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 3822б1пм1  
Вершинина Анастасия Максимовна

:

**Проверил:**

преподаватель каф. МОСТ,

Волокитин В.Д.

Нижний Новгород

2022

Оглавление

[Введение 3](#_Toc122689677)

[1. Постановка цели и задач 3](#_Toc122689678)

[2. Метод решения 4](#_Toc122689679)

[Сортировка вставками (insert sort) 4](#_Toc122689680)

[Сортировка выбором (selection sort) 4](#_Toc122689681)

[Сортировка Хоара (quick sort) 4](#_Toc122689682)

[Сортировка Шелла (shell sort) 4](#_Toc122689683)

[3. Руководство пользователя 5](#_Toc122689684)

[4. Описание программной реализации 6](#_Toc122689685)

[Функции интерфейса 6](#_Toc122689686)

[Функции сортировки 7](#_Toc122689687)

[Сортировка вставками 7](#_Toc122689688)

[Сортировка выбором 7](#_Toc122689689)

[Сортировка Хоара 8](#_Toc122689690)

[Сортировка Шелла 9](#_Toc122689691)

[Дополнительные функции 9](#_Toc122689692)

[Основная программа 10](#_Toc122689693)

[5. Подтверждение корректности 13](#_Toc122689694)

[6. Результаты экспериментов 13](#_Toc122689695)

[Сортировка вставками 13](#_Toc122689696)

[Сортировка выбором 15](#_Toc122689697)

[Сортировка Хоара 17](#_Toc122689698)

[Сортировка Шелла 19](#_Toc122689699)

[Заключение 22](#_Toc122689700)

# Введение

В большинстве программных проектов возникает необходимость обработки большого числа единообразно организованных данных. В таких случаях подобные данные удобно обрабатывать как единое целое, для этого они представляются в виде массива - именованной последовательности данных одного типа. Массив является удобным способом хранения нескольких связанных элементов данных в едином контейнере для большего удобства и эффективности программирования. Массив позволяет сохранять и манипулировать многими элементами данных посредством единственной переменной.

В связи с этим возникает потребность в эффективной работе с ним. Для этого массив сортируют. Видов сортировки очень много. При ее выборе рассматривают три основных критерия:  
1) средняя скорость сортировки  
2) скорость в наилучшем и наихудшем случаях  
3) естественность поведения

С отсортированными данными удобнее и быстрее работать, поэтому выбор наиболее подходящей сортировки позволит улучшить эффективность работы программы.

# Постановка цели и задач

Цель:  
реализация на языке Си сортировки массива вставками (insert sort), выбором (selection sort), сортировки Хоара (quick sort) и сортировки Шелла(shell sort) с наглядным подтверждением корректности выполнения алгоритмов.

Задачи:

1. Реализовать сортировки.
2. Описать программную реализацию и алгоритмы работы сортировок.
3. Проверить корректность выполнения алгоритмов сортировок.
4. Произвести тесты над программой для выявления временной сложности работы алгоритмов, для контроля корректности полученные данные представить в виде графиков.
5. По полученным результатам сделать вывод.

# Метод решения

## Сортировка вставками (insert sort)

Сортировка вставками является одним из основных алгоритмов сортировки. На первом шаге выполняется сортировка двух элементов. Далее алгоритм ставит третий элемент в порядковую позицию, соответствующую его положению относительно первых двух элементов, затем четвертый, пятый и так далее.

## Сортировка выбором (selection sort)

При сортировке выбором алгоритм находит наименьшее из N чисел (где N- число элементов в массиве) и ставит его на первое место. Затем поиск наименьшего происходит среди N-1 чисел (за исключением первого), который ставится уже на второе место. И так далее. Внешний цикл алгоритма осуществляется (N-1) раз, а внутренний - раз.

## Сортировка Хоара (quick sort)

Сортировка Хоара (или quick sort) является улучшенным вариантов сортировки “Пузырьком”. В результате некоторых модификаций из самого неэффективного алгоритма сортировки получился один из самых эффективных, оба построены на базе метода перестановок.   
Сначала выбирают опорный элемент, обычно им назначается элемент в середине массива. От выбора опорного элемента не зависит корректность алгоритма, но в отдельных случаях может сильно зависеть его эффективность. Основная идея этой сортировки заключается в разбиении всего массива на две группы: в одну группу входят опорный элемент и элементы, большие его, в другую- элементы, меньшие опорного. Затем рекурсивно повторяют данный алгоритм с «левым» и «правым» подмассивами до тех пор, пока длина отрезков не станет меньше единицы.

