**СОДЕРЖАНИЕ**

[Постановка задачи 3](#_Toc154701873)

[Метод решения 4](#_Toc154701874)

[Руководство пользователя 6](#_Toc154701875)

[Описание программной реализации 8](#_Toc154701876)

[Подтверждение корректности 14](#_Toc154701877)

[Результаты экспериментов 15](#_Toc154701878)

[Заключение 17](#_Toc154701879)

[Список литературы 18](#_Toc154701880)

[Приложение 19](#_Toc154701881)

# Постановка задачи

В данной лабораторной работе для чисел с плавающей запятой (float) были реализованы такие алгоритмы сортировок как:

1. Сортировка «Пузырёк»
2. Сортировка «Хоара» (быстрая сортировка)
3. Сортировка слиянием
4. Поразрядная сортировка (LSD)

Реализованные алгоритмы необходимо проверить на корректность, а также провести тестирование алгоритма на временную сложность для проведения сравнения по данному критерию на основе полученных в результате тестирования данных.

Сравнение необходимо для выявления наиболее эффективного алгоритма сортировки.

# Метод решения

Для решения поставленной задачи, необходимо реализовать все 4 вида сортировок. Приведу описание их алгоритмов, а также алгоритмов некоторых вспомогательных функций:

1. ***Сортировка «Пузырёк» (\_bubble\_sort)***

Упорядоченный массив создается на том же участке памяти, где находится исходная последовательность. Идея метода состоит в том, чтобы попарно сравнивать соседние элементы. Каждый проход начинается с начала последовательности. Сравнивается первый элемент со вторым: если порядок между ними нарушен, то они меняются местами. Затем сравниваются второй с третьим, третий с четвертым и так далее до конца массива; элементы с неправильным порядком в паре меняются местами. В итоге, после первого прохода, максимальный элемент будет находится на последнем месте в массиве, он как бы «всплывет» наверх. На следующем проходе рассматривается последовательность от 1 до n-1, затем от 1 до n-2, и так до конца. После каждого прохода можно делать проверку: выполнялись ли перестановки элементов. Если не выполнялись, то сортировка завершена.[2]

1. ***Сортировка «Хоара» (быстрая сортировка) (\_hoara\_sort)***

Алгоритм сортировки основывается на выборе опорного элемента — им может быть любой элемент массива. Все остальные элементы массива сравниваются с опорным и те, которые меньше него, ставятся слева от него, а которые больше или равны — справа. Для двух получившихся блоков массива (меньше опорного, и больше либо равны опорному) производится точно такая же операция — выделяется опорный элемент и всё идёт точно так же, пока в блоке не останется один элемент.[1]

1. ***Сортировка слиянием (\_merge\_sort)***

Слияние означает объединение двух (или более) последовательностей в одну упорядоченную последовательность при помощи циклического выбора элементов, доступных в данный момент.

Сначала последовательность разбивается на несколько подпоследовательностей меньшего размера. Затем эти последовательности сортируются с помощью рекурсивного вызова или непосредственно, если их размер достаточно мал. Затем подпоследовательности комбинируются, и мы получаем отсортированную исходную последовательность.

Процедура слияния предполагает объединение двух предварительно упорядоченных подпоследовательностей размерности n/2 в единую последовательность размерности n. Начальные элементы предварительно упорядоченных последовательностей сравниваются между собой, и из них выбирается наименьший. Соответствующий указатель перемещается на следующий элемент. Процедура повторяется до тех пор, пока не достигнут конец одной из подпоследовательностей. Оставшиеся элементы другой подпоследовательности при этом передаются в результирующую последовательность в неизменном виде.[4]

1. ***Поразрядная сортировка (LSD) (\_radix\_sort)***

Алгоритм представляет число в байтовом виде, так что в последствии при проходе через каждый байт вызывается функция, подсчитывающая количество значений байта для данной цифры и формирующая из этих значений предварительную сумму. Затем элементы вспомогательного массива воспроизводятся по вычисленным значениям.

