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»**

**Выполнила**:

студентка группы 3823Б1ПМ1-1

Акчурина А.Н.

**Проверил**:

преподаватель каф. ВВСП,

Волокитин В.Д.

Нижний Новгород

2023

**Содержание**

[Постановка задачи 3](#_Toc160317623)

[Метод решения 4](#_Toc160317624)

[Руководство пользователя 6](#_Toc160317625)

[Описание программной реализации 7](#_Toc160317626)

[Подтверждение корректности 12](#_Toc160317627)

[Результаты экспериментов 13](#_Toc160317628)

[Заключение 18](#_Toc160317629)

[Список литературы 19](#_Toc160317630)

[Приложение: 20](#_Toc160317631)

# Постановка задачи

1. В данной лабораторной работе для чисел с плавающей точкой (double) были реализованы такие алгоритмы сортировок как:
2. Сортировка вставками
3. Сортировка Шелла
4. Сортировка слиянием
5. Поразрядная сортировка
6. Для выполненных сортировок необходимо сделать проверку на корректность и провести тестирование алгоритма на временную сложность для последующего сравнения и выявления на основе полученных результатов более эффективного методов 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 вставками(insert\_sort):**

Алгоритм сортировки основан на просматривании элементов входной последовательности один за другим и распределении каждого нового на свое место среди ранее упорядоченных элементов.

Входные данные представляют собой последовательность чисел(массив). В результате алгоритм должен вернуть перестановку исходной последовательности.

На первом шаге сортировки второй элемент сравнивается с первым, а на втором - третий элемент сравнивается с двумя предыдущими и т.д.

В уже отсортированный массив вставляется i-й элемент, не нарушая порядка. То есть при вставке i-го элемента на j-е место (j <i) элементы с индексами> j и <i увеличиваются на единицу.

1. **Сортировка Шелла(shella\_sort):**

Алгоритм сортировки можно представить в виде сортировки вставками с предварительными «грубыми» проходами.

Идея сортировки состоит с сравнении не только рядом стоящих, но и находящихся на расстоянии элементов.

На первом шаге сравниваются между собой элементы, находящиеся на определенном расстоянии друг от друга, выполняется сортировка вставками. На втором и последующих шагах расстояние уменьшается до тех пор, пока интервал между проверяемыми элементами не станет равным единице. Таким образом происходит сортировка всего массива.

1. **Сортировка слиянием(merge\_sort):**

Алгоритм сортировки рассматривает элементы отдельно друг от друга. На первом шаге выбирается «пороговый» элемент, таким образом, что новый массив разделяется «пороговым» элементом на два подмассива: в левой находятся все элементы, которые располагаются до «порогового», в правой же части все остальные. После этого происходит этап слияния, в ходе которого подмассивы объединяются в результирующий массив.

Процесс слияния осуществляется путем сравнения элементов из различных подмассивов и объединения их в упорядоченном порядке. На втором и последующих шагах используется подобный способ для разделения каждого из новых массивов на две части.

В тот момент, когда в одном из подмассивов заканчиваются элементы, происходит процесс перемещения оставшихся элементов из второго подмассива в основной. Последующие итерации производятся по такому же принципу, только для большей размерности подмассивов, пока они не сольются в массив изначального размера. Если исходный массив имеет размер 1, то сортировка не выполняется.

1. **Поразрядная сортировка(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.

# Руководство пользователя

Каждая сортировка представлена отдельным файлом и работает следующим образом:

1. При запуске программы пользователю необходимо ввести целое положительное число в современной арабской системе счисления, отражающее количество цифр в сортируемом наборе.
2. После программа предлагает выбор заполнения массива : «independent input» - самостоятельный ввод элементов или «filling in with the program» - заполнение программой в рандомном порядке. Для обозначения своего выбора ввести цифру 1 или 2 соответственно.
3. В конце программа выводит на консоль введенное ранее пользователем число отсортированных элементов, корректность выполнения программы (если во время сортировки чисел произошли ошибки, то на консоль выводится «Error», в ином случае “Not error”) и время выполнения сортировки.

