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 3824Б1ПМ1

Котельников И. Е.

**Проверил**:

преподаватель каф. ВВСП,

Волокитин В.Д.

Нижний Новгород

2024

**Содержание**

[Постановка задачи 3](#_Toc26962562)

[Метод решения 4](#_Toc26962563)

[Руководство пользователя 5](#_Toc26962564)

[Описание программной реализации 6](#_Toc26962565)

[Подтверждение корректности 7](#_Toc26962566)

[Результаты экспериментов 8](#_Toc26962567)

[Заключение 9](#_Toc26962568)

[Приложение 10](#_Toc26962569)

# Постановка задачи

В рамках лабораторной работы требовалось реализовать на языке C 4 вида сортировок: вставками, Шелла, слиянием и поразрядную (LSD) для случайных массивов типа данных float. Оценить зависимость времени работы программы и количества операций, производимых ей, от размера массива.

# Метод решения

Первый алгоритм - сортировка вставками (Insertion Sort). Запускается внешний цикл, который проходит по элементами массива (с 1 по последний и шагом равным 1) и сравнивает текущий с предыдущим. Изначально текущий элемент помещается в буфер. Если текущий меньше предыдущего, то запускается внутренний цикл, который идет к началу массива, сравнивая текущий элемент (внутреннего цикла) с буферным. Если буферный меньше текущего, то элемент после текущего (т.е j+1, тогда как текущий это j) получает значение текущего. Если значение текущего меньше буферного, то следующий после текущего элемент (j + 1) получает значение буфера. Аналогично происходит, если внутренний цикл доходит до начала массива. После внутренний цикл прерывается.

Второй алгоритм – сортировка Шелла. Принцип действия аналогичен вставкам, только вместо шага равного 1 используются разные шаги. Первое значение равно длине массива с вычетом 1, это сортирует два крайних элемента. Дальше шаг уменьшается в 2 раза вплоть до момента, когда станет 1. Этот принцип позволяет переместить самые большие элементы в конец, а самые малые в начало, чтобы к моменту, когда шаг станет равным 1 потребовалось меньше итераций внутреннего цикла.

Третий алгоритм – сортировка слиянием. Принцип работы выбран итеративный. Программа сортирует массив путем объединения двух уже отсортированных в один отсортированный. Сначала сортируются пары массивов с единичной длиной, дальше с длиной 2 и так далее по степеням 2 в зависимости от длины исходного массива. Неспаренные пары не берутся в расчет и не сортируются, так как каждая неспаренная пара является уже отсортированным массивом. В конечном итоге получится 2 отсортированных массива которые объединятся в один итоговый, также отсортированный.

Четвертый алгоритм – поразрядная сортировка. Часть памяти с базовым массивом присваивается указателю типа char, длина которого в 4 (так как во float 4 байта) раза больше длины базового. Далее применяется функция индекса суммы, которая подсчитывает количество значений char в каждом 4-ом элементе указателя char, начиная с element-ого (значение этой переменной от 0 до 3, в зависимости от номера итерации сортировки). После происходит подсчет индекса суммы и в каждом из 256 элементов массива индекса суммы указывается номер элемента вспомогательного массива (длинной с базовый), которому будет присваиваться значение базового массива, имеющее в element-ом байте соответствующее значение. После присвоения индекс увеличивается на 1. Далее согласно данным массива индекса суммы все элементы базового массива перетекают во вспомогательный. После аналогичная операция происходит с перетеканием из вспомогательного в базовый, с увеличением element на 1. Так происходит 2 итерации. После сортировки все элементы, чей индекс суммы в последней итерации равен 127 и более инверсируются и вставляются в начало массива.

# Руководство пользователя

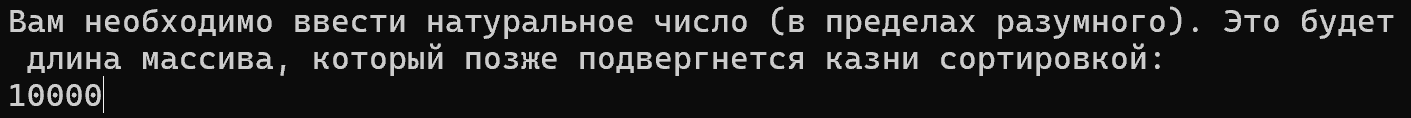
1. После запуска программа просит ввести число, которое после будет длиной массива. Необходимо ввести натуральное число в пределах разумного (чтобы после хватило памяти на хранение и обработку данных):  
   

Рис. 1 Иллюстрация к пункту 1

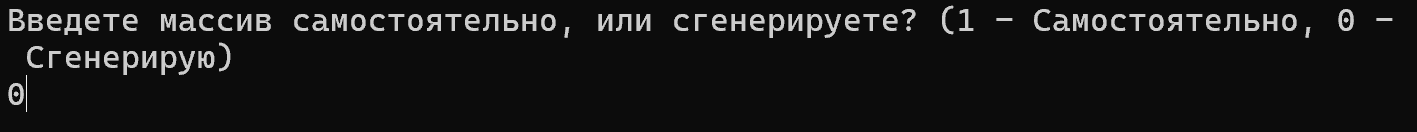
1. Далее программа спрашивает, введет ли пользователь элементы массива самостоятельно, или сгенерирует его. Если пользователь хочет ввести массив, он должен напечатать 1, если нет, то он должен напечатать 0:  
   

Рис. 2 Иллюстрация к пункту 2

1. Программа спрашивает, какую из 4 сортировок пользователь хочет применить к созданному массиву. На выбор есть 4 варианта, нужно написать номер одного из них, соответствующий числу, стоящему напротив названия сортировки в предоставленном программой списке:  
   

