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

Кусайкин Д.О.

**Проверил**:

преподаватель каф. МОСТ,

Волокитин В.Д.

Нижний Новгород

2022

**Содержание**

[Постановка задачи 3](#_heading=h.gjdgxs)

[Метод решения 4](#_heading=h.30j0zll)

[Руководство пользователя 5](#_heading=h.1fob9te)

[Описание программной реализации 6](#_heading=h.3znysh7)

[Подтверждение корректности 7](#_heading=h.tyjcwt)

[Результаты экспериментов 8](#_heading=h.3dy6vkm)

[Заключение 9](#_heading=h.1t3h5sf)

[Приложение 10](#_heading=h.4d34og8)

# Постановка задачи

Цель – изучение алгоритмов сортировки, работающих за сложность O(n**²**), O(nlog n), O(n).

Задачи, необходимые для достижения цели:

1. Изучение алгоритмов упорядочивания массивов с числами с плавающей запятой
2. Написание алгоритмов сортировки
3. Написание пользовательского интерфейса
4. Сравнить скорость и эффективность этих сортировок, корректность работы
5. Проверить соответствие теоретической оценке выполнения

# Метод решения

Выбраны следующие виды сортировок:

1. Сортировка выбором
2. Сортировка Хоара
3. Поразрядная сортировка

Далее подробное описание каждой из сортировок:

Сортировка выбором:

Идея данной сортировки состоит в делении массива на отсортированную (изначально пустую) и неотсортированную часть и нахождении минимального (в случае сортировки по возрастанию) элемента в неотсортированной части N-1 раз, где N – длина массива. После нахождения этот элемент помещается в отсортированную часть справа от остальных (путём swap нужного элемента с первым элементом правой части) и поэтому не рассматривается при последующем выборе минимального. Поиск минимального N-1 необязателен, но при поиске N раз на N итерации в правой части останется лишь один элемент – максимальный среди всех остальных. Следовательно его необязательно помещать в отсортированную часть, а значит массив отсортирован.

Сортировка Хоара (Quick Sort):

Основная идея заключается в том, что мы выбираем ведущий элемент, слева от которого элементы меньше него, а справа больше или равные (может быть наоборот, если необходима сортировка по убыванию). Затем в каждой половине аналогично выбирается ведущий элемент, а справа и слева от него оказываются элементы меньше и больше него соответственно. Одной из самых простых реализаций является рекурсивная, потому что каждый раз мы выполняем упомянутое выше действие для разных подмассивов. Таким образом каждый раз ведущий элемент оказывается на своём месте (кол-во элементов до него и после соответствует необходимому в отсортированном), а так как количество элементов в массивах конечно – мы гарантированно придём к верному результату (отсортированному массиву). Очевидно, что из-за своей реализации данная сортировка будет плохо работать на уже отсортированных массивах (независимо от порядка элементов всё равно будет выбирается элемент). С идеей работы и связана его сложность, так как мы рассматриваем разделения на 2 части, а рассматриваем мы все элементы – выходит O(n).

Линейная сортировка (LSD):

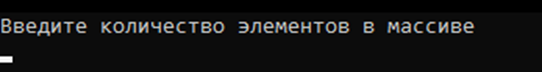
У данной сортировки сложность . Это происходит из-за того, что мы не сравниваем числа, а используем их байты. Мы делим число на 8 частей (по 1 байту каждый), проходимся поочередно по каждому байту (от конца до начала) и каждый раз используем побитовую сортировку. Таким образом достигается максимально возможная скорость сортировки.

Немного про побитовую сортировку.

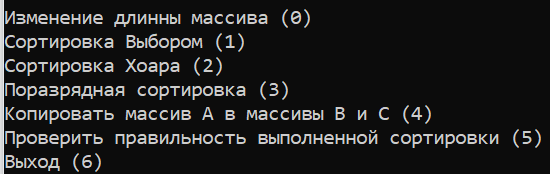
Она представляет собой массив из 256 элементов, в котором лежит количество чисел, с данными байтами. Далее мы расставляем их в массиве так, чтобы байты в массиве соответствовали их числам.

# Руководство пользователя

При запуске программы, пользователя попросят ввести количество элементов массива (для того, чтобы пользователь не смог запустить сортировку на пустом массиве).



Далее пользователь вводит n и начинается основной код программы:



В данном меню находится 5 команд:

1. Изменение длины массива (генерируется массив псевдослучайных чисел этой длины)
2. Сортировка Выбором
3. Сортировка Хоара
4. Поразрядная сортировка
5. Копировать массив А в массивы В и С (для проверки сортировок на идентичных массивах)
6. Проверить правильность выполненной сортировки(собственная сортировка сравнивается со встроенной)
7. Выход

При выборе любой сортировки вернётся меню, а также время выполнения сортировки.