Поскольку проход воспроизводится по каждому байту, то при каждой итерации производится «сортировка подсчетом» по данному разряду. Кроме того, алгоритм меняет адреса исходного и вспомогательного массивов, для работы со свежими элементами и заново вычисляет байтовое представление. Впоследствии массив будет содержать сначала отсортированные положительные элементы, а затем отрицательные, если они присутствовали в изначальном массиве.

В конце реализуется правильное размещение элементов во вспомогательном массиве, который в итоге копируется в основной массив.[2]

# Руководство пользователя

При запуске программы на консоли появляется меню, представленное на *Рисунке 1*.

Перед началом использования следует выполнить ввод данных, введя с клавиатуры «0».

Далее пользователю предлагается ввести количество элементов в массиве (не более 100000), ключ генерации случайных чисел (любое число, на базе которого будут генерироваться псевдослучайные числа), левую и правую границы диапазона значений.

После ввода данных на консоли появится неотсортированный массив.

При выборе пунктов 1-4 выполняется сортировка массива выбранным пользователем методом, после чего на консоль выводится отсортированный массив и время выполнения сортировки.

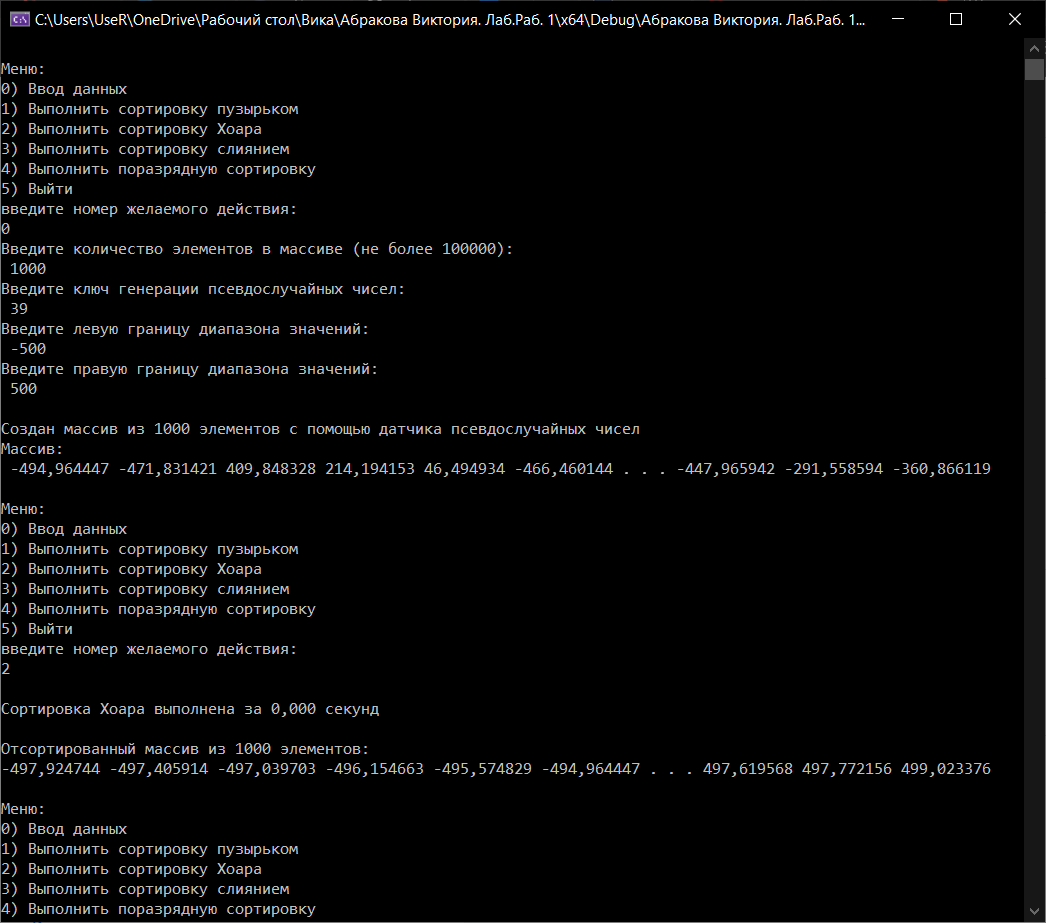
При выборе пункта «Выйти» происходит закрытие консоли.