## Сортировка Шелла (shell sort)

Суть сортировки Шелла заключается в сравнении разделенных на группы элементов массива, находящихся на некотором расстоянии друг от друга. Изначально Дональд Шелл принимал это расстояние за N/2, где N – число элементов в массиве. В каждой такой группе происходила сортировка элементов. Затем расстояние сокращалось еще в два раза. Теперь групп стало меньше, а количество элементов в них больше. Снова происходила сортировка. И так далее. Постепенно расстояние уменьшается до тех пор, пока не станет равным 1,осуществляется последний проход сортировки.

# Руководство пользователя

При запуске программы пользователь должен ввести с клавиатуры числа 1 или 2.

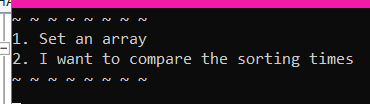


Рисунок 1 – вывод первого меню в консоли.

При вводе 1 пользователю далее предстоит ввести число элементов в массиве.

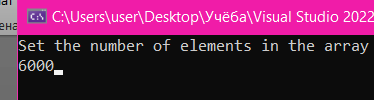


Рисунок 2 – ввод числа элементов в массиве.

Затем необходимо выбрать сортировку.

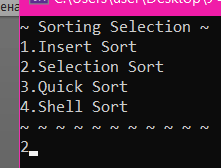


Рисунок 3 – выбор сортировки.

В данном случае программа выведет сообщение о том, что осуществлена успешная проверка правильности сортировки массива и время выполнения выбранного алгоритма сортировки.



Рисунок 4– результат работы программы.

При выборе 2 пункта (рисунок 1) пользователю также будет необходимо ввести число элементов в массиве (рисунок 2), но вывод программы будет отличаться.

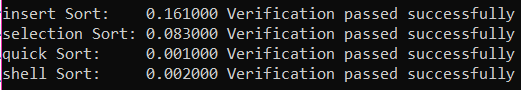
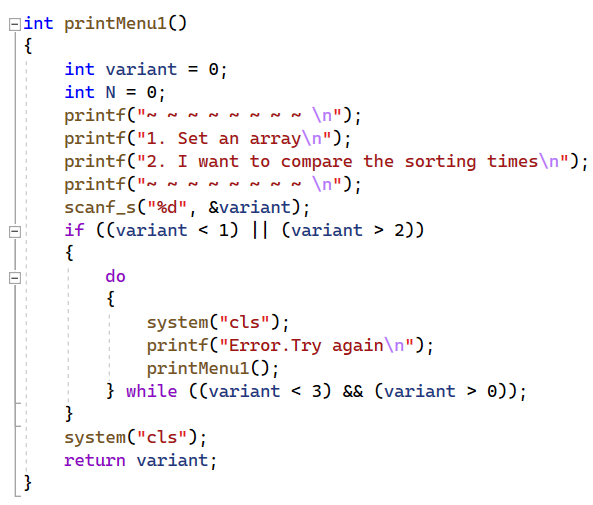


Рисунок 5 – вывод программы при вводе 2, 10000.