Пример работы программы:

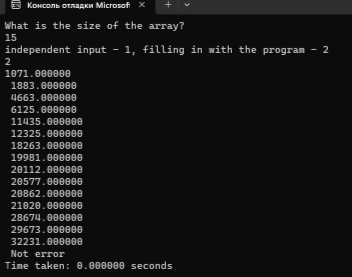


Рисунок 1

# Описание программной реализации

В этой главе будет описана программная реализация алгоритма сортировки.

1. **Программная реализация сортировки вставками:**

void insert\_sort(double\* mas, int size)

{

int j, i;

double tmp;

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

tmp = mas[i];

for (j = i - 1; j >= 0 && mas[j] > tmp; j--) {

mas[j + 1] = mas[j];

mas[j] = tmp;

}

}

}

**Описание работы алгоритма:**

1. Процедура insert\_sort принимает на вход массив.
2. size - 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.
4. **Программная реализации сортировки Шелла:**

void shell\_sort(double\* mas, int size)

{

int i, j, step;

double tmp;

for (step = size / 2; step > 0; step /= 2) {

for (i = step; i < size; i++) {

for (j = i; j >= step && mas[j] < mas[j - step]; j -= step) {

tmp = mas[j];

mas[j] = mas[j - step];

mas[j - step] = tmp;

}

}

}

**Описание работы алгоритма**:

1. Процедура shell\_sort принимает на вход массив.
2. size - переменная, хранящая длину массива.
3. step – переменная хранящая шаг сортировка(длина промежутка)
4. Тройной цикл for используется для прохода по массиву и перестановки элементов. Внешний цикл отвечает за количество проходов по массиву, внутренний - за перемещение по элементам, а последний за сравнение и выполнение перестановок.
5. **Программная реализация сортировки слиянием:**

void merge(double\* arr, double\* arr\_new, int left, int middle, int right) {

double\* p1 = &arr[left], \* p2 = &arr[middle], \* p3 = &arr\_new[left];

while ((p1 < &arr[middle]) && (p2 < &arr[right])) {

if (\*p1 <= \*p2) {

\*p3 = \*p1;

p3++;

p1++;

}

else {

\*p3 = \*p2;

p3++;

p2++;

}

}

while (p1 < &arr[middle]) {

\*p3 = \*p1;

p3++;

p1++;

}

while (p2 < &arr[right]) {

\*p3 = \*p2;

p3++;

p2++;

}

for (int i = left; i < right; i++) arr[i] = arr\_new[i];

}

void mergeSort(double\* arr, double\* arr\_new, int left, int right) {

int middle = (left + right) / 2;

if (right - left == 1) return;

mergeSort(arr, arr\_new, left, middle);

mergeSort(arr, arr\_new, middle, right);

merge(arr, arr\_new, left, middle, right);

}

**Описание работы алгоритма:**

1. Процедура merge\_sort принимает на вход массив.
2. Процедура merge сортирует правый и левый массивы.
3. size - переменная, хранящая длину массива.
4. double\* arr - основной массив.
5. double\* arr\_new - вспомогательный массив.
6. double\* p1 – массив, в котором находятся элементы все элементы, располагающиеся до порогового.
7. double\* p2 – массив, в котором находятся все элементы, не входящие в массив p1.
8. double\* p3 – присваиваем значение массива arr\_new[].
9. left – длина левого массива
10. middle – пороговый элемент.
11. right – длина правого массива.
12. в первом цикле идет сравнение и перенос элементов из подмассивов( p1 и p2) во вспомогательный массив.
13. далее два цикла проверяют массивы на остаточный элемент и добавляют его во вспомогательный массив.
14. цикл for переносит значения из вспомогательного массива в основной.
15. процедура mergeSort – делит основной массив на подмассивы.
16. **Программная реализация поразрядной сортировки:**

void radix\_sort(double arr[], int n, int SIZE) {

double\* arr\_new = (double\*)malloc(SIZE \* 8);

unsigned char\* r0, \* r1;

int num[257];

int r, k, t, tmp, i;

r0 = (unsigned char\*)arr;

r1 = (unsigned char\*)arr\_new;

r = 0;

for (k = 0; k < sizeof(double); k++) /\*заполнение стартового массива\*/

{

for (i = 0; i < 256; num[i++] = 0);

