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

Шарамыгина И.В.

**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

**Цель работы** – изучение алгоритмов с различной эффективностью.

**Задачи:**

* Изучить алгоритмы упорядочивания массивов с плавающей запятой
* Написать программу, содержащую реализацию алгоритмов и пользовательский интерфейс
* Провести проверку правильной работы сортировок
* Сравнить теоретическое время выполнение каждой сортировки с полученным на практике

# Метод решения

Были выбраны три сортировки:

1. Сортировка выбором
2. Сортировка слиянием
3. Поразрядная сортировка (LSD)
4. **Сортировка выбором**

Для сортировки массива находим минимальный элемент и меняем его местами с первым, затем повторяем поиск минимального элемента, не учитывая первый элемент, и меняем второй найденный минимальный элемент со вторым по порядку. Продолжаем до конца массива. Таким образом, после i-ой итерации первые i элементов будут стоять на своих местах. Алгоритм проходится по массиву n² раз для обнаружения наименьшего из оставшихся элементов, следовательно асимптотика: О(n²) в лучшем, среднем и худшем случае.

1. **Сортировка слиянием**

Разделим массив пополам, рекурсивно отсортируем части, после чего выполним процедуру слияния: поддерживаем два указателя, один на текущий элемент первой части, второй – на текущий элемент второй части. Из этих двух элементов выбираем минимальный, вставляем в дополнительный массив и сдвигаем указатель, соответствующий минимуму. Производим копирование из дополнительной памяти в основной массив. Слияние работает за O(n), уровней всего logn, поэтому асимптотика O(nlogn).

1. **Поразрядная сортировка (LSD)**

Для сортировки чисел с плавающей запятой передадим в функцию указатель, приведенный к типу unsigned int. В функции сортировки создадим копию массива, приведенную к типу unsigned char. В отдельном массиве длины 256 (все вариации байта) будем считать количество каждого вида байтов в числах на рассматриваемой позиции, затем изменим массив, чтобы в каждой ячейке лежали позиции, с которого мы должны разместить числа с этим байтом. Разместим числа по соответствующим позициям. Проделаем для каждого байта в числе. Так как в двоичном коде отрицательные числа хранятся дополнительным кодом, они окажутся в конце массива после сортировки. Переместим их в начало, выстраивая по убыванию их модули, используя дополнительную память. Скопируем окончательно отсортированный массив в основной. Поскольку, переходя к новому разряду, мы не теряем упорядоченность, которой достигли ранее, в худшем случае будет выполнено m\*n операций, где n – кол-во элементов в массиве, а m – количество разрядов, отсюда и асимптотика О(n).

# Руководство пользователя

При запуске программы необходимо ввести число – размерность массива для сортировки.

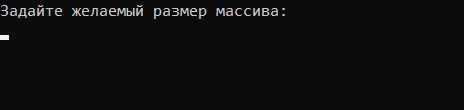


Рисунок 1 - Ввод размерности массива

Далее пользователю предлагается выбрать способ задания массива, для этого нужно ввести номер выбранного варианта:

1. с клавиатуры;

2. с помощью рандомайзера.

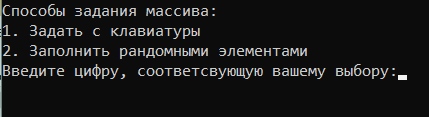


Рисунок 2 - Способы задания массива

При выборе 1 команды пользователю предлагается ввести элементы в количестве соответствующему размерности массива, введенной ранее. Дробные числа пишутся с точкой (пример: 1.5).

При выборе 2 команды массив автоматически заполнится рандомными дробными числами из промежутка от -1000 до 1000.

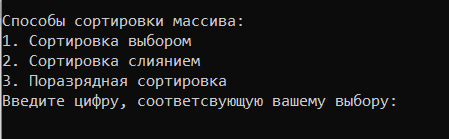
На следующем шаге пользователю предоставляется возможность выбрать вид сортировки. Также необходимо ввести номер команды.  


Рисунок 3 - Выбор способа сортировки

Программа выведет итог проверки работы сортировки (“ok”, “error”) и время ее выполнения в секундах и закончит работу.



Рисунок 4 - Вывод в программе

При введении неверной команды программа выведет предупреждения «Неверный ввод», программу нужно будет запустить заново.

# Описание программной реализации

Программа состоит из единственного файла (laboratory work.cpp), в которой описаны все необходимые функции (10 шт).

void myswap(float\* a, float\* b) – на вход принимает два указателя, меняет местами значения.

int compare(const void\* a, const void\* b) – принимает на вход два указателя, необходима для работы qsort – функции, участвующей в проверке правильности сортировки, возвращает необходимые функции значения.

void select(float\* arr, int size) – принимает указатель на сортируемый массив и размерность массива, производит сортировку массива выбором минимальных элементов и перемещения их в начало массива.

void merge(float\* in, float\* out, int left, int mid, int right) – принимает указатели на массивы входа и выхода, а так же индексы правой и левой границ и середину рассматриваемого участка массивов, выполняет процедуру слияния двух отсортированных частей массивов.

void merge\_sort(float\* arr, float\* tmp, int l, int r) – принимает на вход указатель на массив, массив дополнительной памяти, правую и левую границы рассматриваемой части массива, рекурсивно сортирует данный массив.

