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/ка группы 3821Б1ПМ2

Василевский А.П.

**Проверил**:

преподаватель каф. МОСТ,

Волокитин В.Д.

Нижний Новгород

2021

**Содержание**

[Постановка задачи 3](#_Toc26962562)

[Метод решения 4](#_Toc26962563)

[Руководство пользователя 7](#_Toc26962564)

[Описание программной реализации 8](#_Toc26962565)

[Подтверждение корректности 10](#_Toc26962566)

[Результаты экспериментов 11](#_Toc26962567)

[Заключение 15](#_Toc26962568)

# Постановка задачи

Задачами лабораторной работы являются реализация сортировки выбором, сортировки расчёской, сортировки слиянием и поразрядной сортировки на языке программирования Си для типа данных double, подтверждение корректности реализации и асимптотической сложности.

# Метод решения

**Сортировка выбором**

Суть сортировки выбором заключается в нахождении номера элемента, соответствующего минимальному значению в массиве и обмене этого значения со значением первой несортированной позиции после чего из рассмотрения исключаются уже отсортированные элементы.

Для нахождения номера минимального значения в массиве длинной на первой итерации потребуется сравнений, после чего наименьший элемент будет размещён на первой позиции, а рассматриваемая длина массива для поиска уменьшится на 1. Так как длина рассматриваемая длина массива каждый уменьшается на 1 то и количество сравнений уменьшается на 1 и образует арифметическую прогрессию. Таким образом количество сравнений для лучшего и худшего случая равняется , что соответствует асимптотической сложности .

**Сортировка расчёской**

Сортировка расчёской является улучшенной версией сортировки пузырьком. Основная идея заключается в том, чтобы сравнивать элементы на расстоянии большем чем 1 и тем самым устраняя маленькие элементы в конце массива, которые крайне замедляют сортировку пузырьком.

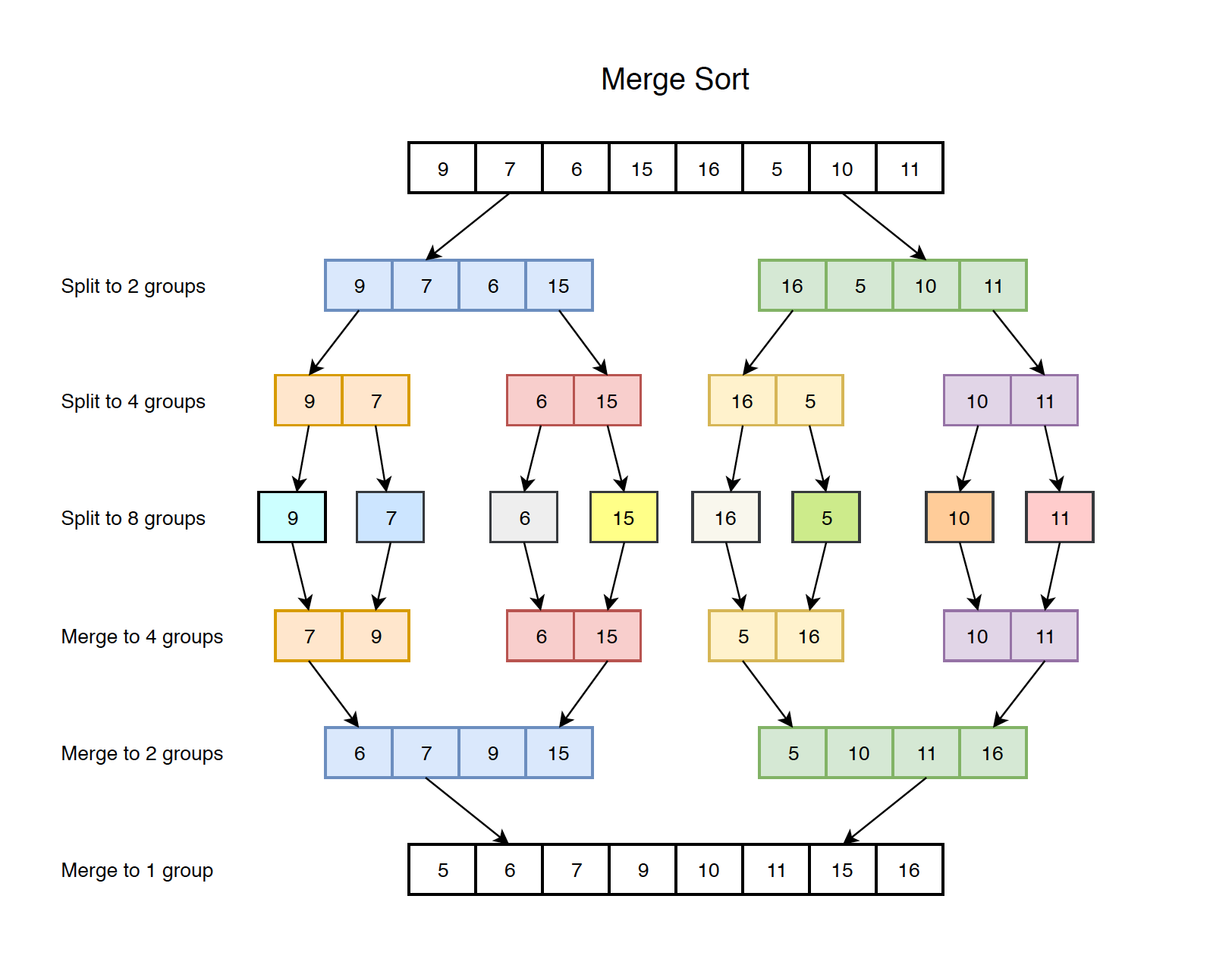
На первой итерации расстояние между сравниваемыми элементами равняется длине массива – 1, а на каждой последующей итерации делится на фактор равный 1.2473309, это происходит до тех пор, пока расстояние между элементами не станет меньше 1. Если сравниваемые элементы не расположены в порядке возрастания, то они меняются местами.