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

if (r == 0) num[r0[8 \* i + k]]++;/\*считает колличество чисел с одним разрядом \*/

/\* k-ый байт i элемента в изначальном первом массиве, идет по каждому элементу\*/

else num[r1[8 \* i + k]]++;

t = 0;/\*заполнение с 1-го элемента массива\*/

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

{

tmp = num[i];

num[i] = t;

t += tmp;

}

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

if (r == 0) arr\_new[num[r0[8 \* i + k]]++] = arr[i];

else arr[num[r1[8 \* i + k]]++] = arr\_new[i];

r = 1 - r;

}

r = 0;

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

if (arr[i] < 0) {

r++;

}

else {

break;

}

}

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

arr\_new[i] = arr[n - i - 1];

}

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

arr\_new[i] = arr[i - r];

}

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

arr[i] = arr\_new[i];

}

**Описание работы алгоритма:**

1. Процедура radix\_sort принимает на вход массив.
2. size - переменная, хранящая длину массива.
3. double\* arr - основной массив.
4. double\* arr\_new - вспомогательный массив.
5. num[r0[8 \* i + k]]++ - счетчик элементов с одним разрядом;
6. r – определяет четность и позволяет перемещаться между массивами.
7. последние циклы for проверяют вводимые значения на знак, так чтобы отрицательные числа шли в отсортированном порядке перед положительными.

# Подтверждение корректности

Для подтверждения корректности работы программ, использовалось сравнение отсортированного массива с отсортированным функцией qsort() из #include <stdlib.h> массивом. [Рисунок 2]

Для каждой программы проверка прошла успешно для достаточно большого количества входных данных (100000) [Рисунок 3].

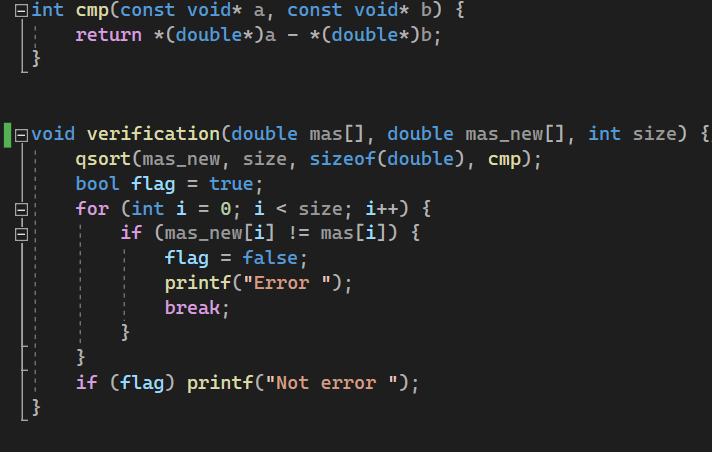


Рисунок 2



Рисунок 3

# Результаты экспериментов

В этой главе реализуется проверка предположения, что сложность сортировки слиянием О(n\*ln(n)), сортировки Шелла f(n)=О(n(logn)^2/loglogn), для сортировки вставками О(n^2), для поразрядной сортировки О(n). Замеры выполнялись последовательно для каждой сортировки для определенного значения размера. Изначально была взята такая последовательность вводимых пользователем размеров массива, как 1000, 10000, 100000, 1000000, но на практике выяснилось, что не для каждой сортировки это целесообразно. По выводимым данным было понятно, что для таких сортировок как поразрядная, Шелла и сортировка слиянием наиболее удобным для проверки является такой диапазон значений как [1000;3000000]. При этом для наиболее качественного построения графика в некоторых из них были внесены изменения. Так в сортировке Шелла и поразрядной были добавлены к изначальной последовательности такие размеры массива, как 50000 и 500000, причем в последней были убраны как излишние элементы 10000 и 100000. В диапазоне значений сортировки вставками были также внесены такие правки. Так были добавлены 20000 и 50000, но убраны 1000000, 2000000, 3000000. При анализе полученных результатов можно сделать вывод, что с такой точностью измерения времени и такой выборкой строго доказать теоретическую сложность алгоритмов на практике невозможно, но с некоторыми допущениями можно сделать следующие суждения о сложности сортировок: сложность сортировки вставками в среднем случае составила O(n^2), сложность сортировки Шелла в среднем случае на практике соответствует с О(n(logn)^2/loglogn), средняя сложность сортировки слиянием: O(nlogn), средняя сложность поразряднойсортировки: O(n). Сложность сортировки вставками в лучшем случае составила O(n).

