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 3822Б1ПМ1

Кириллов А. Д.

**Проверил**:

преподаватель каф. МОСТ,

Волокитин В.Д.

Нижний Новгород

2022

**Содержание**

[Постановка задачи 3](#_Toc26962562)

[Метод решения 4](#_Toc26962563)

[Руководство пользователя 6](#_Toc26962564)

[Описание программной реализации 7](#_Toc26962565)

[Подтверждение корректности 8](#_Toc26962566)

[Результаты экспериментов 9](#_Toc26962567)

[Заключение 11](#_Toc26962568)

[Приложение 12](#_Toc26962569)

# Постановка задачи

Цель – изучение алгоритмов сортировки, работающих за сложность , и .



Для достижения данной цели необходимо поставить следующие задачи:

1. Подготовить тестовую программу, которая умеет изменять размер массив чисел, переключаться между сортировками и выводить корректный ответ;
2. Изучить алгоритмы упорядочивания массива вещественных (типа float) знаковых чисел;
3. Сравнить скорости и эффективности данных сортировок, проверить, что их сложность действительно соответствует теоретической оценке.

# Метод решения

Описание алгоритмов ваших сортировок. Достаточно подробное

Сортировка выбором.

Данный алгоритм работает так, что, условно говоря, делит массив на две части, отсортированная и нет. Во время работы алгоритм проходит по массиву раз. Каждый раз берётся начало не отсортированного подмассива и сравнивается со всеми элементами этого подмассива, для поиска наименьшего элемента. Как только найдётся элемент меньше чем начальный элемент не отсортированного подмассива, его индекс запоминается и продолжается поиск ещё меньше элемента до конца подмассива. Когда был найден самый маленький элемент, то происходит замена элементов, начальный элемент подмассива меняется с наименьшим местами. Затем не отсортированный подмассив уменьшается на один элемент слева, и алгоритм повторяет свои действия с оставшимся не отсортированным подмассивом до тех пор пока длина этого подмассива не станет равна единице – это будет являться остановкой алгоритма так как в этом подмассиве останется только один элемент, который будет самым максимальным.



Сортировка Шелла.

Сортировка Шелла является усовершенствованным вариантом сортировки вставками. Тем самым эта сортировка попадает под категорию быстрые сортировки. Такого результата она добилась тем, что ставит несколько элементов практически на своё место за очень маленькое количество сравнений. Этот результат следует из того, что элементы сравниваются с определённым шагом, а не как в сортировке вставками. Шагом в данной сортировке является деление размера массива нацело на 2, так после первого деления нацело на 2 происходит сравнение элементов одного подмассива с элементами других подмассивов с тем же индексом только к индексу надо прибавить шаг. При сравнение происходит замена элементов (если элемент из левого подмассива оказался больше элемента из правого подмассива, то они меняются местами). Затем шаг вновь делится нацело на 2, и все действия повторяются до того момента, пока шаг не станет равен единице. Как только шаг становится равен единице, подключается сортировка вставками, которой надо сделать намного меньше сравнений и замен. Из-за того, что используется шаг с постоянным делением нацело на 2, асимптотика равна .



Сортировка Радикс(LSD).

Сортировка LSD отличается от всех квадратичных и быстрых сортировок тем, что в её алгоритме не присутствует сравнений элементов. Эта сортировка называется поразрядная, так как для сортировки элементов использует разряды числа, для сортирования элементов надо знать изначально к какому типу данных относится массив. Идея этой сортировки состоит в том, что сначала берётся последний разряд у каждого элемента и происходит сортировка этих последних разрядов, сохраняя порядок элементов с одинаковыми разрядами. Так с последнего разряда и до самого первого происходит сортирование. Так как при сортировке по каждому разряду элементы сортируются с сохранением порядка следования при одинаковых разрядах, итоговый массив будет отсортирован правильно. Для сортировки float нужно работать с мантиссой. В итоге получится, что отрицательные числа будут идти в порядке убывания. Это будет происходить из-за того, что последний бит отвечает за знак: 1. – это отрицательное число, а 0. –положительное.

# Руководство пользователя

При запуске программы, пользователь должен ввести параметры массива:

1) Размер массива;

2) Левую границу массива;

3) Правую границу массива.

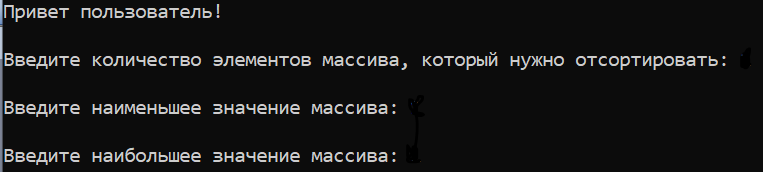


Рисунок - Параметры массива

Затем пользователю предоставляется выбор действия из меню.

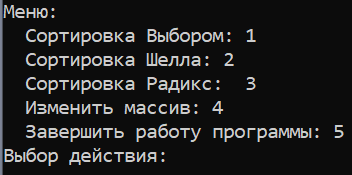


Рисунок 2 - Меню

В меню можно выбрать вид сортировки под вариантами: №1, №2, №3, изменение массива №4, а так же под №5 завершение программы.

Затем программа выводит результат в виде количества неправильно отсортированных элементов массива и времени работы сортировки.



Рисунок 3 – Результат

Как только программа вывела результат, вновь появляется меню.

Программа будет работать до того момента, пока не завершить её действием №5.



