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

Ермольев Т.Д.

**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

Написать 4 алгоритма сортировки (bubble, shell, radix, merge) для массивов из N 32-битных чисел с плавающей запятой (float). Число N задается пользователем в начале выполнения программы. Сравнить скорость выполнения алгоритма и количество операций проделанных алгоритмом с теоретической сложностью алгоритма.

# Метод решения

Описание алгоритмов ваших сортировок. Достаточно подробное

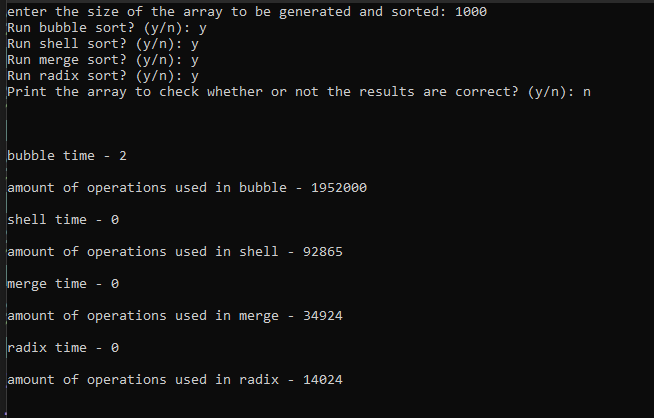
Сортировка пузырьком - рядом стоящие элементы перебираются с 1-го и 2-го по n-1-ый и n-ый, и меняются местами, если левый элемент больше правого. Алгоритм продолжает пробегать массив до тех пор пока элементы не будут больше менятся местами после завершения цикла (другими словами, когда 1 элемент будет меньше 2го, а 2ой меньше 3го, т.д.). Сложность алгоритма равна

Сортировка Шелла - улучшенный вариант сортировки вставками, где элемент «тащится» не с шагом 1, а с постепенно уменьшающимся шагом (причем последний шаг всегда равен 1, чтобы массив правильно отсортировался). Шаги в данной реализации взяты по схеме (3-smooth numbers), что даёт сложность равную

Сортировка слиянием - массив рекурсивно делится пополам на подмассивы, до тех пор пока в каждом не останется по одному элементу. После этого пары подмассивов из одного элемента собираются в один отсортированный из двух. Далее пары подмассивов из двух элементов собираются в один из четырех и т.д. до того пока весь массив не будет собран заново и отсортирован. Сложность -

Поразрядная сортировка - числа сортируются по одному байту сортировкой подсчетом справа налево. Берётся первый байт каждого числа (для этого используется конвертация указателя в char\*, т.к. размер char - 1 байт), считается кол-во вхождений каждого из 256 возможных значений байта, и оригинальный массив восстанавливается по префиксной сумме этих вхождений. Процесс повторяется пока все байты не будут пройдены. Из-за того что самый правый бит в float является знаковым, все отрицательные элементы массива будут в конце, в обратном порядке. Это можно исправить простым дополнительным проходом через весь массив. Сложность -

# Руководство пользователя

Программа запрашивает N, какие сортировки пользователь хочет проверить, а также предлагает возможность распечатать массив для проверки корректности. Она автоматически создаёт массив, и выводит время потраченное на каждую сортировку и количество операций произведённых каждой сортировкой.

# Описание программной реализации

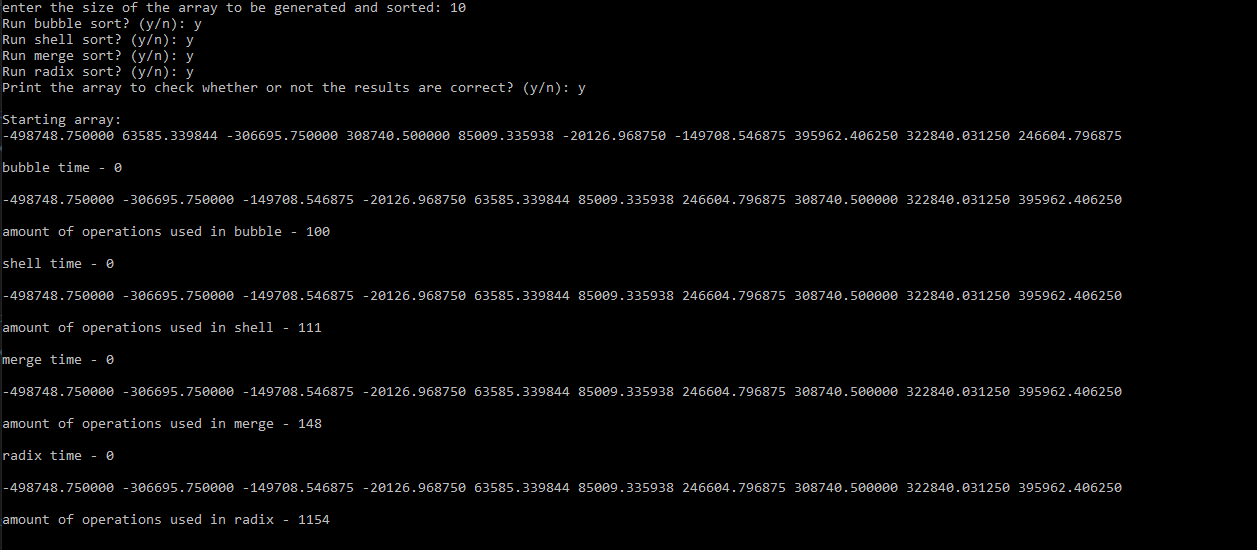
sortingalgs.cpp, для создания использовался IDE Visual Studio и компилятор MSVC.