# Описание программной реализации

Программа состоит из единственного файла (zachet.c)

В которой и описаны все функции (12 шт)

int compare(const void\* a, const void\* b) – принимает указатели на 2 числа, необходима для работы qsort из stdlib (проверки собственной сортировки) – возвращает необходимые для сортировки значения

int check\_mas(double a\*, double b\*, int dlin) – принимает указатели на начало отсортированного и изначального массивов, а также длину, проверяет корректно ли сработала моя сортировка.

void qs(double\* mas, int st, int fn) – принимает указатель на начало массива, левую и правую границы отрезков, необходимые для рекурсии в сортировке Хоара.

void input(double\* mas, int dlin) - принимает указатель на начало массива, а также размер – заполняет псевдослучайными числами

void output(double\* mas, int dlin)– принимает указатель на начало массива, размер, выводит элементы массива на экран

void seal(char\*\* menu) – принимает указатель на начало массива с элементами меню, а также их кол-во – служит для вывода меню

void choice(double\* mas, int dlin)- принимает указатель на начало массива, а также размер.

void copy(double\* mas1, double\* mas2, int dlin) -принимает указатели на начала двух массивов а также размер

void reverse\_mas(double\* mas, long long n, long long k) - принимает указатель на начало массива, а также размер и количество отрицательных чисел, данная функция отвечает за переворот массива.

void radix\_sort(double\* mas, double\* out, long long byte, long long n) - принимает указатели на начала двух массивов(исходного и который пойдет в ответ), размер, номер байта, по которому сортируем.

void LSD(double\* mas, long long dlin, long long k) - принимает указатель на начало массива, а также размер и количество отрицательных чисел.

int main() – основная функция, отвечает за корректную работу, использование подпрограмм, взаимодействие с пользователем

# Подтверждение корректности

Для подтверждения корректности в программе была реализована функция check\_mas, 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 мной и этой функцией массив. Если всё корректно – программа выведет результат моей сортировки, иначе выведет ошибку, что отсортировать массив не удалось.

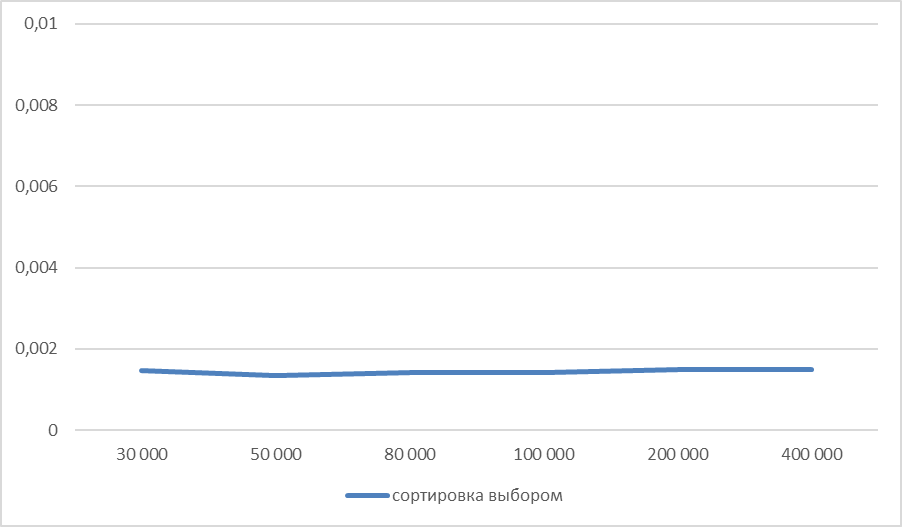
# Результаты экспериментов

По данным экспериментов видно, что сортировка выбором показала самый плохой результат. Быстрей работает сортировка Хоара, начиная с 5 миллиона элементов она начала подниматься, а лучшее время показа поразрядная сортировка, которая начала увеличивать время выполнения лишь при 10 миллионах элементов.



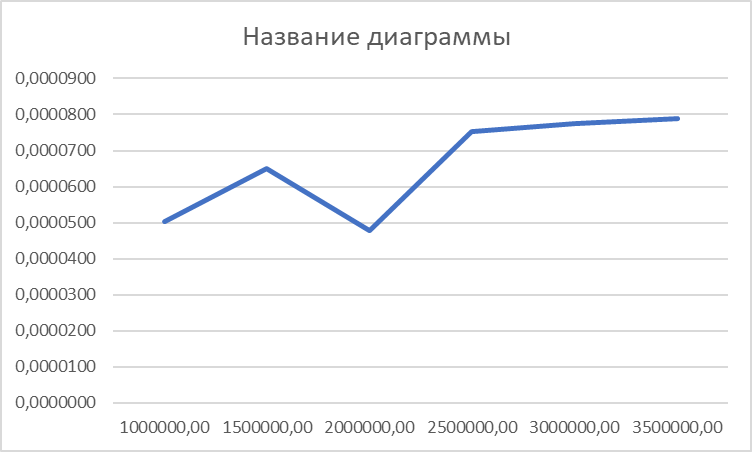
Для того, чтобы подтвердить сложность алгоритма необходимо доказать, что константа сложности (которую мы отбрасываем при определении алгоритма) именно константа по сравнению с самой сложностью. Для этого необходимо разделить время, за которое обработалось данное количество элементов массива, на сложность самого алгоритма с подстановкой самого n.

