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**:

студент группы 3822Б1ПМ1

Сергеев М.О.

**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

Цель – изучение алгоритмов сортировок с теоретической вычислительной сложностью , и , где – количество элементов в массиве.

Задачи:

1. Изучить алгоритмы сортировок и реализовать их.
2. Проверить корректность работы алгоритмов.
3. Измерить скорость выполнения сортировок.
4. Проверить соответствие времени выполнения сортировок с их теоретической сложностью.

# Метод решения

Выбраны следующие алгоритмы сортировок:

1. Сортировка вставками.
2. Сортировка расческой.
3. Поразрядная сортировка.

Сортировка вставками.

Элементы входной последовательности рассматриваются по одному, и каждый новый поступивший элемент размещается в подходящее место среди ранее упорядоченных элементов. Во внешнем цикле алгоритм проходит по всем элементам последовательности, а во внутреннем цикле попарно последовательно сравнивает текущий элемент с уже пройденными, меняя их местами в случае нарушения упорядоченности. Вычислительная сложность алгоритма – .

Сортировка расческой.

Данный алгоритм является улучшением сортировки «Пузырек». В [пузырьковой сортировке](http://algolab.valemak.com/bubble) при каждом проходе по массиву сравниваются соседние элементы. Здесь же сравниваются элементы, находящиеся на фиксированном расстоянии, определяемом фактором уменьшения. При каждом следующем прохождении длина массива делится на этот фактор, и расстояние между сравниваемыми элементами уменьшается, пока на очередном проходе не достигнет единицы. Далее массив заканчивает сортировку обычным "пузырьком". В результате практических тестов и теоретических исследований оптимальное значение для фактора уменьшения определено следующее: , – корень золотого сечения. Вычислительная сложность алгоритма – .

Поразрядная сортировка.

Сравнение производится поразрядно: сначала сравниваются значения крайнего разряда, и элементы сортируются по результатам этого сравнения, затем сравниваются значения следующего разряда, и элементы упорядочиваются, сохраняя относительный порядок, достигнутый при предыдущей сортировке. Затем аналогично делается для следующего разряда. Работая с числами с плавающей запятой, в качестве одного разряда удобно использовать один байт, и сравнивать байты, как натуральные числа, но впоследствии придется переставить отрицательные числа в начало массива. Сортировка по байтам реализуется при помощи выделения дополнительной памяти – массива из 256 элементов, использующегося, как массив префикс-сумм. Вычислительная сложность алгоритма - .

# Руководство пользователя

На вход программа получает натуральное число – количество элементов в массиве.

Выводит два параметра:

validity – целое число, соответствующее количеству элементов в отсортированном массиве, не совпавших с элементами массива, отсортированного встроенной в stdlib.h функцией qsort.

time – примерное время выполнения сортировки в секундах.

# Описание программной реализации

Файлы.

1. Radix.cpp – поразрядная сортировка.
2. Comb.cpp – сортировка расческой.
3. Insert.cpp – сортировка вставками.

Функции.

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

Сравнивает два числа, как два числа типа double.

void countbyte (unsigned long long int \* arr, int n, int count[256], int byte)

Получает на вход: указатель на массив, число элементов в нем, массив целых, текущий байт. Первоначально в каждом элементе массива count хранится количество элементов массива arr, таких что n-ый байт элемента массива arr совпадает с номером элемента в массиве count. После массив переписывается, как массив префикс-сумм.

void uint\_radix\_sort (unsigned long long int\* arr, int size, unsigned long long int\* arr\_tmp)

Алгоритм поразрядной сортировки. Сортирует массив, как массив беззнаковых целых.

void radix\_double (double\* arr, double\* tmp, int n)

Сортирует массив, отсортированный uint\_radix\_sort, как массив double.

int validation (double\* arr1, double\* arr\_origin, int size)

Сравнивает элементы массива arr1 с элементами массива arr\_origin, отсортированного встроенной функцией qsort. Возвращает количество несовпавших элементов.

void fill (double\* arr1, double\* arr2, int size)

Заполняет два массива размера size попарно одинаковыми случайными целыми числами из промежутка [-1000000;1000000].

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

Меняет значения по адресам a и b местами.

void comb (double\* arr, int n)

Алгоритм сортировки расческой.

void insert (double \* arr, int n)

Алгоритм сортировки вставками.

# Подтверждение корректности

Для подтверждения корректности в программе была реализована функция validation, которая сортирует исходный массив при помощи встроенной в stdlib.h функции qsort и сравнивает его элементы с элементами массива, отсортированного при помощи проверяемого алгоритма.

# Результаты экспериментов

По данным экспериментов видно, что при достаточно большом размере массива сортировка вставками оказалось самой медленной, а поразрядная – самой быстрой.

Что бы подтвердить теоретическую сложность алгоритмов достаточно рассмотреть зависимость , где – количество элементов в массиве, – время выполнения сортировки, – функция одного порядка великости с функцией заявленной, как сложность сортировки, при достаточно больших.

Каждая из трех функций стремится к константе, следовательно, алгоритмы сортировок соответствуют заявленной сложности.

# Заключение

В результате проделанной работы изучены и реализованы сортировки вставками, расческой и поразрядная. Подтверждена корректность алгоритмов и их теоретическая сложность.

# Приложение

void countbyte(unsigned long long int \* arr, int n, int count[256], int byte)

{

unsigned char\* arrC;

int bias;

int tmp, temp;

bias = sizeof(unsigned long long int);

arrC = (unsigned char\*)arr;

for (int i = 0; i < 256; i++)

count[i] = 0;

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

count[arrC[j \* bias + byte]]++;

tmp = count[0];

for (int i = 1; i < 256; i++)

{

temp = count[i];

count[i] = tmp;

tmp += temp;

}

count[0] = 0;

}

void uint\_radix\_sort(unsigned long long int\* arr, int size, unsigned long long int\* arr\_tmp)

{

unsigned char\* arrc = (unsigned char\*)arr;

int sizetype = sizeof(unsigned long long int);

int count[256];

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

{

countbyte(arr, size, count, i);

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

arr\_tmp[count[arrc[j \* sizetype + i]]++] = arr[j];

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

arr[k] = arr\_tmp[k];

}

}

void radix\_double(double\* arr, double\* tmp, int n)

{

int count = 0, i;

uint\_radix\_sort((unsigned long long int\*)arr, n, (unsigned long long int\*) tmp);

while (count < n && arr[count++]>0);

count--;

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

tmp[n-count+i] = arr[i];

for (; i < n; i++)

tmp[n - (++count)] = arr[i];

}

void insert(double \* arr, int n)

{

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

{

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

{

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

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

else break;

}

}

}

void comb(double\* arr, int n)

{

double factor = 1.247;

double step = n/factor;

while (step - 1 > 0)

{

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

{

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

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

}

step /= factor;

}

}