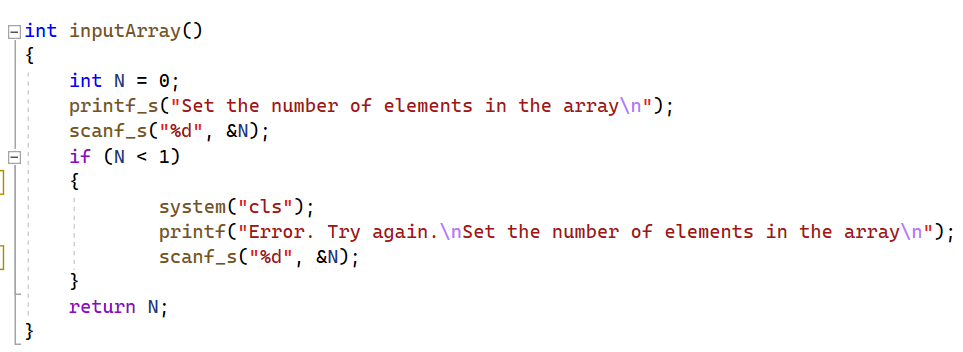
# Описание программной реализации

## Функции интерфейса

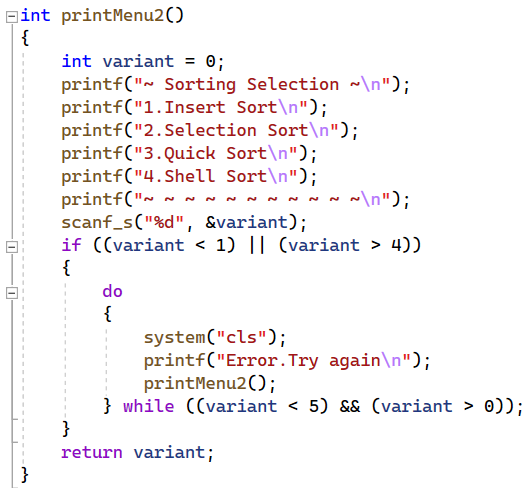
Функция printMenu1() печатает пункты выбора действия для пользователя и возвращает номер выбранного пункта. В случае выхода из допустимого диапазона пунктов предложение ввода повторяется.



Функция inputArrat() возвращает введенное пользователем число элементов в массиве.

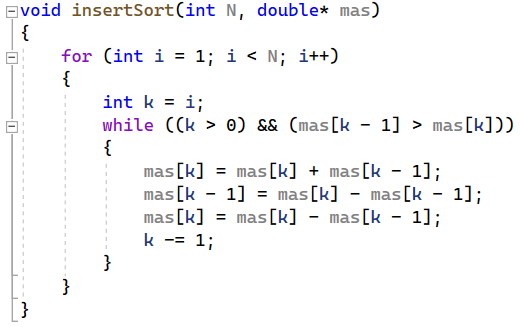


Функция printMenu2() предоставляет пользователю выбор метода сортировки. Возвращает (без она) целое значение выбранного пункта меню.



## Функции сортировки

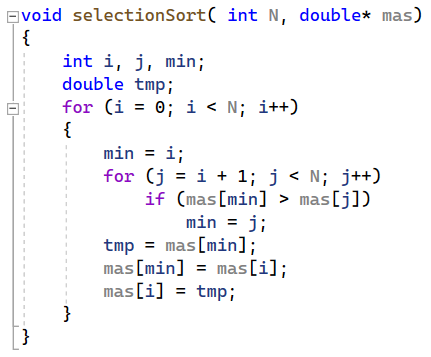
### Сортировка вставками

Функция void insertSort(int N, double\* mas) принимает на вход число элементов в массиве и указатель на начало массива

Во внутреннем цикле while происходит смена мест для двух значений элементов массива без использования третьей переменной.

### Сортировка выбором

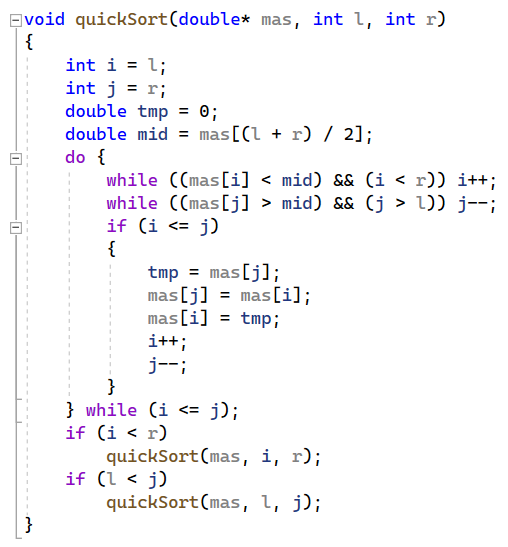
Функция void selectionSort( int N, double\* mas) принимает на вход число элементов в массиве и указатель на начало массива



В цикле for происходит смена мест двух элементов массива с использованием третьей переменной.

