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 382003-1

**Ивлев А.Д.**

**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

Реализовать на языке программирования C сортировку пузырьком, Шелла, слиянием и поразрядную сортировку для чисел типа float. Проверить их теоретическую сложность. И проверить их корректность.

# Метод решения

1. Сортировка пузырьком(Bubble Sort). Проходы по массиву повторяются n-1 (n - кол-во элементов массива) раз. При каждом проходе алгоритма по внутреннему циклу, очередной наибольший элемент массива ставится на своё место в конце массива рядом с предыдущим наибольшим элементом, а наименьший элемент перемещается на одну позицию к началу массива.
2. Сортировка Шелла(Shell Sort). При сортировке Шелла сначала сравниваются и сортируются между собой значения, стоящие один от другого на некотором расстоянии (step-шаг). После этого процедура повторяется для некоторых меньших значений шага, а завершается сортировка Шелла упорядочиванием элементов при step = 1. Я выбрал за шаг последовательность Хиббарда step = 2^k - 1 <= n.
3. Сортировка слиянием(Merge Sort). Для сортировки сортируемый массив разбивается на две части примерно пополам. Каждая из получившихся частей рекурсивно сортируется отдельно. Упорядоченные массивы половинного размера возвращаются в исходный.
4. Поразрядная сортировка(Radix Sort). Линейный алгоритм сортировки, не использующий сравнения. Сначала сортируются значения одного крайнего правого разряда, и элементы группируются по результатам этого сравнения, затем сравниваются значения следующего разряда, соседнего, и элементы упорядочиваются по результатам сравнения значений этого разряда внутри образованных на предыдущем проходе групп сохраняя порядок, достигнутый при предыдущей сортировке. Затем аналогично делается для следующего разряда, и так до конца.

# Руководство пользователя

Пользователь запускает программу вводит количество элементов массива и сам массив. Затем выбирает проверить ли отсортированный массив. Далее выбирает одну из 4-х сортировок. На выход получает количество перестановок, сравнений и отсортированный массив, если выбрал проверку, то Ok, если массив отсортирован правильно или Error, если неправильно.

# Описание программной реализации

MainProgramm.c - главная программа, где вводится длина массива n, сам массив newarray[], пользователь выбирает делать проверку или нет, выбирает одну из сортировок. На выход получает количество перестановок, сравнений и отсортированный массив.

Sort.c - программа с функциями сортировок. Содержит функции сортировок BubbleSort, ShellSort, MergeSort, RadixSort. CreateCounters, Merge и RadixP - подфункции. О входных данных подробнее в комментариях к коду.

TestSort.c - программа с функцией проверки сортировки. (Далее в подтверждении корректности)

Sort.h и TestSort.h - файлы с заголовками функций.

# Подтверждение корректности

Для подтверждения корректности в программе используется функция TestSort. Проверяет, что каждый элемент меньше или равен предыдущему, если это не так, то выводит Error, если верно использует сортировку пузырьком и сравнивает результат. При несовпадении выводит номера этих элементов, если всё совпадает выводит Ok.

# Результаты экспериментов

В таблицах первая строка перестановки, вторая сравнения.

По данным экспериментов -

* Bubble sort
  + Теоретическая сложность O(n^2)
  + Дополнительная память O(0)
  + Не устойчивая

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| n | 5 | 10 | 25 | 100 | 500 | 1000 |
| Лучший случай | 0 | 0 | 0 | 0 | 0 | 0 |
| Лучший случай | 10 | 45 | 300 | 4950 | 124750 | 499500 |
| Средний случай | 2 | 13 | 189 | 2360 | 62420 | 248145 |
| Средний случай | 10 | 45 | 300 | 4950 | 124750 | 499500 |
| Худший случай | 10 | 45 | 300 | 4950 | 124750 | 499500 |
| Худший случай | 10 | 45 | 300 | 4950 | 124750 | 499500 |

Лучший случай - отсортированный массив((n-1)\*n/2 сравнений, 0 перестановок)

Средний случай - случайный массив((n-1)\*n/2 сравнений, от 1 до (n-1)\*n/2 перестановок)

