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**:

студентка группы 3822Б1ПМ1

Шарутина В. Ф.

**Проверил**:

преподаватель каф. МОСТ,

Волокитин В.Д.

Нижний Новгород

2022

**Содержание**

[Введение 3](#_Toc26962562)

[Постановка задачи 4](#_Toc26962563)

[Метод решения 5](#_Toc26962564)

[Руководство пользователя 8](#_Toc26962565)

[Описание программной реализации 9](#_Toc26962566)

[Подтверждение корректности 12](#_Toc26962567)

[Результаты экспериментов 13](#_Toc26962568)

[Заключение 16](#_Toc26962569)

[Приложение 17](#_Toc26962562)

# Введение

Актуальность темы данной работы заключается в том, что сортировки данных в нынешнем мире различных информационных технологий являются неотъемлемой частью большинства процессов, связанных с обработкой информации большого объёма, которые могут встретится и в повседневной жизни, и в каких-то специализированных сферах. Описанные же здесь коды программ явно направлены на упрощение выполнения поставленной задачи – отсортировать данные. Они способны в разы сэкономить время и ресурсы, что действительно достаточно важно в наше время.

# Постановка задачи

В данной лабораторной работе было необходимо написать программу на языке программирования Си, состоящую из нескольких сортировок (в моём случае: сортировку выбором, сортировку вставками, сортировку Шелла и сортировку слиянием), проверить их на правильность выполнения поставленной им задачей (отсортировать элементы определённого массива), а также провести эксперимент на измерение времени, в зависимости от количества элементов в массиве, и проанализировать его результаты.

# Метод решения

**Сортировка выбором**

В данном алгоритме мы условно поделим массив на отсортированную часть и неотсортированную. Изначально воспринимая весь массив, как неотсортированную часть, мы выбираем из него наименьший элемент, и ставим его на первое место, то есть добавляя к отсортированной (до этого ещё пустой) части. После чего мы делаем так со всеми последующими элементами, находя среди них наименьший и ставя его в уже отсортированную часть массива, постепенно её увеличивая.

Общее время, потраченное на выполнение алгоритма, оценивается сверху О(N2) (где N – количество входных элементов массива); так как количество сравнений на каждом цикле для выбора минимального элемента уменьшается на 1, то есть всего сравнений получается n+(n-1)+...+2+1 = (n(n-1))/2.

**Сортировка вставками**

Аналогично сортировке выбором мы условно делим данный массив на отсортированную и неотсортированную части. Первый элемент мы принимаем за отсортированную часть, поэтому сортировка элементов начинается со второго. Прежде всего мы сравниваем его с первым, и если он оказывается меньше него, то он встает на позицию первого (если больше, он остаётся на своей позиции), и при этом наша отсортированная часть увеличивается на один элемент. После этого мы берём уже следующий элемент из неотсортированной части и сравниваем его с элементами из отсортированной и при необходимости меняем их местами, постепенно добавляя по одному элементу к отсортированной части массива. На выходе в итоге должен получиться массив, с расставленными элементами по возрастанию (или убыванию, в зависимости от кода и поставленной задачи).

Общее время, потраченное на выполнение алгоритма, оценивается сверху О(N2) (где N – количество входных элементов массива); так как количество сравнений на каждом цикле уменьшается на 1, то есть всего сравнений получается n+(n-1)+...+2+1 = (n(n-1))/2.

**Сортировка Шелла**

В данной сортировке сначала нам необходимо определиться с шагом, с помощью которого мы разобьём массив на определённые пары элементов. Этот шаг мы получаем делением общего количества элементов изначального массива на два(n/2). После чего мы берём первый элемент и сравниваем его с элементом, находящимся на промежутке равным нашему шагу (то есть он будет на позиции n/2+1), и меняем их местами в случае необходимости, и таким образом проходимся по всему массиву. Потом мы уменьшаем шаг в два раза и проделываем аналогичные действия (разбиваем массив на пары, сравниваем в них элементы и меняем местами), и так до тех пор, пока не получим упорядоченный массив.