Рисунок 4 - Завершение

# Описание программной реализации

Проект состоит из 12 функций, одна из которых main.

void swap(float\* number1, float\* number2)

Функция принимает на вход 2 указателя типа float number1 и number2, которые нужно поменять между собой.

void selectsort(float\* array, int size):

Функция принимает на вход массив array типа float и размер массива size типа int, затем сортирует массив сортировкой выбором.

float\* creat\_array\_fulled (int size, int left\_number,int right\_number):

Функция принимает на вход три параметра типа int для создания массива: размер size, левую границу массива left\_number, правую границу массива right\_number.

По данным параметрам выделяется и заполняется массив.

float\* creat\_array\_empty(int size):

Функция принимает на вход размер size типа int и создает пустой массив.

void examination(float\* mass1, float\* mass2, int size):

Функция принимает на вход 2 массива mass1 и mass2 типа float, отсортированные разными сортировками и их размер size типа int. Производит проверку на правильность сортирования массива mass1 с массивом mass2, который был отсортирован стандартной функцией в С qsort.

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

Функция принимает 2 указателя a и b типа const void, сравнивает их и возвращает -1, 0, 1 (вспомогательная функция).

void array\_copy(float\* mass\_origin, float\* mass\_copy, int size):

Функция принимает на вход 2 массива mass\_origin и mass\_copy типа float и размер size типа int и копирует mass\_origin в mass\_copy.

void shellsort(float\* mass, int size):

Функция принимает на вход массив mass типа float и размер массива size типа int, затем сортирует массив сортировкой Шелла.

void radixfloat(float\* mas, float\* tmp, int size):

Функция принимает на вход 2 массива mas и tmp типа float и их размер size типа float, затем сортирует массив mas и записывает значения в массив tmp.

void radix(unsigned int\* mass, int size, unsigned int\* tmp):

Функция принимает на вход 2 массива типа и их размер типа и производит сортировку Радикс.

void countbyte(unsigned int\* arr, int size, int count[256], int byte):

Функция принимает на вход 2 массива arr и count[256] типа unsigned int и типа int, их размер size типа int и размер выбранного типа данных в байтах byte типа int, используется как вспомогательная функция y radix.

int main():

основная функция

# Подтверждение корректности

Для подтверждения корректности в программе используется функция examination. Ей на вход подаётся 2 массива, один из них был отсортирован одной из выбранных сортировок, а другой встроенной функцией qsort (подаётся не отсортированный массив, размер массива, размер типа данных, дополнительная функция compare). Внутри функции происходит сравнение элементов поэлементно. Если элементы с одинаковыми индексами в отсортированных массивах не совпали, то счётчик неправильно отсортированных элементов увеличивается на один и в результате работы программы пишется количество не правильно отсортированных элементов.

# Результаты экспериментов

Для тестирования сортировок я использовал массивы размером

1. 1000 элементов
2. 10000 элементов
3. 50000 элементов
4. 100000 элементов
5. 250000 элементов
6. 500000 элементов
7. 1000000 элементов
8. 5000000 элементов
9. 10000000 элементов
10. 50000000 элементов
11. 100000000 элементов



Для сортировки выбором было проверено только до 1000000 элементов в отличии от остальных сортировок, так как время ожидания невероятно росло.



По данным экспериментов видно, что сортировка выбором оказалась самой медленной, сортировка Шелла довольно таки быстрая, но не быстрее сортировки Радикс.

# Заключение

После проведения тестов, можно сказать, что лучшей сортировкой оказалась сортировка Радикс, затем по эффективности идёт сортировка Шелла, и самой медленной из рассмотренных сортировок оказалась сортировка выбором. Для проведения тестов и вывода результатов была написана программа, состоящая из алгоритмов сортировок и вспомогательных функция для удобной работы.

# Приложение

void swap(float\* number1, float\* number2)//замена значений

{

float tmp = \*number1;

\*number1 = \*number2;

\*number2 = tmp;

}

Алгоритм сортировки Выбором:

void selectsort(float\* array, int size)

{

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

{

int minindex = i;

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

{

if (array[minindex] > array[j])

{

minindex = j;

}

}

swap(&array[i], &array[minindex]);

}

}

Алгоритм сортировки Шелла:

void shellsort(float\* mass, int size)

{

int i, j, k;

for (i = size / 2; i > 0; i /= 2)

for (j = i; j < size; j++)

for (k = j; k - i >= 0 && mass[k] < mass[k - i]; k -= i)

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

}

Алгоритм сортировки Радикс:

void countbyte(unsigned int\* arr, int size, int count[256], int byte)

{

unsigned char\* arrC;

int k,j,i,bias;

int tmp, temp;

bias = sizeof(unsigned int);

arrC = (unsigned char\*)arr;

for (k = 0; k < 256; k++)

count[k] = 0;

for (j = 0; j < n; j++)

count[arrC[j \* bias + byte]]++;

tmp = count[0];

count[0] = 0;

for (i = 1; i < 256; i++)

{

temp = count[i];

count[i] = count[i - 1] + tmp;

tmp = temp;

}

}

void radix(unsigned int\* mass, int size, unsigned int\* tmp)

{

unsigned char\* arrc = (unsigned char\*)mass;

int i,j,k,sizetype = sizeof(unsigned int);

int count[256];

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

{

countbyte(mass, size, count, i);

for (j = 0; j < size; j++)

tmp[count[arrc[j \* sizetype + i]]++] = mass[j];

for (k = 0; k < size; k++)

mass[k] = tmp[k];

}

}

void radixfloat(float\* mas, float\* tmp, int size)

{

int count = 0, i;

radix((unsigned int\*)mas, size, (unsigned int\*)tmp);

while (count < size && mas[++count] >= 0);

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

{

tmp[i + size - count] = mas[i];

}

for (; i < size; i++)

{

tmp[size - (++count)] = mas[i];

}

}