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 3822Б1ПМ1

Сборнова А.И.

**Проверил**:

преподаватель каф. МОСТ,

Волокитин В.Д.

Нижний Новгород

2022

**Содержание**

[Постановка задачи 3](#_heading=h.gjdgxs)

[Метод решения 4](#_heading=h.30j0zll)

[Руководство пользователя](#_heading=h.1fob9te) 6

[Описание программной реализации](#_heading=h.3znysh7) 7

[Подтверждение корректности](#_heading=h.2et92p0) 8

[Результаты экспериментов](#_heading=h.tyjcwt) 9

[Заключение](#_heading=h.3dy6vkm) 10

[Приложение 1](#_heading=h.1t3h5sf)1

# Постановка задачи

В данной работе необходимо изучить и реализовать сортировки пузырьком, вставками, слиянием и сортировку Хоара. А также сравнить скорость их выполнения и на основе этого сделать выводы по поводу их эффективности.

# Метод решения

1. Сортировка пузырьком

Алгоритм сортировки пузырьком основывается на сравнении соседних элементов, и в случае если больший элемент стоит раньше меньшего, эти элементы меняются местами. Таким образом получается, что мы постепенно “выталкиваем” сначала самый большой элемент в самый конец, потом самый большой из всех кроме него и т. д. Для написания программы используется цикл в котором мы ходим по массиву до тех пор, пока при прохождении цикла производится хотя бы одна перестановка.

1. Сортировка вставками

Алгоритм сортировки вставками основывается на поочередном выборе элементов из массива и размещении их на своем месте в уже отсортированной части массива. Изначально считаем первый элемент, уже отсортированным массивом, берем второй элемент и ищем его место, перед или после первого элемента, после берем третий элемент ищем его место в отсортированном массиве из двух элементов и т. д. Для поиска места элемента в отсортированном массиве используется бинарный поиск, алгоритм которого заключается в разделении массива пополам выборе половины в которой лежит искомый элемент, и разделении этой половины ещё на две равные части и повторяем это снова, и делаем это до тех пор пока не останется один элемент, после сравниваем его с нашим элементом и определяем перед ним он стоит или после.

1. Сортировка Хоара

Алгоритм сортировки Хоара основывается на разделении массива на две части относительно выбранного элемента, в первую часть попадают элементы меньше выбранного, а во вторую больше выбранного. После каждая часть снова разделяется на две таким же образом, и это повторяется до тех пор пока в части не останется только один элемент. При написании программы мы выбираем первый элемент, а после для разделения на две части берем два индекса с начала и с конца и первым индексом отмечаем первый с начала элемент, который больше выбранного, а вторым первый с конца элемент который меньше выбранного. после меняем их местами и продолжаем отмечать такие элементы до тех пор пока индексы не станут равны или поменяются местами, таким образом мы еще и определили местоположение выбранного элемента.

1. Сортировка слиянием

Алгоритм сортировки слиянием основывается на разделении массива пополам, потом каждой из половин еще пополам и так до тех пор пока части массива будут состоять не более чем из двух элементов. после этого упорядочиваем каждую часть и объединяем эти части до частей не более чем из 4 элементов, и т. д. Но при обратном объединении нам так же необходимо отсортировать массив, для этого мы выделяем дополнительную временную память в которую мы будем ненадолго помещать отсортированный массив, и для начала сравниваем первые элементы из обеих объединяемых частей, меньшую помещаем в выделенную память и берем следующий элемент взамен того, что уже поместили память и снова сравниваем. И так до тех пор пока одна из частей не закончится, после этого остаток второй части тоже помещаем во временную память, а после помещаем элементы из временной памяти на место двух объединенных частей.

# Руководство пользователя

При запуске программы выводится меню программы, с помощью которого пользователь взаимодействует с программой. В меню задействованы цифры от 1 до 8, каждая из которых отвечает за свое действие.

Для выбора одного из пунктов меню необходимо ввести соответствующую цифру и нажать enter.

1. Ввести размер массива.

Используется для изменения длины массива (Изначально длина равна 1000). После выбора этого пункта меню необходимо ввести целое число от 1 до 50000, а после нажать enter.

1. Ввести диапазон для рандомных чисел.

Используется для изменения диапазона рандомных чисел для создания массива ( Изначально равен от 0 до 1000) После выбора этого пункта меню необходимо ввести два числа с плавающей запятой через пробел.

1. Заполнить массив.

Используется для обновления массива после внесений в пункты 1-2, или если просто возникла необходимость перезаполнить массив. После выбора этого пункта в меню никаких действий выполнять не надо.

1. Выполнить сортировку пузырьком.
2. Выполнить сортировку вставками.
3. Выполнить сортировку Хоара.
4. Выполнить сортировку слиянием.