Структура программы:

* Функция count – подпрограмма сортировки radix, подсчитывающая и сортирующая один бит числа.
* Функция radix – поразрядная сортировка
* Функция bubble – сортировка пузырьком
* Функция shell – сортировка Шелла
* Функция merge – подпрограмма сортировки слиянием, объединяющая два отсортированных массива в один
* Функция mergesort – сортировка слиянием
* Функция smoothgen – функция генерирующая первые 300 “3-гладких чисел”, необходимых для сортировки Шелла
* Функция main – главная функция программы, которая запрашивает ввод пользователя, создаёт массив, вызывает функции сортировок и выводит их результаты.

# Подтверждение корректности

Для подтверждения корректности в программе можно распечатать каждый массив при вводе “y” на «Print the array to check whether or not the results are correct? (y/n):» после запуска программы. Распечатается оригинальный массив и каждый из отсортирован ных массивов



# Результаты экспериментов

По данным экспериментов видно, что скорость выполнения алгоритмов соотвествует их теоретической сложности. Прикрепляю график скоростей выполнения каждого из алгоритмов. Точками отмечены скорость алгоритма (в тиках) при определённых N (от 5000 до 5000000). Линии - теоретическая сложнность данного алгоритма (). Значения количества использованных операций создают похожую корреляцию

Заметьте -- оси графика растут экспоненциально, чем объясняется «прямота» функций.