Время выполнения алгоритма будет напрямую зависеть от изначальной последовательности: в лучшем и среднем случаях оно будет равно О(N log N), в худшем же – O(N2).

**Сортировка слиянием**

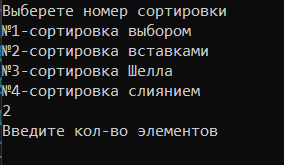
Идея этой сортировки заключается в следующем: первоначально исходный массив мы делим пополам, потом его половинки ещё пополам, и так до тех пор, пока не получим одноэлементные массивы. После чего мы берём каждую пару из таких массивов и применяем к ним процедуру слияния, сравнивая на каждом шаге первые элементы этих массивов, записывая наименьший из них во вспомогательный массив, далее возвращаемся к массиву, из которого выбрали наименьший и переходим к следующему элементу. Когда один из массивов закончится, оставшиеся элементы второго добавляются во вспомогательный массив. Процедура слияния применяется ко всем парам одноэлементных массивов, затем двухэлементным, после четырёхэлентным и так далее, пока во вспомогательном массиве не образуются отсортированный массив, с исходными данными. В итоге остаётся только скопировать элементы из вспомогательного массива в исходный массив.

Сначала во вспомогательный массив сливаются все пары одноэлементных массивов (всего операций 1+1+...+1 = n), затем пары двухэлементных массивов (всего операций 2+2+...+2 [всего n/2 слагаемых] = n) и так далее. Общая сумма операций n+n+...+n, всего слагаемых log2(n), так как массив каждый раз делится пополам. Среднее время работы: O(n\*log(n)).

# Руководство пользователя

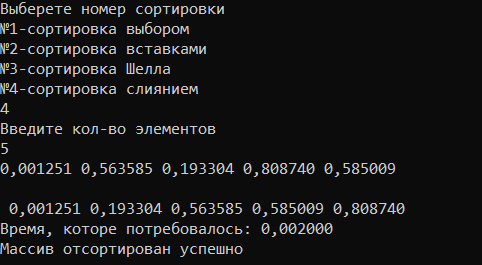
Когда пользователь запустит программу, перед ним на консоли появится подобие меню, в котором можно выбрать одну из четырёх представленных в работе сортировок, работующих с массивами, состоящие из чисел типа float.

После чего ему необходимо будет нажать на клавишу с номером сортировки, которую он хочет запустить. Далее пользователь должен будет ввести количество элементов в его изначальном массиве.



**Рис.1 Меню и ввод кол-ва элементов**

Затем пользователь после нажатия клавиши «enter» может увидеть на консоли, его исходный массив, тот же, но уже отсортированный массив, время, потраченное компьютером на сортировку, а также верно был отсортирован его массив или нет. После чего программа прекращает работу, и при желании поменять сортировку или количество элементов в массиве её нужно перезапустить.



**Рис.2 Результат работы программы**

# Описание программной реализации

Программа состоит из одного файла sorts.cpp, в котором есть все алгоритмы сортировок и вспомогательные функции к ним.

Всего в программе 11 функций, одна из которых main.

* **float\* creating(int size)**

Функция принимает на входе число типа int, отвечающее за размер массива, и создаёт массив из чисел типа float.

* **int compare(const void\* a, const void\* b)**

Функция на входе два указателя типа const void, сравнивает \*a и \*b, возвращает значения: -1, 0, 1, в зависимости от правила, по которому нужно сортировать (вспомогательная функция).

* **int check(float\* mas\_a, float\* mas\_b, int size)**

Функция принимает на входе два указателя на массивы типа float и размер массива типа int, сравнивает два массива одинаковой длины n. Один из них отсортирован встроенной функцией qsort, а другой какой-то из предложенных сортировок, которую выберет пользователь.

* **void swap(float\* a, float\* b)**

Функция на входе принимает два указателя на числа типа float и меняет местами значения a и b.