Рис. 3 Иллюстрация к пункту 3

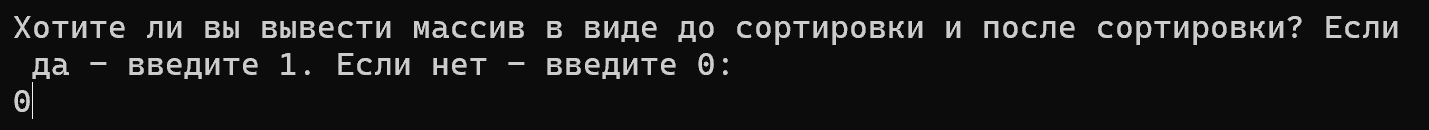
1. Программа спрашивает, желает ли пользователь вывести массив до и после сортировки для визуального наблюдения результата. НУжно ввести 0 для отказа и 1 для согласия:  
   

Рис. 4 Иллюстрация к пункту 4

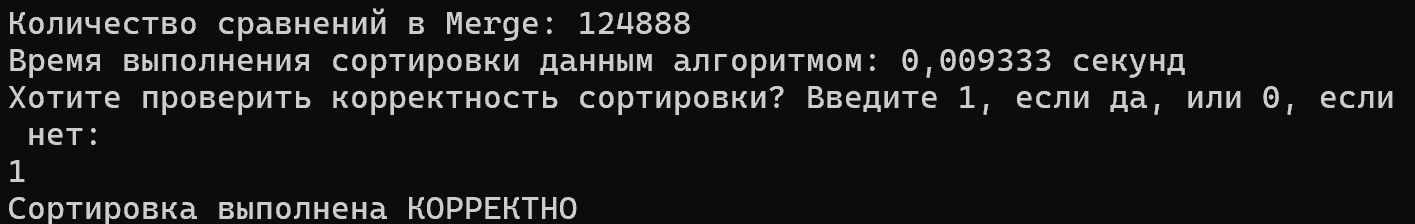
1. Программа выводит количество операций, совершенных в сортировке выбранного массива, а также время, за которое была выполнена сортировка. После она отправляет запрос за желание пользователя проверить корректность сортировки. Если пользователь желает провести проверку - вводит 1, иначе вводит 0. После введения решения пользователя выводится (или не выводится, если пользователь отказался) результат:  
   

Рис. 5 Иллюстрация к пункту 5

# Описание программной реализации

Функция insertionSort() получает на вход указатель на массив, количество элементов и сортирует его вставками, также считая количество сравнений.

Функция shellSort() получает на вход указатель на массив, количество элементов и сортирует его Шеллом, также считая количество сравнений.

Функция mergeSort() получает на вход указатель на массив, количество элементов и сортирует его слиянием, также считая количество сравнений.

Функция radixSort() получает на вход указатель на массив, количество элементов и сортирует его поразрядно, также считая количество присваиваний.

Функция indexSummCreate() получает на вход указатель на массив, количество элементов, шаг, номер рассматриваемого элемента и указатель, в который после срабатывания поместит индекс суммы.

Функция correctCheck() получает на вход указатель на копию первоначального массива, указатель на массив и длину массива, после чего проверяет массив на отсортированность и соответствие исходному массиву по количеству и содержанию.

Функция main() запускает функции выше, (порядок согласно руководству пользователю) и возвращает 0.