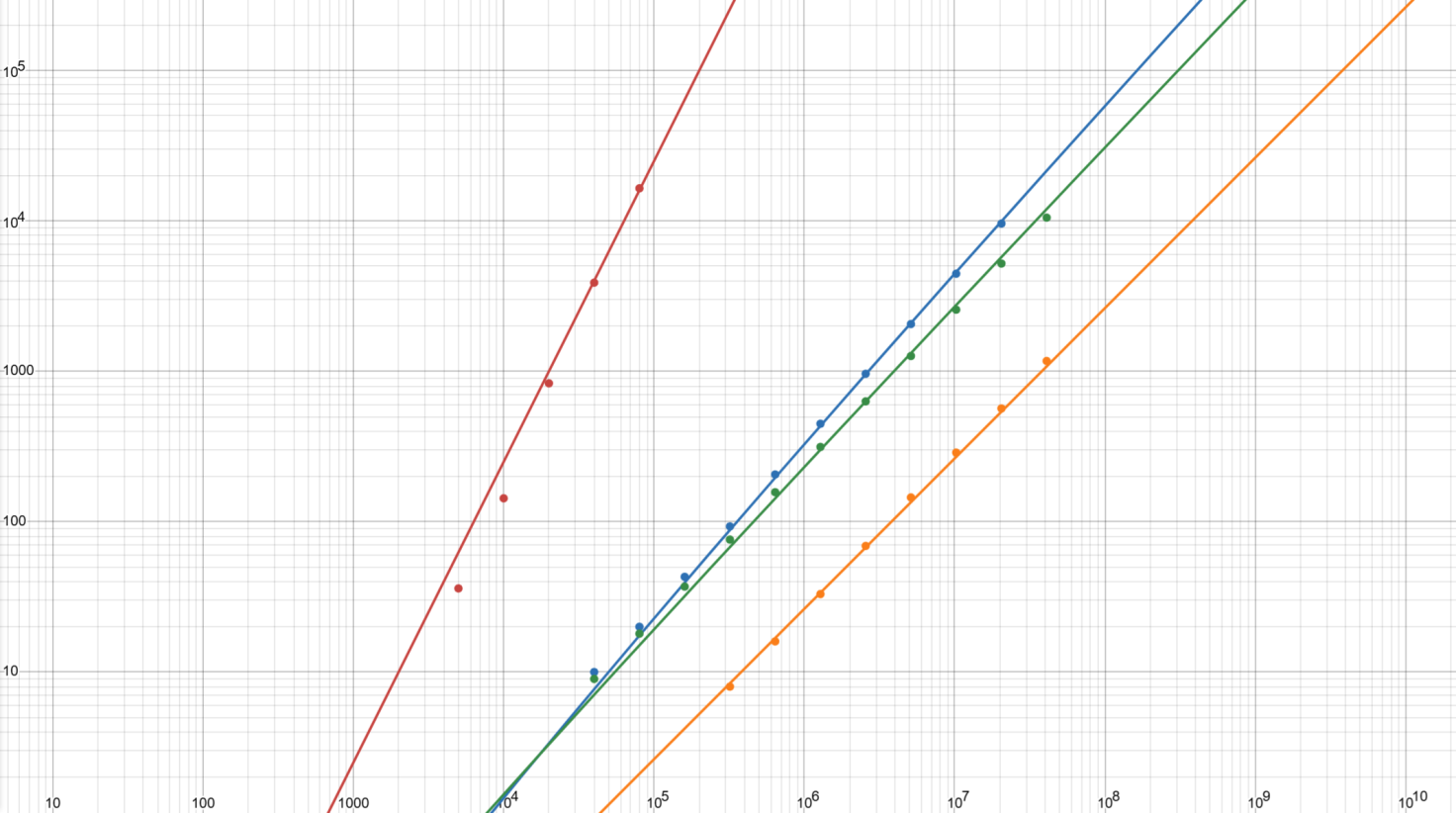
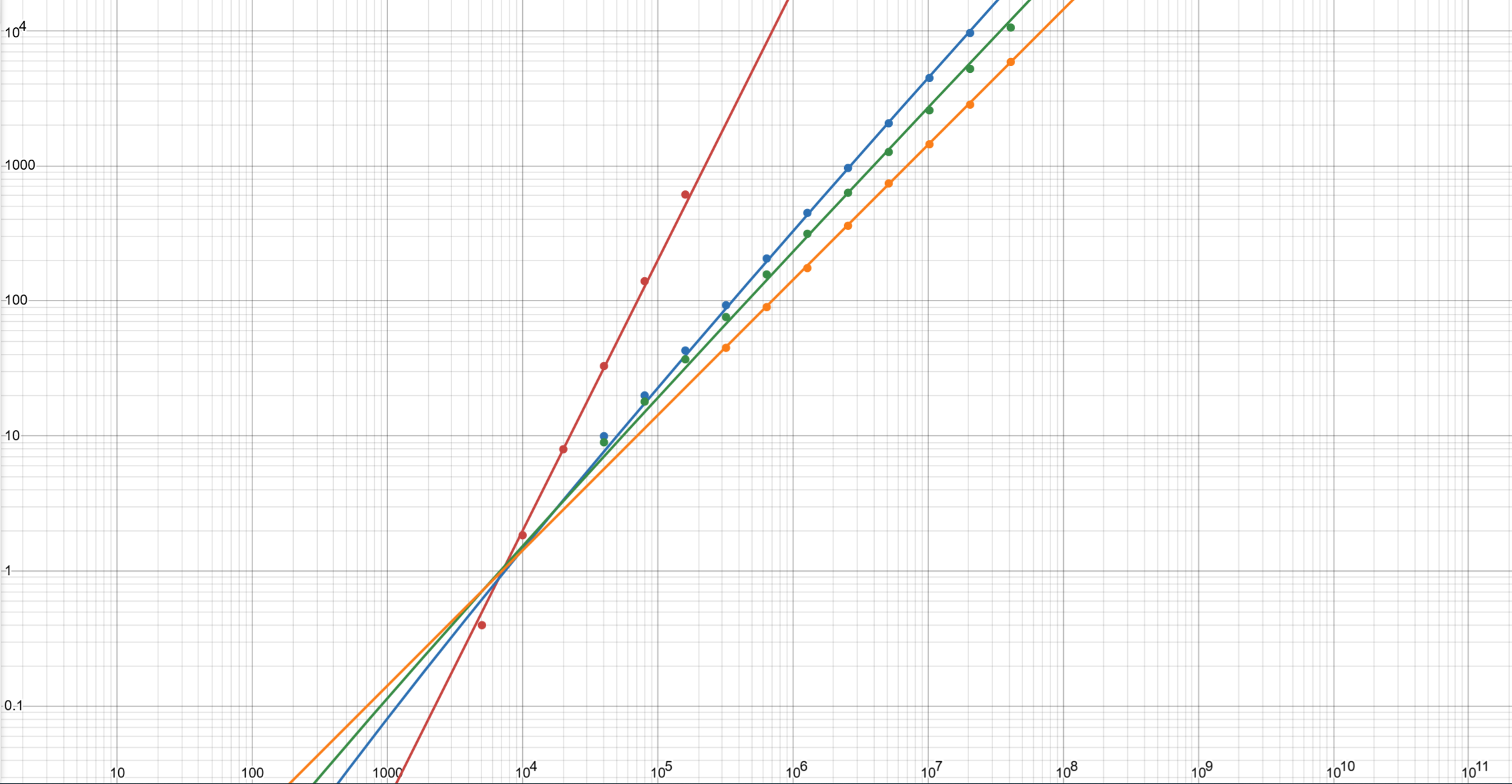


