### 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 3821Б1ПМ2 Сятов Н.А.

## Проверил:

### преподаватель каф. МОСТ, Волокитин В.Д.

### Нижний Новгород 2021

**Содержание**

[Постановка задачи 3](#_bookmark0)

[Метод решения 4](#_bookmark1)

[Руководство пользователя 6](#_bookmark2)

[Описание программной реализации 7](#_bookmark3)

[Подтверждение корректности 8](#_bookmark4)

[Результаты экспериментов 9](#_bookmark5)

[Заключение 16](#_bookmark6)

[Приложение 17](#_bookmark7)

# Постановка задачи

Цель работы: разобрать идею и реализовать на языке C следующие сортировки: сортировка выбором, быстрая сортировка Хоара, сортировка слиянием, поразрядная сортировка. Код данных сортировок я должен был реализовать для типа данных float. Нужно было описать алгоритмы, использованные в коде, подтвердить корректность сортировок, сложность. Описать способы подтверждения сложности и корректности, сделать выводы.

# Метод решения

## Сортировка выбором

### Это одна из простых сортировок. Она достаточно легка в реализации и понимании. Идея заключается в том, чтобы найти минимальный элемент в

### массиве, поставить его на первое место, дальше реализовать тот же алгоритм для оставшегося не отсортированного массива. Так мы проходим N-1 раз по

### Безымянный.pngмассиву, последний оставшийся элемент сам встанет на своё, последнее, место. Сложность этой сортировки О(N2)

## Быстрая сортировка Хоара.

### Одна из самых распространённых сортировок. Я реализовывал её рекурсивно.

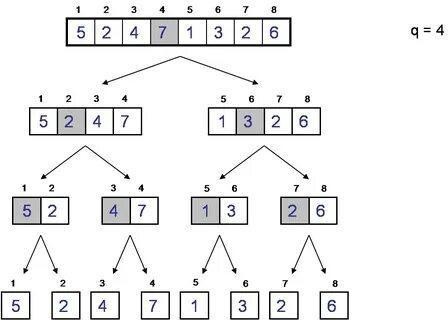
### Выбирается рандомный элемент массива, справа и слева относительно него находят меньший и больший элементы массива и меняют их местами. Теперь

### image090.pngопорным элементом становится тот, который находится справа от выбранного. После снова вызывается та же функция 2 раза: для левой части от опорного элемента и для правой части. Сложность этой сортировки равна О(N\*logN)

## Сортировка слиянием.

Часто эту сортировку тоже реализовывают рекурсивно. Её идея заключается в том, что находят середину массива и вызывают эту же функцию для левой и правой части от середины. Тем самым мы делим массив на массивы из одного элемента, потом сливаем их в отсортированный и продолжаем это до тех пор, пока не сольём все подмассивы в один отсортированный. Это достаточно быстрый алгоритм, но он требует выделение памяти на

дополнительный массив размера N. Сложность этой сортировки О(N\*logN).

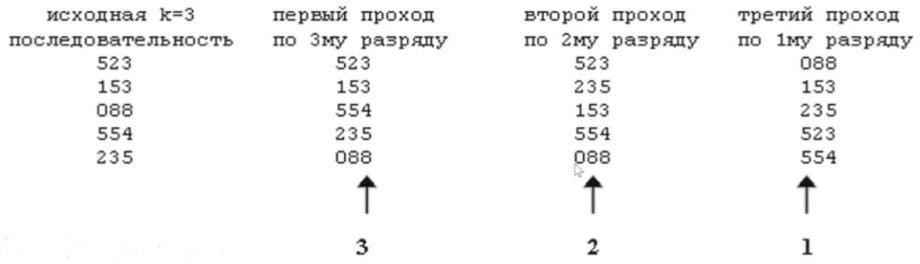


## Поразрядная сортировка

### Одна из самых быстрых сортировок, является расширенной версией

### сортировки подсчётом. Числа делятся по разрядам и сортируются поразрядной сортировкой, начиная с правого разряда. Сортировка устойчивая, поэтому

### числа будут располагаться в правильном порядке. Сложность приближена к О(n). Требуется 2 дополнительных массива: первый размера N, второй размера sizeof(float) \* 256 (1 килобайт).



# Руководство пользователя

Пользователю выводится случайно сгенерированный массив размера N. Высвечивается меню с выбором нужной сортировки. После ввода нужной цифры, массив сортируется выбранной сортировкой и выводится результат, количество произведённых сравнений и перестановок (присвоений). Если введена неправильная информация, то программа выводит исходный массив и пишет об ошибке.

# Описание программной реализации

Я написал сортировки отдельными функциями, которые все находятся в том же файле, что и главная функция main. Я использовал такие библиотеки: stdio.h, stdlib.h, time.h, math.h, locale.h, memory.h.

