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

Измайлов И.Р.

**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 8](#_Toc59802146)

[Подтверждение корректности 9](#_Toc59802147)

[Результаты экспериментов 10](#_Toc59802148)

[Заключение 11](#_Toc59802149)

[Приложение 12](#_Toc59802150)

# Постановка задачи

Передо мной была поставлена задача – повторить весь материал по языку программирования Си, который мы успели пройти в первом семестре. На основе этих знаний создать приложение в среде разработки Visual studio, предназначенное для сортировки набора вещественных чисел, введенных пользователем. В этом приложении нужно было реализовать четыре вида сортировки: Сортировка вставкой, Сортировка «Шелла», Сортировка «расческой», «Поразрядная сортировка»; ко всему к этому необходимо было рассчитать сложность выполнения каждого из них и сравнить их показатели друг с другом. По выполнении всех этих задач - выложить свой код на GitHub.

# Метод решения

**Сортировка вставкой (Insertion Sort)**

**Особенности:**

1. Простота реализации
2. Эффективность на частично упорядоченных последовательностях или на небольших наборах чисел
3. Высокая вычислительная сложность

Идея алгоритма: делим исходный массив, состоящий из набора чисел на отсортированную и неотсортированную часть. Поскольку изначально последовательность не отсортирована, то за отсортированный массив берется первый элемент структуре данных. На каждом шаге алгоритма выбирается элемент, находящийся справа от отсортированной части. Этот элемент сравнивается с каждым элементом в отсортированном списке и ставится на «нужную позицию». Тем самым отсортированная часть массива увеличилась на один элемент. Алгоритм выполняется до тех пор, пока набор входных неотсортированных данных не будет исчерпан.

**Сортировка «расческой»(Comb Sort)**

1. Усовершенствованный алгоритм сортировки «пузырьком» (Bubble Sort)
2. Является быстрой сортировкой

В «Bubble sort» происходит сравнение двух соседних элементов массива. Идея же сортировки расческой заключается в сравнении элементов, находящихся на большем расстоянии друг от друга. Сначала берется самый большое расстояние, то есть сравниваем первый и последний элементы в массиве, на каждом шаге уменьшается расстояние между элементами на конкретное значение. Это конкретное значение носит название фактор уменьшения, которое получили, используя следующую формулу: 1/(1-e-φ) ≈ 1.247, где е – основание натурального логарифма, а φ – золотое сечение. Алгоритм выполняется до тех пор , пока разность индексов больше единицы, затем массив сортируется пузырьковой сортировкой. Тем самым по мере выполнения программы, мы избавлялись от маленьких значений в конце массива, которые сильно замедляли сортировку пузырьком.

**Сортировка слиянием(Merge Sort)**

**Особенности**

1. Реализация на основе принципа «разделяй и властвуй»
2. Быстрая сортировка
3. Первая сортировка из нашего списка, основанная на рекуррентном вызове функции
4. Требуется дополнительная память равная по объему сортируемому массиву

Сначала мы разбиваем сортируемый нами массив на две примерно одинаковые части. После этого, рекурсивным вызовом функции, мы делим каждую из половинок еще на две одинаковые части. Такое разбиение массива происходит до тех пор, пока не дойдем до массива длиной один, который будем считать отсортированным.

После этого начинается процедура слияния, означающая, что мы будем объединять две предварительно упорядоченные подпоследовательности в вспомогательный массив. Сначала сравниваются первые элементы упорядоченных последовательностей и из них выбирается наименьший, который сливается в вспомогательный массив. В подмассиве, где был выбран элемент, указатель перемещается на следующий элемент. Процедура выполняется, пока не будет достигнут конец одной из подпоследовательностей. Остальные элементы другой подпоследовательности передаются в вспомогательный массив без изменения порядка. В начале сравниваются единичные подмассивы(которые уже отсортированы), затем подпоследовательности из двух элементов, четыре и так далее.

**Поразрядная сортировка(Radix Sort)**

**Особенности:**

1. Не использует сравнение сортируемых элементов
2. Ключ, по которому происходит сортировка, необходимо разделить на разряды
3. Линейная сортировка
4. Требуется О(n) дополнительной памяти

В отличие от предыдущих сортировок поразрядная сортировка предполагает сравнение элементов неупорядоченного массива поразрядно. В качестве одного разряда для удобства возьмем 1 байт. Используется сортировка подсчетом и ее свойство устойчивости. Основная идея: каждый ключ сортировки можно рассматривать как k- значное число, каждая цифра которого находится в диапазоне от 0 до m-1. Поочередно используется устойчивая сортировка(например сортировка подсчетом) для каждой цифры справа налево.