* **void select(float\* arr, int N)**

Функция принимает на входе указатель на массив типа float и его размер типа int. В ней имеются два цикла: внешний и внутренний. Внутренний отвечает за поиск элемента с минимальным значением в неотсортированной части массива, а внешний в свою очередь переставляет наименьший элемент в конец отсортированной части массива.

* **void insert(float\* arr, int N)**

Функция принимает на входе указатель на массив типа float и его размер типа int. В ней имеются так же, как и в предыдущей функции два цикла: внешний и внутренний. Внешний цикл просто проходит по всем элементам массива, а внутренний здесь отвечает за сравнение элемента из неотсортированной части массива с элементами из отсортированной и при необходимости в нём они меняются местами.

* **void shell(float\* arr, int N)**

Функция принимает на входе указатель на массив типа float и его размер типа int. В ней имеются три цикла. Внешний считает условный шаг, через который будут выбираться два элемента и сравниваться между собой. Прохождение по всем элементам массива, сравнение условно выбранных элементов и их перестановка при необходимости выполняются в двух внутренних циклах.

* **void merge(float\* in, float\* out, int l1, int l2, int r)**

Функция принимает исходный и вспомогательный массивы, переданные в mergesort, а также границы подмассивов: l1 — элемент с которого начнется слияние первого подмассива (включительно), l2 — элемент, на котором закончится сортировка первого подмассива (невключительно) и начнется сортировка второго подмассива (включительно), r — элемент, на котором закончится сортировка второго подмассива (невключительно).

* **void mergesort(float\* in, float\* out, int l, int r)**

Функция принимает на входе исходный массив, вспомогательный пустой массив, а также границы сортировки исходного массива: индекс l элемента, с которого нужно начать сортировку (включительно), и индекс r на котором нужно ее закончить (невключительно). Подпрограмма mergesort реализована рекурсивно. Внутри самой функции считается индекс m центрального элемента в массиве, а затем функция вызывает саму себя от параметров l и r, а также от m и r*.* Так функция делит исходный массив на каждом шаге на две части, пока эти подмассивы не будут состоять только из одного элемента. После вспомогательная функция merge()сливает подмассивы во вспомогательный массив out,благодаря чему и происходит сортировка.

* **int main()**

Основная функция, которая ничего на входе не принимает и в конце возвращает по сути 0. Она отвечает за выбор сортировки и их работу, а также вывод результата эксперимента на время в зависимости от введённого количества элементов массива.

# Подтверждение корректности

В данной программе для подтверждения корректности используются функция check, принимающая на входе два указателя на массивы типа float и размер массива типа int; оба массива являются исходными, но один из них будет отсортирован одним из представленных вариантов сортировок на выбор, а другой встроенной функцией qsort, для работы которой необходима вспомогательная функция (компаратор – в моём случае compare), что будет являться критерием сортировки. После чего происходит поэлементное сравнение двух отсортированных массивов: в случае несовпадения хотя бы одного элемента вернётся 0, в обратной же ситуации вернётся 1, что будет означать, что массивы равны и выбранная пользователем сортировка прошла успешно.

# Результаты экспериментов

Суть эксперимента в данной работе заключаются в том, чтобы измерить время работы определённых сортировок в зависимости количества элементов в массиве с помощью встроенной функции clock и сравнить результаты с теоретическими подсчётами. Результаты измерений занесены в таблицу, представленную ниже, а также на графике. Первый столбец представляет из себя вводимое количество элементов массива, в строках написано время работы сортировок в секундах.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | сортировка выбором | сортировка вставками | сортировка Шелла | сортировка слиянием |
| 1000 | 0,002 | 0,005 | 0,004 | 0,003 |
| 5000 | 0,037 | 0,059 | 0,147 | 0,005 |
| 10000 | 0,147 | 0,252 | 0,372 | 0,011 |
| 25000 | 0,893 | 1,47 | 1,887 | 0,028 |
| 50000 | 3,645 | 5,997 | 7,8 | 0,071 |
| 100000 | 14,378 | 30,446 | 29,433 | 0,174 |
| 500000 | 359,961 | 729,115 | 739,562 | 0,95 |
| 1000000 | 1452,393 | 2547,364 | 3594,639 | 1,865 |

