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

Сунцова М.В.

**Проверил**:

преподаватель каф. МОСТ,

Волокитин В.Д.

Нижний Новгород

2022

**Содержание**

[Постановка задачи 3](#_Toc123755243)

[Метод решения 4](#_Toc123755244)

[Руководство пользователя 5](#_Toc123755245)

[Описание программной реализации 6](#_Toc123755246)

[Подтверждение корректности 8](#_Toc123755247)

[Результаты экспериментов 9](#_Toc123755248)

[Заключение 11](#_Toc123755249)

[Приложение 12](#_Toc123755250)

# Постановка задачи

Требуется создать программу на языке Си и реализовать в ней сортировки: вставками, выбором, расческой, Хоара для массивов с числами типа double на языке Си. Необходимо проверить корректность работы сортировок, сравнить их скорость, подтвердить их теоретическую сложность.

# Метод решения

Сортировка вставками.

Изначально первый элемент в массиве считается отсортированной частью массива, затем для каждого следующего элемента при помощи бинарного поиска ищется позиция в отсортированной части, элементы начиная с этого места до того, где стоит этот элемент сдвигаются вправо на 1, затем на свою позицию встает этот элемент. Алгоритм заканчивает работу после того, как вставлен последний неотсортированный элемент массива.

Сложность алгоритма О(, где n- количество элементов.

Сортировка выбором.

Выбирается элемент и среди последующих ищется наименьший элемент меньше выбранного, если такого элемента нет, если такой элемент нашелся, то они меняются местами, и ищется наименьший элемент для следующего элемента. Алгоритм заканчивает работу, когда предпоследний элемент стоит на месте.

Сложность алгоритма О(

Сортировка расческой.

В ходе сортировки сравниваются и меняются местами элементы на различном расстоянии. Изначально расстояние равно n-1 (n-количество элементов в массиве) и сравниваются крайние элементы массива, затем расстояние делится на фактор равный 1,246 ( где ) и следовательно уменьшается, сравниваются все пары элементов на этом расстоянии, потом все это повторяется, последними будут сравниваться элементы с расстоянием 1, то есть соседние, после их сравнения и перестановок алгоритм заканчивает работу.

Сложность алгоритма O()

Сортировка Хоара.

Выбирается опорный элемент в массиве и нужно чтобы все элементы меньше опорного стояли справа от него, больше слева. l идет слева направо, пока элементы не превышают опорного по значению, индекс r идет справа налево пока элемент не меньше опорного по значению, иначе элементы l и r меняются местами. Опорный элемент стоит на месте и известен его индекс, рекурсивно таким же образом упорядочиваются элементы справа и слева от опорного.

Сложность алгоритма O()

# Руководство пользователя

При запуске программы пользователю предлагается ввести количество элементов в массиве.

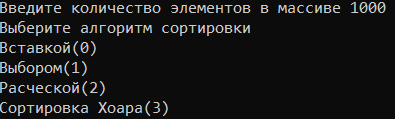
Затем появляется возможность выбрать тип сортировки, вводя числа от 0 до 3, при введении других значений появляется сообщение об ошибке.

Рисунок 1, выбор сортировок

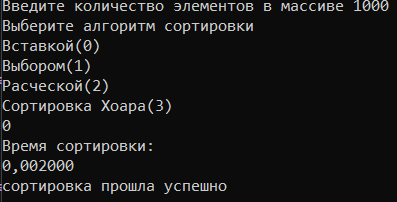
После выполнения сортировки выводится время её выполнения, результат проверки на корректность.

Рисунок 2. вывод времени сортировок, результата проверки на корректность

# Описание программной реализации

Проект содержит один файл sorts.cpp.

**void fill(double\* a, int n)** принимает указатель на массив, заполняет массив с помощью функции rand() псевдослучайными числами в диапазоне от -100 до 100.

**int comp(const void\* a, const void\* b)**  вспомогательная функция для qsort. Принимает указатели на сравниваемые числа, в зависимости от результатов сравнения выводит -1, 0 или 1.

**void copy(double\* a, double\* b, int n)** принимает указатели на 2 массива, количество элементов, копирует массив а в массив b. Копирует основной массив в проверочный.

**void check(double\* a, double\* b, int n)** функция для проверки корректности, принимает указатели на 2 массива и количество элементов.

**int bplace(double\* a, double r, int k)** бинарный поиск принимает указатель на массив, элемент, количество отсортированных элементов. Возвращает индекс элемента r в отсортированном массиве. Функция входит в сортировку вставками.

**void shift(double\* a, int s, int k)** функция сдвига. Принимает указатель на массив, индекс r в отсортированной части, количество элементов. Сдвигает элементы от индекса, который должен иметь r, до индекса на который имеет r который равен количеству элементов в отсортированной части.

**void insert(double\* a, int n)** основная функция сортировки вставками. Принимает указатель на массив и количество элементов. В цикле для каждого элемента с 1 до n-ного индекса вызывает bplace и shift, ставит элемент на нужное место в отсортированной части массива.

**void select(double\* a, int n)**  функция сортировки выбором. Принимает указатель на массив и количество элементов. В цикле по i от 0 до n-1 элемента выбирает элемент и в цикле по j от i+1 до n-ного индекса ищет элемент по значению наименьший при выполнении этого условия элементы меняются местами между собой.

**void comb(double\* a, int n)** функция сортировки расческой. Принимает указатель на массив, количество элементов, сортирует массив.

**int partition(double\* a, int l, int r)** функция для сортировки Хоара. Выбирает опорный элемент как стоящий в середине массива(или его части),относительно его располагает остальные элементы, возвращает индекс опорного элемента.