Сортировка выбором. У данной сортировки сложность n2. Докажем это приведенным выше способом.



Видно, что график вышел на константу – значит функция соответствует прогнозируемой сложности

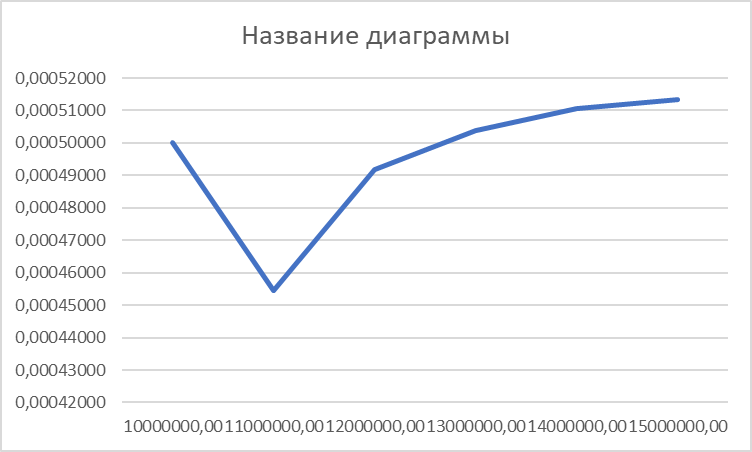
Сортировка Хоара



Возьмём основание логарифма за 2, в таком случае график сходится к константе, значит сложность так же определена верно

Поразрядная сортировка.

У данной сортировки сложность n .



Тоже в итоге выходит на константу, а значит соответствует поставленной сложности.

# Заключение

В результате работы была реализована сортировка для чисел, с плавающей точкой, подробно разобраны такие виды, как сортировка Выбором, сортировка Хоара и поразрядная сортировка, для работы с которыми был реализован пользовательский интерфейс. Также была сравнена скорость и эффективность работы этих сортировок. Сортировка Выбором – наименее эффективна на больших массивах, далее идёт сортировка Хоара, которая имеет среднюю эффективность из этих 3-х сортировок, самой лучшей же оказалась поразрядная сортировка.

# Приложение

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

#include <locale.h>

#include <time.h>

#include <string.h>

#define MENU\_SIZE 7

#define EPSILON 2.220446e-16

char\* menu[MENU\_SIZE] = { "Изменение длинны массива (0)", "Сортировка Выбором (1)", "Сортировка Хоара (2)", "Поразрядная сортировка (3)", "Копировать массив А в массивы В и С (4)", "Проверить правильность выполненной сортировки (5)", "Выход (6)"};

void seal(char\*\* menu)

{

printf\_s("\n\n");

for (int i = 0; i < MENU\_SIZE; i++)

{

printf\_s("%s\n", menu[i]);

}

}

void input(double\* mas, int dlin)

{

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

{

mas[i] = rand() / (rand() / 100.0) - rand() / (rand() / 100.0);

}

}

void output(double\* mas, int dlin)

{

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

{

printf("%lf\n", mas[i]);

}

printf("\n");

}

void choice(double\* mas,int dlin)

{

for (int i = 0; i < dlin - 1; i++)

{

double peremen = mas[i];

double temp = peremen;

int nomer = i;

for (int j = i + 1; j < dlin; j++)

{

if (mas[j]<peremen)

{

peremen = mas[j];

nomer = j;

}

}

mas[i] = peremen;

mas[nomer] = temp;

}

}

/\*void bubble(double\* mas, int dlin)

{

for (int i = 0; i < dlin - 1; i++)

{

int flag = 1;

for (int j = 0; j < dlin - i - 1; j++)

{

if (mas[j+1]<mas[j])

{

double temp = mas[j];

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

mas[j + 1] = temp;

flag = 0;

}

}

if (flag)

{

break;

}

}

}\*/

void copy(double\* mas1, double\* mas2, int dlin)

{

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

{

mas2[i] = mas1[i];

}

}

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

{

if (\*(double\*)a - \*(double\*)b < 0)

return -1;

else if (fabs(\*(double\*)a - \*(double\*)b) < EPSILON)

return 0;

else

return 1;

}

int check\_mas(double\* mas1, double\* mas2, int dlin)

{

qsort(mas1, dlin, sizeof(double), compare);