В худшем случае, когда фактор подобран неверно асимптотическая сложность , в лучшем случае , и (p – количество приращений) в среднем, что сопоставимо с .

**Сортировка слиянием**

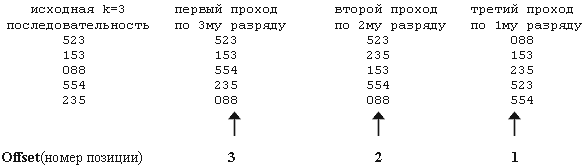
Суть этого алгоритма заключается в том, что массив разбивается на подмассивы, что происходит до тех пор, пока длина подмассива не станет равной 1, после чего подмассивы попарно начинают объединять.

Разбиение массива выполняется за операций, а объединение двух массивов длинны n происходит за . Таким образом сложность алгоритма составляет и требует дополнительной памяти.



**Поразрядная сортировка**

В поразрядной сортировке используется сортировка подсчётом и свойство её устойчивости. Предполагается, что каждый ключ сортировки можно рассматривать как k-значное число с разрядом в 1 байт. Поочерёдно используется устойчивая сортировка для каждой цифры справа налево. Однако числа в памяти хранятся в обратном порядке (от младшего байта к старшему), поэтому сортировка будет идти по тому порядку в котором число хранится в памяти. Т.к. отрицательные числа в памяти имеют 1 в старшем байте (в первом бите) то они будут восприниматься как большие положительные числа, поэтому нужно разместить отрицательные числа в обратном порядке в начале массива после будут следовать положительные.



# Руководство пользователя

При запуске программы будет выводиться сообщение “Specify the length of the array : “, после чего пользователю необходимо ввести длину массива.

Далее будет выведено “select the input method: g - generate automatically e - enter yourself”, если будет введён символ ‘g’, то программа сгенерирует массив автоматически, если будет введён символ ‘e’ программа выведет сообщение   
“enter an array:” после которого следует ввести элементы массива (после каждого элемента нажимается enter), в случае, когда введённый символ не является ‘g’ или ‘e’ программа будет повторно выводить сообщение и ожидать ввод символа.

После задания массива программа выводится сообщение “Output an array? y – yes n – no”, если будет введён символ ‘e’ программа выведет элементы массива, если будет введён символ ‘n’ программа продолжит своё выполнение, в случае, когда введённый символ не является ‘e’ или ‘n’ программа будет повторно выводить сообщение и ожидать ввод символа.

На следующем этапе выводится сообщение “select the sort type: s - selection sort c - comb sort m - merge sort r - radix sort”. При вводе символа будет использоваться соответствующая ему сортировка (Сортировка выбором для 's', сортировка расчёской для ‘c’, сортировка слиянием для ‘m’ и поразрядная сортировка для ‘r’). В случае, когда введённый символ не является ‘s’ или ‘c’ или ‘m’ или ‘r’ программа будет повторно выводить сообщение и ожидать ввод символа.