### Сортировка Хоара

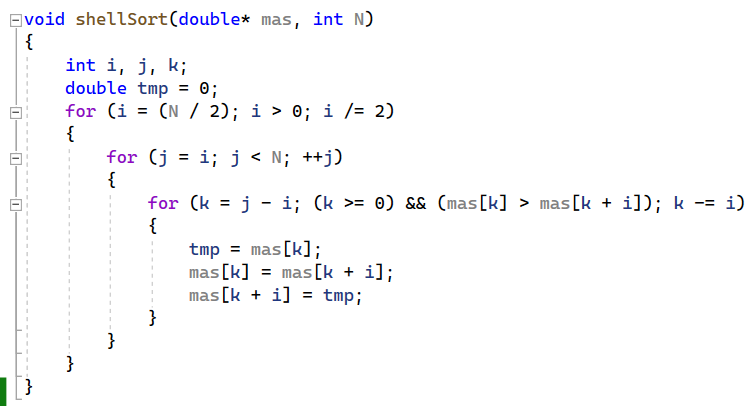
Функция (quick sort) void quickSort(double\* mas, int l, int r) принимает на вход указатель на начало массива, номер первого, «левого» сортируемого элемента, и номер последнего, «правого» сортируемого элемента.



Опорному элементу присваивается значение массива от «серединного элемента». Счетчикам i и j присваивается значение l и r соответственно. В цикле do while происходит распределение всех элементов массива на две группы: в первой элементы больше и равные опорному, во второй - меньшие опорного. Этот цикл работает до тех пор, пока счетчики i и j не станут равными. Далее происходит рекурсивный вызов функции. Только теперь алгоритм Шелла работает на отрезках от i до r и от l до j.

## Сортировка Шелла

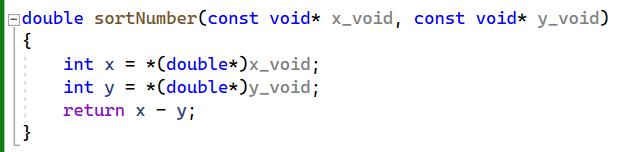
Функция void shellSort(double\* mas, int N) принимает на вход указатель на начало массива и количество элементов в массиве.



Во втором внутреннем цикле происходит замена двух элементов массива с помощью третьей переменной tmp.

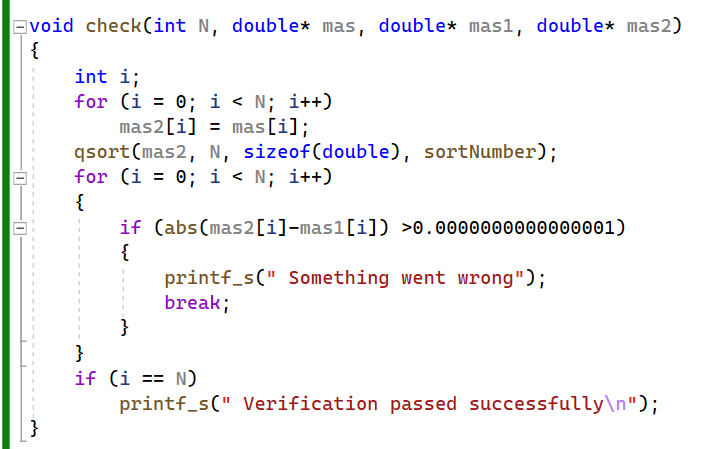
### Дополнительные функции

Функция double sortNumber(const void\* x\_void, const void\* y\_void), принимает на вход два константных указателя типа void. Функция sortNumber необходима для принятия на вход библиотечной функцией qsort (содержащейся в библиотеке <stdlib.h>) в качестве третьего аргумента.



В теле функции пустые указатели void\* x\_void и void\* y\_void преобразуется в указатель типа double.

Функция void check(int N, double\* mas, double\* mas1, double\* mas2) принимает на вход количество элементов в массиве и три указателя на массив. Первый массив – исходный, второй – сортируется собственными алгоритмами сортировки, третий – копия первого, который сортируется библиотечной функцией qsort.



