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**

**«Алгоритмы сортировки массивов»**

**Выполнила**:

студентка группы 382003-1

Замчалова К.В.

**Проверил**:

ассистент каф. МОСТ,

Волокитин В.Д.

Нижний Новгород

2020

**Содержание**

[Постановка задачи 3](#_Toc26962562)

[Метод решения 4](#_Toc26962563)

[Руководство пользователя 6](#_Toc26962564)

[Описание программной реализации 7](#_Toc26962565)

[Подтверждение корректности 8](#_Toc26962566)

[Результаты экспериментов 9](#_Toc26962567)

[Заключение 12](#_Toc26962568)

[Приложение 13](#_Toc26962569)

# Постановка задачи

Перед нами стоит задача отсортировать массив так, чтобы элементы следовали в порядке возрастания, т.е. на нулевой позиции должен стоять наименьший элемент, а все последующие элементы должны быть больше предыдущего.

# Метод решения

Для решения поставленной задачи будем использовать четыре метода сортировки массива: сортировка выбором, расчёской, слиянием и поразрядную сортировку.

**Алгоритм сортировки выбором:**

1) Находим номер минимального элемента в массиве

2) Меняем местами значения минимального элемента и элемента на нулевой позиции массива (или оставляем всё как есть, если минимальный элемент уже стоит на нулевой позиции)

3) Не обращая внимания на уже перемещённые в начало массива элементы, сортируем оставшийся массив таким же образом

**Алгоритм сортировки расчёской:**

1) Сравниваем два элемента на расстоянии размер массива минус один, меняем их местами, если элемент, находящийся ближе к началу массива, больше элемента, стоящего ближе к концу массива.

2) Пройдя массив с этим шагом, уменьшаем разрыв между сравниваемыми элементами путём деления его на так называемый фактор уменьшения, он примерно равен 1,247

3) Пройдя по массиву с новым шагом, вновь делим шаг на фактор уменьшения и ещё раз проходим по массиву, сравнивая и переставляя элементы

4) Продолжаем эту процедуру до тех пор, пока не будут сравниваться соседние элементы

**Алгоритм сортировки слиянием:**

1) Исходный массив делится на два равных.

2) Каждый из двух половинных массивов сортируется отдельно тем же алгоритмом деления (массив длины 1 – упорядоченный)

а) Берём меньший из первых двух элементов упорядоченных массивов и ставим его в начало нового массива. Счётчик элемента нового массива и массива, из которого был взят элемент, увеличиваем на один. Продолжаем слияние по этому же принципу.

б) Когда один подмассив закончится, записываем в выходной массив все оставшиеся элементы.

3) Выполняем слияние двух исходных половинчатых массивов.

