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

Шокуров Д. А.

**Проверил**:

ассистент каф. МОСТ,

Волокитин В.Д.

Нижний Новгород

2020

**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

Задача данной лабораторной работы - реализовать программу для сортировки массива чисел типа double, вводимых пользователем через консоль. В качестве алгоритмов сортировки использовать сортировку вставками, сортировку Шелла, сортировку слиянием и поразрядную сортировку. Замерить выше перечисленные алгоритмы на количество совершенных перестановок и операций сравнения.

# Метод решения

**Сортировка вставками**

Идея: алгоритм сортировки, в котором каждый элемент массива по очереди занимает свое место в отсортированной части массива.

В начальный момент отсортированная часть массива пуста, но на каждом шаге алгоритма в неотсортированной части массива выбирается элемент и путем последовательных сравнений помещается в нужную позицию в отсортированной части массива.

Вычислительная сложность – О(N^2)

**Сортировка Шелла**

Идея: разбить массив на подмассивы. В качестве подмассива выбираются элементы, удаленные на d шагов. Сортируем подмассивы сортировкой вставками и уменьшаем шаг d. Продолжаем до тех пор, пока d не станет равным 1.

Вычислительная сложность – O(N\*log(N))

**Cортировка слиянием**

Идея: сортируемый массив разделяется на два подмассива приблизительно одинакового размера. Рекурсивное разбиение продолжается до тех пор, пока размер массива не достигнет единицы. Каждая из частей сортируется отдельно. Два упорядоченных массива обьединяются в третий результирующий массив.

Вычислительная сложность – O(N\*log(N))

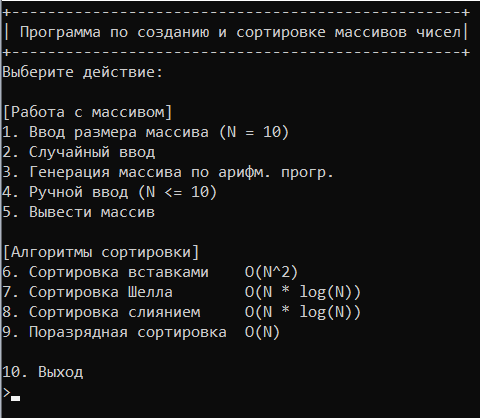
**Поразрядная сортировка**

Идея: использовать сортировку подсчетом для каждого разряда от младшего к старшему.

Идея сортировки подсчетом: если известно, что у K чисел разряд равен x и количество L чисел, у которых разряд меньше x, то К элементов должны располагаться в отсортированном массиве по индексам от L+1 до L+K.

Вычислительная сложность – O(N)

# Руководство пользователя

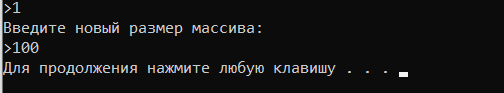


1.1 Главное меню

Главное меню программы содержит 10 команд: 5 по работе с массивом, 4 для вызова сортировок и команду «Выход». Для выбора команды необходимо ввести в консоль её номер. Далее приведено описание команд:

**Работа с массивом**

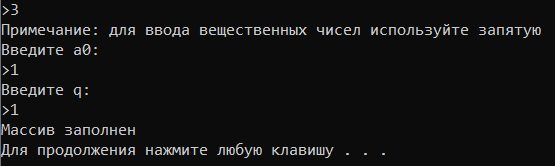
1. Ввод размера массива»: Для выбора введите 1. Для изменения размера массива введите число, равное её новому размеру.



1.2 Окно выбора размера массива

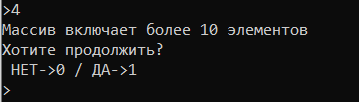
2) «Случайный ввод»: Для выбора введите 2. Заполняет массив случайными числами типа double

3) «Генерация массива по арифметической прогрессии»: Для выбора введите 3. Введите начальные переменные a0(первый член арифм. прогр.) и q (разность прогрессии).