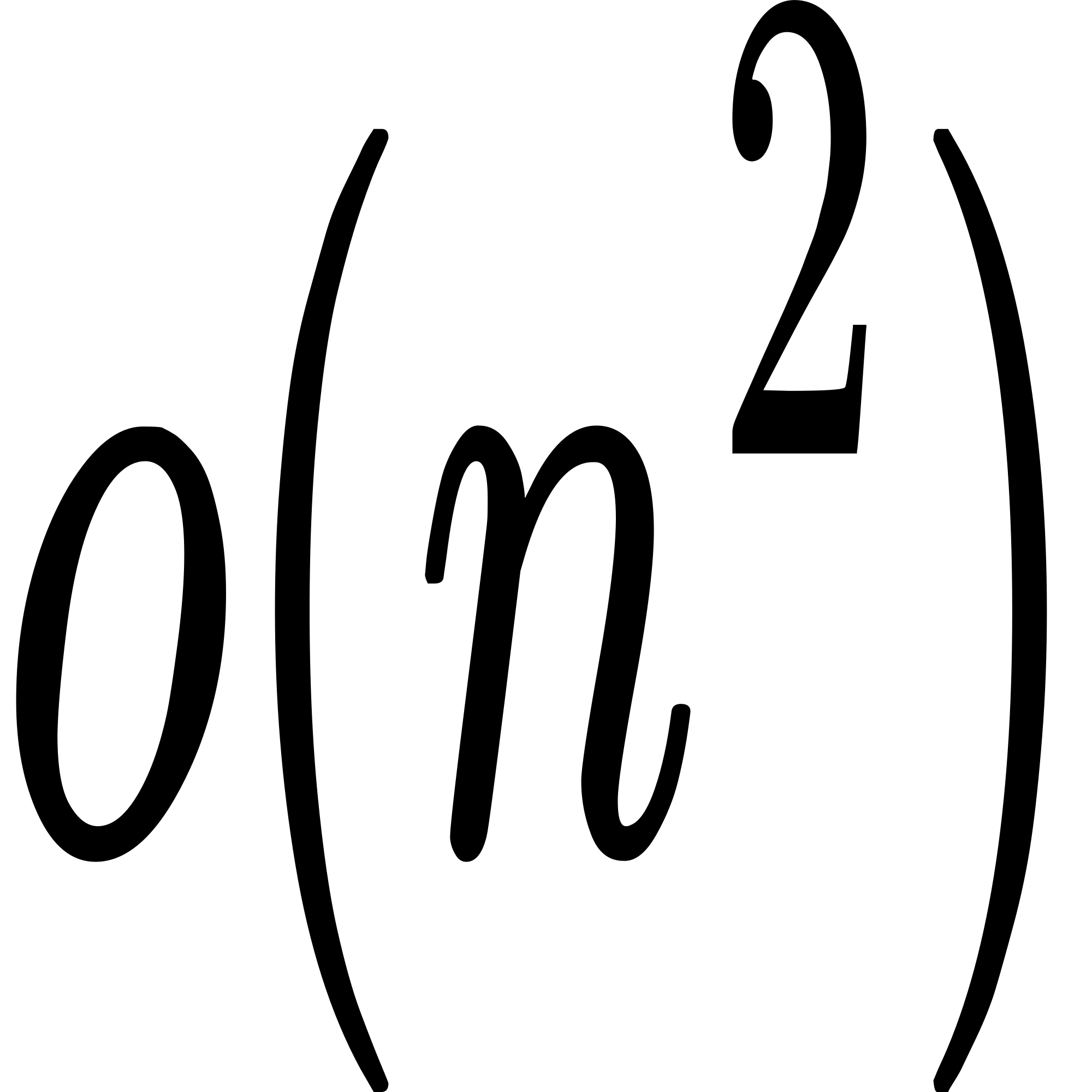
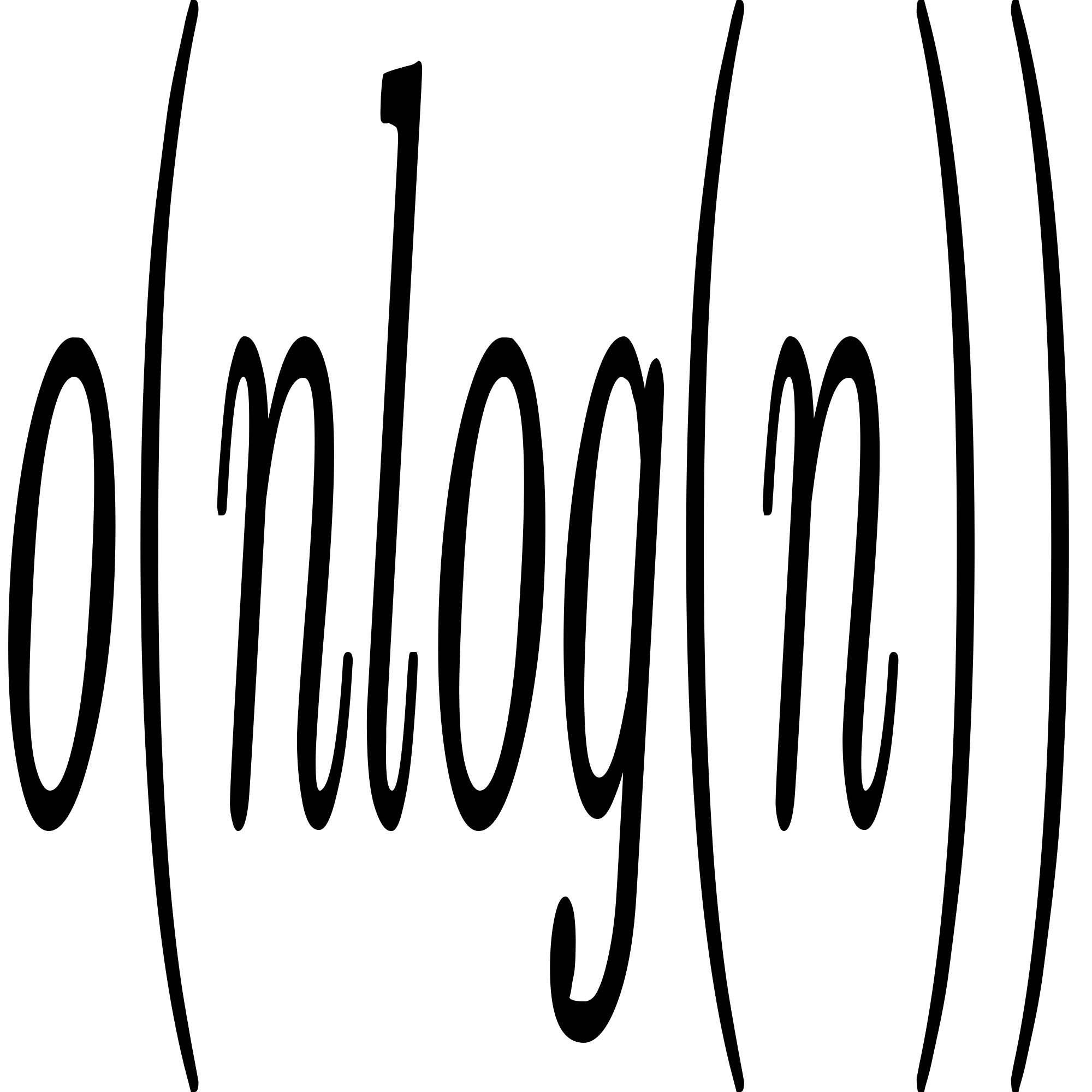
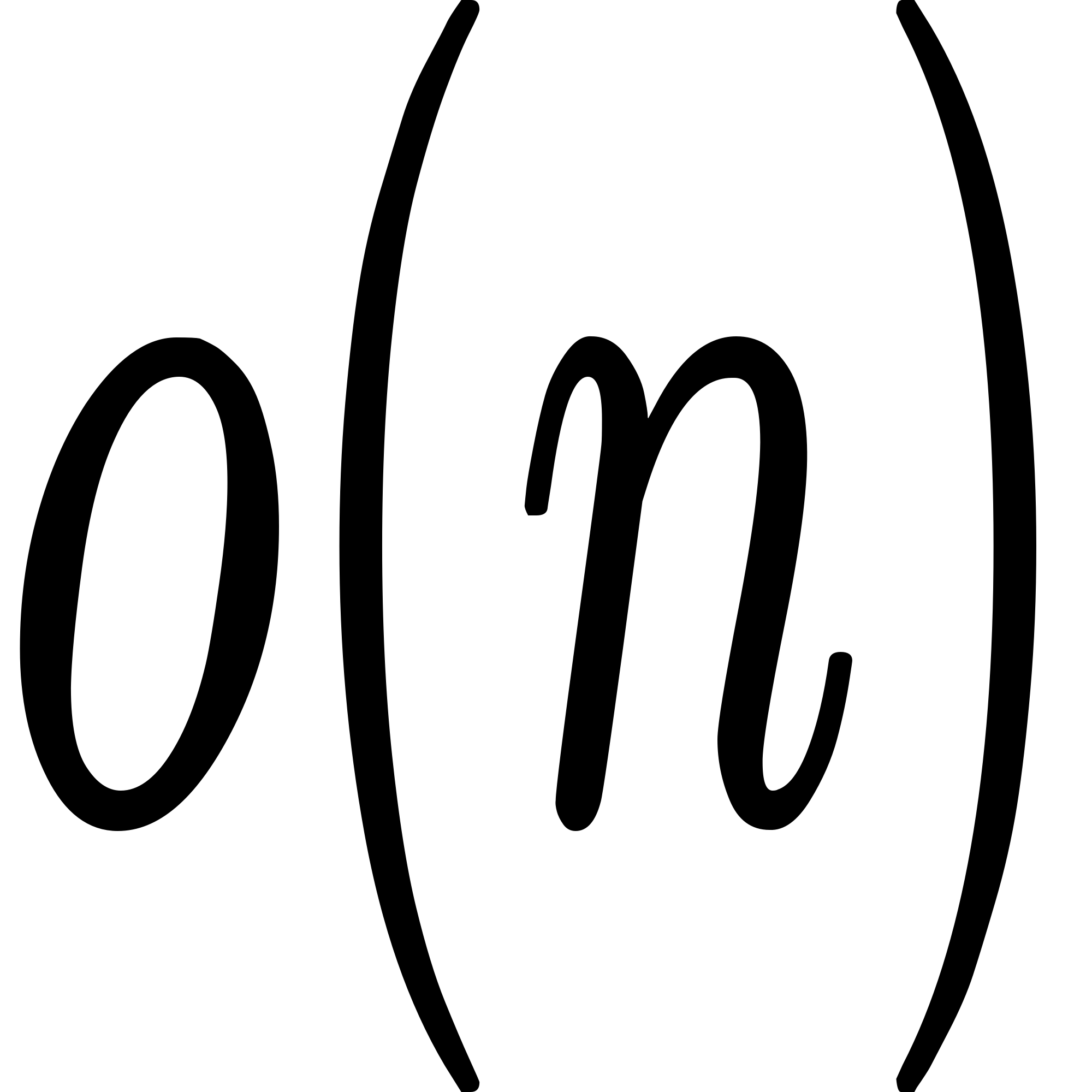
# Подтверждение корректности