Для реализации данного алгоритма важно помнить, как организованы числа в памяти. Поскольку моя лабораторная работа предполагает работу с числами с плавающей запятой, то не будет лишним упомянуть, как они хранятся в памяти. Число с плавающей запятой: . Существует несколько стандартов для представления таких чисел. Наиболее известным (исходя из поисковой сети Яндекс) считается IEEE-754. Этот стандарт предполагает запись числа в виде [знак][порядок][мантисса]. Числа double имеют порядок из 11 бит, мантиссу из 52 бит.

Если значение[порядка] у одного числа больше соответствующего значения у другого, то первое число больше второго. То есть сначала сравниваем двоичные цифры, затем младшие. Можно интерпретировать положительные числа стандарта как целые соответствующей длины: a > b равносильно (unsigned long int ) a > (unsigned long int) b

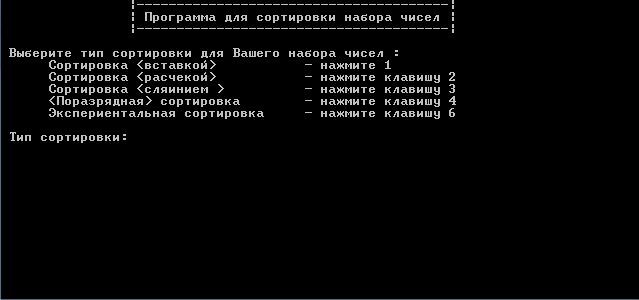
Если мы имеем дело с отрицательными числами, то у него первый бит равен единице, так что данное число, как и в случае без знаковых целых чисел, является очень большим положительным a >0, b<0 <====> (unsigned long int)a < (unsigned long int)b

Следует еще помнить a , b < 0 <====> (unsigned long int)a < (unsigned long int)b.

Это означает, что у нас сначала будут стоять отсортированные положительные числа, затем отсортированные в обратном порядке отрицательные числа. Нам нужно будем после сортировки написать цикл который, расположит такой набор чисел так как нужно.

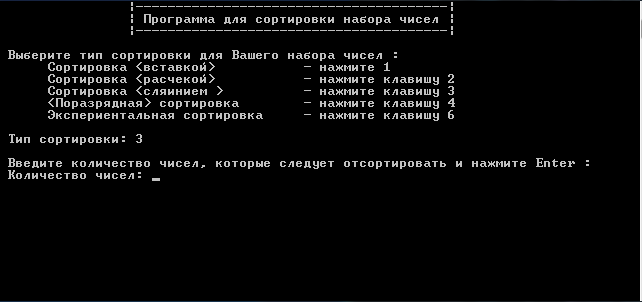
# Руководство пользователя

1. Запуск программы

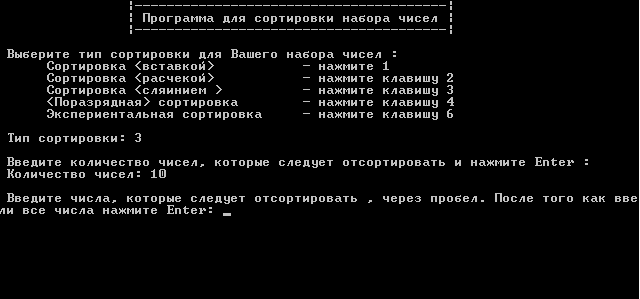


При запуске программы пользователю будет предоставлен на выбор 4 вида сортировки. После выбора нужной сортировки, нужно нажать на клавишу Enter. Клавиша 1 – соответствует сортировке вставкой, 2 – расческой, 3 – слиянием, 4 – поразрядной сортировке. Чтобы произвести экспериментальную сортировку массива необходимо ввести число 6. Данная сортировка реализует автоматическое заполнение массива длиной n, которое должен ввести пользователь. Результатом выполнения «Экспериментальной сортировки» будет лишь количество перестановок и сравнений первый трех сортировок. Важно! Первые три сортировки позволяют упорядочить любые последовательности, элементы которых относятся к множеству вещественных чисел, а четвертая сортировка – лишь для целых чисел.

Для наглядности примера мы введем цифру 3 и нажмем Enter.



