Федеральное агентство по образованию Российской Федерации

Государственное образовательное учреждение

высшего профессионального образования

Нижегородский государственный университет им. Н.И. Лобачевского

Институт информационных технологий математики и механики

Отчёт по лабораторной работе

Лабораторная работа №2.

Выполнил:

студент ИИТММ гр. 3824Б1ФИ1

Галочкин Д. А.

Проверил:

старший преподаватель кафедры ВВСП

Лебедев И. Г

Нижний Новгород

2024 г.

Содержание

[Введение 3](file:///C:\Users\Misha\Desktop\лаба.doc#_Toc270962758)

[Постановка задачи 4](file:///C:\Users\Misha\Desktop\лаба.doc#_Toc270962759)

[Руководство пользователя 5](file:///C:\Users\Misha\Desktop\лаба.doc#_Toc270962760)

[Руководство программиста 6](file:///C:\Users\Misha\Desktop\лаба.doc#_Toc270962761)

[Описание структур данных 6](file:///C:\Users\Misha\Desktop\лаба.doc#_Toc270962762)

[Описание функций 6](file:///C:\Users\Misha\Desktop\лаба.doc#_Toc270962763)

[Эксперименты 7](file:///C:\Users\Misha\Desktop\лаба.doc#_Toc270962765)

Заключение 8

Приложение 9

# Введение

Сортировка массивов данных является одной из основополагающих задач в области информатики и программирования. Эффективные алгоритмы сортировки не только упрощают работу с данными, но и значительно повышают производительность различных приложений. В данной лабораторной работе мы рассмотрим и сравним несколько алгоритмов сортировки на языке C, таких как сортировка пузырьком, выбором, вставками, слиянием, быстрая сортировка (сортировка Хоара), сортировка Шелла и сортировка подсчетом.

В рамках работы будет реализована программа, способная создавать массивы данных различными способами — случайным образом, через ввод с клавиатуры или чтением из текстового файлы. Это позволит исследовать поведение алгоритмов сортировки на различных объемах данных и различных типах входных массивов. Сравнение времени выполнения различных сортировок даст возможность выявить их сильные и слабые стороны, что окажется полезным для выбора наиболее подходящего алгоритма в практических задачах.

Кроме основной задачи, в лабораторной работе предусмотрены дополнительные задания, направленные на углубление навыков работы с массивами и понимание различных норм векторов.

Данная лабораторная работа даст возможность ознакомиться с различными алгоритмами сортировки и получить ряд практических знаний, необходимых для эффективной работы с массивами данных в языке C. Полученные результаты и выводы помогут в дальнейшем выбирать наиболее подходящие алгоритмы в зависимости от конкретной задачи и требований к производительности.

# Постановка задачи

В задаче можно выделить следующие этапы:

1. **Изучить и реализовать алгоритмы сортировки:**
   * Сортировка пузырьком
   * Сортировка выбором
   * Сортировка вставками
   * Сортировка слиянием
   * Быстрая сортировка (сортировка Хоара)
   * Сортировка Шелла
   * Сортировка подсчетом
2. **Создать программу на языке C, которая будет:**
   * Генерировать массивы случайных чисел заданного пользователем размера и диапазона, вводить массив с клавиатуры или файла.
   * Производить сортировку массива с использованием различных алгоритмов по выбору пользователя.
   * Выводить массив в консоль или записывать его в файл.
   * Измерять и выводить время выполнения каждого алгоритма сортировки для различных размеров массивов.
3. **Разработать пользовательский интерфейс, который будет:**
   * Позволять пользователю выбирать алгоритм сортировки и входные данные.
   * Обеспечивать возможность ввода массива с клавиатуры или загрузки из текстового файла.
   * Предоставлять возможность сброса и повторного выбора исходных данных.
4. **Выполнить дополнительные задания:**
   * Реализовать функции для вычисления норм вектора: первой, второй, Гельдеровой и бесконечной норм.
   * Разработать функцию для нормализации вектора.
   * Провести анализ эффективности алгоритмов сортировки на различных типах и объемах данных.
5. **Сравнить производительность различных алгоритмов,** проанализировав полученные временные результаты и сделав выводы о преимуществе того или иного метода в зависимости от входных данных.