Для подтверждения корректности в программе присутствует проверка. Массив проверяется на соответствие изначальному по количеству и составу элементов, а также на неудовлетворение условию [i] > [i + 1].

# Результаты экспериментов

В результате экспериментов каждая сортировка была проверена на удовлетворение условию своей теоретической сложности. Для этого были построены графики зависимости количества операций и времени выполнения программы от количества элементов входного массива. Для проверки каждая из сортировок была проверена на 300 разных случайно сгенерированных массивах.

Каждая из сортировок имеет следующую сложность:

1. Вставки - 
2. Шелла - wps
3. Слияние - 
4. Поразрядная - 

Если график зависимости горизонтален, то программа соответствует своей теоретической сложности.

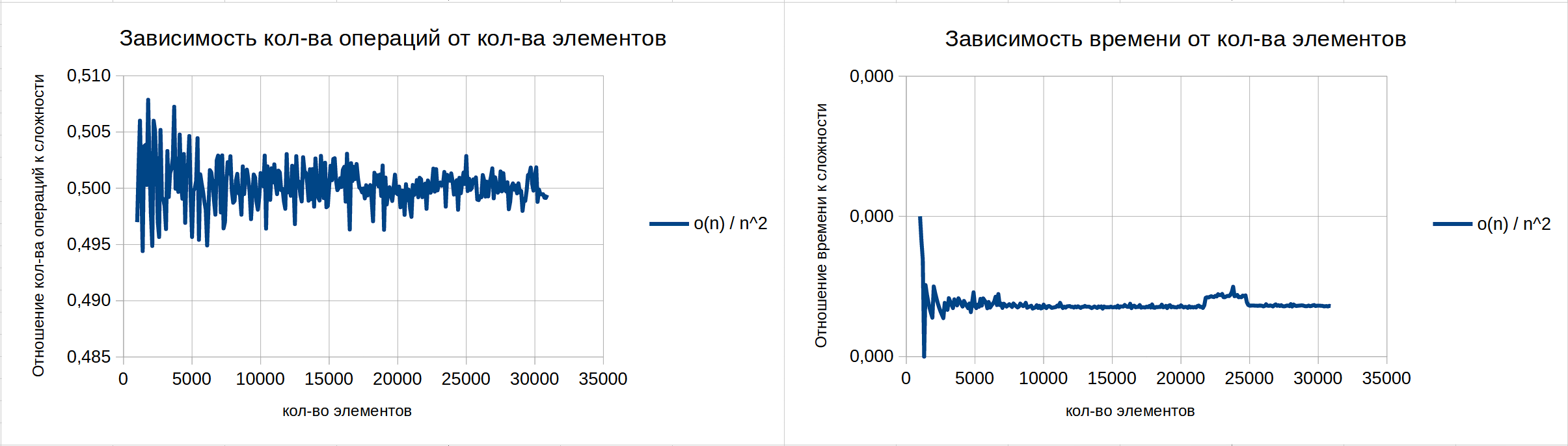


Рис. 6 Графики зависимости для сортировки Вставками.