1. Дальнейшие действия

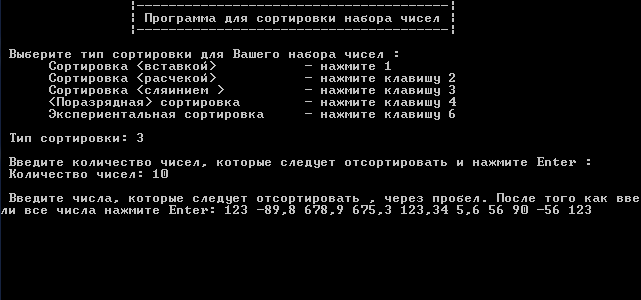


На консоли появится надпись «Введите количество чисел, которые следует отсортировать и нажмите Enter» и следующая строка «Количество чисел». Пользователю необходимо посчитать количество чисел в сортируемой последовательности, ввести это количество и нажать на Enter.

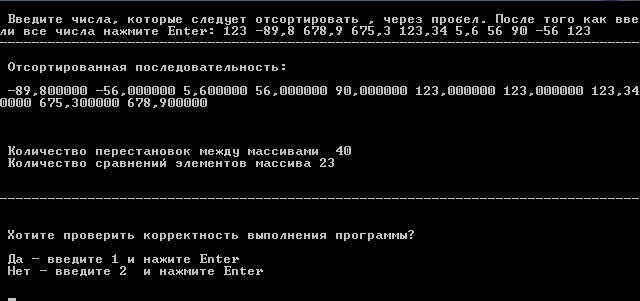
На нашем примере мы будем сортировать 10 чисел. Введем число 10 и нажмем Enter.

После нажатия клавиши Enter консоли появится строчка «Введите числа, которые следует отсортировать, через пробел. После того как ввели все числа нажмите Enter». Пользователю необходимо ввести все числа через пробел и после ввода нажать на клавишу Enter. Важно!!!

Ввод вещественных чисел производятся с помощью знака «,», а не «.» . Так же следует помнить, что если мы выбрали 4 сортировку(поразрядная сортировка), то вводятся только целые и вещественные числа:



В конкретном примере мы будем сортировать числа: **123 -89,8 678,9 675,3 123,34 5,6 56 90 -56 123**

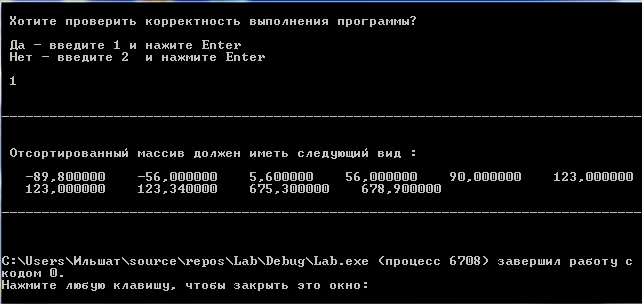
****

После ввода всех чисел нажмем на клавишу Enter.

3.Результат выполнения программы

Появится окошечко, котором будет выведена отсортированная последовательность. В этом же окошечке будет выведено количество перестановок между элементами и количество сравнений элементов массива. Чтобы убедиться в том, что программа сработала правильно пользователю дается возможность проверить корректность выполнения программы. Если он хочет проверить, то ему необходимо ввести 1 и нажать на Enter, если нет – 2 и нажать на Enter.

Чтобы показать все возможности нашей программы мы нажмем на клавишу 1.



В итоге у нас появилось дополнительное окошечко, в котором показывается, как должна была бы выглядеть отсортированная последовательность. Сравнив эти две получившиеся последовательности, приходим к выводу, что программа сработала корректно.

# Описание программной реализации

Мой проект имеет название lab. В ней всего лишь один файл с названием source.cpp.Int main – первая функция, с которой начинается выполнение программы. В зависимости от того, какой вид сортировки был выбран пользователем, будет выполняться та или иная функция. Под кажую сортировку написана своя отдельная функция. После выполнения сортировки в фунцкии main происходит вывод результат выполнения программы: количество сравнений, количество перестановок и отсортированный массив.

Сортировку вставками реализует функция insertion sort. Сортировку расческой – comb\_sort. Сортировку слиянием - umsort, merge. Поразрядную сортировку – signedRadixSort, signedRadixLastPass, radixPass, Counters.

Существуют еще две функции, которые отвечают за корректность выполнения программы: CheckSortI, CheckSortD.

При реализации сортировок, необходимо было посчитать сложность работы тех или иных сортировок, при разных размерах массивов. Для этих целей была создана дополнительная функция Experiment, которая будет забивать массив длиной n случайными числами и проводить 3сортировки для них. За экспериментальную сортировку вставкой отвечает – comb\_sortExp, за слияние – umsortExp, MergeExp, за расческу – comb\_sortExp, за поразрядную сортировку – signedRadixSortExp, signedRadixLastPassExp, radixPassExp, CountersExp.

