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»**

**Выполнил**:

студентка группы 3822Б1-ПМ1

Сосновская У.А.

**Проверил**:

преподаватель каф. МОСТ,

Волокитин В.Д.

Нижний Новгород

2022

**Содержание**

[Постановка задачи 3](#_Toc26962562)

[Метод решения 4](#_Toc26962563)

[Руководство пользователя 6](#_Toc26962564)

[Описание программной реализации 7](#_Toc26962565)

[Подтверждение корректности 8](#_Toc26962566)

[Результаты экспериментов 9](#_Toc26962567)

[Заключение 11](#_Toc26962568)

[Приложение 12](#_Toc26962569)

# Постановка задачи

После окончания изучения работы и структуры сортировок на языке Си, нам требовалось написать программу, которая сортирует произвольный массив заданного размера с помощью одной из реализуемых сортировок. После этого мы должны были замерить время работы сортировки для массивов разных размеров.

**Цель работы**: создание программы с сортировками

**Задачи**:

* Создание программы с сортировками
* Подтверждение корректности работы сортировок
* Сравнение практических результатов с теоретическими
* Написание отчета

# Метод решения

Для данной лабораторной работы необходимо было реализовать четыре сортировки.

### Сортировка «пузырек» с остановкой

На вход функция получает неотсортированный массив и его размер, вводится переменная flag. Первый цикл for проходит по всему массиву, перед тем как зайти во второй цикл переменная flag обнуляется. Второй цикл for проходит от начала массива до конца его неотсортированной части (она находится в начале). Внутри происходит сравнение пары элементов, если элемент, стоящий раньше, оказывается больше того, что стоит позже, они меняются местами с помощью функции swap, в этом случает значение переменной flag становится равным единице, если элементы не менялись, значение flag также остается неизменным.

Если после выхода из внутреннего цикла переменная flag не поменяла своего значения, то мы можем сделать вывод, что массив полностью отсортирован, функция прекращает свою работу.

Если же значение переменной меняется, то значение переменной i внешнего цикла увеличивается, тем самым сдвигая границу неотсортированного массива влево, уменьшая его. За один шаг внешнего цикла на свое место гарантированно встает самый большой элемент в массиве. Это повторяется до тех пор, пока массив не будет отсортирован.

### Сортировка вставками

На вход функция получает неотсортированный массив и его размер. Внешний цикл идет по всему массиву, внутренний по части массива, которая будет отсортирована после завершения этого цикла. Один элемент мы считаем отсортированным, поэтому сортировка начинается с двух элементов. После каждого завершения внутреннего цикла, мы получаем отсортированную часть массива, при увеличении переменной i, к отсортированной части добавляется еще один элемент, который после завершения внутреннего цикла встает на свое место. Так после завершения внешнего цикла, мы получаем отсортированный массив.

### Сортировка Хоара (быстрая сортировка)

На вход функция получает неотсортированный массив, его левую и правую границу. Пока указатель на левую границу меньше или равен указателю на правую границу, функция работает. Объявляется переменная p, значение которой определяется результатом вызова функции partition. Эта функция должна выбрать ведущий элемент, разместить все числа, которые больше или равны ведущему, справа от него, а числа, которые меньше, слева и вернуть порядковый номер ведущего элемента.

На вход функция partition получает массив, его левую и правую границу. Объявляется переменная tmp, значение которой равно порядковому номеру среднего между правой и левой границей элемента. Также объявляется переменная mid, равная его значению. Внешний цикл while работает, пока указатель на левую границу массива меньше указателя на правую. Первый внутренний цикл while сдвигает левую границу вправо, если значение элемента, являющегося граничным, меньше значения среднего элемента, и указатели на левую и правую границу не равны. Второй внутренний цикл while сдвигает правую границу влево, если значение элемента, являющегося граничным, больше значения среднего элемента, и указатели на левую и правую границу не равны.

Таким образом, после завершения внутренних циклов мы получаем элемент, который больше среднего элемента, но находится слева от него, и элемент, который меньше среднего, но расположен справа от него, и с помощью функции swap меняем их местами.

Если слева появляется элемент, равный значению среднего, но отличный от него по положению, то с помощью функции swap он меняется местами с элементом, который меньше среднего, но расположен справа от него. После этого левый указатель сдвигается на один элемент вправо.