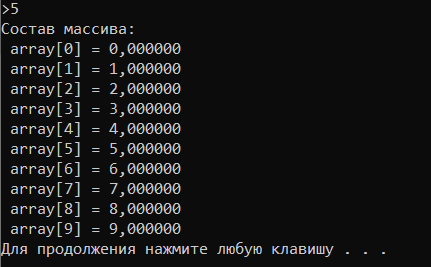
1.3 Инициализация переменных арифм. прогрессии

4) «Ручной ввод»: Для выбора введите 4. Введите N чисел для заполнения массива. При размере массива больше 10 попросит подтвердить ввод. Для подтверждения введите 1, для отмены 0.



1.4 Подтверждение ручного ввода

5) «Вывод массива»: Для выбора введите 5. Выводит массив с указанием индексов элементов.



1.5 Вывод массива

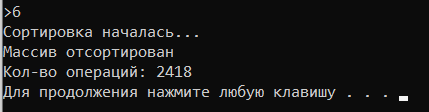
**Алгоритмы сортировки**

6) «Сортировка вставками»: Для выбора введите 6. Сортирует массив сортировкой вставками и выводит количество сделанных операций

7) «Сортировка Шелла»: Для выбора введите 7. Сортирует массив сортировкой Шелла и выводит количество сделанных операций.

8) «Сортировка слиянием»: Для выбора введите 8. Сортирует массив сортировкой слиянием и выводит количество сделанных операций.

9) «Поразрядная сортировка»: Для выбора введите 9. Сортирует массив поразрядной сортировкой и выводит количество сделанных операций.

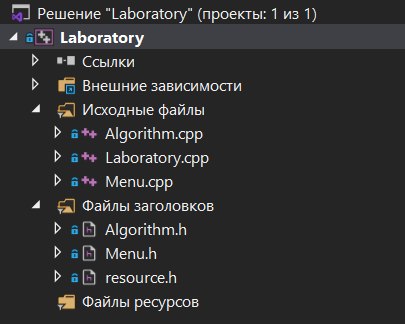


1.6 Пример работы алгоритмов сортировки

10) «Выход»: Для выбора введите 10. Завершает работу программы.

# Описание программной реализации

Для удобства программа была разделена на три файла «Algorithm.cpp», «Laboratory.cpp», «Menu.cpp». К каждому из файлов прилагается заголовок.



2.1 Решение

Файл «Laboratory.cpp» содержит точку входа в программу - функцию main(), в которой вызываются два метода MenuInitialize() и UpdateMenu(), реализация которых находится в файле «Menu.cpp»

**Menu.cpp**

В данном файле располагается реализация меню программы.

Глобальные переменные:

double\* arr – указатель на массив

int Exit – код выхода из программы

int N – размер массива

int limit – ограничения на ручной ввод и вывод массива

**UpdateMenu();**

Для обработки ввода в консоль используется метод UpdateMenu(), который отображает главное меню программы и обрабатывает ввод консоли. При наличии команды, соответствующей введенному числу, данный метод вызывает ее, в противном случае выводит в консоль «Команда не найдена».

**MenuInitialize();**

Предварительная инициализация переменных Exit, N, limit. Заполнение массива значениями от 0 до N-1.

Вызов дальнейших методов зависит от введенного в консоль числа.

1. **int input\_size();**

Данная функция вызывается из UpdateMenu() в случае ввода пользователем 1. В данном методе удаляется прежний и создается новый массив, заполняемый числами от 0 до N-1.

1. **int random\_array();**

Данная функция вызывается из UpdateMenu() в случае ввода пользователем 2. Заполняет массив случайными значениями типа double.

1. **int generating\_progression();**

Данная функция вызывается из UpdateMenu() в случае ввода пользователем 3. В данном методе происходит запрос на ввод начальных значений для формирования арифметической прогрессии и дальнейшее заполнение массива элементами последовательности.

1. **int input\_array()**

Данная функция вызывается из UpdateMenu() в случае ввода пользователем 4.