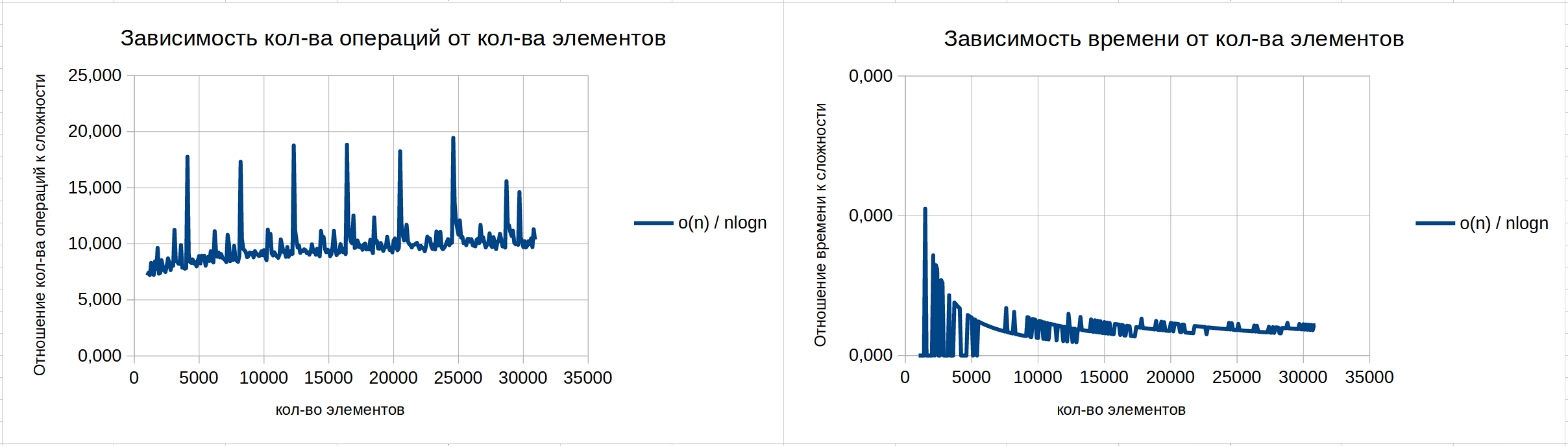


Рис. 7 Графики зависимости для сортировки Шелла.

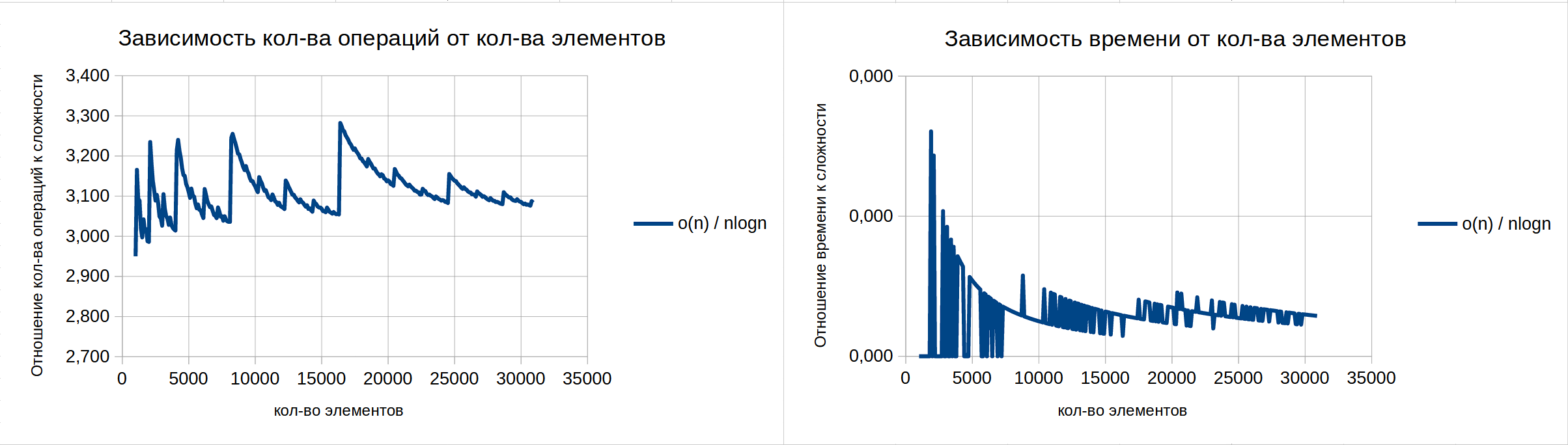


Рис. 8 Графики зависимости для сортировки Слиянием.