После сортировки массива программа выводится сообщение “Output a sorted array? y – yes n – no”, если будет введён символ ‘e’ программа выведет отсортированный массив и завершит работу, если будет введён символ ‘n’ программа завершит работу. В случае, когда введённый символ не является ‘e’ или ‘n’ программа будет повторно выводить сообщение и ожидать ввод символа.

# Описание программной реализации

void selectionSort(double\* arr, int len) - на вход функция принимает указатель на сортируемый массив, длину сортируемого массива. Выполняет сортировку выбором.

void comb\_sort(double array[], int size) - на вход функция принимает указатель на сортируемый массив, длину сортируемого массива. Выполняет сортировку расчёской.

void merge(double\* array1, double\* array2, int l, int q, int r) - на вход функция принимает указатель на сортируемый массив, указатель на вспомогательный массив, и три числа по которым определяются части массива которые необходимо объединить. Объединяет две части сортируемого массива (номера элементов с l по q и с g+1 по r) записывая в вспомогательный массив, после чего переносит значения в сортируемый массив.

void merge\_sort(double\* array1, double\* array2, int l, int r) - на вход функция принимает указатель на сортируемый массив, указатель на вспомогательный массив той же длины, индекс первого элемента, индекс последнего элемента. Выполняет сортировку слиянием

void createCounters(double\* data, int\* counters, int N) - на вход принимает указатель на массив (длины N), указатель на вспомогательный массив (длины 256\*sizeof(double)\*sizeof(int) ) , и длину этого массива. Функция подсчитывает сколько раз в сортируемом массиве встретилось значение каждого байта числа, записывая все в вспомогательный массив.

void radixPass(short Offset, int N, double\* source, double\* dest, int\* count) -функция принимает позицию ,начиная с которой нужно вставлять число в выходной массив, длину сортируемого массива, указатель на первый элемент исходного массива, указатель на первый элемент выходного массива , указатель на начало позиций сортируемого разряд из вспомогательного массива для createCounters. Функция выполняет поразрядную сортировку.

void sign(double\* in, double\* out, int N) - на вход принимает указатель на массив, указатель на второй массив (после radixPass эти массивы одинаковые) и длина массива. Размещает отрицательные элементы в начало массива в правильном порядке.

void radixSort(double\* in, double\* out, int\* counters, int N) - на вход принимает указатель на массив (длины N), указатель на массив (длины N), указатель на вспомогательный массив (длины 256\*sizeof(double)\*sizeof(int) ) и длину массива N. Выполняет поразрядную сортировку с учётом знака.

# Подтверждение корректности

Для подтверждения корректности каждой из функций сортировки в отдельной программе использовался цикл for(len = 100; len < 100000; len += 500) в котором создавались два массива длины len и заполнялись случайными числами типа double (так чтобы массивы были равны).

Далее вызывалась функция, корректность которой необходимо проверить, для первого массива и встроенная функция qsort() из библиотеки stdlib работающая корректно

Для qsort() использовалась функция int cmp(const void\* a, const void\* b), получающая адреса двух чисел a и b и возвращающая -1 если a<b, 0 если a=b и 1 если a>b.

После чего первый массив (сортируемый функцией корректность которой необходимо проверить) и второй массив (сортируемый встроенной функцией) поэлементно сравниваются. Если при одинаковом индексе элементы в массивах будут различны, то будет выведено сообщение об ошибке (“SORT ERROR”).

После того как корректность каждой из функций для сортировки была подтверждена эти функции были объединены в одну программу.

# Результаты экспериментов

**Сортировка выбором**

Для проверки предполагаемой асимптотической сложности данной сортировки равной построим график зависимости количества сравнений/присвоений от длины массива n. Если разделить каждое полученной значение на то получится график сходящийся к константе. Для сравнений и присвоений эта константа равняется 0,5.

