double\* mas, int s, int k) –** вспомогательная функция для сортировки «Вставками». Принимает указатель на начало исходного массива, номер первого сдвигаемого элемента и количество упорядоченных элементов. Функция делает сдвиг элементов с индексами s,…,k-1 на 1 позицию вправо.

**void insert(double\* mas, int na) –** сортировка «Вставками». Принимает указатель на начало исходного массива и количество элементов в массиве. В функции 2 вложенных цикла: внешний фиксирует первый элемент в неотсортированной части массива, а внутренний цикл вставляет этот элемент на нужное место в отсортированной части.

**void shell(double\* mas, int na) –** функция сортировки Шелла. Принимает указатель на начало исходного массива и длину его. Основной массив делает шаг прохода по массиву, первый вложенный цикл делает проход по массиву, второй вложенный делает сравнение.

**void merge(double\* mas, int first, int last) –** вспомогательная функция для сортировки слиянием. Принимает указатель на начало массива, индекс на первый элемент и индекс на последний элемент. Функция сортирует «раздробленные» массивы.

**void mergesort1(double\* mas, int st, int end1) –** функция сортировки слиянием. Принимает указатель на начало массива, индекс на первый элемент и индекс на последний элемент. Функция «разделяет» массив на два и выполняет функцию merge.

**void mergesort(double\* mas, int na) –** вспомогательная функция для сортировки слиянием. Принимает указатель на начало массива и его длину. Функция выполняет сортировку слиянием и засекает время на ее работу.

# Подтверждение корректности

Для подтверждения корректности работы сортировок в программе используется функция **int check(double\* mas1, double\* mas, int na).** Функция принимает указатель на начало исходного массива, указатель на начало отсортированного массива и длину массива. В функции с помощью массива arr копируется изначальный массив, после сортируется он с помощью библиотечной сортировки qsort. Далее сравниваются элементы из массива arr соответственно с элементами из отсортированного массива разными сортировками. Если массив отсортирован верно и все элементы совпадают, функция возвращает значение 1, в противном случае – 0.

# Результаты экспериментов

По данным экспериментов (рис.4) видно, что медленнее всего работает сортировка bubble, быстрее нее, но также относительно медленно работает сортировка «Вставками». Далее идет сортировка Шелла, которая до n=750000 была приблизительно наравне с сортировкой слиянием. Но уже при больших значениях сортировка слиянием оказалась самой быстрой.

Рис. 4

Чтобы проверить соответствие теоретической оценке выполнения, разделим полученное время на теоретическое: для bubble и «Вставками» на *n2,* для Шелла и «слиянием» *n\*log2(n),* где n – количество элементов.

# Заключение

В результате лабораторной работы на языке Си была реализована сортировка для вещественных чисел типа double, подробно рассмотрены и разобраны сортировки: сортировка bubble, сортировка вставками, сортировка Шелла и сортировка слиянием, для работы с которыми был разработан и реализован пользовательский интерфейс. Также были сравнены такие значения, как скорость и эффективность работы этих сортировок. Наименее эффективная сортировка оказалась сортировка bubble, после нее идет сортировка «вставками». Самыми эффективными оказались сортировки Шелла и слиянием. Из них двоих более эффективно работает сортировка слиянием.

# Приложение

void swap(double\* a, double\* b)

{

double t = \*a;

\*a = \*b;

\*b = t;

}

//сортировка пузырьком

void bubble(double \*mas, int na) {

clock\_t start, end;

double ti = 0;

int flag = 0;

start = clock();

for (int i = 0; i < na - 1; i++)

{

flag = 0;

for (int j = 0; j < na - 1 - i; j++)

if (mas[j] > mas[j + 1]) {

swap(&mas[j], &mas[j + 1]);

flag = 1;

}

if (flag == 0) break;

}

end = clock();

ti = (end - start)/(double)(CLOCKS\_PER\_SEC);

/\*printf("Отсортированный массив: ");

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

printf("%lf ", mas[i]);\*/

printf("\nзатраченное время: %lf", ti);

}

//сортировка вставками

//определение местоположения элемента

int place(double\* mas, double r, int k)

{

int s = k;

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

if (mas[i] > r)

{

s = i;

break;

}

return s;

}

//сдвиг для места элемента

void shift(double\* mas, int s, int k)

{

for (int i = k; i > s; i--)

mas[i] = mas[i - 1];

}

void insert(double\* mas, int na)

{

int k, s;

double r;

clock\_t start, end;

double ti = 0;

start = clock();

for (k = 1; k < na; k++)

{

r = mas[k];

s = place(mas, r, k);

shift(mas, s, k);

mas[s] = r;

}

end = clock();

ti = (end - start) / (double)(CLOCKS\_PER\_SEC);

/\*printf("Отсортированный массив: ");

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

printf("%lf ", mas[i]);\*/

printf("\nзатраченное время: %lf", ti);

}

//соритровка Шелла

void shell(double\* mas, int na)

{

clock\_t start, end;

double t;

double ti = 0;

start = clock();

for (int i = na / 2; i > 0; i /= 2)//шаг между элементами

for (int j = i; j < na; j++)

for (int k = j - i; k >= 0 && mas[k] > mas[k + i]; k -= i)

swap(&mas[k], &mas[k - i]);

end = clock();

ti = (end - start) / (double)(CLOCKS\_PER\_SEC);

/\*printf("Отсортированный массив: ");

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

printf("%lf ", mas[i]);\*/

printf("\nзатраченное время: %lf", ti);

}

//сортировка слиянием

//обмен местами

void merge(double\* mas, int first, int last)

{

int l, r, m;

double\* a;

a = (double\*)malloc((last - first) \* sizeof(double));

m = (first + last) / 2;

l = first;

r = m + 1;

for (int i = first; i <= last; i++)

{

if ((l <= m) && ((r > last) || (mas[l] < mas[r])))

{

a[i - first] = mas[l];

l++;

}

else

{

a [i - first] = mas[r];

r++;

}

}

for (int j = first; j <= last; j++)

mas[j] = a [j + first];

}

void mergesort1(double\* mas, int st, int end1)

{

if (st < end1)

{

mergesort1(mas, st, (st + end1) / 2);

mergesort1(mas, (st + end1) / 2 + 1, end1);

merge(mas, st, end1);

}

}

void mergesort(double\* mas, int na)

{

clock\_t start, end;

double ti = 0;

start = clock();

mergesort1(mas, 0, na-1);

end = clock();

ti = (end - start) / (double)(CLOCKS\_PER\_SEC);

//printf("Отсортированный массив: ");

//for (int i = 0; i < na; i++)

// printf("%lf ", mas[i]);

printf("\nзатраченное время: %lf", ti);

}