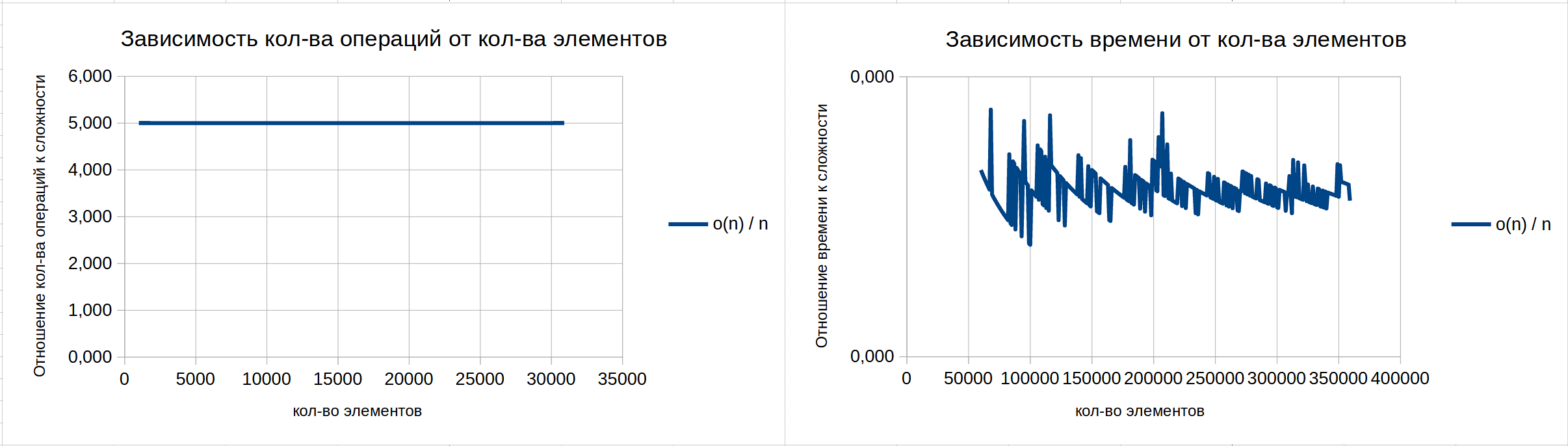
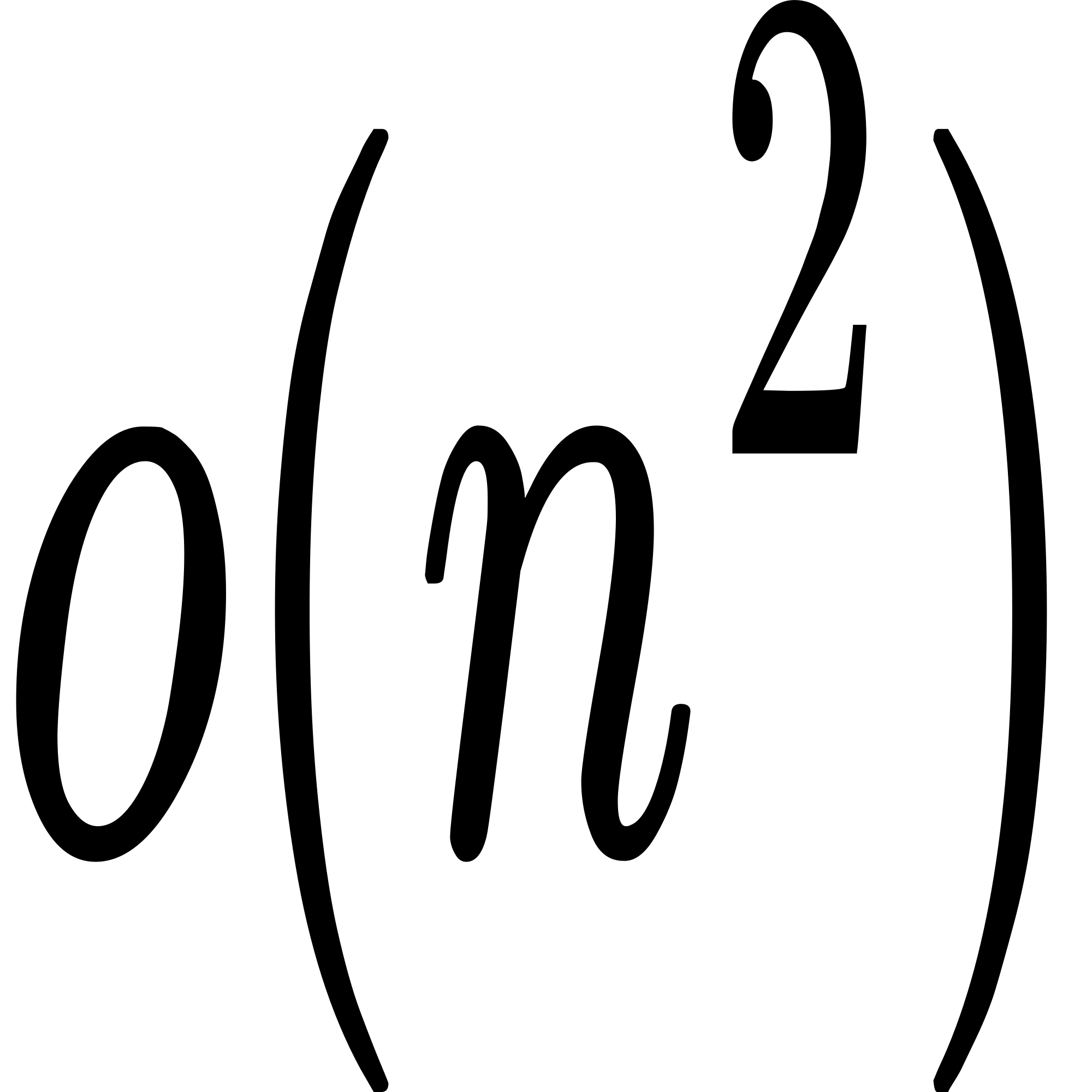