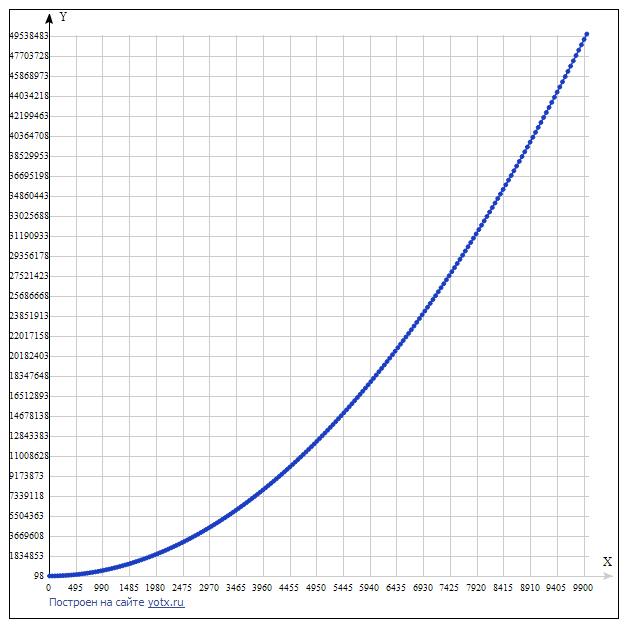
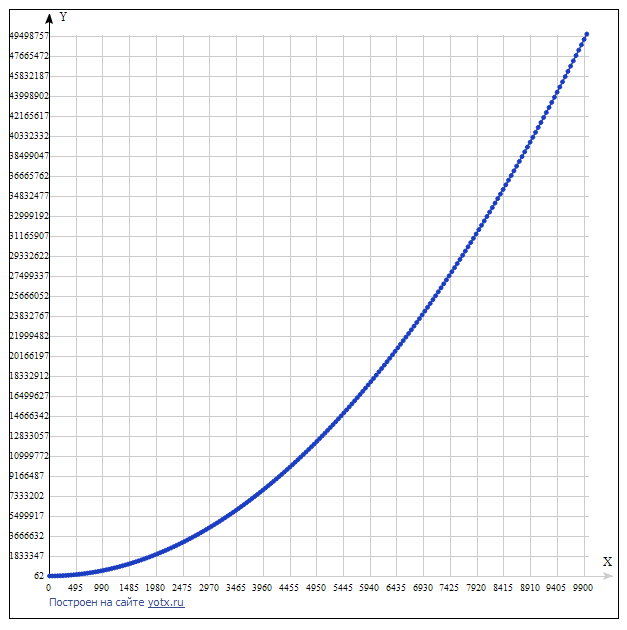


График зависимости сравнений от длины массива График зависимости присвоений от длины массива

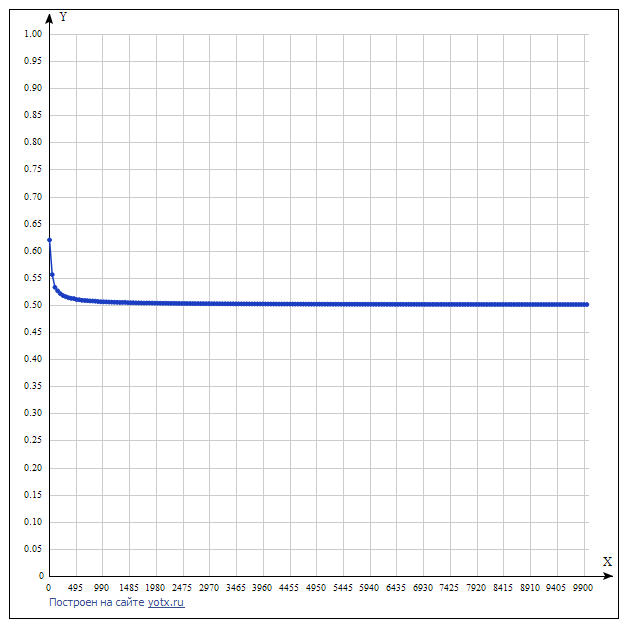
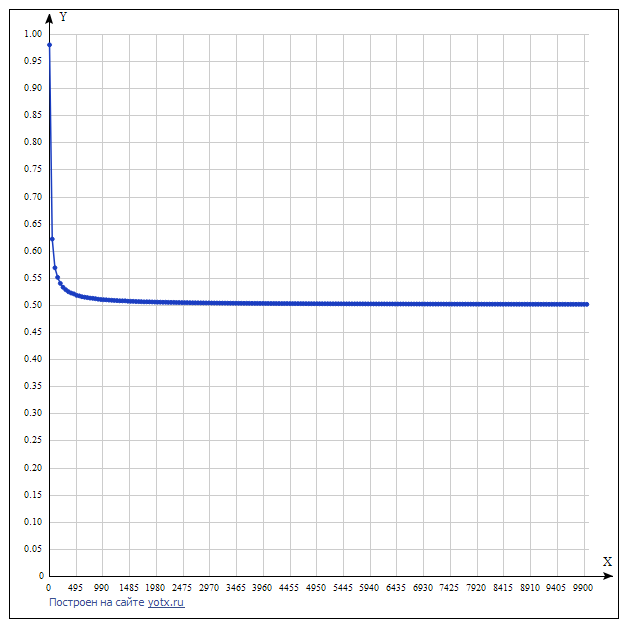
 

График зависимости сравнений от длины массива График зависимости присвоений от длины массива

после деления на после деления на

**Сортировка расчёской**

Для проверки предполагаемой асимптотической сложности данной сортировки равной (p – количество приращений), что сопоставимо с построим график зависимости количества сравнений/присвоений от длины массива n. Если разделить каждое полученной значение на то получится график сходящийся к константе. Для сравнений эта константа равняется 3,17, а для присвоений равняется 4,1.