Рисунок 1. Пример работы программы

# Описание программной реализации

Описание работы алгоритма сортировки пузырьком:

1. Процедура \_bubble\_sort принимает на вход массив.
2. n - переменная, хранящая длину массива.
3. Двойной цикл for используется для прохода по массиву и перестановки элементов. Внешний цикл отвечает за количество проходов по массиву, а внутренний - за выполнение сравнений и перестановок.

Описание работы алгоритма сортировки Хоара:

1. Процедура \_hoara\_sort принимает на вход массив.
2. n - переменная, хранящая длину массива.
3. Функция \_partition выбирает опорный элемент и переставляет все элементы, меньшие его, перед ним, а все большие - после него.
4. Функция \_qsort вызывается рекурсивно для двух подмассивов до тех пор, пока половина массива не будет состоять из одного элемента.

Описание работы алгоритма сортировки слиянием:

1. Процедура \_merge\_sort принимает на вход массив.
2. n - переменная, хранящая длину массива.
3. step - переменная, хранящая длину подмассива.
4. Функция \_merge принимает два подмассива и результирующий массив, в котором они будут объединены.
5. an и bn - переменные, хранящие длины подмассивов.
6. ia, ib, ir - переменные, хранящие индексы массивов.
7. Функции \_min и \_max возвращают минимальное и максимальное значение соответственно.

Описание работы алгоритма поразрядной сортировки:

1. Процедура \_radix\_sort принимает на вход массив.
2. n- переменная, хранящая длину массива.
3. float\* a - основной массив.
4. float\* a2 - вспомогательный массив.
5. count - массив, который используется для распределяющего подсчёта/
6. Последние циклы for проверяют вводимые значения на знак, так чтобы отрицательные числа шли в отсортированном порядке перед положительными.

# Подтверждение корректности

Для проверки корректности реализованных алгоритмов была проведена их проверка на случайных данных. Все алгоритмы были проверены на случайно сгенерированных данных различного объёма. Для проверки правильности выполнения алгоритма применялось сравнение с qsort() из #include <stdlib.h>, qsort() представленна на *Рисунке 2*. Результаты выполнения написанной сортировки сравнивались с результатом стандартной функции.