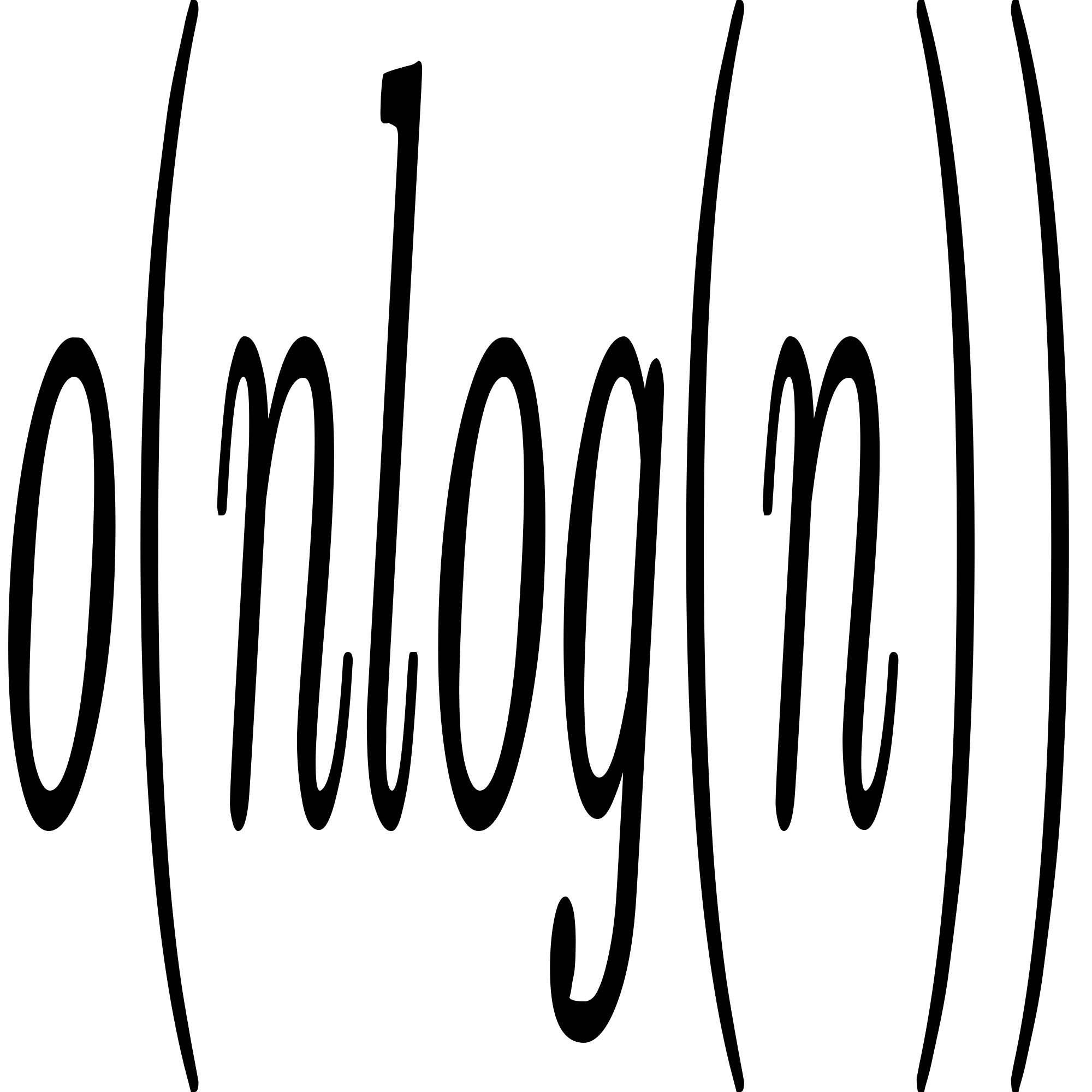
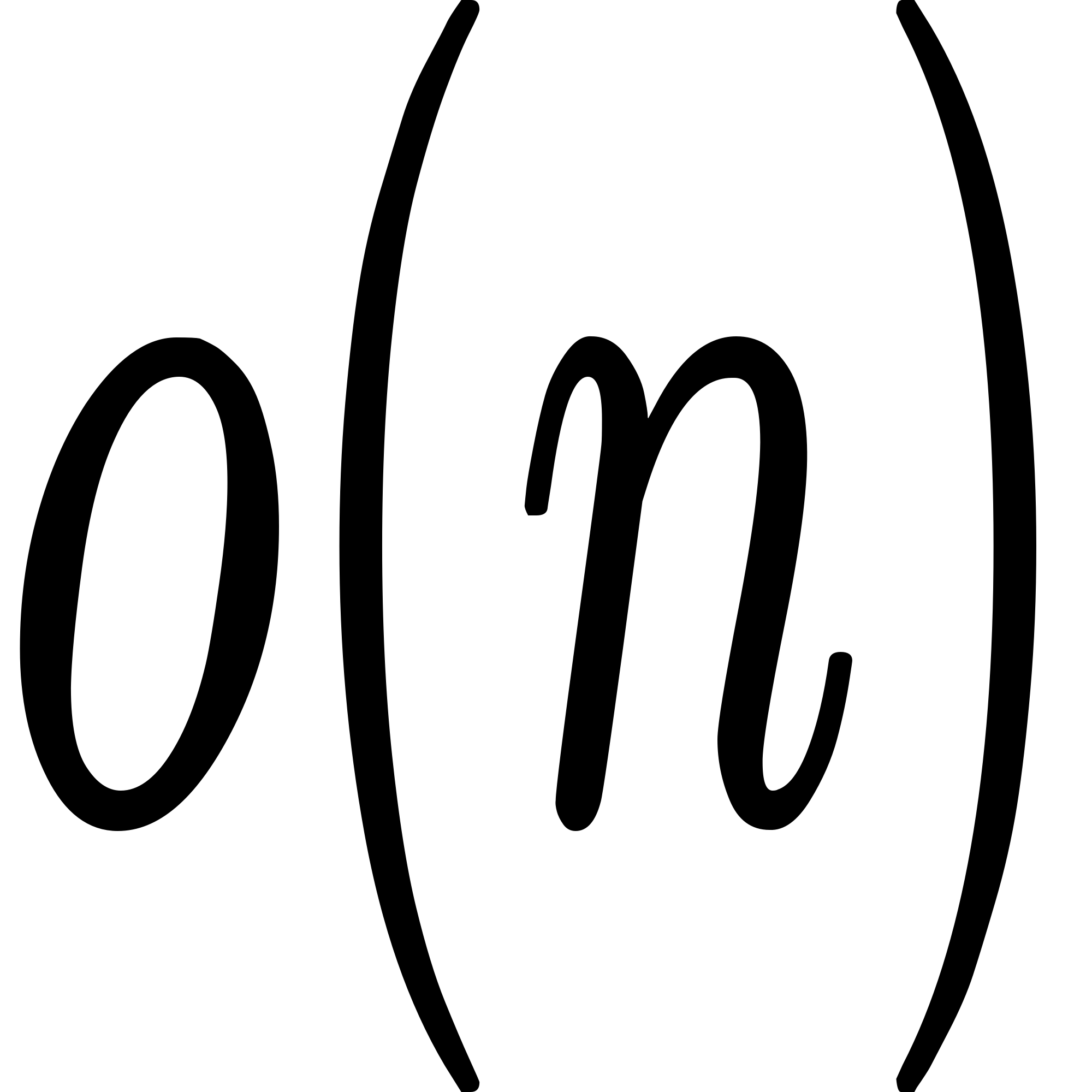