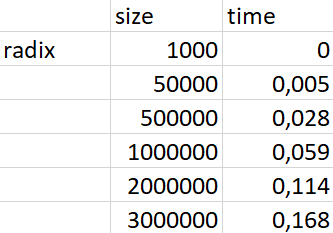


Рисунок 3

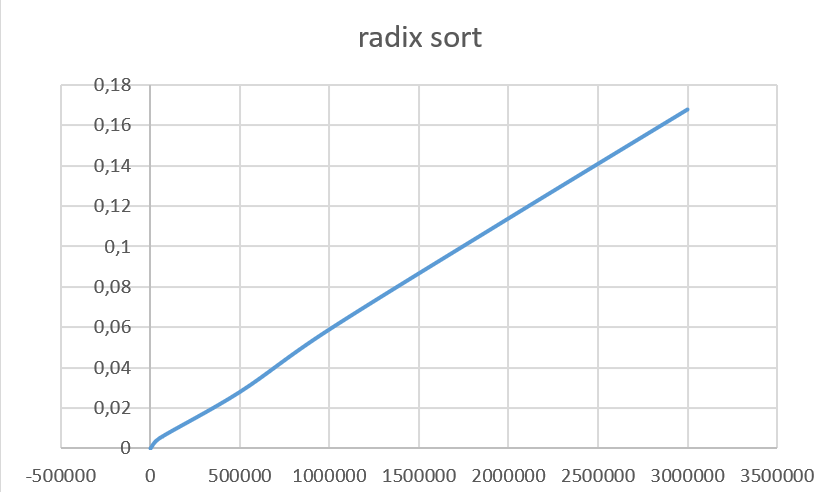


Рисунок 4

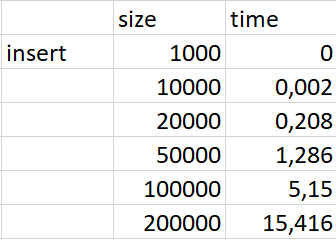


Рисунок 5

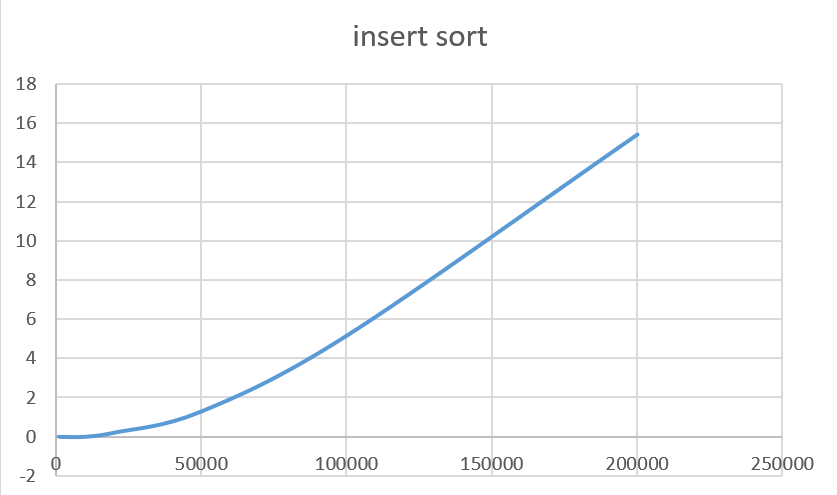


Рисунок 6

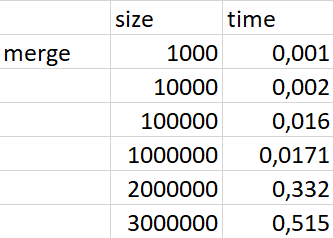


Рисунок 7

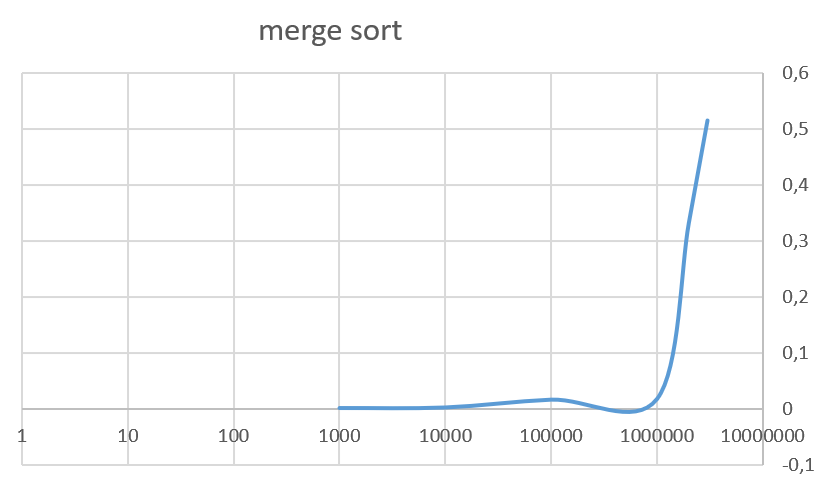


Рисунок 8

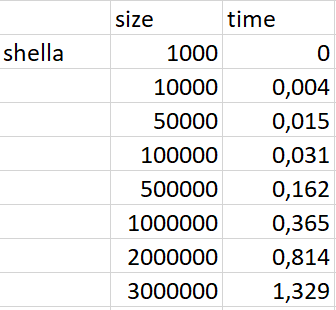


Рисунок 9

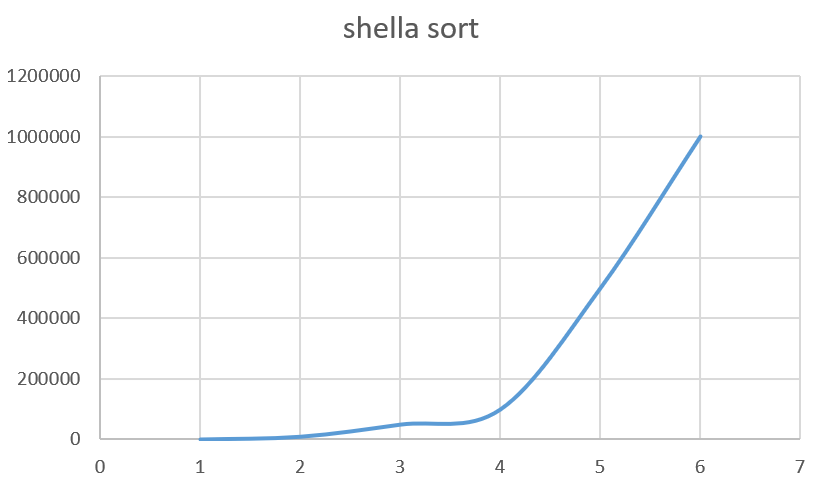


Рисунок 10

# Заключение

В ходе выполнения эксперимента были сделаны следующие выводы:

1. При работе с небольшими n наиболее рационально использовать несложные алгоритмы, которые имеют временную сложность работы O(n).
2. Наиболее быстрой для больших значений n является поразрядная сортировка, которая имеет временную сложность О(n). Данный алгоритм требует дополнительных действий для получения результата несмотря на линейную сложность в связи с тем, что сортировка работает поразрядно и у отрицательных элементов последний разряд имеет большее значение, в отличии от положительных.

