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**

**«Вычисление арифметических выражений»**

**Выполнил**:

студент группы 382003-1

Мухин Г.С.

**Проверил**:

ассистент каф. МОСТ,

Волокитин В.Д.

Нижний Новгород

2020г.

**Содержание**

[Постановка задачи 3](#_Toc59802143)

[Метод решения 4](#_Toc59802144)

[Руководство пользователя 7](#_Toc59802145)

[Описание программной реализации 9](#_Toc59802146)

[Подтверждение корректности 10](#_Toc59802147)

[Результаты экспериментов 11](#_Toc59802148)

[Заключение 13](#_Toc59802149)

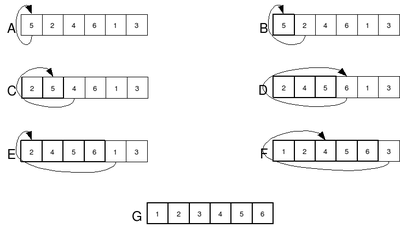
[Приложение 14](#_Toc59802150)

# Постановка задачи

При выполнении данной лабораторной работы, передо мной была поставлена задача написать программу на языке C, используя тип данных float, которая сортирует сгенерированный массив, с целью исследования различных алгоритмов сортировки («Сортировка вставками», «Сортировка расческой», «Сортировка слиянием», «Поразрядная сортировка»), а также для создания наглядного примера теоретической сложности алгоритмов, путем экспериментов с замером количества сравнений и перестановок при использовании каждой сортировки.

# Метод решения

**Сортировка вставками (Insertion Sort)**

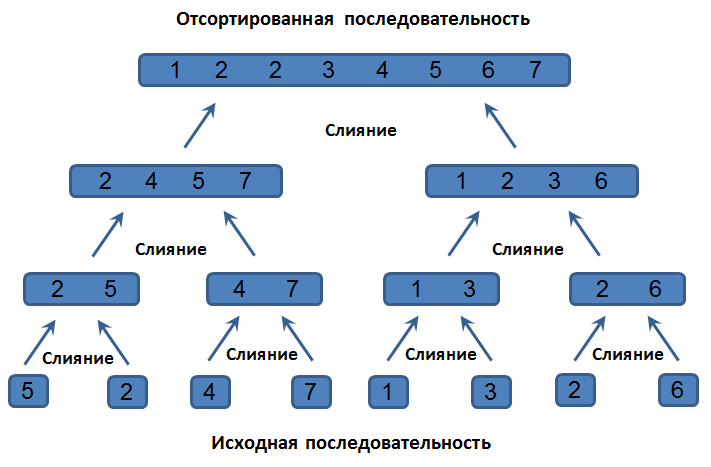
Идея алгоритма: сортировки вставками всегда делят массив на 2 части — отсортированную и неотсортированную. Из неотсортированной части извлекается любой элемент. Поскольку другая часть массива отсортирована, то в ней достаточно быстро можно найти своё место для этого извлечённого элемента. Элемент вставляется куда нужно, в результате чего отсортированная часть массива увеличивается, а неотсортированная уменьшается. Алгоритм выполняется до тех пор, пока набор входных неотсортированных данных не будет исчерпан.  


**Сортировка «расческой»(Comb Sort)**

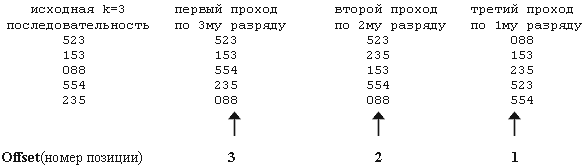
Основная идея «расчёски» в том, чтобы первоначально брать достаточно большое расстояние между сравниваемыми элементами и по мере упорядочивания массива сужать это расстояние вплоть до минимального. Таким образом, мы как бы причёсываем массив, постепенно разглаживая на всё более аккуратные пряди. Первоначальный разрыв между сравниваемыми элементами лучше брать с учётом специальной величины, называемой фактором уменьшения, оптимальное значение которой равно примерно 1,247. Сначала расстояние между элементами максимально, то есть равно размеру массива минус один.

**Сортировка слиянием(Merge Sort)**

 Эта сортировка — хороший пример использования принципа «разделяй и сортируй». Сначала задача разбивается на несколько подзадач меньшего размера. Затем эти задачи решаются с помощью рекурсивного вызова или непосредственно, если их размер достаточно мал. Наконец, их решения комбинируются, и получается решение исходной задачи.   
Если описать подробнее, то вводимый массив разбивается на две примерно равные части (левая и правая), после чего каждая из них тоже разбивается на части и так далее. В конце разбиеиия получаются массивы, состоящие из 1 элемента, их можно считать базой для объединения, ведь такой массив точно отсортирован.