Если справа появляется элемент, равный значению среднего, но отличный от него по положению, то с помощью функции swap он меняется местами с элементом, который больше среднего, но расположен слева от него. После этого правый указатель сдвигается на один элемент влево. Так как эта проверка стоит раньше, чем проверка на равный элемент слева, то, после смены элементов, левый указатель будет стоять на равном элементе и вернет его вправо, не нарушив условия.

После выхода из цикла, функция partition возвращает расположение среднего элемента, и Quicksort вызывает себя же, но с измененными данными о левой и правой границе. Сначала функция вызывается с левой границы до ведущего элемента. Это происходит до тех пор, пока не останется один элемент, который по умолчанию является отсортированным. После этого вызов происходит с правой границы до конечного элемента.

После завершения всех вызовов массив будет отсортирован.

### Сортировка «расческой»

Эта сортировка является более усовершенствованной версией сортировки «пузырьком».

На вход функция получает неотсортированный массив и его размер. Объявляется переменная factor, равная 1,246. Это фактор уменьшения шага, он равен , где e это основание натурального алгоритма, а φ – золотое сечение. Также объявляется переменная step, равная размеру массива, уменьшенному на единицу.

Внешний цикл while работает, пока переменная step не равна 0. Внутренний цикл работает, пока сумма i-ого элемента и step меньше размера массива. Если i-ый элемент больше i+step элемента, то они с помощью функции swap меняются местами. После того, как внутренний цикл завершил свою работу, step уменьшается.

Таким образом происходит попарная сортировка элементов массива, которые находятся на расстоянии step друг от друга, и к моменту, когда step будет равен 1 останутся лишь пару соседних элементов, которые нужно поменять местами.

# Руководство пользователя

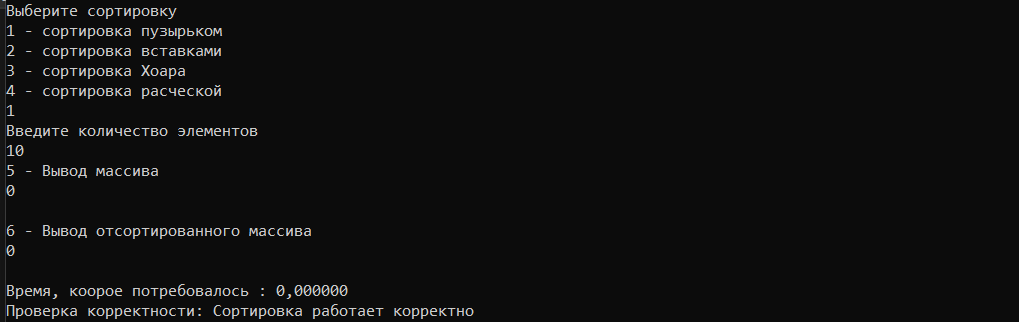
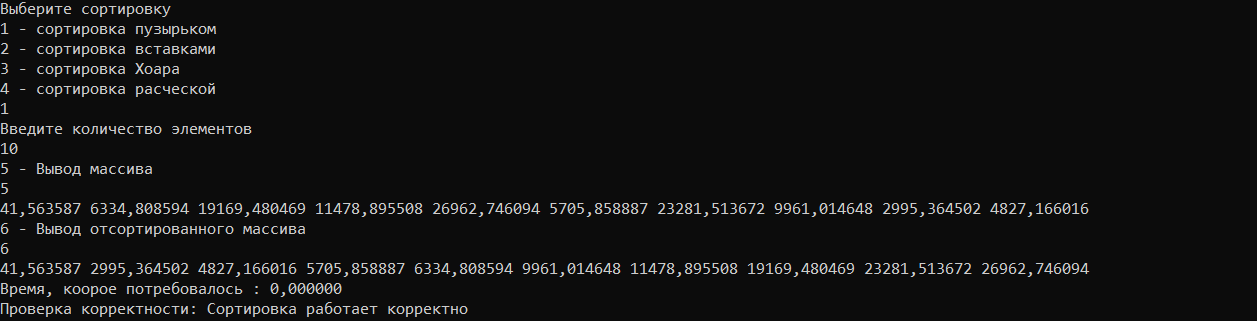
 После запуска программы перед пользователем появляется окно, куда он должен ввести необходимые данные (какую именно сортировку он хочет использовать, размер массива, который должен быть отсортирован). Затем пользователю будет предложено вывести созданный и отсортированный массивы на экран, при вводе нужного числа с клавиатуры, массивы будут выведены, при вводе чего-либо другого, массивы выведены не будут. В конце будет выведено время, затраченное на сортировку данного массива, и проверка корректности работы сортировки.