Основные фунуции:

1. swap(float\* f, float\* s) – функция меняет местами два элемента
2. randomArr(float arr[], int size) – функция генерирует случайный массив типа float размера size, диапазон чисел от -1000.999999 до 1000.999999.
3. SelectionSort(float arr[], int size, int\* swaping, int\* check) – функция сортировки выбором, принимает массив, его размер, также дополнительно подсчитывает количество сравнений и перестановок внутри функции.
4. QuickSort(float arr[], int left, int right, int\* swaping, int\* check) – функция быстрой сортировки, принимает массив, его самый левый и самый правый индекс, также дополнительно подсчитывает количество сравнений и перестановок внутри функции.
5. MergeSort(float arr[], float second[], int left, int right, int\* swaping, int\* check) – функция сортировки слиянием, принимает исходный массив, дополнительный массив, самый левый и самый правый индекс исходного массива, также дополнительно подсчитывает количество сравнений и перестановок внутри функции.
6. RadixSort(float in[], float out[], long counters[], int\* swaping, int\* check) – функция поразрядной функции, принимает исходный массив, дополнительный массив, массив с счётчиками.

# Подтверждение корректности

Для подтверждения корректности в программе использовал стандартную сортировку qsort из библиотеки stdlib.h. Для этого я создал массив, который дублирует неотсортированный исходный массив. После чего исходный сортируется моей сортировкой, а дубликат с помощью qsort. Потом по индексам сравниваются эти два массива. Если хотя бы один элемент не сходится, программа выводит ошибку.

# Результаты экспериментов

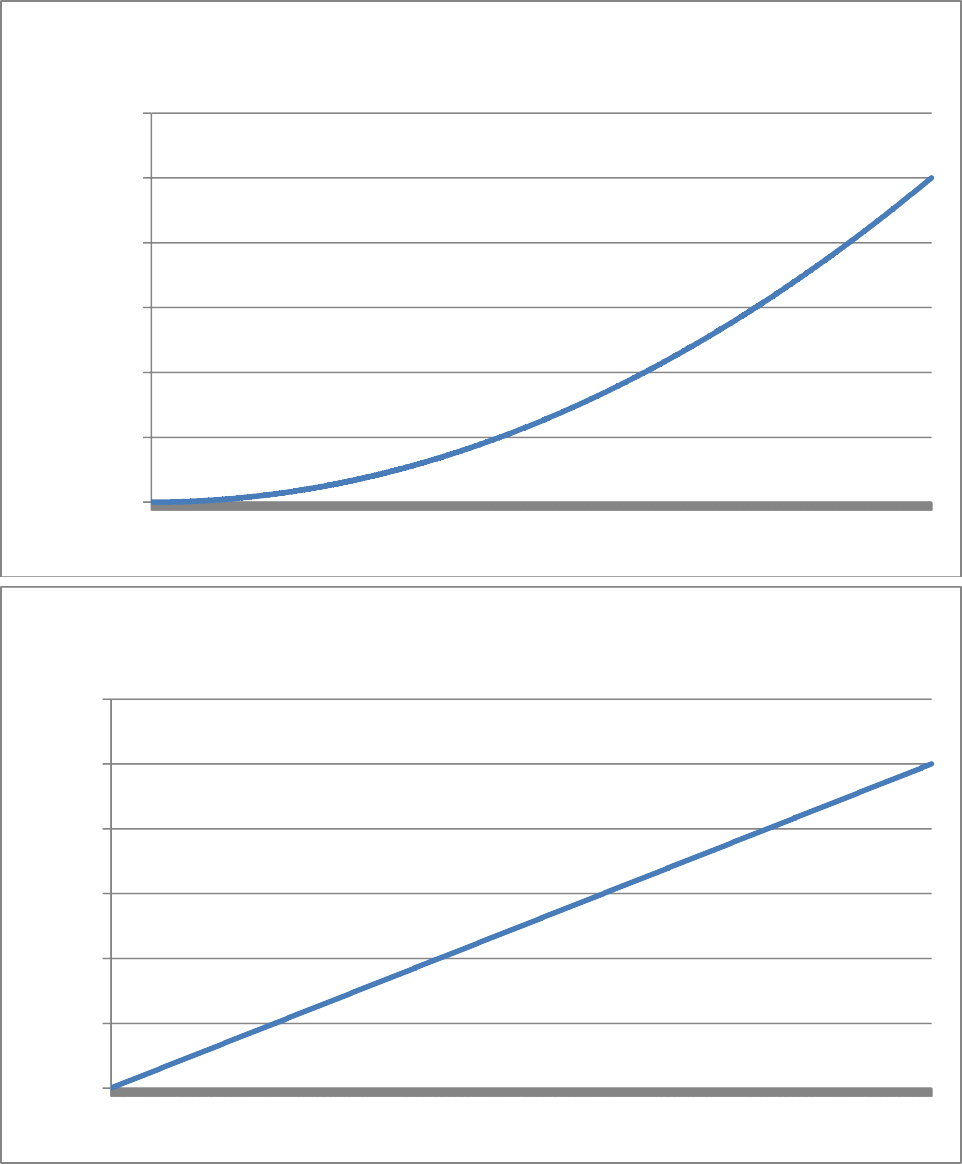
Программа генерировала случайные массивы размер от 10 до 10 000 с шагом 10.