Худший случай - инверсивный отсортированный массив((n-1)\*n/2 сравнений, (n- 1)\*n/2 перестановок)

* Shell sort
  + Теоретическая сложность O(n^(3/2))
  + Дополнительная память O(0)
  + Устойчивая

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| n | 5 | 10 | 25 | 100 | 500 | 1000 |
| Лучший случай | 0 | 0 | 0 | 0 | 0 | 0 |
| Лучший случай | 6 | 19 | 74 | 480 | 3498 | 7987 |
| Средний случай | 2 | 10 | 41 | 398 | 2792 | 6973 |
| Средний случай | 6 | 19 | 74 | 280 | 3498 | 7987 |

Лучший случай - отсортированный массив(от n^(1.1) до n^(1.5) сравнений, 0 перестановок)

Средний случай - случайный массив(от n^(1.1) до n^(1.5) сравнений, от 1 до n^(1.5) перестановок)

Худший случай трудно проверить экспериментальным путём

* Merge sort
  + Теоретическая сложность O(n\*log(n))
  + Дополнительная память O(n)
  + Устойчивая

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| n | 5 | 10 | 25 | 100 | 500 | 1000 |
| Лучший случай | 7 | 19 | 64 | 356 | 2272 | 5044 |
| Лучший случай | 12 | 34 | 118 | 672 | 4488 | 9976 |
| Средний случай | 7 | 19 | 64 | 356 | 2272 | 5044 |
| Средний случай | 12 | 34 | 118 | 672 | 4488 | 9976 |

Лучший случай - отсортированный массив(~n\*log2(n) сравнений, ~n\*log3(n) перестановок)

Средний случай - случайный массив(~n\*log2(n) сравнений, ~n\*log3(n) перестановок)

Худший случай трудно проверить экспериментальным путём

* Radix sort
  + Теоретическая сложность O(k\*(n+m))
  + Дополнительная память O(n+m)
  + Устойчивая

Так как Radix Sort - линейный алгоритм сортировки, не использующий сравнения, то можно не рассматривать количество сравнений (всегда 0) и различные случаи.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| n | 5 | 10 | 25 | 100 | 500 | 1000 |
| Перестановки | 20 | 40 | 100 | 400 | 2000 | 4000 |

Сложность (4\*n).

# Заключение

В ходе работы мне удалось реализовать на языке программирования C сортировку пузырьком, Шелла, слиянием и поразрядную сортировку для чисел типа float и проверить их корректность.

Полученная сложность алгоритмов совпала с теоретической сложностью с точностью до констант.

# Приложение

Смотреть весь код в файлах MainProgramm.c, Sort.c, TestSort.c.

Пример кода функции сортировки слиянием(Merge Sort).

void Merge(unsigned int f, unsigned int l, float array[], float dparray[])

//Подпрограмма сортировки слиянием

//Сортирует и возвращает под массивы

//f - элемент начала сортировки (0 по умолчанию, сортирует с начала)

//l - элемент конца сортировки (n-1 по умолчанию, сортирует до конца)

//array[] - массив для сортировки, dparray[] - дополнительный массив размером n

{

unsigned int mid = (f + l)/2;

unsigned int st = f;

unsigned int fin = mid + 1;

for (unsigned int j = f; j<=l; j++)

{

comparison++;

if ((st<=mid) && ((fin>l) || (array[st]<array[fin])))

{

dparray[j] = array[st];

st++;

permutation++;

}

else

{

dparray[j] = array[fin];

fin++;

}

}

for (unsigned int i = f; i <= l; i++)

{

array[i] = dparray[i];

}

}

void MergeSort(unsigned int f, unsigned int l, float array[], float dparray[])

//Сортировка слиянием

//f - элемент начала сортировки (0 по умолчанию, сортирует с начала)

//l - элемент конца сортировки (n-1 по умолчанию, сортирует до конца)

//array[] - массив для сортировки, dparray[] - дополнительный массив размером n

//n - длина массива для сортировки

{

if (f<l)

{

unsigned int mid = (f + l)/2;

MergeSort(f, mid, array, dparray);

MergeSort(mid+1, l, array, dparray);

Merge(f, l, array, dparray);

}

}