Рис. 2

Рис. 1

# Описание программной реализации

Программа состоит из одного файла (ConsoleApplication1.cpp), где были реализованы все необходимые сортировки, проверка их корректности, подсчет времени их работы.

int compare(const void\* a, const void\* b) – функция, которая используется для реализации встроенной сортировки qsort. Принимает указатели, после этого определяет тип указателей, с которыми мы работаем и сравнивает их. Если первый элемент больше второго, то возвращается 1, если меньше, то -0, если они раны, возвращается 0.

int check(float\* mas\_a, float\* mas\_b, int size) – функция, которая используется для подтверждения работы программы. Она принимает на вход массив равный исходному неотсортированному массиву, отсортированный массив и их размер. После этого происходит встроенной функции qsort, которая принимает неотсортированный массив, его размер, размер элемента в байтах, и результат вызова функции compare. После этого поэлементно сравниваем разницу между элементами первого и второго массива, если она больше погрешности, то массив работает некорректно и функция возвращает 0, если меньше, то возвращает 1.

void\* malloc(size\_t size); - функция, необходимая для создания массива. Выделяет память на size байт.

float\* creature(int size) – функция создания массива с произвольным набором заданного количества чисел. Принимает на вход размер массива, выделяет на него память, заполняет произвольными элементами. Возвращает готовый массив.

void swap(float\* a, float\* b) – функция, которая меняет местами два элемента массива. Принимает на вход указатели на элементы, которые необходимо поменять местами.

void bubble(float\* array, int size) – сортировка «пузырьком». Принимает на вход массив и его размер, возвращает отсортированный массив.

void Insert(float\* array, int size) - сортировка выбором. Принимает на вход массив и его размер, возвращает отсортированный массив.

int partition(float\* mas, int l, int r) – функция необходимая для реализации сортировки Хоара. Принимает на вход массив, левую правую границу массива или его части, с которой мы работаем. Распределяет элементы относительно ведущего, возвращает номер ведущего элемента.

void Quicksort(float\* array, int left, int right) – сортировка Хоара. Принимает массив его левую и правую границу. Рекурсивно делит массив на части, сортирует каждую из них и возвращает отсортированный массив.

void comb(float\* arr, int size) – сортировка «расческой». Принимает массив и его размер, возвращает отсортированный массив.

int main() – основная функция. В ней подсчитывается время работы сортировок, через нее вызывается функция с необходимой сортировкой. Создается массив, необходимый для проверки работы корректности сортировки. Выводит на экран результат работы сортировок, вывод о корректности их работы.

# Подтверждение корректности

Для подтверждения корректности в программе используется функция check. Она принимает на вход массив равный исходному неотсортированному массиву, отсортированный массив и их размер. После этого происходит встроенной функции qsort, которая принимает неотсортированный массив, его размер, размер элемента в байтах, и результат вызова функции compare. После этого поэлементно сравниваем разницу между элементами первого и второго массива, если она больше погрешности, то массив работает некорректно и функция возвращает 0, если меньше, то возвращает 1.

Функция compare принимает указатели, после этого определяет тип указателей, с которыми мы работаем и сравнивает их. Если первый элемент больше второго, то возвращается 1, если меньше, то -0, если они раны, возвращается 0.

Я провела проверку корректности, используя массивы с 100000 элементов. Тестирование подтвердило корректность написанных сортировок.

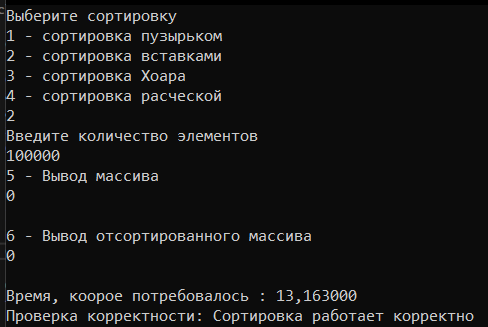
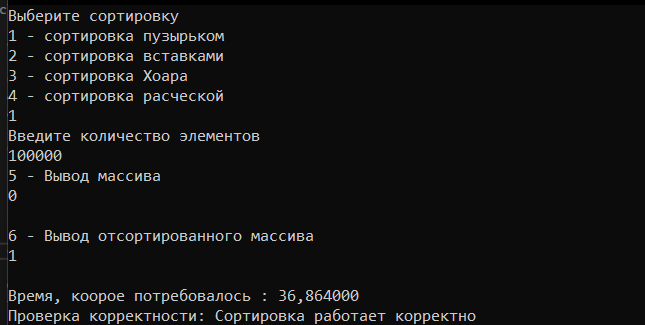