# Список литературы

1. Дональд Кнут. Искусство программирования. Том 3. Сортировка и поиск, 2-е изд. Гл. 5.2.1. ISBN 5-8459-0082-4
2. <https://intellect.icu/sortirovka-v-yazyke-si-4485>
3. <https://training.epam.com/ru/blog/449>
4. <https://education.yandex.ru/journal/osnovnye-vidy-sortirovok-i-primery-ikh-realizatsii>
5. Учебное пособие Е.К. Липачѐв. Технология программирования. Методы сортировки данных.
6. Брайан Керниган, Деннис Ритчи. Язык программирования Си. — Москва: Финансы и статистика, 1992. — 272 с. — ISBN 5-279-00473-1.
7. Брайан Керниган, Деннис Ритчи. Язык программирования Си. — Санкт-Петербург: Невский диалект, 2000. — 352 с. — (Библиотека программиста). — ISBN 5-7940-0045-7.
8. Брайан Керниган, Деннис Ритчи. Язык программирования C. — Москва: Вильямс, 2015. — 304 с. — ISBN 978-5-8459-1975-5.

# Приложение:

1. **Сортировка вставками:**

void insert\_sort(double\* mas, int size) {

int j, i;

double tmp;

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

tmp = mas[i];

for (j = i - 1; j >= 0 && mas[j] > tmp; j--) {

mas[j + 1] = mas[j];

mas[j] = tmp;

}

}

}

1. **Сортировка Шелла:**

void shell\_sort(double\* mas, int size)

{

int i, j, step;

double tmp;

for (step = size / 2; step > 0; step /= 2) {

for (i = step; i < size; i++) {

for (j = i; j >= step && mas[j] < mas[j - step]; j -= step) {

tmp = mas[j];

mas[j] = mas[j - step];

mas[j - step] = tmp;

}

}

}

}

1. **Сортировка слиянием:**

void mergeSort(double\* arr, double\* arr\_new, int left, int right) {

int middle = (left + right) / 2;

if (right - left == 1) return;

mergeSort(arr, arr\_new, left, middle);

mergeSort(arr, arr\_new, middle, right);

merge(arr, arr\_new, left, middle, right);}

void merge(double\* arr, double\* arr\_new, int left, int middle, int right) {

double\* p1 = &arr[left], \* p2 = &arr[middle], \* p3 = &arr\_new[left];

while ((p1 < &arr[middle]) && (p2 < &arr[right])) {

if (\*p1 <= \*p2) {

\*p3 = \*p1;

p3++;

p1++;

}

else {

\*p3 = \*p2;

p3++;

p2++;

}

}

while (p1 < &arr[middle]) {

\*p3 = \*p1;

p3++;

p1++;

}

while (p2 < &arr[right]) {

\*p3 = \*p2;

p3++;

p2++;

}

for (int i = left; i < right; i++) arr[i] = arr\_new[i];

}

int cmp(const void\* a, const void\* b) {

return \*(double\*)a - \*(double\*)b;

}

1. **Поразрядная сортировка:**

void radix\_sort(double arr[], int n, int SIZE) {

double\* arr\_new = (double\*)malloc(SIZE \* 8);

unsigned char\* r0, \* r1;

int num[257];

int r, k, t, tmp, i;

r0 = (unsigned char\*)arr;

r1 = (unsigned char\*)arr\_new;

r = 0;

for (k = 0; k < sizeof(double); k++) /\*заполнение стартового массива\*/

{

for (i = 0; i < 256; num[i++] = 0);

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

if (r == 0) num[r0[8 \* i + k]]++;/\*считает колличество чисел с одним разрядом \*/

/\* k-ый байт i элемента в изначальном первом массиве, идет по каждому элементу\*/

else num[r1[8 \* i + k]]++;

t = 0;/\*заполнение с 1-го элемента массива\*/

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

{

tmp = num[i];

num[i] = t;

t += tmp;

}

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

if (r == 0) arr\_new[num[r0[8 \* i + k]]++] = arr[i];

else arr[num[r1[8 \* i + k]]++] = arr\_new[i];

r = 1 - r;

}

r = 0;

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

if (arr[i] < 0) {

r++;

}

else {

break;

}

}

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

arr\_new[i] = arr[n - i - 1];

}

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

arr\_new[i] = arr[i - r];

}

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

arr[i] = arr\_new[i];

}

free(arr\_new);}