Эти пункты меню используются для выполнения одной из сортировки. После выбора этого пункта меню не нужно выполнять никаких действий, а также вы узнаете время за которое выполняется сортировка.

1. Завершить программу.

Этот пункт меню завершает программу, после его выбора программа самостоятельно закроется, не нужно выполнять никаких действий.

# Описание программной реализации

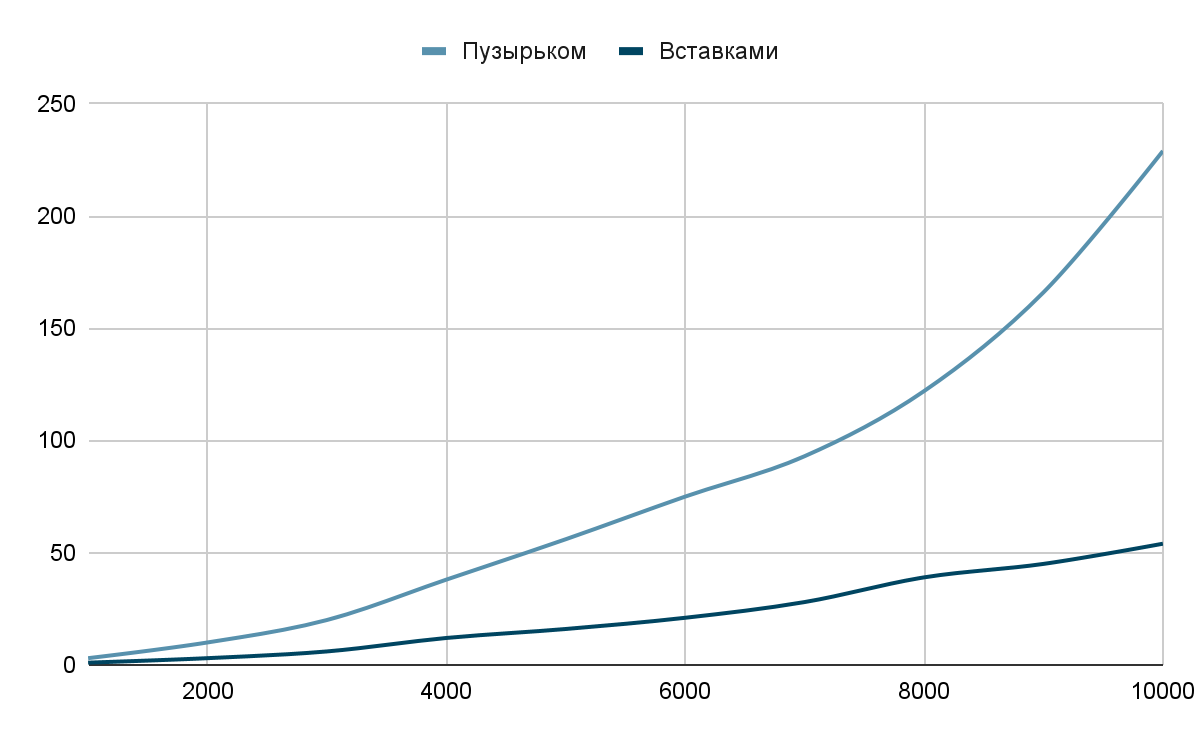
Один файл под названием Сортировки.cpp. В нем содержится реализация всех сортировок, а также реализация меню для общения с пользователем.