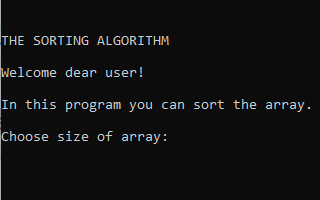
Затем соединяются отсортированные массивы. Выделяется дополнительная память под результирующий массив, в которую будут переносится нужные элементы. На каждом шаге берем наименьший из первых элементов массивов и заносим в результирующий, а затем увеличиваем счетчики результирующего массива и того, из которого брали элемент. Если один из двух массивов заканчивается, то результирующий до конца заполняется элементами из другого.   


**Поразрядная сортировка(Radix Sort)** Основная идея, заключается в том, что сравнение производится поразрядно: сначала сравниваются значения одного крайнего разряда, и элементы группируются по результатам этого сравнения, 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. Затем аналогично делается для следующего разряда, и так до конца.   
Алгоритм пригоден для сортировки любых объектов, запись которых можно поделить на «разряды», содержащие сравнимые значения. Например, так сортировать можно не только числа, записанные в виде набора цифр, но и строки, являющиеся набором символов, и вообще произвольные значения в памяти, представленные в виде набора байт.

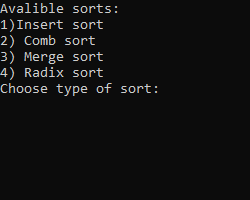


# Руководство пользователя

1. Запуск программы

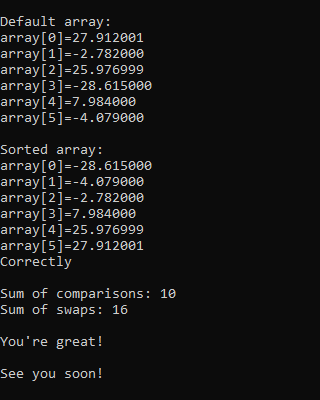
   
При запуске программы, пользователь может выбрать размер массива.

1. Выбор типа сортировки.



Далее пользователь может выбрать тип сортировки, введя одну из предложенных цифр: «1», «2», «3», «4» ; где 1 – сортировка «вставками», «2» - сортировка «расческой», «3» - сортировка «слиянием», «4» - «поразрядная» сортировка.

1. Результат, завершение программы.

  
В конечном итоге, пользователь видит весь первоначально сгенерированный массив, а сразу под ним – результат сортировки, а также количество сравнений (sum of comparisons) и количество перестановок(sum of swaps)

# Описание программной реализации

Моя программа состоит из одного файла, внутри которого прописаны несколько функций для каждой из сортировок. Название проекта – labamukhin

Функции делятся на основные и вспомогательные.

1.К основным я отношу те, что непосредственно реализуют сортировки. И это

1. insrt\_sort();
2. comb\_sort();
3. merge\_sort();
4. radix\_sort().

2.К вспомогательным относятся:

1. merge()
2. createCounters()
3. radix\_Lpass()
4. radix\_pass()

Структура же в моем проекте следующая

1. Подключение необходимых библиотек и объявление 2 переменных, отвечающих в последствии за подсчет сравнений и перестановок.
2. Функции в порядке 1.1; 1.2; 2.1; 1.3; 2.2; 2.4; 2.3; 1.4
3. Основная часть программы «меню», внутри которой пользователь вводит размер массива и выбирает тип сортировки из предложенных. Сразу после начинается заполнение массива случайными числами и его вывод на экран. Далее происходит сортировка и также вывод массива, но уже отсортированного. После чего проводится простейшая проверка на корректность работы произведенной сортировки. В конечном итоге выводиться кол-во сравнений и перестановок.

# Подтверждение корректности

Для подтверждения корректности в программе происходит перебор выведенного массива, где каждый следующий элемент должен быть больше предыдущего.