Рис. 9 Графики зависимости для Поразрядной сортировки.

На графиках заметно, что постепенно значения дроби, отмечающиеся на вертикальной оси начинают отличаться последующее от предыдущего все меньше и меньше, постепенно образуя горизонтальную линию (в случае сортировки LSD левый график это чисто линия). Это значит, что отношение данных величин сохраняется и равняется примерно одному значению, т.е предполагаемая теоретическая сложность является фактической сложностью соответствующего алгоритма.

# Заключение

В рамках лабораторной работы были реализованы 4 алгоритма сортировки: сортировка вставками, Шелла, слиянием и поразрядная. Результатом является программа, которая позволяет запустить любой из этих алгоритмов, для массива, случайного или введенного с клавиатуры, проверить корректность полученного результата и изучить зависимость времени работы или количества проведенных операция сортировкой, от длины массива. Исследования этих алгоритмов позволяют прийти к выводу, что, в данной реализации, сортировка вставками работает за  Шелла и слияниями за причем, сортировка слияниями работает быстрее, а поразрядная - за 

# Приложение

Поразрядная сортировка:

int\* indexSummCreate(float\* basicArr, int baseLen, int step, int element, int\* indexSumm) {

int tempBefore = 0;

int tempAfter = 0;

unsigned char\* internalArr;

memset(indexSumm, 0, 256 \* sizeof(int));

internalArr = (unsigned char\*)basicArr;

for (int i = 0; i < baseLen; i++) {

indexSumm[internalArr[i \* step + element]]++;

}

tempBefore = indexSumm[0];

indexSumm[0] = 0;

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

tempAfter = indexSumm[i];

indexSumm[i] = indexSumm[i - 1] + tempBefore;

tempBefore = tempAfter;

}

return indexSumm;

}

float\* radixSort(float\* basicArr, int baseLen, int number) {

int finalCount = 0;

long long int count = 0;

float\* supportArr;

int\* indexSumm = (int\*)malloc(256 \* sizeof(int));

unsigned char\* internalArr;

int step = 4;

int element = 0;

supportArr = (float\*)malloc(baseLen \* sizeof(float));

for (int i = 0; i < 2; i++) {

internalArr = (unsigned char\*)basicArr;

indexSumm = indexSummCreate(basicArr, baseLen, step, element, indexSumm);

for (int j = 0; j < baseLen; j++) {

supportArr[indexSumm[internalArr[j \* step + element]]] = basicArr[j];

count++;

indexSumm[internalArr[j \* step + element]]++;

}

element++;

internalArr = (unsigned char\*)supportArr;

indexSumm = indexSummCreate(supportArr, baseLen, step, element, indexSumm);

for (int j = 0; j < baseLen; j++) {

basicArr[indexSumm[internalArr[j \* step + element]]] = supportArr[j];

count++;

indexSumm[internalArr[j \* step + element]]++;

}

element++;

}

for (int i = baseLen - 1; i > indexSumm[127] - 1; i--) {

supportArr[finalCount] = basicArr[i];

count++;

finalCount++;

}

for (int i = 0; i < indexSumm[127]; i++) {

supportArr[finalCount] = basicArr[i];

count++;

finalCount++;

}

basicArr = supportArr;

if (number == 3)

printf("\n Количество присваиваний в Radix: %d\n", count);

return basicArr;

}

Сортировка слиянием:

float\* mergeSort(float\* basicArr, int baseLen, int number) {

float\* supportArr;

long long int count = 0;

supportArr = (float\*)malloc(baseLen \* sizeof(float));

int order = 1, iFirst, iSecond, ending, orderSecond, orderFirst;

for (int j = 0; j < baseLen; j++) {

supportArr[j] = basicArr[j];

}

while (order < baseLen) {

count++;

for (int i = 0; i < ((baseLen + order - 1) / (order \* 2)); i++) {

iFirst = 0;

iSecond = 0;

ending = 0;

orderFirst = order;

orderSecond = order - ((i \* order \* 2 + order \* 2) / (baseLen + 1)) \* (order - (baseLen % order));

for (int k = i \* order \* 2; iFirst < orderFirst && iSecond < orderSecond; k++) {

if (basicArr[i \* order \* 2 + iFirst] < basicArr[i \* order \* 2 + order + iSecond]) {

supportArr[k] = basicArr[i \* order \* 2 + iFirst];

iFirst++;

}

else {

supportArr[k] = basicArr[i \* order \* 2 + order + iSecond];

iSecond++;

}

ending = k + 1;

count++;

}

for (int k = iFirst; k < orderFirst; k++) {

supportArr[ending] = basicArr[i \* order \* 2 + k];

ending++;

}

for (int k = iSecond; k < orderSecond; k++) {

supportArr[ending] = basicArr[i \* order \* 2 + order + k];

ending++;

}

}

for (int j = 0; j < baseLen; j++) {

basicArr[j] = supportArr[j];

}

order \*= 2;

}

if (number == 3)

printf("\n Количество сравнений в Merge: %d\n", count);

return basicArr;

Сортировка Шелла:

float\* shellSort(float\* basicArr, int baseLen, int number) {

float temp;

int k;

int step = baseLen - 1;

long long int count = 0;

while (step > 0) {

count++;

for (int i = step; i < baseLen; i++) {

temp = basicArr[i];

k = i - step;

while (k >= 0 && basicArr[k] >= temp) {

basicArr[k + step] = basicArr[k];

k -= step;

count += 2;

}

count++;

k += step;

basicArr[k] = temp;

}

step /= 2;

}

if (number == 3)

printf("\n Количество сравнений в Шелла: %d\n", count);

return basicArr;

}

Сортировка вставками:  
float\* insertionSort(float\* basicArr, int baseLen, int number) {

float temp;

int j;

long long int count = 0;

for (int i = 1; i < baseLen; i++) {

temp = basicArr[i];

j = i - 1;

while (j >= 0 && basicArr[j] >= temp) {

basicArr[j + 1] = basicArr[j];

j--;

count += 2;

}

j++;

count++;

basicArr[j] = temp;

}

if (number == 3)

printf("\n Количество сравнений во вставках: %d\n", count);

return basicArr;

}