Чтобы проверить сложность данных сортировок, достаточно посчитать количество перестановок и сравнений и поделить их на теоретическую сложность. В итоге

должен получиться график, сходящийся к константе.

* 1. Сортировка выбором

Это одна из простых сортировок, её предполагаемая сложность О (N2) Далее представлены графики сравнений и перестановок:



**Сравнения**

60000000

50000000

40000000

30000000

20000000

10000000

0

**перестановки**

12000

10000

8000

6000

4000

2000

0

1

45

89

133

177

221

265

309

353

397

441

485

529

573

617

661

705

749

793

837

881

925

969

Теперь разделим полученные данные на N2:

1

41

81

121

161

201

241

281

321

361

401

441

481

521

561

601

641

681

721

761

801

841

881

921

961

1

40

79

118

157

196

235

274

313

352

391

430

469

508

547

586

625

664

703

742

781

820

859

898

937

976

10

3760

7510

11260

15010

18760

22510

26260

30010

33760

37510

41260

45010

48760

52510

56260

60010

63760

67510

71260

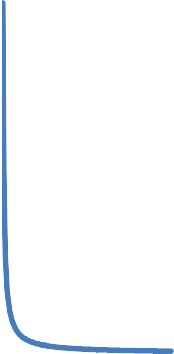
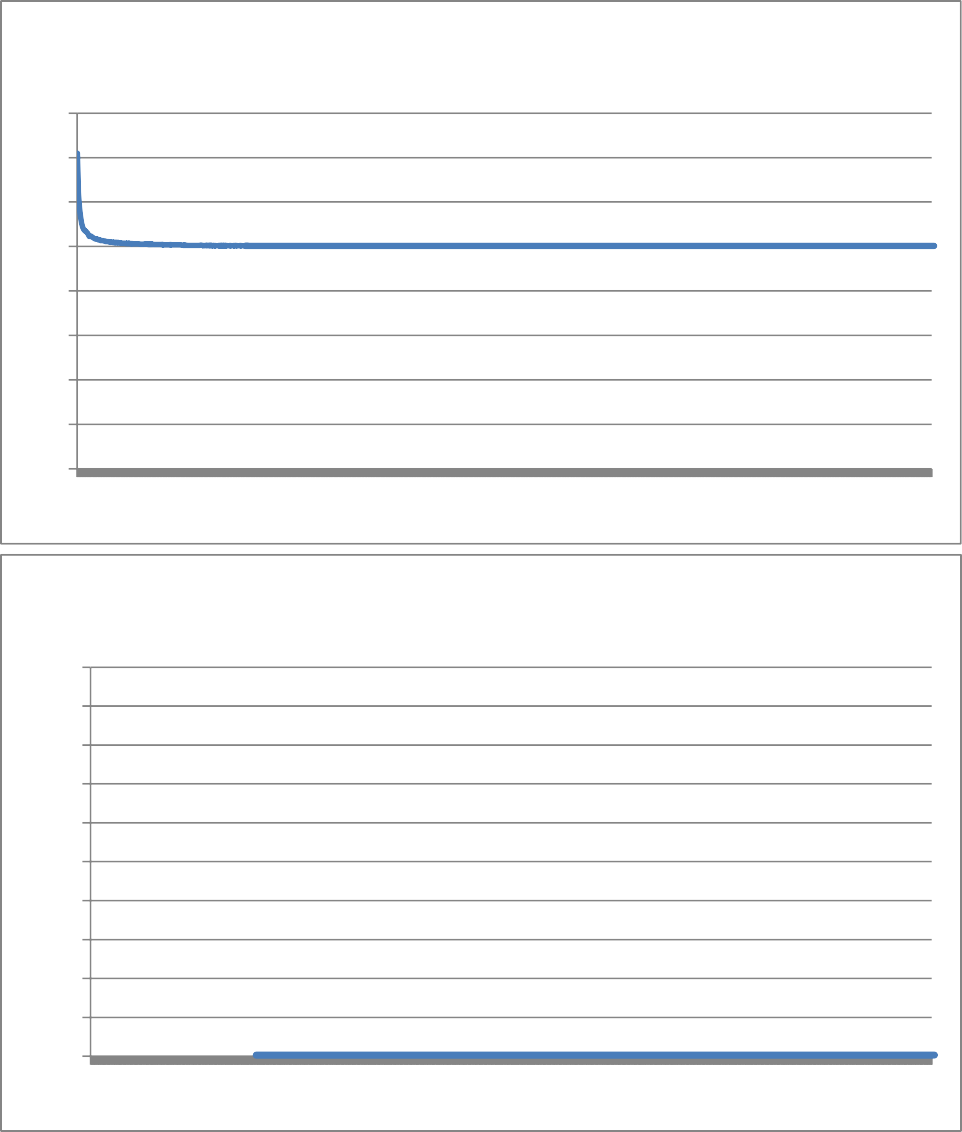
75010

78760

82510

86260

Мы получили константы, значит, сложность определена верно.