Рис. 4

Рис. 3

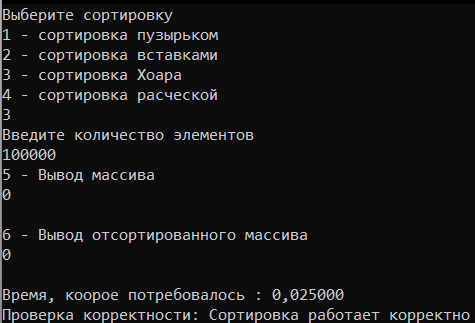
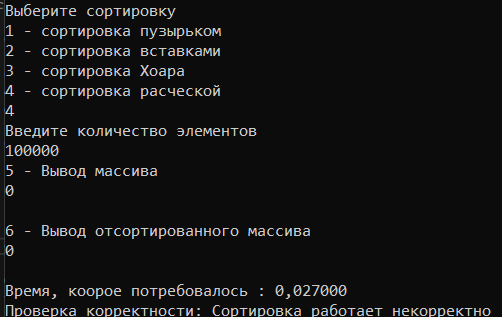


Рис. 6

Рис. 5

# Результаты экспериментов

Целью проведения эксперимента было выявление зависимости времени работы сортировок от количества элементов в сортируемом массиве. Изучив результаты экспериментов, мы можем заметить, что график времени работы сортировки «пузырек» и сортировки «вставками» похож на график параболы, который является оценкой сложности функции этих сортировок в среднем (средним назовем случай, когда элементы массива расположены в произвольном порядке). Графики быстрой сортировки с сортировки «расческой» похожи на графики функции n\*log(n), которая также является оценкой сложности этих алгоритмов в среднем.

### Сортировка «Пузырек» с остановкой

|  |  |
| --- | --- |
| Количество  элементов | Время, с |
| 10 | 0 |
| 100 | 0 |
| 1000 | 0,001 |
| 5000 | 0,062 |
| 10000 | 0,276 |
| 50000 | 8,076 |
| 100000 | 35,307 |
| 200000 | 166,82 |
| 300000 | 373,946 |
| 400000 | 567,724 |
| 500000 | 1026,844 |
| 600000 | 1407,901 |
| 700000 | 1780,398 |

### Сортировка «Вставками»

|  |  |
| --- | --- |
| Количество элементов | Время, с |
| 10 | 0 |
| 100 | 0 |
| 1000 | 0,002 |
| 5000 | 0,029 |
| 10000 | 0,117 |
| 50000 | 2,936 |
| 100000 | 13,826 |
| 200000 | 60,749 |
| 300000 | 127,198 |
| 400000 | 209,573 |
| 500000 | 290,539 |
| 600000 | 442,109 |
| 700000 | 588,747 |

### Сортировка Хоара (быстрая сортировка)

|  |  |
| --- | --- |
| Количество элементов | Время, с |
| 10 | 0 |
| 100 | 0 |
| 1000 | 0 |
| 5000 | 0,001 |
| 10000 | 0,002 |
| 50000 | 0,008 |
| 100000 | 0,019 |
| 200000 | 0,048 |
| 300000 | 0,067 |
| 400000 | 0,081 |
| 500000 | 0,111 |
| 600000 | 0,122 |
| 700000 | 0,133 |
| 800000 | 0,168 |
| 900000 | 0,185 |
| 1000000 | 0,202 |

### Сортировка «расческой»

|  |  |
| --- | --- |
| Количество элементов | Время, с |
| 10 | 0 |
| 100 | 0 |
| 1000 | 0,001 |
| 5000 | 0,001 |
| 10000 | 0,002 |
| 50000 | 0,01 |
| 100000 | 0,026 |
| 200000 | 0,053 |
| 300000 | 0,091 |
| 400000 | 0,123 |
| 500000 | 0,161 |
| 600000 | 0,189 |
| 700000 | 0,227 |
| 800000 | 0,228 |
| 900000 | 0,268 |
| 1000000 | 0,27 |

# Заключение

Написав программу, содержащую сортировки, проверив корректность их работы, мы смогли использовать результат их работы для проведения экспериментов. Они показали нам, что зависимость времени работы сортировок от количества элементов в массиве растет так же, как сложность сортировки.