**Рис.3 Результаты измерений**

Для того, чтобы сравнить полученные результаты с теоретическими необходимо их поделить полученное время у сортировок вставками и выбором на N2, а у сортировок Шелла и слиянием на log2(N)\*N, где N – количество элементов.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | сортировка выбором | сортировка вставками | сортировка Шелла | сортировка слиянием |
| 1000 | 0,000000002 | 0,000000005 | 4,01373E-07 | 3,0103E-07 |
| 5000 | 1,48E-09 | 2,36E-09 | 2,39263E-06 | 8,13821E-08 |
| 10000 | 1,47E-09 | 2,52E-09 | 2,79958E-06 | 8,27832E-08 |
| 25000 | 1,4288E-09 | 2,352E-09 | 5,16645E-06 | 7,66617E-08 |
| 50000 | 1,458E-09 | 2,3988E-09 | 9,99382E-06 | 9,09694E-08 |
| 100000 | 1,4378E-09 | 3,0446E-09 | 1,77204E-05 | 1,04758E-07 |
| 500000 | 1,43984E-09 | 2,91646E-09 | 7,813E-05 | 1,00361E-07 |
| 1000000 | 1,45239E-09 | 2,54736E-09 | 0,000180349 | 9,35702E-08 |

Из диаграмм, представленных ниже видно, что значение почти на каждом графике стремится к какой-то константе, из чего можно сделать вывод, что теоретическая оценка алгоритмов была верна.

**Рис.4-7 Сравнение теоретических расчётов времени с действительными**

# Заключение

В результате данной работы была выполнена выше поставленная цель: была написана программа на языке программирования Си, содержащая в себе несколько сортировок, которые были проверены на корректность их работы, а также был проведён эксперимент на измерение времени, в зависимости от количества элементов в массиве, результаты которого совпали с теоретической оценкой времени.

# Приложение

void swap(float\* a, float\* b)

{

float tmp = \*a;

\*a = \*b;

\*b = tmp;

}

void select(float\* arr, int N)

{

for (int i = 0; i < N - 1; i++)

{

int minind = i;

for (int j = i + 1; j < N; j++)

{

if (arr[minind] > arr[j])

minind = j;

}

swap(&arr[minind], &arr[i]);

}

}

void insert(float\* arr, int N)

{

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

{

for (int j = i; j > 0; j--)

{

if (arr[j] < arr[j - 1])

swap(&arr[j], &arr[j - 1]);

else

break;

}

}

}

void shell(float\* arr, int N)

{

for (int i = N / 2; i > 0; i /= 2)

{

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

{

for (int k = j; k - i >= 0; k -= i)

{

if (arr[k] < arr[k - i])

swap(&arr[k], &arr[k - i]);

}

}

}

}

void merge(float\* in, float\* out, int l1, int l2, int r)

{

int i, j, ind;

i = l1;

j = l2;

ind = l1;

while ((i < l2) && (j < r))

{

if ((in[i] - in[j]) < pow(10, -7))

{

out[ind] = in[i];

i++;

ind++;

}

else

{

out[ind] = in[j];

j++;

ind++;

}

}

while (i < l2)

{

out[ind] = in[i];

i++;

ind++;

}

while (j < r)

{

out[ind] = in[j];

j++;

ind++;

}

}

void mergesort(float\* in, float\* out, int l, int r)

{

int m, i;

if (l < r - 1)

{

m = r - ((r - l) + ((r - l) % 2)) / 2;

mergesort(in, out, l, m);

mergesort(in, out, m, r);

merge(in, out, l, m, r);

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

{

in[i] = out[i];

}

}

}