График результатов выполнения алгоритмов 1: красный - пузырёк и O(N^2), синий - Шелл и O(N\*logn\*logn), зелёный - слияние и O(n\*logn), желтый - поразрядная и O(n)

Чтобы доказать что графики действительно растут с разными скоростями, домножим никоторые из них на константу, чтобы они пресекались в одной точке. Видно, что угол наклона всех 4-ёх графиков разный

# Заключение

Были написаны все 4 алгоритма сортировки, для массивов из N чисел с плавающей запятой. Резульаты сортировок были проверены, и было найдено, что они действительно соответствуют своей теоретической сложности.

# Приложение

void count(unsigned char\* ar, int n, int period)

{

#ifdef count\_checks

radixchecks += 256 + n + n;

#endif // count\_checks

int i;

int\* cnt = (int\*)calloc(256, sizeof(int));

float\* ans = (float\*)calloc(n, sizeof(float));

for (i = 0; i < n; i++) cnt[ar[i \* sizeof(float) + period]]++;

for (i = 1; i < 256; i++) cnt[i] += cnt[i - 1];

for (i = n - 1; i >= 0; i--) ans[--cnt[ar[i \* sizeof(float) + period]]] = ((float\*)ar)[i];

for (i = 0; i < n; i++) ((float\*)ar)[i] = ans[i];

free(cnt);

free(ans);

}

void radix(float\* ar, int n)

{

#ifdef count\_checks

radixchecks += 2 \* n + 2 \* n + n;

#endif // count\_checks

int i, period, ptr;

float\* ans;

for (i = 0; i < sizeof(float); i++)

{

count((unsigned char\*)ar, n, i);

}

ans = (float\*)calloc(n, sizeof(float));

ptr = 0;

for (i = n - 1; i >= 0; i--) { if (ar[i] < 0) { ans[ptr] = ar[i]; ptr++; } }

for (i = 0; i < n; i++) { if (ar[i] >= 0) { ans[ptr] = ar[i]; ptr++; } }

for (i = 0; i < n; i++) { ar[i] = ans[i]; }

free(ans);

}

void bubble(float\* ar, int n)

{

int i, haschanged = 1;

float swap;

while (haschanged)

{

haschanged = 0;

#ifdef count\_checks

bubblechecks += n + n;

#endif // count\_checks

for (i = 0; i < n - 1; i++)

{

if (ar[i] > ar[i + 1])

{

haschanged = 1;

swap = ar[i];

ar[i] = ar[i + 1];

ar[i + 1] = swap;

}

}

}

}

void shell(float\* ar, int n, int\* gaps, int gapsize)

{

int gapi, i, j;

float swap;

for (gapi = gapsize - 1; gapi >= 0; gapi--)

{

int gap = gaps[gapi];

for (i = gap; i < n; i++)

{

float swap = ar[i];

for (j = i; (j >= gap) && (ar[j - gap] > swap); j -= gap)

{

ar[j] = ar[j - gap];

}

ar[j] = swap;

#ifdef count\_checks

shellchecks += 3;

#endif // count\_checks

}

}

}

void merge(float\* ar, int n, int l, int mid, int r)

{

float\* copar = (float\*)malloc((r - l) \* sizeof(float));

int ptr = mid;

int coparptr = 0;

int i;

for (i = l; i < mid; i++)

{

while (ptr < r && ar[ptr] < ar[i]) { copar[coparptr++] = ar[ptr++]; }

copar[coparptr++] = ar[i];

}

while (ptr < r) { copar[coparptr++] = ar[ptr++]; }

for (i = l; i < r; i++) { ar[i] = copar[i - l]; }

#ifdef count\_checks

mergechecks += (r - l + 1) \* 3;

#endif // count\_checks

}

void mergesort(float\* ar, int n, int l, int r)

{

int mid = r - ((r - l) / 2);

#ifdef count\_checks

mergechecks++;

#endif // count\_checks

if (r - l <= 1) { return; }

mergesort(ar, n, l, mid);

mergesort(ar, n, mid, r);

merge(ar, n, l, mid, r);

}