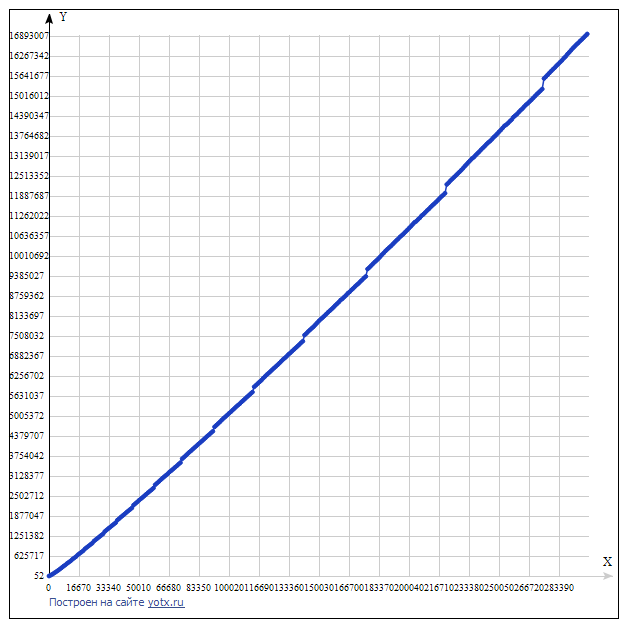
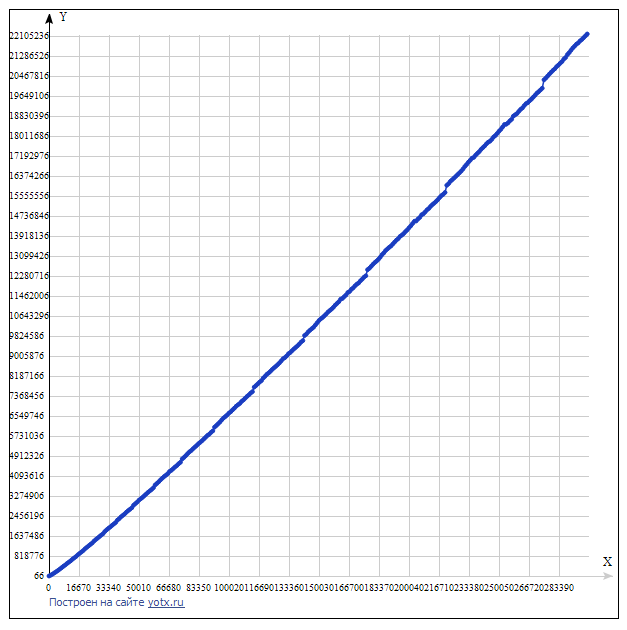
 

График зависимости сравнений от длины массива График зависимости присвоений от длины массива