void countbyte(unsigned int\* arr, int size, int count[256], int byte) – принимает на вход указатель на беззнаковый целый тип данных, размерность массива, массив длины 256 и переменную с указанием рассматриваемого байта, определяет индексы вхождения элементов с каждым видом байта в массив.

void radix\_unsigned(unsigned int\* arr, int size, unsigned int\* arr\_tmp) – принимает на вход указатель на беззнаковый целый тип данных, размерность массива, массив дополнительной памяти, проходя по каждому байту чисел расставляет числа в заданном функцией countbyte порядке.

void radix\_float(float\* arr, float\* tmp, int size) – принимает два указателя на основной и дополнительной памяти массивы, размерность массивы, выполняет сортировку данного массива типа float.

int check(float\* arr, float\* arr\_test, int size) – принимает на вход два указателя на заданный и отсортированный массивы, размерность, выполняет сортировку исходного массива и поэлементное сравнение с итогом реализации выбранной пользователем сортировки, возвращает 1, если элементы стоят в одинаковом (верном) порядке, или 0, если в сортировке была допущена ошибка.

int main() – основная функция, отвечает за корректную работу, использование подпрограмм, взаимодействие с пользователем, возвращает 0 в случае корректной работы программы.

# Подтверждение корректности

Для подтверждения корректности в программе была реализована функция check, которая по исходному массиву генерирует такой же, а затем сортирует его встроенной в stdlib.h функцией qsort, которая принимает массив, тип его данных и компоратор (критерий, по которому сортируется массив), а затем поэлементно сравнивает отсортированный моей программой и этой функцией массивы. Если ошибок не найдено и сортировки сработали одинаково программа выведет «ok» в итоге сортировки, иначе будет выведено «error».

# Результаты экспериментов

По данным экспериментов видно, что сортировка выбором является самой медленной среди выбранных сортировок. Сортировка слиянием выполняется заметно быстрее, но все же уступает поразрядной.

Для проверки соответствия теоретической сложности и сложности, полученной на практике, поделим время выполнения программы на ее предполагаемую сложность. В случае получения константы можно сделать вывод, что сложность сортировок была определена верно. Результаты по каждой сортировке представлены в виде графиков.

Видно, что график вышел на константу – функция соответствует предполагаемой сложности O(n²).

При делении на nlog(n) график стремится к числу 1,95Е-08 следовательно сложность так же определена верно.

При выбранных n было получено одно значение. Сложность определена верно.

# Заключение

В работе были реализованы такие виды сортировок как выбор, слияние и поразрядная, проверены их сложности, выполнено сравнение времени их выполнения при разных размерностях. Сортировка выбором наименее эффективна, среднюю эффективность имеет сортировка слиянием, самые же лучшую скорость работы показывает поразрядная сортировка.

Было проверено соответствие теоретической асимптотики на практике. Для всех сортировок была получена константа при делении времени выполнения, на сложность алгоритма.

# Приложение

void select(float\* arr, int size)

{

int i, j;

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

{

int min\_ind=i;

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

if (arr[min\_ind] > arr[j])

min\_ind = j;

myswap(&arr[i], &arr[min\_ind]);

}

}

void merge(float\* in, float\* out, int left, int mid, int right)

{

int l1 = left;

int l2 = mid+1;

int ind=0;

int i;

while (l1 <= mid && l2 <= right)

{

if (in[l1] <= in[l2])

out[ind++] = in[l1++];

else

out[ind++] = in[l2++];

}

while (l1>mid && l2<= right)

out[ind++] = in[l2++];

while (l2 > right && l1 <= mid)

out[ind++] = in[l1++];

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

in[left + i] = out[i];

}

void merge\_sort(float\* arr, float\* tmp, int l, int r) {

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

if (l == r) {

return;

}

merge\_sort(arr, tmp, l, m);

merge\_sort(arr, tmp, m + 1, r);

merge(arr, tmp, l, m, r);

}

void countbyte(unsigned int\* arr, int size, int count[256], int byte) //исхоодный массив, размер массива, символьный массив, текущий разряд

{

unsigned char\* masc = (unsigned char\*)malloc(sizeof(unsigned char)\*size);

int i, tmp1, tmp2, bias = sizeof(unsigned int);

masc = (unsigned char\*)arr;

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

//printf\_s("%d) %d\n", i, masc[i]);

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

count[i] = 0;

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

count[masc[i \* bias + byte]]++;

tmp1 = count[0];

count[0] = 0;

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

{

tmp2 = count[i];

count[i] = count[i - 1] + tmp1;

tmp1 = tmp2;

}

}

void radix\_unsigned(unsigned int\* arr, int size, unsigned int\* arr\_tmp)

{

unsigned char\* masc = (unsigned char\*)malloc(sizeof(unsigned char)\*4\*size);

int count[256], sizetype = sizeof(unsigned int);

int j, i;

masc = (unsigned char\*)arr;

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

{

countbyte(arr, size, count, i);

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

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

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

arr[j] = arr\_tmp[j];

}

}

void radix\_float(float\* arr, float\* tmp, int size)

{

int i=0;

int count = 0;

radix\_unsigned((unsigned int\*)arr, size, (unsigned int\*)tmp);

while (i < size)

{

if (arr[i] >= 0)

count++;

i++;

}

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

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

}

count = size - count;

for (count = count--; count >= 0; count--) {

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

}

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

arr[i] = tmp[i];

}

}