**Алгоритм поразрядной сортировки:**

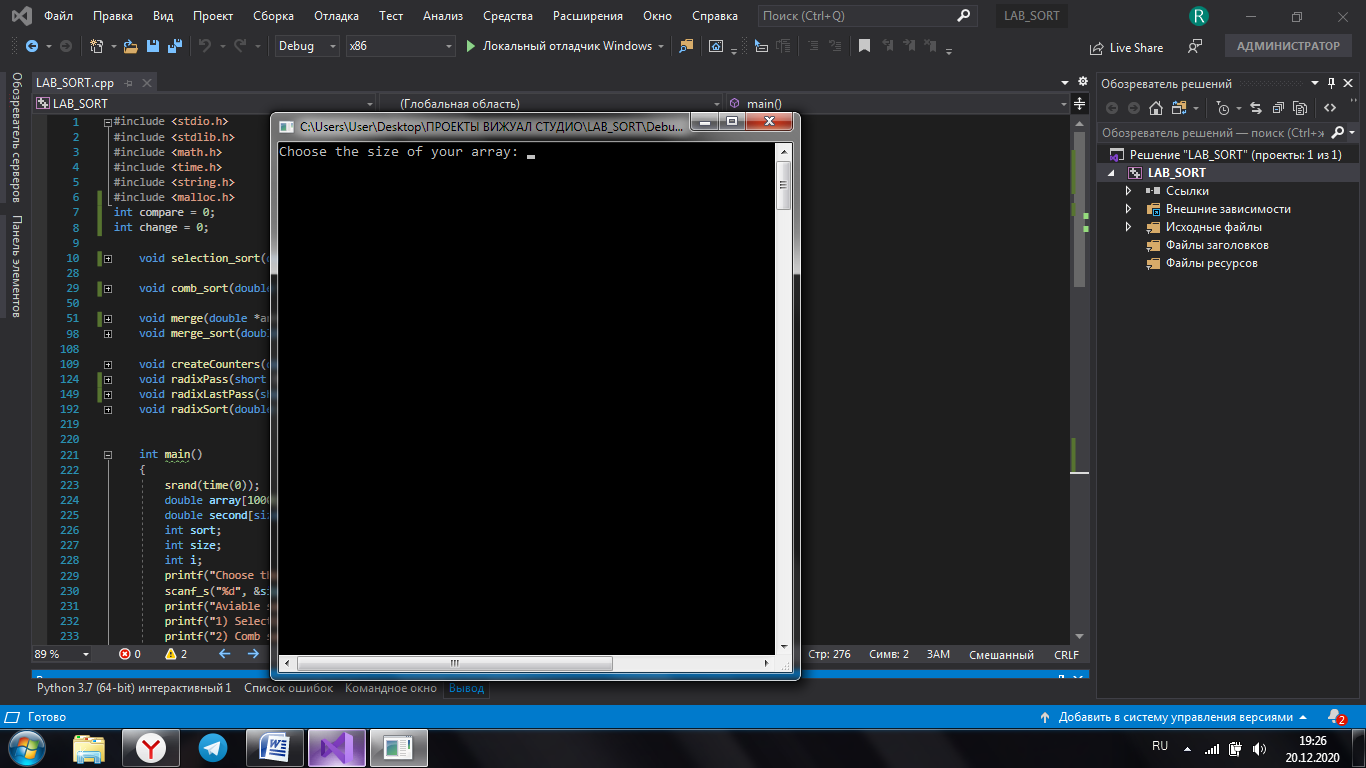
1) Сравнение производится поразрядно. Сначала сравниваются значения одного крайнего разряда, и элементы группируются по результатам этого сравнения

2) Затем сравниваются значения следующего разряда, соседнего, и элементы либо упорядочиваются по результатам сравнения значений этого разряда внутри образованных на предыдущем проходе групп, либо переупорядочиваются в целом, но сохраняя относительный порядок, достигнутый при предыдущей сортировке.