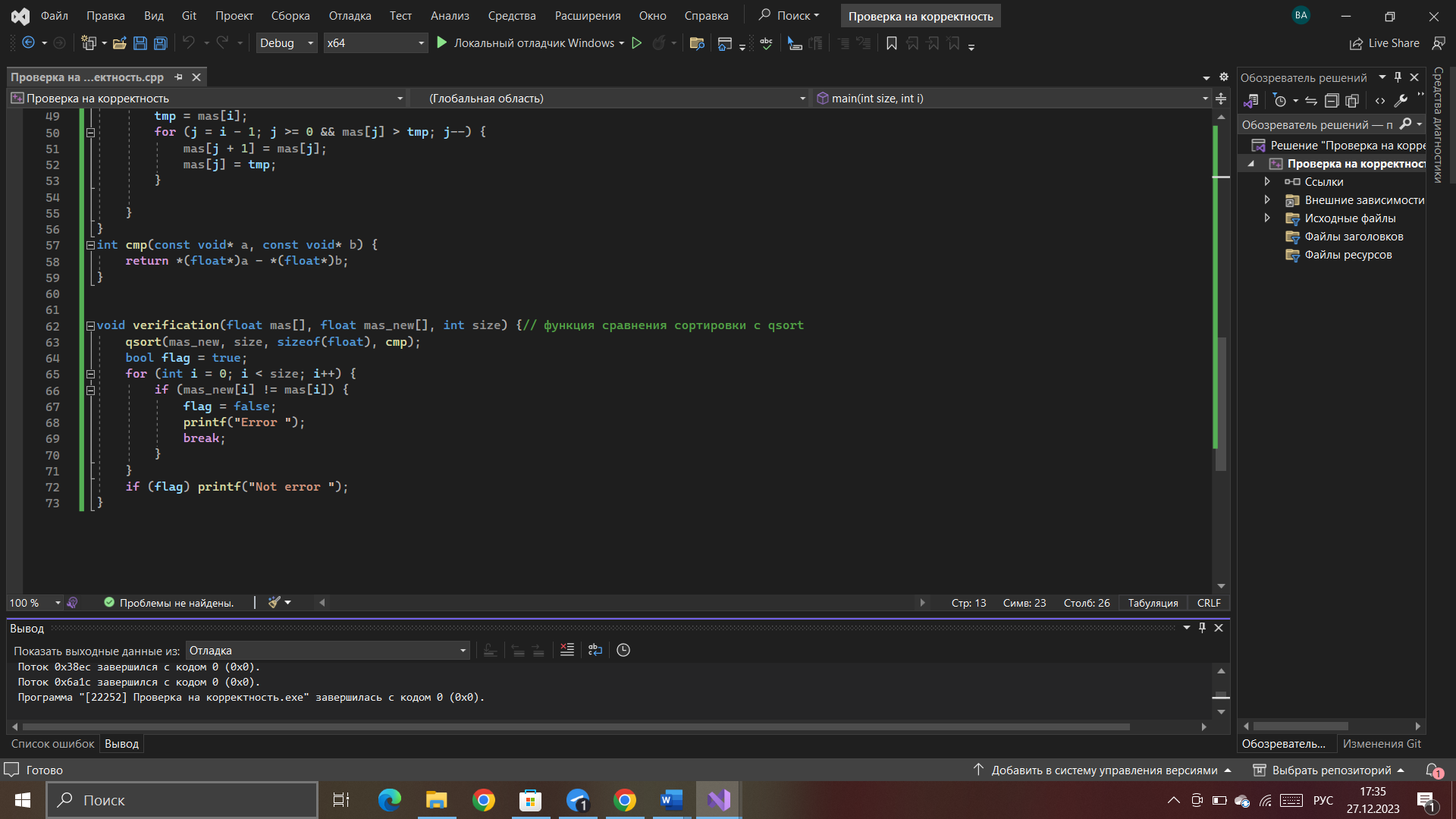
В итоге проверки, можно сделать вывод, что все алгоритмы корректно сортируют данные.

Рисунок 2. Сортировка qsort() из библиотеки #include <stdlib.h>

# Результаты экспериментов

Докажем асимптотическую сложность каждого из алгоритма. В качестве доказательства будет приводиться график отношения времени выполнения (t) программы к предполагаемой сложности. Если предположение будет верно, то график функции должен быть близок к некоторой константе.

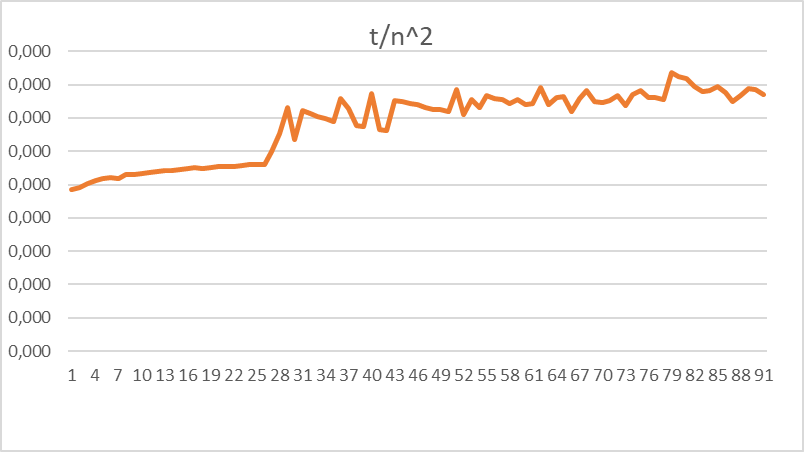
Для сортировки пузырьком мы предполагали, что сложность алгоритма O(n^2). Как видно на *Рисунке 3*, предположение верно, потому что данная зависимость стремится к некоторой константе.