**Сранения / N^2**

0,8

0,7

0,6

0,5

0,4

0,3

0,2

0,1

0

**Перестановки / N^2**

0,1

0,09

0,08

0,07

0,06

0,05

0,04

0,03

0,02

0,01

0

1

41

81

121

161

201

241

281

321

361

401

441

481

521

561

601

641

681

721

761

801

841

881

921

961

* 1. Быстрая сортировка Хоара

Теоретическая сложность этой сортировки О (N\*log2N), чтобы график был наглядней, расширим диапазон до 90 000:



**Сравнения**

2000000

1800000

1600000

1400000

1200000

1000000

800000

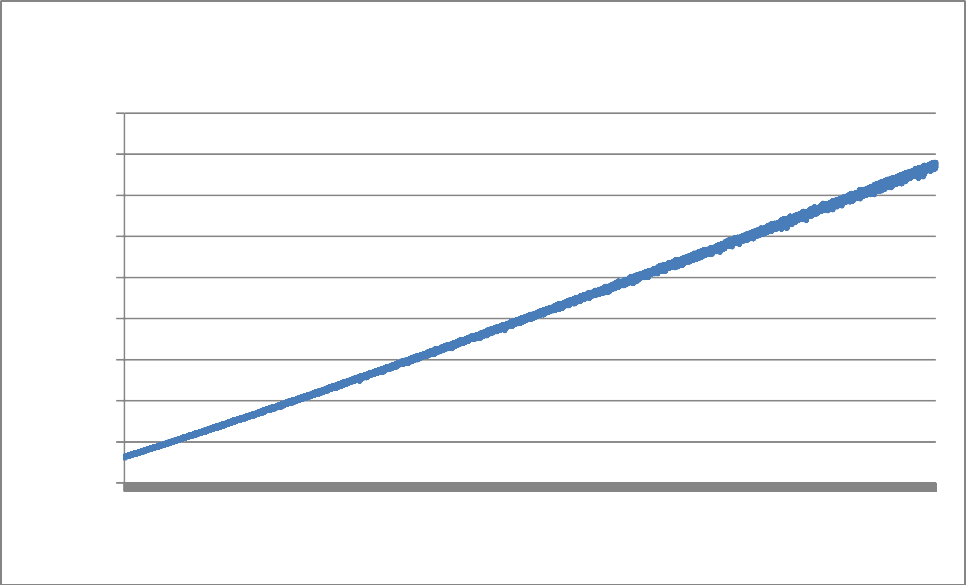
600000

400000

200000

0

Разделим полученные данные на N\*log2N:



**Перестановки**

450000

400000

350000

300000

250000

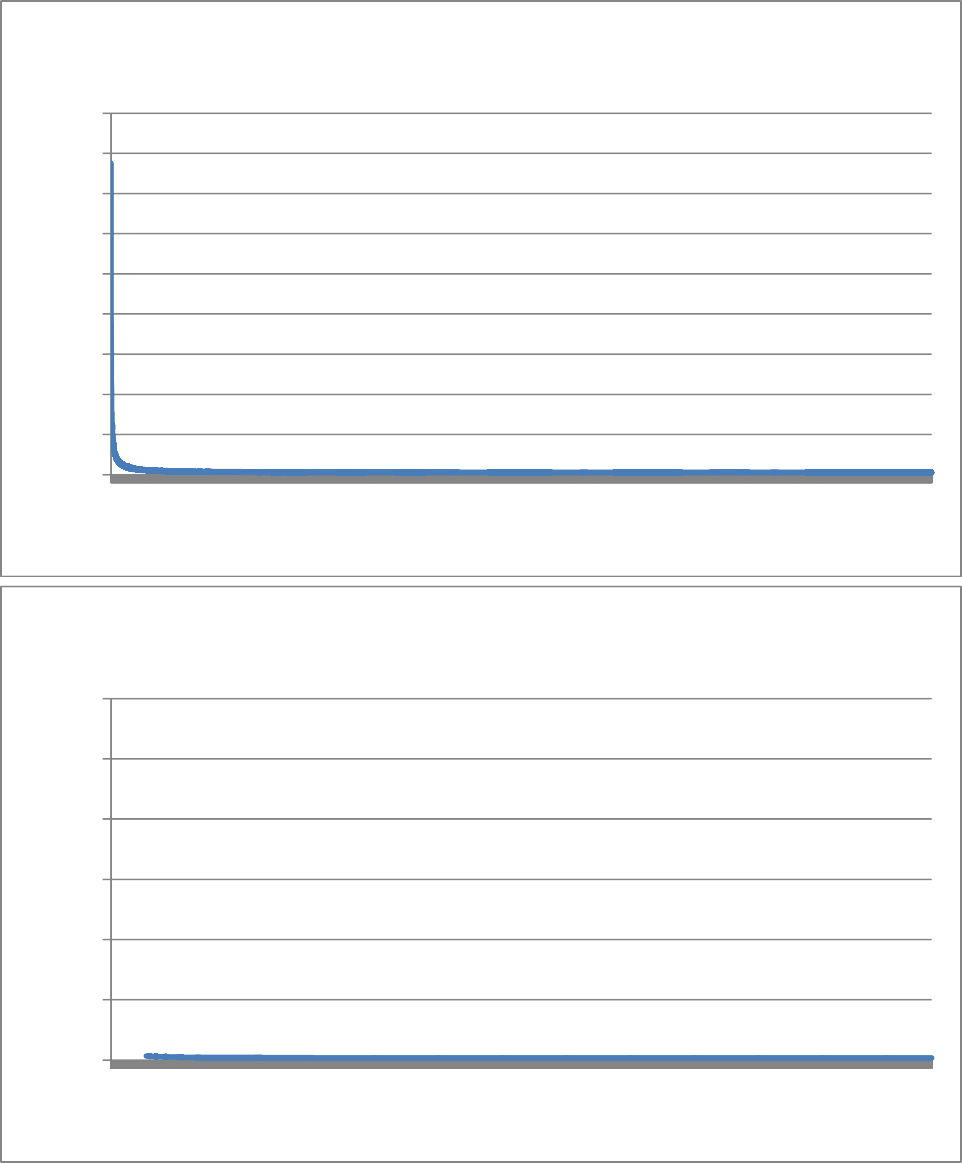
200000

150000

100000

50000

0



**Сравнения / N\*log2N**

45000

40000

35000

30000

25000

20000

15000

10000

5000

0

**Перестановки / N\*log2N**

12000

10000

8000

6000

4000

2000

0

10

3610

7210

10810

14410

18010

21610

25210

28810

32410

36010

39610

43210

46810

50410

54010

57610

61210

64810

68410

72010

75610

79210

82810

86410

10

3760

7510

11260

15010

18760

22510

26260

30010

33760

37510

41260

45010

48760

52510

56260

60010

63760

67510

71260

75010

78760

82510

86260

Мы получили константу, значит, теоретическая сложность совпала с практической в точности до константы

10

3610

7210

10810

14410

18010

21610

25210

28810

32410

36010

39610

43210

46810

50410

54010

57610

61210

64810

68410

72010

75610

79210

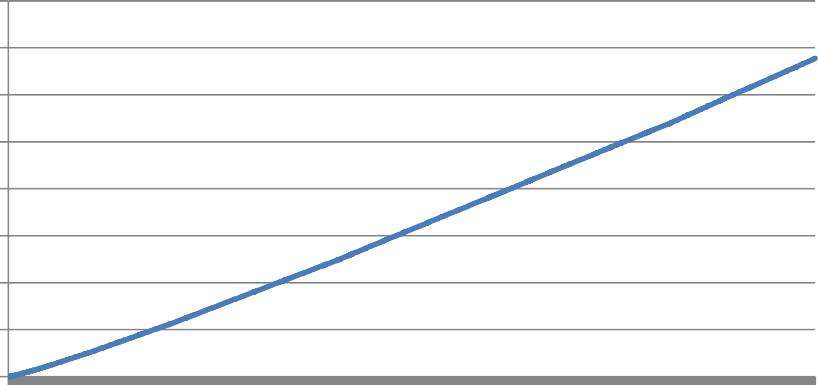
82810

86410

* 1. Сортировка слиянием

Сложность этой сортировки совпадает с сложностью быстрой, тоже О (N\*log2N)

Построим графики:



**Сравнения**

400000

350000

300000

250000

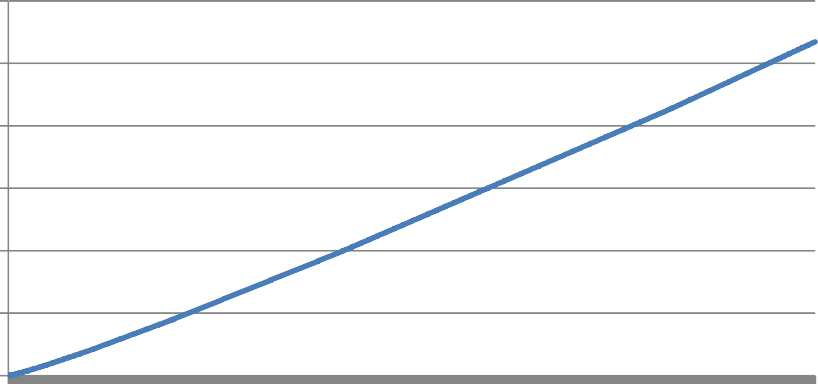
200000

150000

100000

50000

0



**Присвоения**

300000

250000

200000

150000

100000

50000

0

10

430

850

1270

1690

2110

2530

2950

3370

3790

4210

4630

5050

5470

5890

6310

6730

7150

7570

7990

8410

8830

9250

9670

Разделим на сложность:

10

430

850

1270

1690

2110

2530

2950

3370

3790

4210

4630

5050

5470

5890

6310

6730

7150

7570

7990

8410

8830

9250

9670

10

400

790

1180

1570

1960

2350

2740

3130

3520

3910

4300

4690

5080

5470

5860

6250

6640

7030

7420

7810

8200

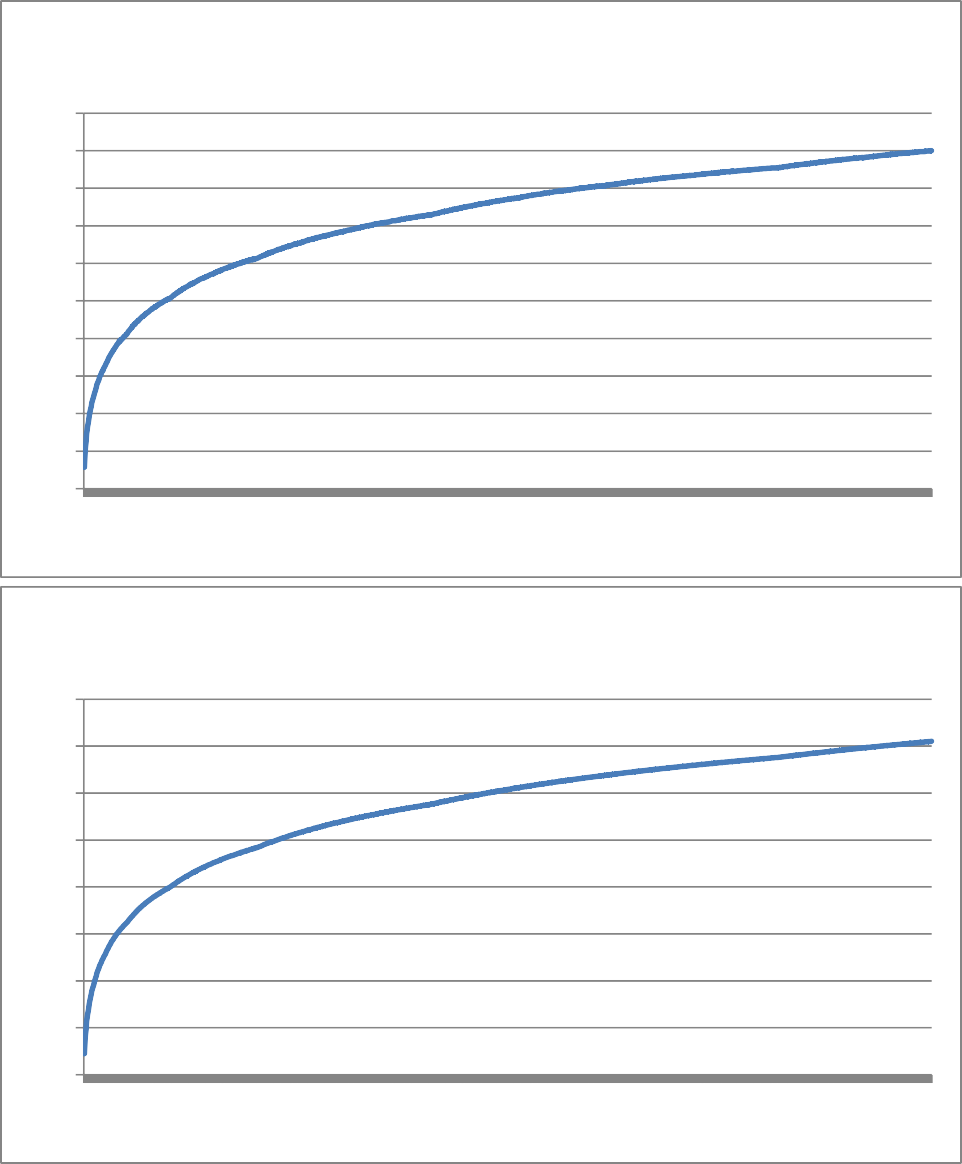
8590

8980

9370

9760

Мы получили с определенного размера константу, значит, подтвердили теоретическую сложность.