В данном методе происходит запрос на ввод N чисел для заполнения массива.

1. **int show\_array()**

Данная функция вызывается из UpdateMenu() в случае ввода пользователем 5.

Выводит массив с указанием индексов.

**6-9) ulong insertion\_sort(), ulong Shell\_sort(), ulong mergesort(), ulong radixSort().**

Вызов одной из сортировок под соответствующим номером.

Все вышеописанные функции возвращают 0.

**Algorithm.cpp**

В данном файл располагается реализация алгоритмов сортировки.

**ulong insertion\_sort(double arr[], const int size);**

Сортировка вставками. В качестве аргументов использует указатель на массив arr[] и его размер. Выводит количество совершенных операций.

**ulong Shell\_sort(double arr[], const int size)**

Сортировка Шелла. В качестве аргументов принимает указатель на массив типа double и его размер. Содержит три цикла, первый из которых уменьшает шаг между сортируемыми элементами, второй проходит по всем значениям, начиная со step и заканчивая size-1. Третий меняет элементы на расстоянии step местами, пока выполняется условие j-step>=0 и arr[j - step] > arr[j]. Данная функция выводит количество совершенных операций.

**ulong mergesort(double arr[], double second[], int l, int r);**

Сортировка слиянием. В качестве аргументов принимает указатель на массив типа double, указатель на вспомогательный массив и индексы левой и правой границы сортируемого массива. Внутри функция рекурсивно вызывает саму себя, пока l < r после чего происходит вызов функции слияния merge(). После копирует элементы массива second[] в массив arr[]. Возвращает количество совершенных операций сравнения и копирования значений.

**ulong merge(double arr[], double second[], int left, int middle, int right);**

Функция слияния двух упорядоченных массивов. В качестве аргументов принимает указатель на сортируемый и вспомогательный массивы, левую и правую границы сортируемого массива и значение middle, равное индексу последнего элемента левого массива. В начале работы сравнивает левую и правую границы. Если они равны, то заносит в вспомогательный массив значение исходного массива, иначе запускает цикл, который выбирает из упорядоченных массивов наименьшее значение и заносит его в вспомогательный массив. Продолжает свою работу, пока один из массив не опустеет, после чего переносит оставшиеся злементы и выводит количество совершенных операций.

**ulong \_mergesort(double arr[], const int size);**

Сортировка слиянием с автоматическим созданием вспомогательного массива и передачи границ. После работы удаляет вспомогательный массив и возвращает количество совершенных операций.

**ulong radixSort(double arr[], long size);**

Поразрядная сортировка. В качестве аргументов принимает указатель на массив типа double и его размер. Внутри себя создает и заполняет массив counters[] – массив счетчиков, создает вспомогательный массив dest[] и производит сортировку с помощью функции radixPass() чисел в массиве arr[] по их разрядам от младшего к старшему. Для сортировки последнего разряда использует функцию radixLastPass(). Выводит количество совершенных операций.

**ulong createCounters(double\* arr, long\* counters, long size);**

Функция для создания массива счетчиков. В качестве аргументов принимает сортируемый массив, вспомогательный массив и размер сортируемого массива. Внутри себя запускает цикл while(), который считает количество разрядов для каждого массива и заносит результаты в вспомогательный массив counters. Выводит количество совершенных операций.

**ulong radixPass(short Offset, long size, double\* arr, double\* dest, long\* count);**

Сортировка массива чисел по одному разряду. В качестве аргументов принимает номер разряда, размер сортируемого массива, указатель, указатель на сортируемый массив, указатель на массив для вывода отсортированных чисел, указатель на массив счетчиков для данного разряда. Внутри себя считает префикс-сумму массива count и смещает все его значения на единицу. После этого значения в массиве count указывают на позицию элемента в выходном массиве. Затем размещает элементы исходного массива по указанным индексам. Возвращает количество совершенных операций сравнения.

**ulong radixLastPass (short Offset, long size, double\* arr, double\* dest, long\* count);**

Сортировка массива чисел с учетом знака. Используется для сортировки последнего разряда в поразрядной сортировке.