Рисунок 3. График отношения времени выполнения сортировки пузырьком к сложности О(n^2)

Для сортировки Хоара мы предполагали, что сложность алгоритма O(n\*log(n)). Как видно на *Рисунке 4*, предположение верно, потому что данная зависимость стремится к некоторой константе.

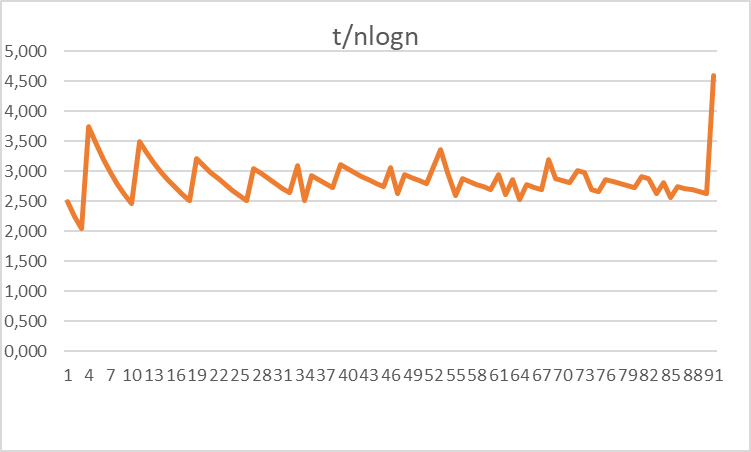


Рисунок 4. График отношения времени выполнения сортировки Хоара к сложности О(nlogn)

Для сортировки слиянием мы предполагали, что сложность алгоритма O(n^2). Как видно на *Рисунке 5*, предположение верно, потому что данная зависимость стремится к некоторой константе.

Рисунок 5. График отношения времени выполнения сортировки слиянием к сложности О(n^2)

Для LSD-сортировки мы предполагали, что сложность алгоритма O(n). Как видно на *Рисунке 6*, предположение верно, потому что данная зависимость стремится к некоторой константе.

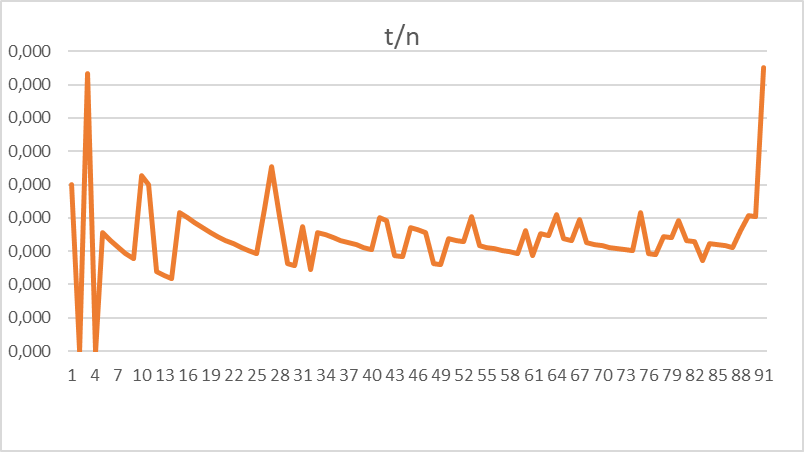


Рисунок 6. График отношения времени выполнения поразрядной сортировки к сложности О(n)

# Заключение

В ходе лабораторной работы было реализовано 4 сортировки: сортировка пузырьком, сортировка Хоара (быстрая сортировка), сортировка слиянием и поразрядная сортировка. Все сортировки прошли проверку на корректность, а также удалось доказать асимптотическую сложность каждой из них.

По проделанной работе можно сделать несколько выводов:

1. Реализация сортировки пузырьком очень проста, однако она подходит для сравнительно небольших по размеру массивов, в противном случае время её выполнения будет велико.
2. Самая быстрая сортировка, подходящая для массивов достаточно больших размеров – поразрядная.