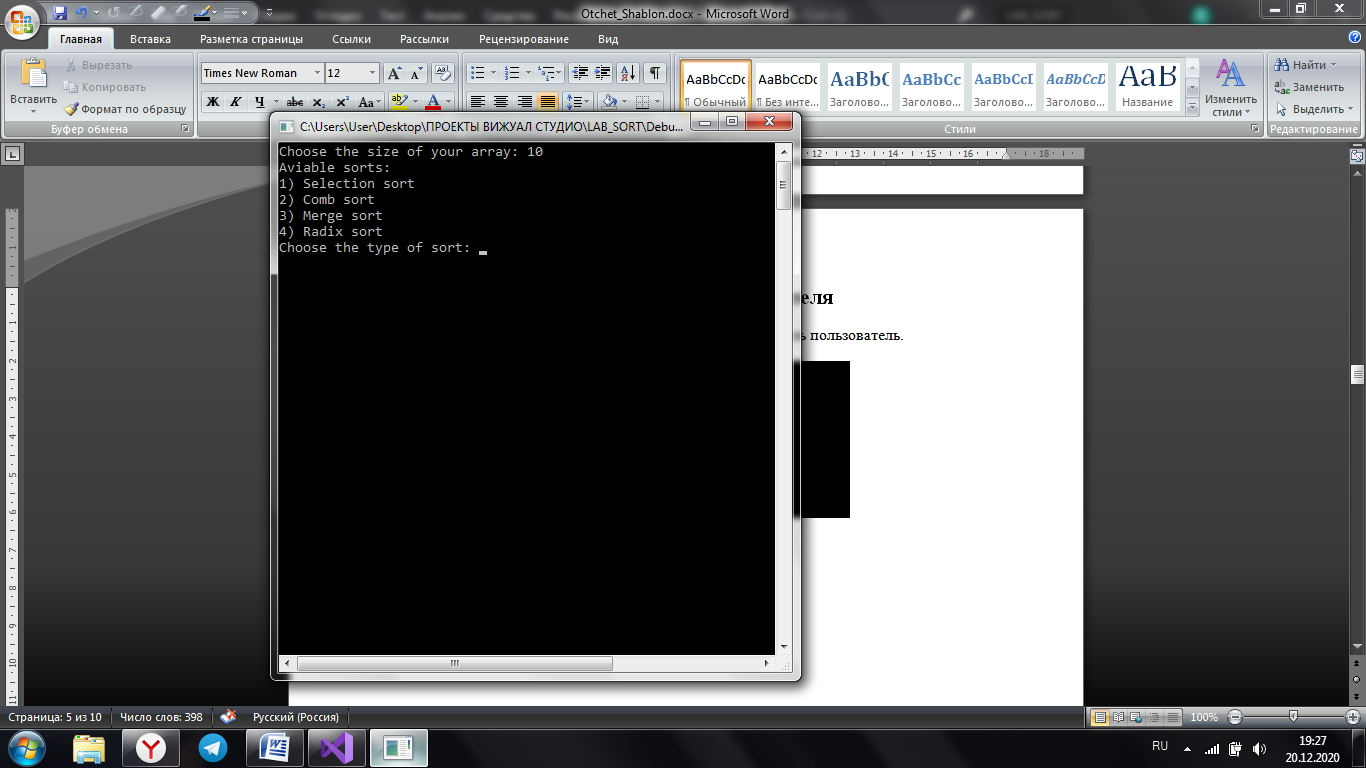
3) Затем аналогично делается для следующего разряда, и так до конца.