**Сравнения / N\*log2N**

500

450

400

350

300

250

200

150

100

50

0

**Присвоения / N\*log2N**

400

350

300

250

200

150

100

50

0

10

400

790

1180

1570

1960

2350

2740

3130

3520

3910

4300

4690

5080

5470

5860

6250

6640

7030

7420

7810

8200

8590

8980

9370

9760

* 1. Поразрядная сортировка

Одна из самых быстрых сортировок, потому что она является линенйной, её сложность О(4(N + 256) + N)

Построим графики:

10

400

790

1180

1570

1960

2350

2740

3130

3520

3910

4300

4690

5080

5470

5860

6250

6640

7030

7420

7810

8200

8590

8980

9370

9760

10

430

850

1270

1690

2110

2530

2950

3370

3790

4210

4630

5050

5470

5890

6310

6730

7150

7570

7990

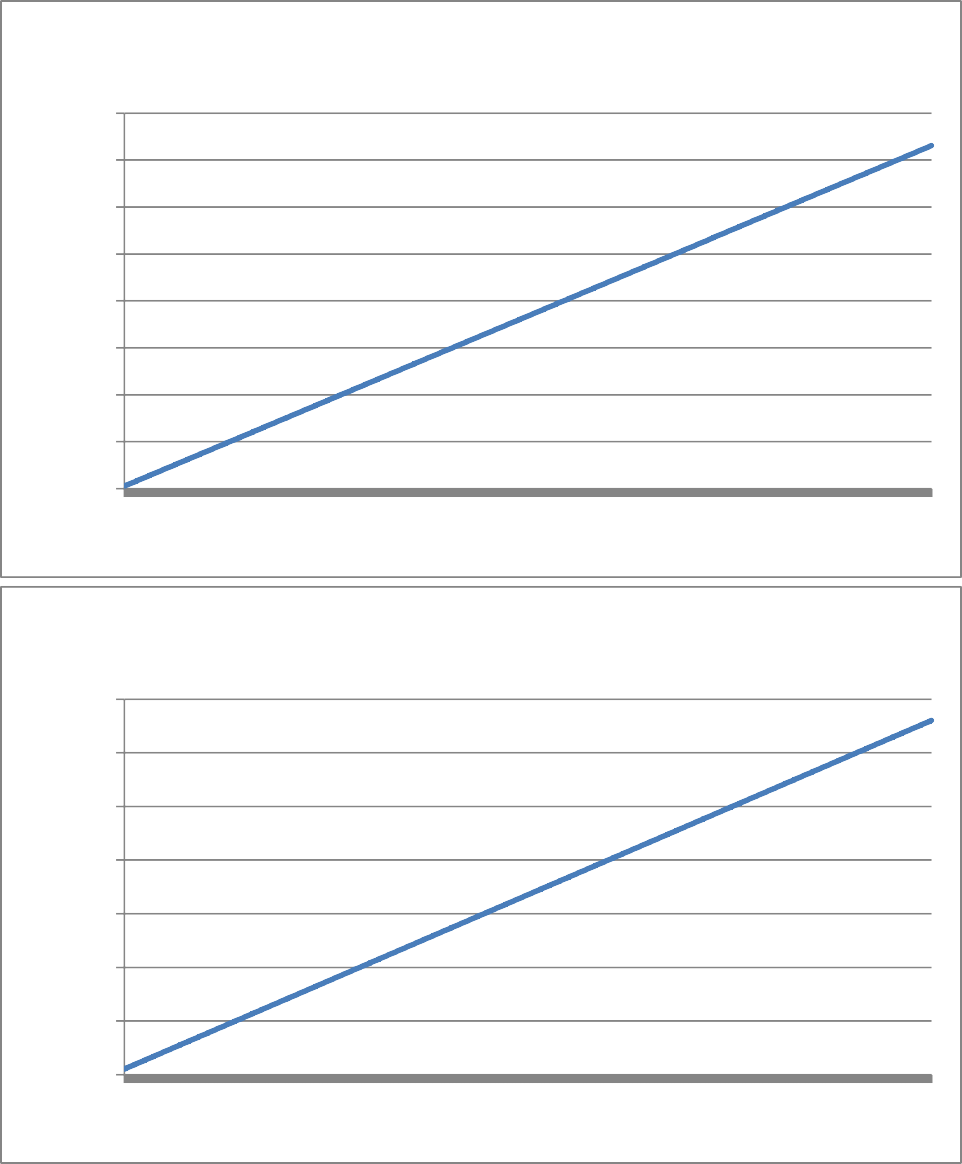
8410

8830

9250

9670

По рисунку видно, что график похож на прямую, докажем сложность, разделив на 5\*N+1024:



**Сравнения**

160000

140000

120000

100000

80000

60000

40000

20000

0

**Присвоения**

140000

120000

100000

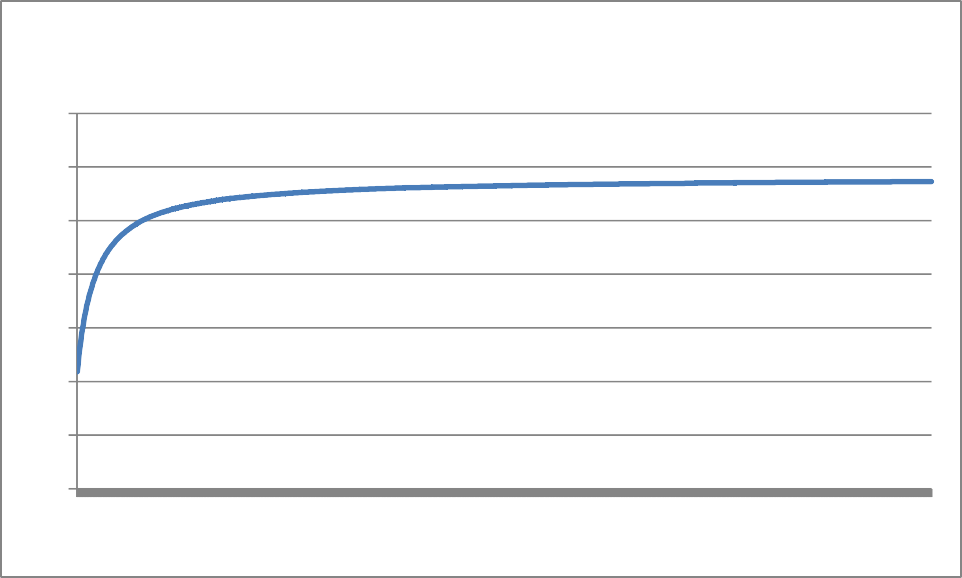
80000

60000

40000

20000

0



**Сравнения / (5\*N+1024)**

3,5

3

2,5

2

1,5

1

0,5

0

10

430

850

1270

1690

2110

2530

2950

3370

3790

4210

4630

5050

5470

5890

6310

6730

7150

7570

7990

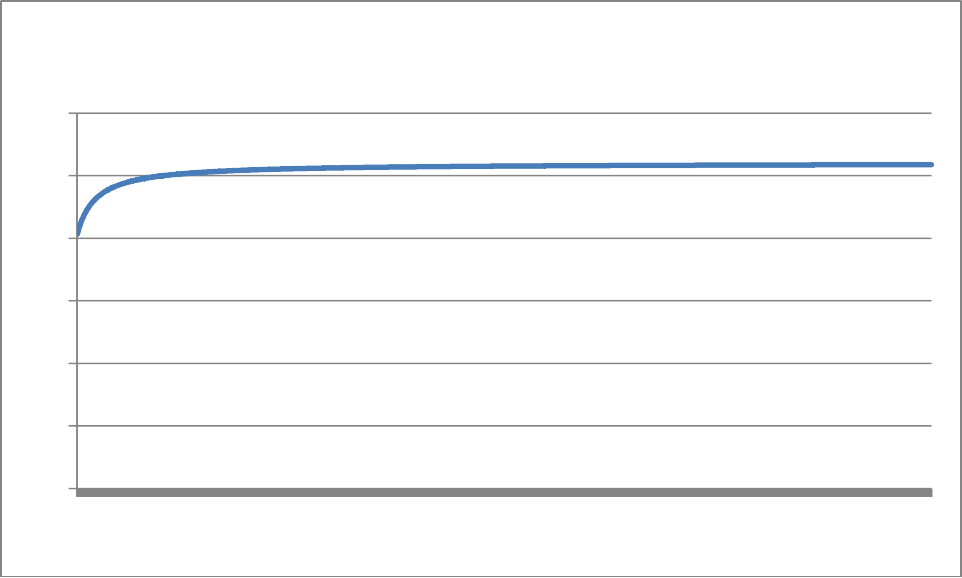
8410

8830

9250

9670

Снова получаем константу, что и требовалось доказать.



**Присвоения / (5\*N+1024)**

3

2,5

2

1,5

1

0,5

0

10

400

790

1180

1570

1960

2350

2740

3130

3520

3910

4300

4690

5080

5470

5860

6250

6640

7030

7420

7810

8200

8590

8980

9370

9760

# Заключение

Я реализовал на языке С 4 сортировки: выбором, быструю, слиянием, поразрядную. Описал их алгоритмы работы, проверил корректность при больших данных, доказал их теоретическую сложность с помощью подсчета сравнений и перестановок, научился работать в системе Git, работать с большими данными.

# Приложение