Данная функция проверяет разность двух элементов массива типа double на допустимую погрешность (а именно

## Основная программа

int main()

{

int N = 1;

int i;

double\* mas;

double\* mas1;

double\* mas2;

int variant = 0;

double startTime = 0;

double endTime = 0;

int f = 1;

variant = printMenu1();

switch (variant)

{

case 1:

{

f = 1;

break;

}

case 2:

{

f = 0;

N=inputArray();

mas = (double\*)malloc(N \* sizeof(double));

mas1 = (double\*)malloc(N \* sizeof(double));

mas2 = (double\*)malloc(N \* sizeof(double));

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

mas[i] = ((double)rand() / (double)RAND\_MAX);

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

mas1[i] = mas[i];

startTime = clock();

insertSort(N, mas1);

endTime = clock();

printf\_s("insert Sort: ");

printf("%f", ((double)(endTime - startTime) / CLOCKS\_PER\_SEC));

check(N, mas, mas1, mas2);

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

mas1[i] = mas[i];

startTime = clock();

selectionSort(N, mas1);

endTime = clock();

printf\_s("selection Sort: ");

printf("%f", ((double)(endTime - startTime) / CLOCKS\_PER\_SEC));

check(N, mas, mas1, mas2);

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

mas1[i] = mas[i];

startTime = clock();

quickSort(mas1, 0, N - 1);

endTime = clock();

printf\_s("quick Sort: ");

printf("%f", ((double)(endTime - startTime) / CLOCKS\_PER\_SEC));

check(N, mas, mas1, mas2);

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

mas1[i] = mas[i];

startTime = clock();

shellSort(mas1, N);

endTime = clock();

printf\_s("shell Sort: ");

printf("%f", ((double)(endTime - startTime) / CLOCKS\_PER\_SEC));

check(N, mas, mas1, mas2);

free(mas);

free(mas1);

free(mas2);

break;

}

}

if (f == 1)

{

N = inputArray();

mas = (double\*)malloc(N \* sizeof(double));

mas1 = (double\*)malloc(N \* sizeof(double));

mas2 = (double\*)malloc(N \* sizeof(double));

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

mas[i] = ((double)rand() / (double)RAND\_MAX);

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

mas1[i] = mas[i];

system("cls");

variant=printMenu2();

switch (variant)

{

case 1:

{

startTime = clock();

insertSort(N, mas1);

endTime = clock();

check(N, mas, mas1, mas2);

break;

}

case 2:

{

startTime = clock();

selectionSort(N, mas1);

endTime = clock();

check(N, mas, mas1, mas2);

break;

}

case 3:

startTime = clock();

quickSort(mas1, 0, N - 1);

endTime = clock();

check(N, mas, mas1, mas2);

break;

case 4:

startTime = clock();

shellSort(mas1, N);

endTime = clock();

check(N, mas, mas1, mas2);

break;

}

printf("\n");

printf("%f", ((double)(endTime - startTime) / CLOCKS\_PER\_SEC));

free(mas);

free(mas1);

free(mas2);

}

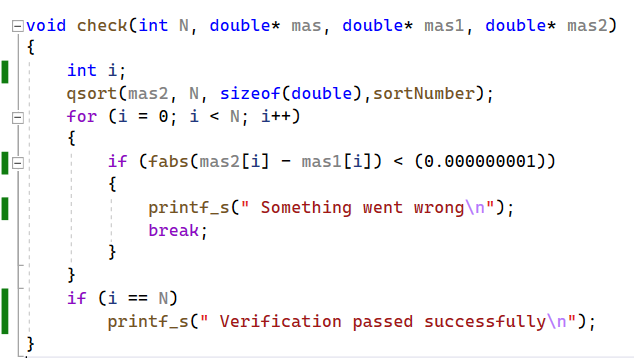
return 0;

}

# Подтверждение корректности

Для подтверждения корректности поэлементно сравниваются массивы: отсортированные собственными алгоритмами (mas1) и отсортированные с помощью библиотечной функции qsort (mas2).

Для этого массив mas перед каждой сортировкой копируется в массив mas2. Затем mas1 сортируется определенным алгоритмом, а mas2 сортируется стандартной функцией qsort. Затем массивы поэлементно сравниваются. А именно: если абсолютная разница между элементами двух массивов меньше 0.000000001, то выводится сообщение об ошибке и программа прекращает работу. Если этого не произошло, то выводится сообщение об успешно пройденной проверке.