# Руководство пользователя

При запуске программы через консоль, передайте 3 параметра:

1. Имя файла, в котором содержится массив.

2. Имя файла, в который нужно записать отсортированный массив.

3. Имя сортировки:

*bubble* – сортировка пузырьком

*selection* – сортировка пузырьком

*insertion* – сортировка пузырьком

*quick* – сортировка пузырьком

*merge* – сортировка пузырьком

*shell* – сортировка пузырьком

*count* – сортировка пузырьком

После выполнения программы в файле выхода будет записан отсортированный массив.

При обычной работе с программой:

1. Сгенерировать массив случайных чисел

1.1. Введите n – размерность массива.

1.2 Введите min – нижняя граница генерируемых чисел

1.3 Введите max – верхняя граница генерируемых чисел

2. Ввести массив с клавиатуры

1.1 Введите n – размерность массива.

1.2 Введите n чисел – элементы массива.

3. Ввести массив из файла

Введите inputName - имя файла, из которого нужно прочитать массив.

4. Записать массив в файл

Введите outputName – имя файла, в который нужно записать массив.

5. Сортировка массива.

5.1 Сортировка пузырьком

5.2 Сортировка выбором

5.3 Сортировка вставками

5.4 Сортировка слиянием

5.5 Быстрая сортировка

5.6 Сортировка Шелла

5.7 Сортировка подсчётом

По окончании программа выведет время сортировки массива.

6. Вывести в консоль норму вектора

6.1 Первая норма

6.2 Вторая норма

6.3 Гельдерова норма

6.4 Бесконечная норма

6.5 К-ая норма

Введите k.

7. Вывести нормированный вектор.

8. Сбросить вектор.

9. Вывести вектор в консоль.

10. Выйти

# Руководство программиста

## Описание структуры программы

Программа состоит из трех файлов

*main.c* – содержит интерфейс программы.

*vector.c* – содержит функции для работы с векторами.

*vector.h* – заголовочный файл с функциями для работы с векторами.

***Описание функций программы***

***main.c***

*void ClearScreen()* – очищает экран консоли

*void Pause()* – приостанавливает работу программы

*void ReadString(char\*\* s) –* читает строку из консоли

***vector.c***

*void swap(int \*i, int \*j)* – меняет местами значения двух переменных.

*void FreeVector(int\* n, int\* vector) –* освобождает вектор.

*void CreateVectorFromFile(int\* n, int\*\* vector, char\* fileName)* – создать вектор из файла

*void WriteVectorToFile(int n, int\* vector, char\* fileName)* – записать вектор в файл.

*void CreateVector(int n, int\*\* vector)* – выделить память под вектор заданного размера.

*void FillVectorRandom(int n, int\* vector, int min, int max)* – заполнить вектор случайными числами из заданного диапазона

*void FillVectorKeyboard(int n, int\* vector)* – заполнить вектор с клавиатуры.

*void PrintVector(int n, int\* vector)* – вывести вектор в консоль.

*void BubbleSort(int n, int\* vector) –* сортировка пузырьком

*void SelectionSort(int n, int\* vector)* – сортировка выбором

*void InsertionSort(int n, int\* vector)* – сортировка вставками

*void Merge(int l, int r, int mid, int n, int\* vector, int\* auxVector)* – слияние двух отсортированных массивов, участвует в сортировке слиянием

*void MergeRecursive(int l, int r, int n, int\* vector, int\* auxVector)* – рекурсивная процедура сортировки слиянием.