1. *ulong – unsigned long*

# Подтверждение корректности

Для подтверждения корректности каждый из алгоритмов сортировки прошел процедуру, описанную ниже:

Создается два одинаковых упорядоченных массива arr1 и arr2. Значения массива arr2 перемешиваются случайным образом и подаются для сортировки одному из алгоритмов. После этого значения массива arr2 сравниваются с упорядоченным массивом arr1.

Зависимось количества совершенных операций каждым из алгоритмов изображена на графиках:

В процессе проведения эксперимента использовались случайные данные.

**Сортировка вставками**

Рисунок 1 График зависимости количества совершенных операций алгоритмом сортировки вставками от размера массива

Синий – результаты экспериментов. Красный – кривая y = (N^2) / 4.

Можно заметить, что график почти совпадает с кривой у = (N^2) / 4, что говорит о том, что сложность сортировки вставками сравнима с O((N^2) / 4) ~ O(N^2).

**Сортировка Шелла**

Рисунок 2. График зависимости количества совершенных операций сортировкой Шелла от размера массива.

Синий – результаты экспериментов. Красный – кривая y = N\*ln(N).

Как видно по графику, большинство результатов экспериментов расположены вблизи функции y = N\*ln(N), но при случайных данных функция резко деградирует (это ярко демонстрирует точка N = 2560).

**Сортировка слиянием**

Рисунок 3. График зависимости количества совершенных операций сортировкой слиянием от размера массива

Синий – результаты экспериментов. Красный – кривая y = 2,9\*N\*ln(N).

График функции y = 2,9\*N\*ln(N) в промежутке от 0 до 3\*10^5 сливается с результатами экспериментов, что говорит о том, что сортировка слиянием имеет сложность O(C\*N\*ln(N)) ~ O(N\*ln(N)).

**Поразрядная сортировка**

Рисунок 4. График зависимости количества совершенных операций поразрядной сортировки от размера массива

Cиний – результаты экспериментов. Красный – прямая y = 8 \* N.

Можно заметить, что график совпадает с прямой y = 8 \* N, где N – размер массива, что означает, что вычислительная сложность поразрядной сортировки сравнима с O(8\*N) ~ O(N).

# Результаты экспериментов

По данным экспериментов видно, что каждый из алгоритмов соответствует своей теоретической вычислительной сложности.

# Заключение

Выполняя лабораторную работу я научился реализовывать сортировку вставками, сортировку Шелла, сортировку слиянием и поразрядную сортировку, научился создавать программы с консольным интерфейсом и убедился в теоретической сложности алгоритмов.

# Приложение

Algorithm.cpp

#include <stdlib.h>

#include <string.h>

typedef unsigned char uchar;

typedef unsigned long ulong;

using namespace std;

ulong insertion\_sort(double arr[], const int size) {

ulong amount = 0;

int i;

double temp;

for (i = 1;i < size;i++) {

//по возрастанию

for (int j = i; j > 0 && arr[j - 1] > arr[j];j--) {

temp = arr[j];

arr[j] = arr[j - 1];

arr[j - 1] = temp;

amount++;

}

}

return amount;

}

ulong Shell\_sort(double arr[], const int size) {

ulong amount = 0;

int step, i, j;

double temp;

for (step = size / 2; step > 0;step /= 2) {

for (i = step; i < size;i++) {

//по возрастанию

for (j = i;j - step >= 0 && arr[j - step] > arr[j];j -= step) {

temp = arr[j];

arr[j] = arr[j - step];

arr[j - step] = temp;

amount++;

}

}

}

return amount;

}

ulong merge(double arr[], double second[], int left, int middle, int right) {

int i;

ulong amount = 0;

int l\_cur = left, r\_cur = middle + 1;

if (left == right) {

second[left] = arr[left];

amount++;

return amount;