# Подтверждение корректности

Для подтверждения корректности в программе реализована функция checkSort, которая копирует массив и сравнивает результат работы сортировки с результатом работы функции. Функция CheckSort – обычная сортировка пузырьком, алгоритм и программная реализация которой разбиралась на парах. CheckSortD – функция проверки правильной работы сортировки вещественных последовательностей. То есть он вызывается в первых трех сортировках. CheckSortI – функция проверки корректной работы сортировки целых чисел. Поскольку поразрядная сортировка реализована лишь для целых чисел, то вызываться он будет лишь, при этой сортировке.

# Результаты экспериментов

Для того чтобы провести эксперимент и сравнить показатели сортировок между собой я введу в рассмотрение еще одну функцию experiment, который будет заполнять массив длиной, который будет задаваться вручную, случайными числами.

Количество перестановок

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **len** | **insertionSort** | | **comb\_sort** | **umsort** | **radix\_sort** |
| **1** | 0 | | 0 | 0 | 7 |
| **2** | 1 | | 0 | 4 | 14 |
| **5** | 5 | | 3 | 20 | 35 |
| **10** | 18 | | 10 | 40 | 70 |
| **100** | 2609 | | 226 | 800 | 700 |
| **500** | 59776 | | 1348 | 5000 | 3500 |
| **1000** | 248594 | | 2845 | 10000 | 7000 |
| **10000** | 24935268 | | 28880 | 140000 | 70000 |
| **500000** | Выяснить не удалось | 1454829 | | 10000000 | 3500000 |
| **1000000** | Выяснитьнеудалось | 2915498 | | 20000000 | 7000000 |

Количество сравнений

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Len | **insertionSort** | **comb\_sort** | **umsort** | **signedRadixSort** |
| **1** | 0 | 1 | 0 | 0 |
| **2** | 1 | 1 | 1 | 0 |
| **5** | 7 | 10 | 3 | 0 |
| **10** | 25 | 39 | 23 | 0 |
| **100** | 2706 | 1233 | 549 | 0 |
| **500** | 60273 | 9536 | 3847 | 0 |
| **1000** | 249591 | 22034 | 8737 | 0 |
| **10000** | 24945265 | 329653 | 123425 | 0 |
| **500000** | Выяснить не удалось | 24978555 | 8833169 | 0 |
| **1000000** | Выяснить не удалось | 52956975 | 18662077 | 0  Получившиеся результаты для наглядности были представлены в виде двух таблиц: одна из них показывает количество перестановок внутри каждой сортировки, а вторая – количество перестановок в той или иной сортировке.  Можно заметить, что во второй таблице результат количества сравнений у radix-sort везде равен нулю. Напомним, что он имеет такие показатели из-за того, что поразрядная сортировка линейная и не использует в своей программе сравнения.  Простая сортировка(insertion sort) и линейная сортировка (comb sort) на небольших массивах( примерно до 100 элементов) работают за меньшее время, чем сортировки umsort и radix sort. Это можно заметить по количеству перестановок. Однако уже после преодоления этой отметки мы видим, как простая сортировка начинает заметно проигрывать во времени трем остальным сортировкам. Показатели между insertion sort и остальными сортировками различаются примерно в 3 раза. И с увеличением количества элементов, мы замечаем, как буквально в геометрической прогрессии увеличиваются показатели сравнений у insertion sort. В момент когда количество элементов в массиве достигает 500000, то количество перестановок становится так много, что компилятор уже не может выдать результат. В остальных же сортировках такого «потолка не наблюдается».  Когда количество сортируемых элементов становится примерно 500, в скорости выполнения программы уже начинает отставать comb sort. Количество перестановок внутри comb sort примерно уже 3 раз меньше чем у двух остальных.  После преодоления отметки в 1000 элементов, мы замечаем, как линейная сортировка оправдывает сое название и начинает выполнять программу за гораздо меньшее время.  Если располагать сортировки в порядке возрастания колиество сравнений, то до 500 элементов, мы получим, что insertion sort, combsort, umsort. А после этой отметки мы получим противоположную ситуацию. |

# Заключение

**1.Сортировка вставкой**

**Сложность алгоритма:** Худшее время O(N2) сравнений, обменов. Лучше время O(n)сравнений, О(1) обменов, О(n2) сравнений, обменов