*void MergeSort(int n, int\* vector)* – сортировка слиянием

*int Partition(int l, int r, int n, int\* vector)* – разбиение массива относительно опорного элемента, участвует в быстрой сортировке

*void Qsort(int n, int\* vector, int l, int r)* – рекурсивная процедура быстрой сортировки

*void QuickSort(int n, int\* vector)* – быстрая сортировка

*void ShellSort(int n, int\* vector) –* сортировка Шелла

*void CountSort(int n, int\* vector) –* сортировка подсчётом

*int FirstNorm(int n, int\* vector)* – первая норма вектора

*int InfNorm(int n, int\* vector)* – бесконечная норма вектора

*double NthNorm(int n, int\* vector, int k)* – К-ая норма вектора

*void Normalize(int n, int\* vector, double\*\* vectorResult)* – создает нормализированный вектор.

# Эксперименты

Эксперименты проводились на компьютере со следующими характеристиками:

Процессор - Intel(R) Xeon(R) CPU E5-2650 v2 @ 2.60GHz

Оперативная память – 32 ГБ

Диапазон значений (-100000, 100000)

Таблица . Время работы сортировки пузырьком

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 100 | 1000 | 10000 | 100000 | 1000000 | 10000000 | 100000000 |
| 0.0000s | 0.0050s | 0.4080s | 40.5690s | - | - | - |

Таблица 2. Время работы сортировки выбором

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 100 | 1000 | 10000 | 100000 | 1000000 | 10000000 | 100000000 |
| 0.0000s | 0.0010s | 0.1480s | 14.7430s | - | - | - |

Таблица 3. Время работы сортировки вставками

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 100 | 1000 | 10000 | 100000 | 1000000 | 10000000 | 100000000 |
| 0.0000s | 0.0020s | 0.1910s | 19.4530s | - | - | - |

Таблица 4. Время работы быстрой сортировки

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 100 | 1000 | 10000 | 100000 | 1000000 | 10000000 | 100000000 |
| 0.0000s | 0.0000s | 0.0010s | 0.0160s | 0.1650s | 1.6510s | 18.3730s |

Таблица 5. Время работы сортировки слиянием

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 100 | 1000 | 10000 | 100000 | 1000000 | 10000000 | 100000000 |
| 0.0000s | 0.0000s | 0.0020s | 0.0190s | 0.2330s | 2.5110s | 27.2300s |

Таблица 6. Время работы сортировки Шелла

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 100 | 1000 | 10000 | 100000 | 1000000 | 10000000 | 100000000 |
| 0.0000s | 0.0010s | 0.0030s | 0.0310s | 0.4160s | 4.8300s | 57.8830s |

Таблица 7. Время работы сортировки подсчётом

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 100 | 1000 | 10000 | 100000 | 1000000 | 10000000 | 100000000 |
| 0.0000s | 0.0000s | 0.0000s | 0.0010s | 0.0090s | 0.0870s | 0.9270s |

# Заключение

Сортировка подсчётом оказалась самой эффективной. Однако стоит отметить ее недостаток – при больших диапазонах генерируемых чисел она будет требовать слишком много памяти.  
Время работы – O(n + a), затраты памяти – O(a), где, a – диапазон.

Быстрая сортировка оказалась менее эффективна, чем сортировка подсчётом, но ее расходы по памяти минимальны, благодаря чему она больше подойдёт в ситуациях, когда необходимо экономить память. Однако она имеет и недостаток – на специально сгенерированном наборе ее сложность будет квадратичной.  
Время работы в среднем – O(nlogn), худшее время работы – O(n2), затраты памяти – O(1).

Сортировка слиянием работает немного медленнее, чем быстрая сортировка. Но она компенсирует это тем, что гарантирует сложность O(nlogn) в любом случае.  
Время работы – O(nlogn), затраты памяти – O(n).

Сортировка Шелла, несмотря на квадратичную сложность в худшем случае, работает достаточно быстро. Ее достоинством является баланс между сложностью ее написания и времени работы.  
Лучшее время работы – O(nlog2n), худшее время работы – O(n2), затраты памяти – O(1).

Сортировки пузырьком, выбором и вставками имеют более ознакомительный характер, нежели практическое применение. Все они имеют квадратичную сложность и не способны конкурировать в сравнении с другими, более эффективными алгоритмами.  
Время работы – O(n2), затраты памяти – O(1).

**Приложение:**

<https://github.com/dmgalochkin/first-year-second-lab>