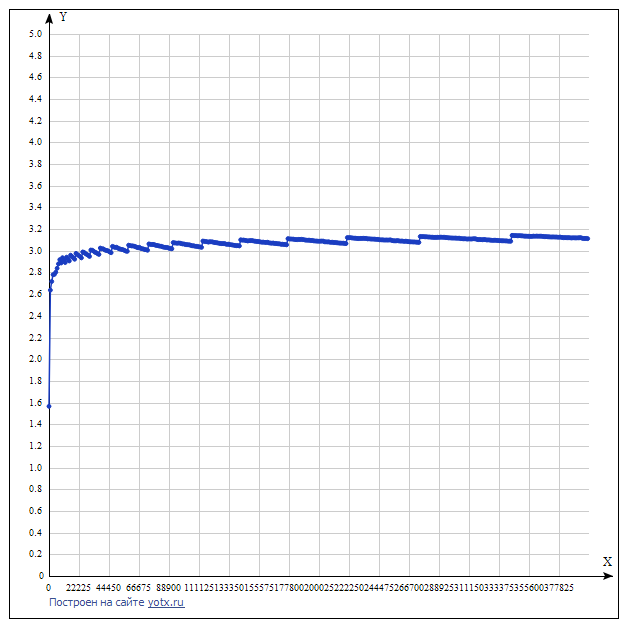
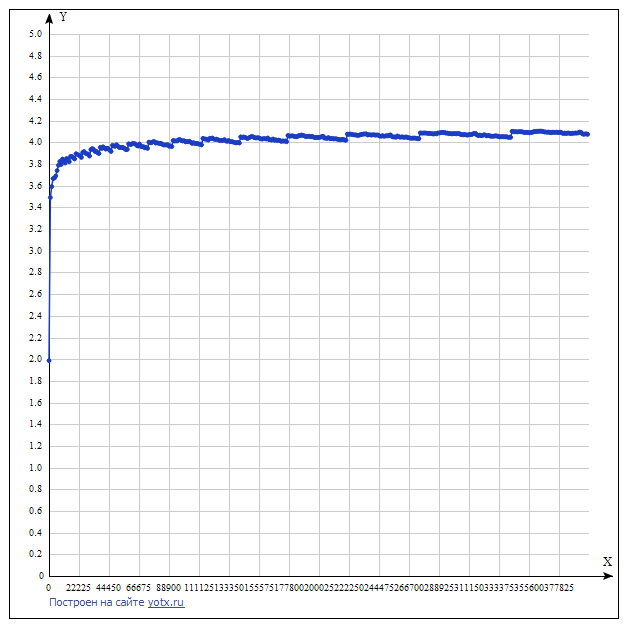
 

График зависимости сравнений от длины массива График зависимости присвоений от длины массива

после деления на после деления на

**Сортировка слиянием**

Для проверки предполагаемой асимптотической сложности данной сортировки равной построим график зависимости количества сравнений/присвоений от длины массива n. Если разделить каждое полученной значение на то получится график сходящийся к константе. Для сравнений эта константа равняется 3,47, а для присвоений равняется 3,3.

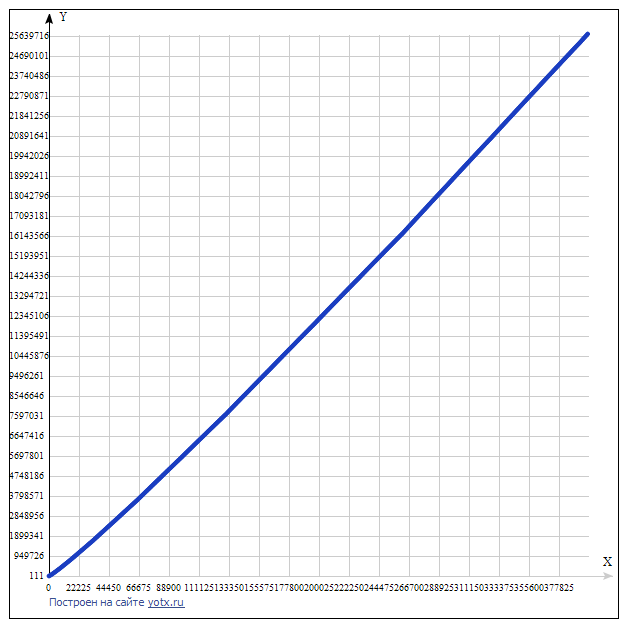
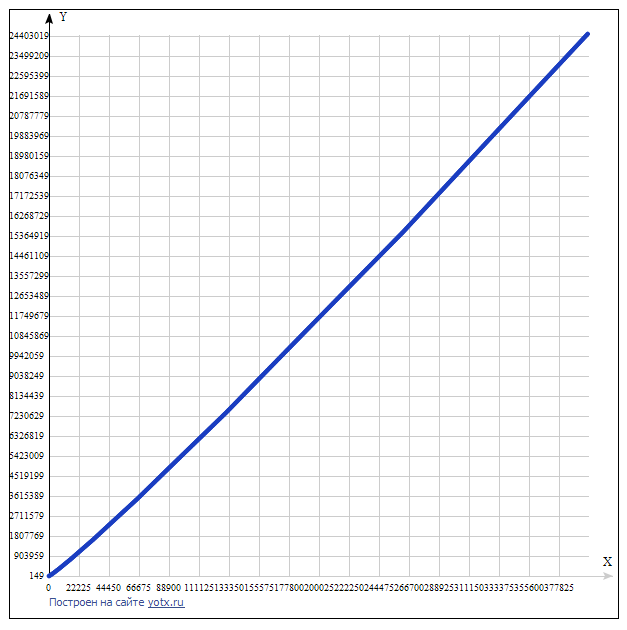
 