# Список литературы

1. Как работает быстрая сортировка: [Электронный ресурс] // КОД. URL: https://thecode.media/qsort/. (Дата обращения: 25.12.2023)
2. Поразрядная сортировка :: Radix sort: [Электронный ресурс] // AlgoLab. URL: http://algolab.valemak.com/radix. (Дата обращения: 25.12.2023)
3. Сортировка пузырьком на Си: [Электронный ресурс] // vscode. URL: <https://vscode.ru/prog-lessons/sortirovka-puzyirkom-na-si.html>. (Дата обращения: 25.12.2023)
4. Сортировка слиянием: [Электронный ресурс] // Программирование. URL: https://prog-cpp.ru/sort-merge/. (Дата обращения: 25.12.2023)

# Приложение

***Сортировка пузырьком***:

void \_bubble\_sort(float a[], int n){

int i, j;

float tmp;

int flag = 0;

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

flag = 0;

for (j = n - 1; j > i; j--){

if (a[j] < a[j - 1]) {

tmp = a[j];

a[j] = a[j-1];

a[j-1] = tmp;

flag = 1;

}

}

if (flag == 0)

break;

}

}

***Сортировка Хоара:***

int \_partition(float\* a, int s, int f) {

float b = a[s];

float tmp;

f--;

while (s <= f) {

while (a[s] < b)

s++;

while (a[f] > b)

f--;

if (s <= f) {

tmp = a[s];

a[s] = a[f];

a[f] = tmp;

s++;

f--;

}

}

return s;

}

void \_qsort(float\* a, int s, int f) {

if (f - s < 2)

return;

int p = \_partition(a, s, f);

\_qsort(a, s, p);

\_qsort(a, p, f);

}

void \_hoara\_sort(float a[], int n) {

\_qsort(a, 0, n);

}

***Сортировка слиянием:***

void \_merge(float\* a, float\* b, float\* res, int an, int bn){

int ia = 0, ib = 0, ir = 0;

while (an > ia && ib < bn){

if (a[ia] < b[ib])

res[ir++] = a[ia++];

else{

res[ir++] = b[ib++];

}

}

for (; ia < an; ia++){

res[ir++] = a[ia];

}

for (; ib < bn; ib++){

res[ir++] = b[ib];

}

}

int \_max(int a, int b) {

if (a > b) {

return a;

}

return b;

}

int \_min(int a, int b) {

if (a < b) {

return a;

}

return b;

}

void \_mergesort(float\* a, int n) {

int step = 1, i;

float\* temp;

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

for (; step < n; step \*= 2) {

for (i = 0; i < n; i += 2 \* step) {

\_merge(&a[i], &a[i + step], &tmpdata[i], \_max(0, \_min(step, n - i)), \_max(0, \_min(step, n - step - i)));

}

temp = a;

a = tmpdata;

tmpdata = temp;

}

}

void \_merge\_sort(float a[], int n){

\_mergesort(a, n);

}

***Поразрядная сортировка:***

void foffset(unsigned char\* a, int n, int offset, int count[257]) {

int i = 0;

int tmp;

int temp;

for (i = 0; i < 257; count[i++] = 0) {};

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

count[

a[offset + i \* sizeof(float)]

]++;

}

tmp = count[0];//байт == 0

count[0] = 0;

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

{

temp = tmp;

tmp = count[i + 1];

count[i + 1] = temp;

}

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

}

}

void \_radixsort(float\* a, int n) {

int i = 0, j = 0;

unsigned int negativeDigits = 0;

int count[257];

float\* tempp;

float\* a2 = (float\*)malloc(n \* sizeof(float));

unsigned char\* pm = (unsigned char\*)a;

for (; i < sizeof(float); i++) {

foffset(pm, n, i, count);

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

a2[

count[

pm[j \* sizeof(float) + i]

]++

] = a[j];

}

tempp = a;

a = a2;

a2 = tempp;

pm = (unsigned char\*)a;

}

for (i = n - 1; i > -1; i--) {

if (a[i] >= 0) {

break;

}

negativeDigits++;

}

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

a2[i] = a[n - i - 1];

}

for (i = negativeDigits; i < n; i++) {

a2[i] = a[i - negativeDigits];

}

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

a[i] = a2[i];

}

free(a2);

}

void \_radix\_sort(float a[], int n){

\_radixsort(a, n);

}