# Результаты экспериментов

## Сортировка вставками

Количество сравнений в этом алгоритме зависит от исходной упорядоченности массива.  
Если массив отсортирован в нужном порядке, то количество сравнений равно , где N – число элементов в массиве.   
Если массив не упорядочен, то количество сравнений равно )- в среднем случае и -в наихудшем случае.

Время работы сортировки зависит 1)от количества элементов в массиве, 2) от исходной упорядоченности массива.   
В лучшем случае временная сложность алгоритма: ;  
В среднем и худшем – ).

Рисунок 6 – график зависимости времени сортировки вставками от исходной упорядоченности массива (при каждом повторении сортировки массив «перезадается» с помощью датчика псевдослучайных чисел) и изначально заданного числа элементов N в нём (N=1000,N=2000,N=4000).

Рисунок 7 – график зависимости времени сортировки вставками от исходной упорядоченности массива (при каждом повторении сортировки массив перезадается с помощью датчика псевдослучайных чисел) и изначально заданного числа элементов N в нём (N=6000,N=8000,N=10000).

Рисунок 8 – объединенный график с рисунков 6 и 7.

## Сортировка выбором

Сортировка выбором требует сравнений.  
Количество перестановок:   
3(N-1) – в лучшем случае;   
- худшем.

Время работы сортировки зависит 1)от количества элементов в массиве, 2) от исходной упорядоченности массива.   
В лучшем случае временная сложность алгоритма:   
В среднем случае: , где y- константа Эйлера, равная 0,577216;  
В худшем случае: ).

Рисунок 9 – график зависимости времени сортировки выбором от исходной упорядоченности массива (при каждом повторении сортировки массив «перезадается» с помощью датчика псевдослучайных чисел) и изначально заданного числа элементов N в нём (N=1000,N=2000,N=4000).

Рисунок 10 – график зависимости времени сортировки выбором от исходной упорядоченности массива (при каждом повторении сортировки массив «перезадается» с помощью датчика псевдослучайных чисел) и изначально заданного числа элементов N в нём (N=6000,N=8000,N=10000).

Рисунок 11 – объединенный график с рисунков 9 и 10.

## Сортировка Хоара

В среднем случае выполняется сравнений.  
В среднем случае количество перестановок равно .  
Временная сложность алгоритма:  
в лучшем случае )  
в среднем случае )  
в худшем случае ).

Рисунок 12 – график зависимости времени сортировкой Хоара от исходной упорядоченности массива (при каждом повторении сортировки массив «перезадается» с помощью датчика псевдослучайных чисел) и изначально заданного числа элементов N в нём (N=5000,N=20000,N=30000).

Рисунок 13 – график зависимости времени сортировкой Хоара от исходной упорядоченности массива (при каждом повторении сортировки массив «перезадается» с помощью датчика псевдослучайных чисел) и изначально заданного числа элементов N в нём (N=50000,N=65000,N=80000).

Рисунок 14 – объединенный график с рисунков 12 и 13.

## Сортировка Шелла

Временная сложность алгоритма:  
В лучшем случае: ;  
В среднем случае:- но зависит от выбора длины шага.;  
В худшем случае: ).

Рисунок 15 – график зависимости времени сортировкой Шелла от исходной упорядоченности массива (при каждом повторении сортировки массив «перезадается» с помощью датчика псевдослучайных чисел) и изначально заданного числа элементов N в нём (N=5000,N=10000,N=15000).

Рисунок 16 – график зависимости времени сортировкой Шелла от исходной упорядоченности массива (при каждом повторении сортировки массив «перезадается» с помощью датчика псевдослучайных чисел) и изначально заданного числа элементов N в нём (N=20000,N=30000,N=40000).

Рисунок 17 – объединенный график с рисунков 15 и 16.

# Заключение

В ходе лабораторной работы были реализованы на языке Си алгоритмы сортировки вставками, выбором, сортировка Хоара и сортировка Шелла. Были описаны алгоритмы работы данных сортировок, их программная реализация и проведенные эксперименты для замера их временной сложности. В ходе экспериментов была проведена проверка корректности сортировок.