Лучший случай – отсортированный массив

Средний случай - случайно заполненный массив

Худший случай – отсортированный в обратно порядке массив

**2.Сортировка расческой**

**Сложность алгоритма:** Худшее время - O(n2) сравнений, обменов. Лучше и 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 – Чтобы создать наихудший случай сортировки слиянием, операция слияния, которая привела к отсортированной последовательности, должна произвести максимум сравнений. Разберем на конкретном примере. Пусть отсортированный массив будет следующим {11, 12, 13, 14, 15, 16, 17, 18 19, 20}. Для худшего случая правый и левый подмассивы, должны хранить альтернативные элементы. Т.е. Левый подмассив должен иметь вид {1,3,5,7}, а правый {2,4,6,8}. Мы применим ту же логику для левого и правого помассива до тех пор пока не получим подпоследовательность длиной 2. Таким образом, наихудший случай будет выглядеть так {1,5,3,7,2,6,4,8}.

**4.Поразрядная сортировка**

**Сложность алгоритма: O(n\*k)** k- число разрядов

# Приложение

int main()

{

setlocale(LC\_ALL, "Rus");

int a;

int kol;

printf("\t\t|---------------------------------------| \n");

printf("\t\t| Программа для сортировки набора чисел |\n");

printf("\t\t|---------------------------------------| \n\n");

printf(" Выберите тип сортировки для Вашего набора чисел :\n");

printf(" Сортировка <вставкой> - нажмите 1 \n");

printf(" Сортировка <расчекой> - нажмите клавишу 2\n");

printf(" Сортировка <сляинием > - нажмите клавишу 3\n");

printf(" <Поразрядная> сортировка - нажмите клавишу 4\n");

printf(" Экспериентальная сортировка - нажмите клавишу 6\n\n");

printf(" Тип сортировки: ");

scanf(" %d", &a);

if (a == 1 || a == 2 || a == 3) {

printf("\n");

printf(" Введите количество чисел, которые следует отсортировать и нажмите Enter : \n ");

printf("Количество чисел: ");

scanf("%d", &kol);

double\* array = (double\*)malloc(kol \* sizeof(double));

double\* check = (double\*)malloc(kol \* sizeof(double));

printf("\n");

printf(" Введите числа, которые следует отсортировать , через пробел. После того как ввели все числа нажмите Enter: ");

for (int i = 0; i < kol; i++)

scanf("%lf", &array[i]);

for (int i = 0; i < kol; i++)

check[i] = array[i];

if (a == 1)

insertionSort(array, kol, check);

if (a == 2)

comb\_sort(array, kol, check);

if (a == 3)

umsort(array, kol, check);

}

if(a==4) {

int vib = 0;

printf("\n");

printf(" Введите количество чисел, которые следует отсортировать и нажмите Enter : \n");

printf(" Количество чисел: ");

scanf("%d", &kol);

int\* array = (int\*)malloc(kol \* sizeof(int));

int\* arrayk = (int\*)malloc(kol \* sizeof(int));

int\* check = (int\*)malloc(kol \* sizeof(int));

printf("\n");

printf(" Введите целые числа, которые следует отсортировать , через пробел\n");

printf(" После того как ввели все числа нажмите Enter: ");

for (int i = 0; i < kol; i++)

scanf("%d", &array[i]);

for (int i = 0; i < kol; i++)

check[i] = array[i];

int pers[1] = { 0 };

int srvs[1] = { 0 };

signedRadixSort(array, arrayk, kol, pers, srvs);

printf("--------------------------------------------------------------------------------\n");

printf(" Отсортированная последовательность:\n\n");

printf(" ");

int j = 0;

while (array[j] >= 0 && j < kol)

j++;

if (j != 0) {

int i = 0;

for (int l = kol - 1; l >= j; l--)

printf("%d ", array[l]);

for (int i = 0; i < j; i++)

printf("%d ", array[i]);

}

else {

for (int i = 0; i < kol; i++)

printf("%d", array[i]);

}

printf("\n\n");

printf(" Количество перестановок : %d\n", pers[0]);

printf("--------------------------------------------------------------------------------\n\n");

printf(" Хотите проверить корректность выполнения программы?\n\n");

printf(" Да - введите 1 и нажите Enter\n");

printf(" Нет - введите 2 и нажмите Enter\n\n");

printf(" ");

scanf(" %d", &vib);

if (vib == 1) {

printf("\n\n");

CheckSortI(check, kol);

}

free(check);

}

if (a == 6) {

Experiment();

}

}