**void quicksort(double\* a, int l, int r)** сортировка Хоара. Принимает указатель на массив, индекс крайнего левого элемента и крайнего правого элемента. Вызывает partition для определения индекса опорного элемента и упорядочивания массива относительно его. Затем 2 раза вызывает себя же, для элементов слева от опорного и справа от опорного.

В функции **int main()** создаются динамические массивы(основной и проверочный), выводится на экран меню, в зависимости от выбора пользователя вызывается сортировка, засекается и выводится на экран время ее работы.

# Подтверждение корректности

Для подтверждения корректности в программе используется функция **void check(double\* a, double\* b, int n),** которая принимает указатели на отсортированный массив и его неотсортированную копию, созданную про помощи функции copy, количество элементов, копия сортируется при помощи qsort из библиотеки stdlib.h. Создается и зануляется счетчик. Затем числа в массивах поэлементно сравниваются, если они не равны, то в счетчике прибавляется 1. Если счетчик принимает значение 1 или больше, то выводится сообщение о некорректности сортировки (рис. 3), иначе на экран выводится сообщение об успешной сортировке.

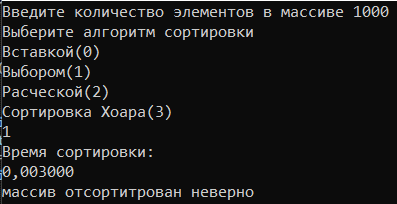


Рисунок 3. Сообщение о некорректной сортировке

# Результаты экспериментов

По результатам экспериментов составлена таблица, где представлено время сортировки в секундах.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Сортировка/  Кол-во элементов | Вставка,  сек | Выбор,  сек | Расческа,  сек | Сортировка Хоара,  сек |
| 1000 | 0,001 | 0,001 | 0,000 | 0,000 |
| 5000 | 0,0016 | 0,029 | 0,001 | 0,000 |
| 10000 | 0,085 | 0,116 | 0,002 | 0,001 |
| 50000 | 1,599 | 2,892 | 0,011 | 0,007 |
| 100000 | 6,373 | 11,523 | 0,023 | 0,015 |
| 250000 | 39,891 | 72,371 | 0,061 | 0,040 |
| 750000 | 360,594 | 649,497 | 0,199 | 0,139 |
| 1000000 | 638,55 | 1156,174 | 0,268 | 0,199 |
| 1250000 |  |  | 0,341 | 0,260 |
| 1500000 |  |  | 0,407 | 0,327 |
| 2000000 |  |  | 0,553 | 0,487 |

Сортировка Хоара работает быстрее сортировки расческой.

Из сортировок за О( сортировка вставкой была быстрее сортировки выбором.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Кол-во элементов | (Время сортировки вставкой/n2)\* | (Время сортировки выбором/n2)\* | (Время сортировки расческой/n\*)\* | (Время сортировки Хоара/n\*)\* |
| 1000 | 10 | 10 | 0 | 0 |
| 5000 | 0,64 | 11,6 | 16,276 | 0 |
| 10000 | 8,5 | 11,6 | 15,052 | 7,526 |
| 50000 | 6,396 | 11,568 | 14,094 | 8,969 |
| 100000 | 6,373 | 11,523 | 13,847 | 9,031 |
| 250000 | 6,383 | 11,580 | 13,607 | 8,923 |
| 750000 | 6,410 | 11,547 | 13,595 | 9,496 |
| 1000000 | 6,385 | 11,562 | 13,446 | 9,984 |
| 1250000 |  |  | 13,469 | 10,267 |
| 1500000 |  |  | 13,225 | 10,626 |
| 2000000 |  |  | 13,210 | 10,961 |

При делении времени на сложность значения стремятся к константе, следовательно, сортировки соответствуют своей теоретической сложности.

# Заключение

Создав программу с сортировками, проверив их на корректность и проведя эксперименты с сортировкой массивов с разным количеством элементов на время. Я убедилась, в том, что время сортировки массива увеличивается в зависимости от количества элементов как сложность сортировки.

# Приложение

Сортировка вставками:

int bplace(double\* a, double r, int k) {

int mid, st = 0, fn = k - 1;

while (st <= fn) {

mid = (st + fn) / 2;

if (r < a[mid])

fn = mid - 1;

else if (r > a[mid])

st = mid + 1;

else

return mid;

}

return st;

}

void shift(double\* a, int s, int k) {

int i;

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

a[i] = a[i - 1];

}

void insert(double\* a, int n) {

int k;

double r;

int s;

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

r = a[k];

s = bplace(a, r, k);

shift(a, s, k);

a[s] = r;

}

}

Сортировка выбором:

void select(double\* a, int n) {

int i;

int j;

int mini;

double tmp;

for (i = 0; i < n - 1; i++) {

mini= i;

for (j = i+1; j < n; j++)

if (a[j] < a[mini])

mini = j;

tmp = a[mini];

a[mini] = a[i];

a[i] = tmp;

}

}

Сортировка расческой:

void comb(double\* a, int n) {

double factor = 1.246;

double step = n - 1;

double tmp;

while (step > 1)

{

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

if (a[i] > a[i + (int)step]) {

tmp = a[i];

a[i] = a[i + (int)step];

a[i + (int)step] = tmp;

}

}

step /= factor;

}

}

Сортировка Хоара:

int partition(double\* a, int l, int r) {

double tmp;

double mid = a[l + (r - l) / 2];

while (l < r) {

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

l++;

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

r--;

if (l < r) {

if (a[l] > a[r]) {

tmp = a[l];

a[l] = a[r];

a[r] = tmp;

}

else if (a[l] == a[r])

r--;

}

}

return r;

}

void quicksort(double\* a, int l, int r) {

int p;

if (l <= r) {

p = partition(a, l, r);

quicksort(a, l, p - 1);

quicksort(a, p + 1, r);

}

}