График зависимости сравнений от длины массива График зависимости присвоений от длины массива

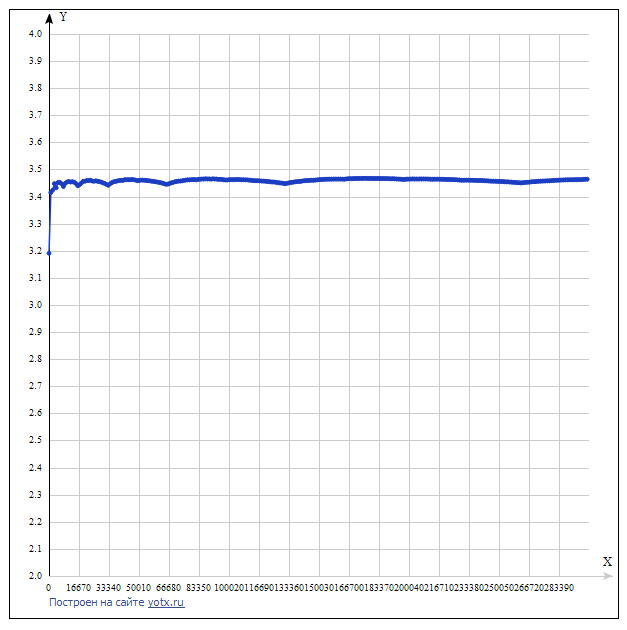
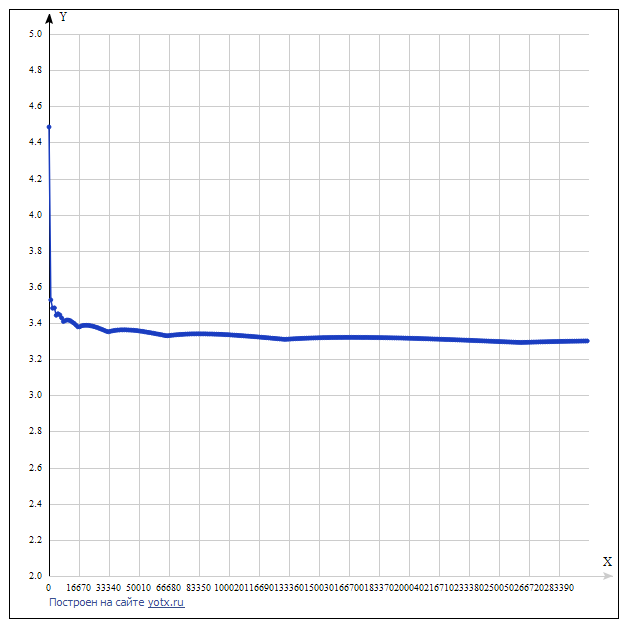
 

График зависимости сравнений от длины массива График зависимости присвоений от длины массива

после деления на после деления на

**Поразрядная сортировка**

Для проверки предполагаемой асимптотической сложности данной сортировки равной , где *k*-длинна 1 сортируемого числа в байтах, *m* – возможные значения в одном разряде построим график зависимости количества сравнений/присвоений от длины массива n. Если разделить каждое полученной значение на то получится график сходящийся к константе. Для сравнений эта константа равняется 2,9, а для присвоений равняется 8,22.