# Результаты экспериментов

Результаты занесены в таблицу ниже

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 10 | | 50 | | 100 | | 500 | | 1000 | |
|  | cmprs | swps | cmprs | swps | cmprs | swps | cmprs | swps | cmprs | swps |
| «1» | 25 | 75 | 590 | 1800 | 2500 | 7400 | 60000 | 190000 | 25000 | 750000 |
| «2» | 39 | 11 | 480 | 196 | 1233 | 255 | 9536 | 1735 | 22034 | 4105 |
| «3» | 25 | 34 | 230 | 286 | 550 | 672 | 3920 | 4488 | 8820 | 9976 |
| «4» | 0 | 40 | 0 | 200 | 0 | 400 | 0 | 2000 | 0 | 4000 |

Результатом каждого эксперимента, является выявление среднего случая (в ходе эксперимента проводилось несколько запусков программы для каждого размера массива)

**1.Сортировка вставкой**

Сложность алгоритма:

*Худшее время - O(n2) сравнений, обменов.*

*Лучше время - O(n) сравнений, 0 обменов.*

*Среднее время - О(n2) сравнений, обменов.*

*Лучший случай – отсортированный массив*

*Средний случай - случайно заполненный массив*

*Худший случай – отсортированный в обратно порядке массив*

В ходе эксперимента было подтверждено:

*Среднее время - О(n2) сравнений, обменов.*

**2.Сортировка расческой**

Сложность алгоритма:

*Худшее время - O(n2) сравнений, обменов.*

*Лучше время - O(n\*log(n))сравнений, обменов  
Среднее время O(n\*log(n))сравнений, обменов*

*Лучший случай – отсортированный массив*

*Средний случай - случайно заполненный массив*

*Худший случай – все зависит от элемента, который мы взяли в качестве опорного.*

**3.Сортировка слиянием**

Сложность алгоритма:

*Худшее время - O(n\*log(n)) сравнений, обменов.*

*Лучшее время О(n\*log(n)).*

*Среднее время: О(n\*log(n))*

*Лучший случай – отсортированный массив*

*Средний случай - случайно заполненный массив*

*Худший случай – Такой массив, в котором будет произведено максимально возможное кол-во сравнений. (10 12 14 16 18 20 22 24 11 13 15 17 19 21 23 25)*

**4.Поразрядная сортировка**

Сложность алгоритма: *O(w\*n)*

# Заключение

Поставленные задачи решены. Мной была написана программа со всеми типами сортировок с которыми я должен был её написать, во время создания которой я лучше стал понимать работу алгоритмов сортировки на языке С. Эксперименты были поставлены и их результаты подтвердили вычислительную сложность каждого из алгоритмов.   
Также был сделан вывод, что для массивов с большим числом элементов лучше использовать поразрядную сортировку, т.к. она затрачивает значительно меньше времени.

# Приложение

int main()

{

srand(time(0));

float arr[1000];

float arr2[sizeof arr];

int sort;

int t;

int i;

int a;

printf(" \n");

printf(" \n");

printf("THE SORTING ALGORITHM \n");

printf(" \n");

printf("Welcome dear user!\n ");

printf(" \n");

printf("In this program you can sort the array.\n");

printf(" \n");

printf("Choose size of array: ");

scanf("%d", &t);

printf("Avalible sorts:\n");

printf("1)Insert sort \n");

printf("2) Comb sort\n");

printf("3) Merge sort\n");

printf("4) Radix sort\n");

printf("Choose type of sort: ");

scanf("%d", &sort);

printf(" \n");

for (i = 0; i < t; i++)

arr[i] = rand() \* 0.001 \* pow(-1, i);

cmp = 0;

swp = 0;

printf("Default array: \n");

for (i = 0; i < t; i++)

{

printf("array[");

printf("%d", i);

printf("]=");

printf("%f ", arr[i]);

printf(" \n");

}

printf(" \n");

if (sort == 1)

insrt\_sort(arr, t);

if (sort == 2)

comb\_sort(arr, t);

if (sort == 3)

merge\_sort(arr, arr2, 0, t - 1);

if (sort == 4)

radix\_sort(arr, t);

printf("Sorted array: \n");

for (i = 0; i < t; i++)

{

printf("array[");

printf("%d", i);

printf("]=");

printf("%f ", arr[i]);

printf(" \n");

}

if (arr[i] <= arr[i + 1])

printf("Correctly");

else

printf("Uncorrectly");

printf(" \n");

printf(" \n");

printf("Sum of comparisons: ");

printf("%d", cmp);

printf("\n");

printf("Sum of swaps: ");

printf("%d", swp);

printf(" \n");

printf(" \n");

printf("You're great!\n");

printf(" \n");

printf("See you soon!\n");

printf(" \n");

}