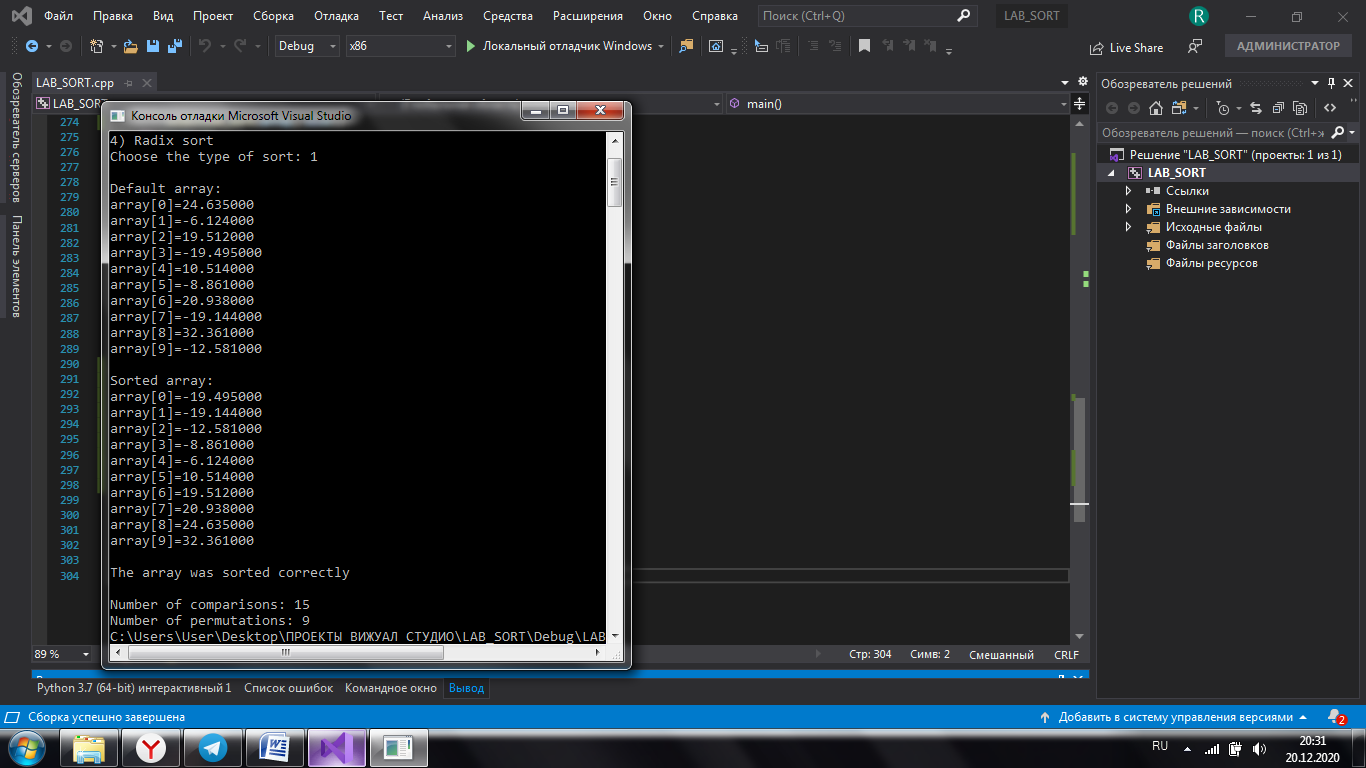
# Руководство пользователя

По запуску программы вам предлагается ввести количество элементов массива. Весь ввод выполняется с клавиатуры.



После этого необходимо выбрать сортировку из предложенных четырёх. Для этого введите её номер по списку



На экране появится случайный исходный массив, его отсортированная версия, количество перестановок и сравнений и оповещение о корректности сортировки. 

# Описание программной реализации

Используемые функции:

1) bubble\_sort – сортировка пузырьком, предназначенная для подтверждения корректности следующих сортировок, проходит по массиву, сравнивая и переставляя его элементы в порядке возрастания, если не один элемент не был переставлен, выходим из цикла. В качестве аргументов принимает указатель на массив типа double и его размер типа int. Ничего не возвращает.

2) selection\_sort – сортировка выбором, сортирует массив по вышеописанному алгоритму. Принимает указатель на массив типа double и его размер типа int. Ничего не возвращает.

3) comb\_sort – сортировка расчёской, сортирует массив по вышеописанному алгоритму. Принимает указатель на массив типа double и его размер типа int. Ничего не возвращает.

4) merge – упорядочивает две половины массива, после чего сортирует их как один целый массив. Принимает указатель на исходный массив и дополнительный пустой массив типа double, номер первого элемента левой части массива, номер последнего элемента левой части массива и номер последнего элемента правой части массива типа int. Ничего не возвращает.

5) merge\_sort – сортировка слиянием. Принимает указатель на исходный и дополнительный массив типа double, номер первого и последнего элемента части массива, которую нужно отсортировать, типа int. Ничего не возвращает.

6) createCounters – создаёт корзины для всех возможных значений разряда числа, принимает указатель на исходный массив типа double, указатель на выходной массив с корзинами типа long и размер исходного массива. Ничего не возвращает.

7) radixPass – записывает элементы массива, отсортированные по заданному разряду. Принимает номер байта типа short, размер исходного массива, указатель на исходный массив типа double, указатель на массив для записи отсортированных чисел типа double, указатель на массив корзин по разряду типа long. Ничего не возвращает.