}

for (i = left;i <= right;i++) {

if ((l\_cur <= middle) && (r\_cur <= right)) {

//по возрастанию

if (arr[l\_cur] <= arr[r\_cur]) {

second[i] = arr[l\_cur];

l\_cur++;

}

else {

second[i] = arr[r\_cur];

r\_cur++;

}

}

else if (l\_cur <= middle) {

second[i] = arr[l\_cur];

l\_cur++;

}

else {

second[i] = arr[r\_cur];

r\_cur++;

}

amount++;

}

return amount;

}

ulong mergesort(double arr[], double second[], int l, int r) {

int i, q;

ulong amount = 0;

if (l < r) {

q = (l + r) / 2;

amount += mergesort(arr, second, l, q);

amount += mergesort(arr, second, q + 1, r);

amount += merge(arr, second, l, q, r);

for (i = l;i <= r;i++) {

arr[i] = second[i];

amount++;

}

}

return amount;

}

ulong \_mergesort(double arr[], const int size) {

ulong amount = 0;

double\* second = (double\*)malloc(sizeof(double) \* size);

amount += mergesort(arr, second, 0, size - 1);

free(second);

return amount;

}

ulong createCounters(double\* arr, long\* counters, long N) {

int i;

ulong amount = 0;

uchar\* bp = (uchar\*)arr;

uchar\* end = (uchar\*)(arr + N);

//количество возможных вариантов\*кол-во разрядов(по 256)\*тип данных массива counters

memset(counters, 0, 256 \* sizeof(double) \* sizeof(long));

while (bp != end) {

for (i = 0;i < sizeof(double);i++) {

counters[256 \* i + \*(bp++)]++;

amount++;

}

}

return amount;

}

ulong radixPass(short Offset, long size, double\* arr, double\* dest, long\* count) {

ulong amount = 0;

double\* sp;

long s, c, i, \* cp;

uchar\* bp;

s = 0;

cp = count;

for (i = 256; i > 0; --i, ++cp) {

c = \*cp;

\*cp = s;

s += c;

}

bp = (uchar\*)arr + Offset;

sp = arr;

for (i = size; i > 0; --i, bp += sizeof(double), ++sp) {

cp = count + \*bp;

dest[\*cp] = \*sp;

(\*cp)++;

amount++;

}

return amount;

}

ulong radixLastPass(short Offset, long size, double\* arr, double\* dest, long\* count) {

ulong amount = 0;

double\* sp;

long sum, c, i, \* cp;

uchar\* bp;

long numNeg = 0;

for (i = 128;i < 256;i++) numNeg += count[i];

sum = numNeg;

cp = count;

for (i = 0; i < 128; ++i, ++cp) {

c = \*cp;

\*cp = sum;

sum += c;

}

cp = count + 255;

sum = 0; \*cp = 0;

for (i = 255;i >= 128;i--,cp--) {

sum += \*cp;

\*cp = sum;

}

bp = (uchar\*)arr + Offset;

sp = arr;

for (i = size; i > 0; i--, bp += sizeof(double), sp++) {

cp = count + \*bp;

if (\*bp < 128) {

dest[\*cp] = \*sp;

(\*cp)++;

}

else {

(\*cp)--;

dest[\*cp] = \*sp;

}

amount++;

}

return amount;

}

ulong radixSort(double arr[], const long size) {

ulong amount = 0;

int i;

double\* temp;

double\* dest = (double\*)malloc(sizeof(double) \* size);

long\* counters = (long\*)malloc(sizeof(double) \* 256 \* sizeof(long));

long\* count;

amount += createCounters((double\*)arr, counters, size);

for (i = 0; i < sizeof(double) - 1; i++) {

count = counters + 256 \* i;

amount += radixPass(i, size, arr, dest, count);

temp = arr;

arr = dest;

dest = temp;

}

count = counters + 256 \* (sizeof(double) - 1);

amount += radixLastPass(sizeof(double) - 1, size, arr, dest, count);

temp = arr;

arr = dest;

dest = temp;

free(dest);

free(counters);

return amount;

}