# Приложение

#include "stdio.h"

#include "stdlib.h"

#include "time.h"

#include "locale.h"

int compare(const void\* a, const void\* b)

{

const float\* arg1, \* arg2;

arg1 = (const float\*)a;

arg2 = (const float\*)b;

if ((\*arg1 - \*arg2) < 0) return -1;

else

{

if ((\*arg1 - \*arg2) > 0) return 1;

else return 0;

}

}

int check(float\* mas\_a, float\* mas\_b, int size)

{

qsort(mas\_a, size, sizeof(float), compare);

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

if ((mas\_a[i] - mas\_b[i]) > 1e-8 || (mas\_a[i] - mas\_b[i]) < -(1e-8))

return 0;

return 1;

}

void\* malloc(size\_t size);

float\* creature(int size)

{

float\* mas;

mas = (float\*)malloc(sizeof(float\*) \* size);

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

mas[i] =rand() + ((float)rand() / ((float)RAND\_MAX));

return mas;

}

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

{

float tmp = \*a;

\*a = \*b;

\*b = tmp;

}

void bubble(float\* array, int size)

{

int flag = 0;

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

{

flag = 0;

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

{

if (array[j] > array[j + 1])

{

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

flag = 1;

}

}

if (flag == 0) return;

}

}

void Insert(float\* array, int size)

{

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

for (int j = i; j > 0; j--)

if (array[j] < array[j - 1])

swap(&array[j], &array[j - 1]);

else break;

}

int partition(float\* mas, int l, int r)

{

int tmp = l + (r - l) / 2;

float mid = mas[tmp];

while (l < r)

{

while ((mas[l] < mid) && (l != r))

l++;

while ((mas[r] > mid) && (r != l))

r--;

if (mas[r] == mid && r != tmp && r != l)

{

swap(&mas[r], &mas[l]);

r--;

}

else if (mas[l] == mid && l != tmp && r != l)

{

swap(&mas[r], &mas[l]);

l++;

}

else swap(&mas[r], &mas[l]);

}

return r;

}

void comb(float\* arr, int size)

{

float factor = 1.246;

int step = size - 1;

while (step != 0)

{

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

if (arr[i] > arr[i + step])

swap(&arr[i], &arr[i + step]);

step /= factor;

}

}

void Quicksort(float\* array, int left, int right)

{

if (left <= right)

{

int p = partition(array, left, right);

Quicksort(array, left, p-1);

Quicksort(array, p+1, right);

}

}

int main()

{

clock\_t start, finish;

int n, vibor, itog, nachalo;

float\* arr;

float\* arr1;

setlocale(LC\_ALL, "Rus");

printf("Выберите сортировку\n");

printf("1 - сортировка пузырьком\n");

printf("2 - сортировка вставками\n");

printf("3 - сортировка Хоара\n");

printf("4 - сортировка расческой\n");

scanf\_s("%i", &vibor);

printf("Введите количество элементов\n");

scanf\_s("%i", &n);

arr = creature(n);

arr1 = (float\*)malloc(sizeof(float\*) \* n);

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

arr1[i] = arr[i];

printf("5 - Вывод массива \n");

scanf\_s("%i", &nachalo);

if (nachalo == 5)

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

printf("%f ", arr[i]);

printf("\n");

if (vibor == 1)

{

start = clock();

bubble(arr, n);

finish = clock();

}

else if (vibor == 2)

{

start = clock();

Insert(arr, n);

finish = clock();

}

else if (vibor == 3)

{

start = clock();

Quicksort(arr, 0, n-1);

finish = clock();

}

else if (vibor == 4)

{

start = clock();

comb(arr, n);

finish = clock();

}

printf("6 - Вывод отсортированного массива \n");

scanf\_s("%i", &itog);

if (itog == 6)

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

printf("%f ", arr[i]);

printf("\n");

printf("Время, коорое потребовалось : ");

printf("%f\n", (finish - start) / (double)(CLOCKS\_PER\_SEC));

printf("Проверка корректности: ");

if (check(arr1, arr, n) == 1)

printf("Сортировка работает корректно");

else

printf("Сортировка работает некорректно");

return 0;

}