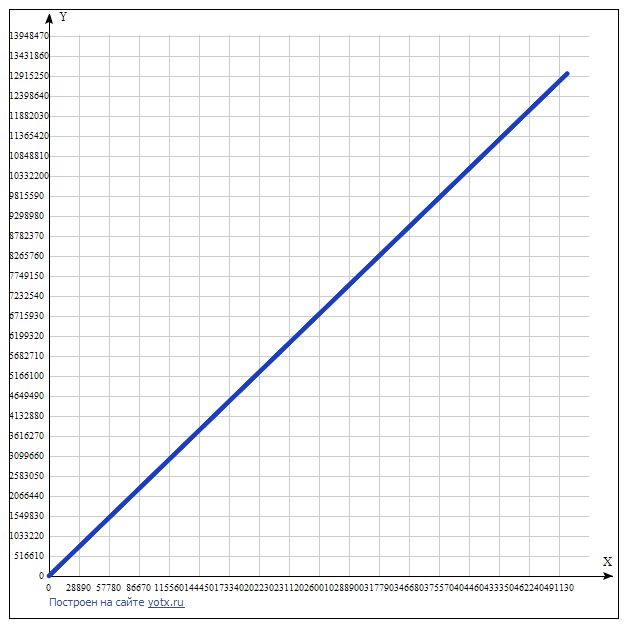
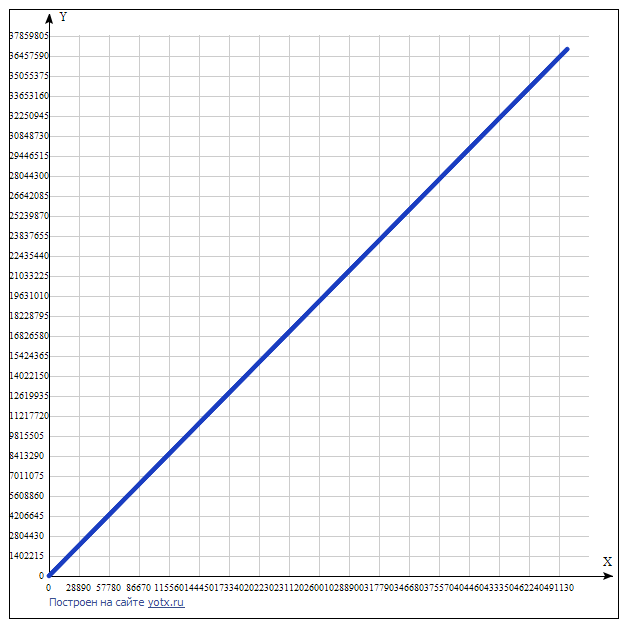
 

График зависимости сравнений от длины массива График зависимости присвоений от длины массива

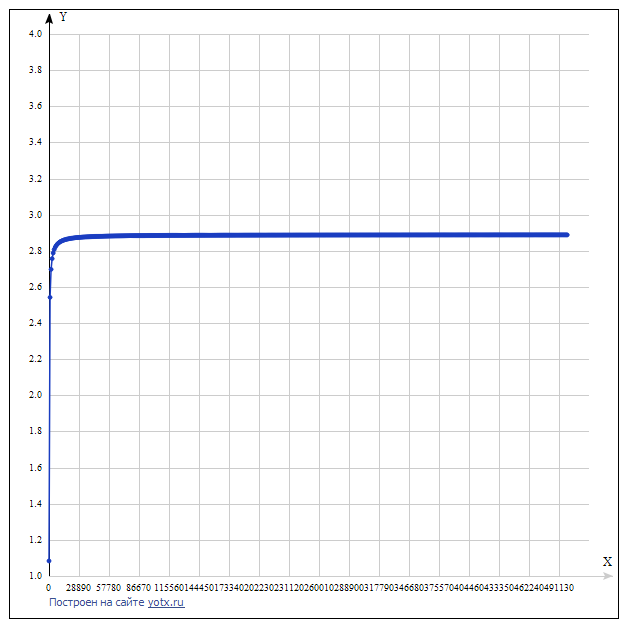
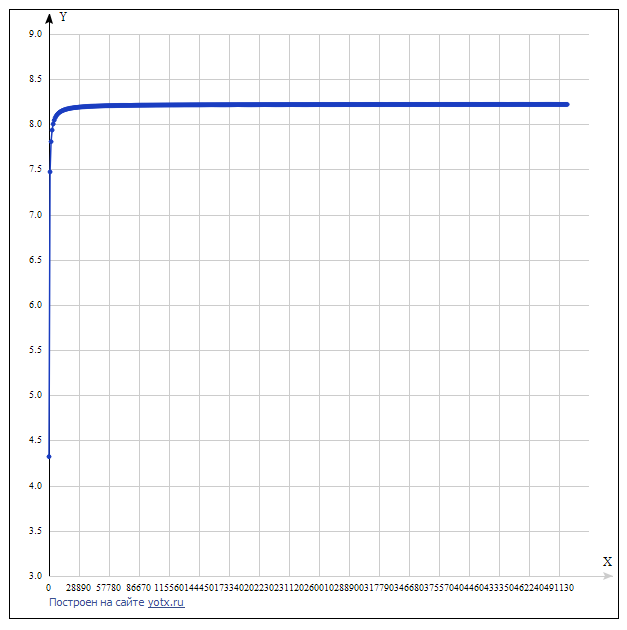
 

График зависимости сравнений от длины массива График зависимости присвоений от длины массива

после деления на после деления на

# Заключение

В ходе лабораторной работы были реализованы сортировка выбором, сортировка расчёской, сортировка слиянием и поразрядная сортировка на языке программирования Си и проведена проверка корректности. Были описаны алгоритмы работы данных сортировок и проведенные эксперименты для подтверждения их теоретический сложности.