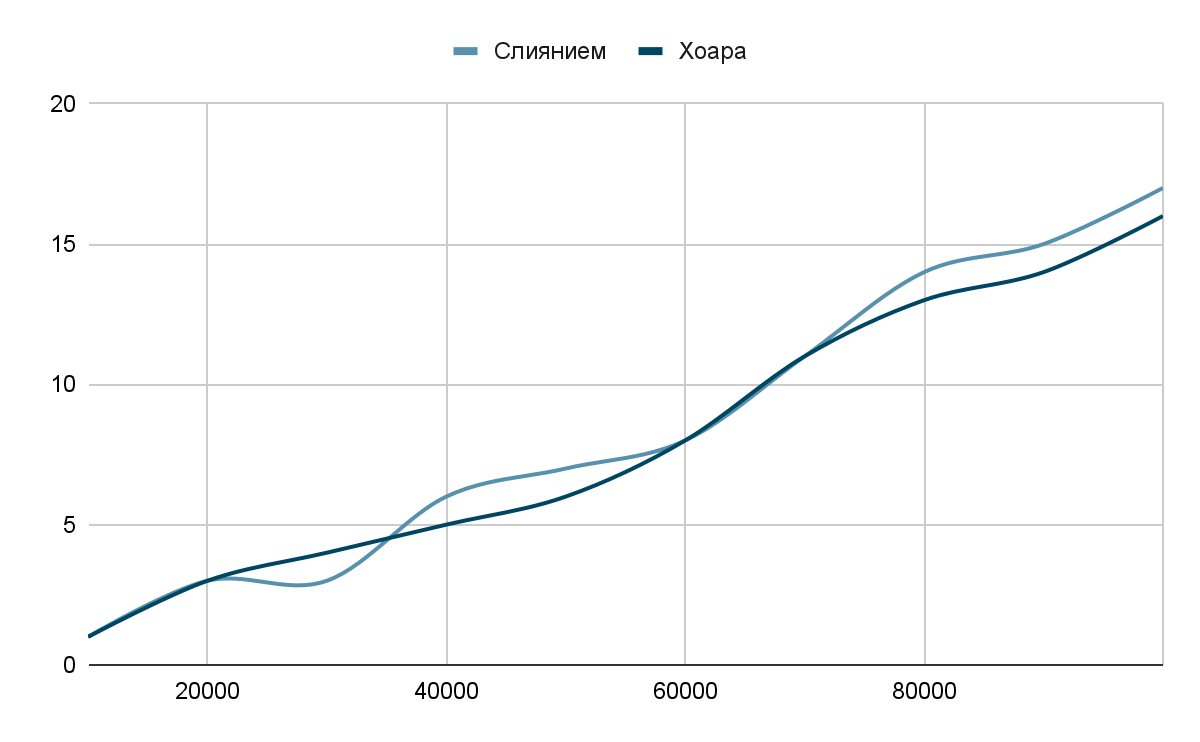
# Подтверждение корректности

Для подтверждения корректности в программе мы сравниваем результаты написанной нами сортировки со стандартной сортировкой языка С, qsort().

# Результаты экспериментов

По данным экспериментов видно, что сортировка Хоара и сортировка слиянием на порядок быстрее чем сортировка пузырьком и вставками. При этом сортировка пузырьком значительно медленней сортировки вставками.





# Заключение

В ходе данной работы я изучила и написала сортировки пузырьком, вставками, слиянием и сортировку Хоара. Также я опытным путем проверила скорость их работы и выяснила какие сортировки наиболее быстрые, а какие наиболее медленные.

# Приложение

Сортировка пузырьком

void bable(double b[], int n) {

double k;

int flag = 1;

for (; flag > 0;) {

flag = 0;

n--;

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

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

k = b[j];

b[j] = b[j + 1];

b[j + 1] = k;

flag++;

}

}

}

}

Сортировка вставками

int biplace(double b[], int r, int k) { // Вспомогательная функция для сортировки вставками

int st = 0; // бинарный поиск места для эл-та r

int fn = k - 1;

int mid;

int s;

while (st < fn) {

mid = (st + fn) / 2;

if (r > b[mid])

st = mid + 1;

else

fn = mid - 1;

}

if (r > b[st])

s = st + 1;

else

s = st;

return s;

}

void shift(double b[], int s, int k) { // Вспомогателная функция для сортировки вставками

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

b[i] = b[i - 1];

}

}

void insert(double b[], int n) {

int s, k;

double r;

for (k = 1; k < n; k++) {

r = b[k];

s = biplace(b, r, k);

shift(b, s, k);

b[s] = r;

}

}

Сортировка Хоара

void hoar(double b[], int first, int last)

{

double pivot;

int p;

int l\_hold = first;

int r\_hold = last;

pivot = b[first];

while (first < last)

{

while ((b[last] >= pivot) && (first < last))

last--;

if (first != last)

{

b[first] = b[last];

first++;

}

while ((b[first] <= pivot) && (first < last))

first++;

if (first != last)

{

b[last] = b[first];

last--;

}

}

b[first] = pivot;

p = first;

first = l\_hold;

last = r\_hold;

if (first < p)

hoar(b, first, p - 1);

if (last > p)

hoar(b, p + 1, last);

}

Сортировка слиянием

void merge(double b[], int n)

{

int mid = n / 2;

if (n % 2 == 1)

mid++;

int h = 1;

double\* c = (double\*)malloc(n \* sizeof(double));

int step;

while (h < n)

{

step = h;

int i = 0;

int j = mid;

int k = 0;

while (step <= mid)

{

while ((i < step) && (j < n) && (j < (mid + step)))

{

if (b[i] < b[j])

{

c[k] = b[i];

i++; k++;

}

else {

c[k] = b[j];

j++; k++;

}

}

while (i < step)

{

c[k] = b[i];

i++; k++;

}

while ((j < (mid + step)) && (j < n))

{

c[k] = b[j];

j++; k++;

}

step = step + h;

}

h = h \* 2;

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

b[i] = c[i];

}

}