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

if (mas1[i]!=mas2[i])

return 0;

return 1;

}

void qs(double\* mas, int st, int fn)

{

int i = st, j = fn;

double t, x = mas[(st + fn) / 2];

do {

while (mas[i] < x) i++;

while (mas[j] > x) j--;

if (i <= j) {

if (i < j) {

t = mas[i];

mas[i] = mas[j];

mas[j] = t;

}

i++;

j--;

}

} while (i <= j);

if (i < fn) qs(mas, i, fn);

if (st < j) qs(mas, st, j);

}

void reverse\_mas(double\* mas, long long n, long long k)

{

for (int i = 0; i < (n - k) / 2; ++i)

{

double temp = mas[i];

mas[i] = mas[(n - k) - i - 1];

mas[(n - k) - i - 1] = temp;

}

}

void radix\_sort(double\* mas, double\* out, long long byte, long long n)

{

unsigned char\* masc = (unsigned char\*)mas;

long long counter[256];

long long tmp;

long long i;

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

for (i = 0; i < n; ++i) counter[masc[8 \* i + byte]]++;

tmp = counter[0];

counter[0] = 0;

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

{

long long b = counter[i];

counter[i] = tmp;

tmp += b;

}

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

{

out[counter[masc[8 \* i + byte]]] = mas[i];

counter[masc[8 \* i + byte]]++;

}

}

void LSD(double\* mas, long long n, long long k)

{

long long i;

double\* out = (double\*)malloc(n \* sizeof(double));

for (i = 0; i < 8; ++i)

{

if (i % 2 == 0)

{

radix\_sort(mas, out, i, n);

}

else

{

radix\_sort(out, mas, i, n);

}

}

reverse\_mas(mas, n, k);

reverse\_mas(mas, n, 0);

}

void minus(double\* mas, int dlin, int\* k)

{

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

{

if (mas[i] < 0)

{

(\*k)++;

}

}

}

int main()

{

setlocale(LC\_CTYPE, "Rus");

srand(time(NULL));

int dlin, knopki, flag = 1;

printf\_s("Введите длину массива:\n");

scanf\_s("%d", &dlin);

double\* mas\_a = (double\*)malloc(dlin \* sizeof(double));

double\* mas\_b = (double\*)malloc(dlin \* sizeof(double));

double\* mas\_c = (double\*)malloc(dlin \* sizeof(double));

input(mas\_a, dlin);

copy(mas\_a, mas\_b, dlin);

copy(mas\_a, mas\_c, dlin);

seal(menu);

while (flag != 0)

{

scanf\_s("%d", &knopki);

switch (knopki)

{

case 0: {printf\_s("Введите длину массива: \n");

scanf\_s("%d", &dlin);

mas\_a = (double\*)malloc(dlin \* sizeof(double));

mas\_b = (double\*)malloc(dlin \* sizeof(double));

mas\_c = (double\*)malloc(dlin \* sizeof(double));

input(mas\_a, dlin);

copy(mas\_a, mas\_b, dlin);

copy(mas\_a, mas\_c, dlin);

seal(menu);

break; }

/\*case 1: {long long start, finish; start = clock();

bubble(mas\_b, dlin);

finish = clock();

printf\_s("\nВремя сортировки Пузырьком: %.40lf\n", (finish - start) \* 1.0 / CLOCKS\_PER\_SEC);

seal(menu);

break; }\*/

case 1: {long long start, finish;

start = clock();

choice(mas\_b, dlin);

finish = clock();

printf\_s("\nВремя сортировки Выбором: %.40lf\n", (finish - start) \* 1.0 / CLOCKS\_PER\_SEC);

seal(menu);

break; }

case 2: {long long start, finish;

start = clock();

qs(mas\_b, 0, dlin-1);

finish = clock();

printf\_s("\nВремя сортировки Хоара: %.40lf\n", (finish - start) \* 1.0 / CLOCKS\_PER\_SEC);

seal(menu);

break; }

case 3: {int k = 0;

minus(mas\_a, dlin, &k);

long long start, finish; start = clock();

LSD(mas\_b, dlin, k);

finish = clock();

printf\_s("\nBремя сортировки Поразрядной: %.40lf\n", (finish - start) \* 1.0 / CLOCKS\_PER\_SEC);

seal(menu);

break; }

case 4: {copy(mas\_a, mas\_b, dlin);

copy(mas\_a, mas\_c, dlin);

seal(menu);

break; }

case 5: {if (check\_mas(mas\_c, mas\_b, dlin))

{

printf\_s("Сортировка выполнена верно\n");

}

else

{

printf\_s("Сортировка выполнена не верно\n");

}

seal(menu);

break; }

case 6: {flag = 0;

break; }

case 9: {output(mas\_b, dlin);

seal(menu);

break; }

}

}

}