8) radixLastPass – проход по знаковому разряду чисел, перестановка отрицательных чисел в начало массива. Принимает номер байта типа short, размер исходного массива, указатель на исходный массив типа double, указатель на массив для записи отсортированных чисел типа double, указатель на массив корзин по разряду типа long. Ничего не возвращает.

9) radixSort – поразрядная сортировка. Принимает на вход массив double и его размер. Ничего не возвращает.

# Подтверждение корректности

Для подтверждения корректности в программе присутствует проверка сортировкой пузырьком с флагом. Если ни один элемент массива не оказывается больше следующего, то значение переменной flag остаётся нулевой, на экран выводится сообщение о корректной сортировке массива. Иначе же выполняется полная сортировка пузырьком, массив сортируется, но за более длительное время. На экран выводится сообщение о том, массив был неправильно отсортирован. Перестановки и сравнения в сортировке пузырьком не учитываются. В ходе экспериментов ни одного сообщения о некорректной сортировке выявлено не было.

# Результаты экспериментов

1. Сортировка выбором

Теоретическая сложность алгоритма – O(n2) (по арифметической прогрессии – (n2-n)/2 перестановок в любом случае). Проведём сортировку массивов разных размеров. Лучший для сортировки случай – тот, в котором входной массив уже отсортирован, средний – массив перемешан, худший – массив отсортирован в обратном порядке. В таблице представлено количество сравнений и перестановок (через дробь) для каждого случая и размера.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 100 | 300 | 500 | 800 | 1000 |
| Лучший | 4950/0 | 44850/0 | 124750/0 | 319600/0 | 499500/0 |
| Средний | 4950/315 | 44850/1351 | 124750/2355 | 319600/4171 | 499500/5502 |
| Худший | 4950/2461 | 44850/22500 | 124750/62381 | 319600/159409 | 499500/247743 |

По приведённым результатам экспериментов мы видим, что сложность алгоритма действительно O(n2)

2. Сортировка расчёской

Сложность данной сортировки зависит от случая. Теоретическая сложность в лучшем случае – O(n\*log n), в среднем – O(n2/2p), где p – количество приращений, в худшем - O(n2). В данной сортировке обратная упорядоченность является средним случаем, а набор случайных чисел - худшим

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 100 | 300 | 500 | 800 | 1000 |
| Лучший | 1233/0 | 5144/0 | 9536/0 | 16835/0 | 22034/0 |
| Худший | 1233/237 | 5144/926 | 9536/1832 | 16835/3119 | 22034/4178 |
| Средний | 1233/106 | 5144/406 | 9536/688 | 16835/1183 | 22034/1524 |

3. Сортировка слиянием

Сортировка имеет общую сложность O(n\*log n) как в среднем, так и в лучшем и худшем случае, по сути, все случаи для неё – средние. Рассмотрим стандартный средний (случайный) и лучший (упорядоченный) случай.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 100 | 300 | 500 | 800 | 1000 |
| Лучший | 356/672 | 1308/2488 | 2272/4488 | 4048/7776 | 5044/9976 |
| Средний | 551/672 | 2125/2488 | 3916/4488 | 6809/7776 | 8827/9976 |

4. Поразрядная сортировка

На сложность алгоритма входные значения не влияют, так как он работает с байтами числа. Сложность в любом случае линейная – O(n)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 100 | 300 | 500 | 800 | 1000 |
| Средний | 0/800 | 0/2400 | 0/4000 | 0/6400 | 0/8000 |

# Заключение

Мы рассмотрели четыре способа сортировки массивов. Все они хорошо справляются со своей задачей, однако для массивов более крупных размеров (на миллион элементов и больше) лучше использовать алгоритмы с наименьшей сложностью, например поразрядную сортировку, чтобы сэкономить время.

# Приложение

