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 3823Б1ПМ1

Алемаев М.В.

**Проверил**:

преподаватель каф. ВВСП,

Волокитин В.Д.

Нижний Новгород

2023

**Содержание**

[Постановка задачи 3](#_Toc26962562)

[Метод решения 4](#_Toc26962563)

[Руководство пользователя 6](#_Toc26962564)

[Описание программной реализации 7](#_Toc26962565)

[Подтверждение корректности](#_Toc26962566) 10

[Результаты экспериментов 11](#_Toc26962567)

[Заключение 13](#_Toc26962568)

[Приложение 14](#_Toc26962569)

# Постановка задачи

* Реализовать сортировки выбором, расческой, слиянием и поразрядную, используя тип данных float в языке программирования Си
* Создать пользовательский интерфейс для сортировки и анализа сортировок
* Провести тестирование сортировок, проверить на корректность
* Провести ряд эксперементов, в ходе которых доказать асимптотическую сложность алгоритмов, построить график отношения времени к асимптотической сложности и найти константу.

# Метод решения

1. Сортировка выбором (Selection sort)

Средний случай – O(n^2)

Лучший случай – O(n^2)

Худший случай – O(n^2)

Сначала отыскивается наименьший элемент массива, затем он меняется местами с элементом, стоящим первым в сортируемом массиве. Далее находится следующий минимальный элемент и меняется местами с элементом, стоящим вторым в исходном массиве и так до последнего элемента.

1. Сортировка расческой (Comb sort)

Средний случай – O(nlog(n))

Лучший случай – O(nlog(n))

Худший случай – O(n^2)

Сортировка расчёской — улучшение сортировки пузырьком. Её идея состоит в том, чтобы «устранить» элементы с небольшими значения в конце массива, которые замедляют работу алгоритма. При «расчёсывании» сначала берётся достаточно большое расстояние между сравниваемыми значениями, а потом оно сужается вплоть до минимального.

Первоначальный разрыв нужно выбирать не случайным образом, а с учётом специальной величины — фактора уменьшения – 1.2473309… Сначала расстояние между элементами будет равняться размеру массива, и на каждом последующем шаге расстояние будет снова делиться на фактор уменьшения — и так пока шаг не станет равным 1.

1. Сортировка слиянием (Merge sort).

Средний случай – O(nlog(n))

Лучший случай – O(nlog(n))

Худший случай – O(nlog(n))

Сортировка происходит по следующему алгоритму[[1]](#footnote-1):

* **Разделение**: сортируемая последовательность, состоящая из n элементов, разбивается на две меньшие последовательности, каждая из которых содержит n/2 элементов.
* **Покорение**: сортировка обеих вспомогательных последовательностей методом слияния.
* **Комбинирование**: слияние двух отсортированных последовательностей для получения окончательного результата.

1. Поразрядная сортировка (Radix sort)

Средний случай – O(n)

Лучший случай – O(n)

Худший случай – O(n)

Сравнение производится поразрядно: сначала сравниваются значения одного крайнего разряда, и элементы группируются по результатам этого сравнения, затем сравниваются значения следующего разряда, соседнего, и элементы либо упорядочиваются по результатам сравнения значений этого разряда внутри образованных на предыдущем проходе групп, либо переупорядочиваются в целом, но сохраняя относительный порядок, достигнутый при предыдущей сортировке. Затем аналогично делается для следующего разряда, и так до конца.

Поскольку мы рассматриваем вариацию LSD, то элементы переупорядочатся полностью.

# Руководство пользователя

После запуска программы, на экране монитора пользователя появится выбор сортировки:

1 – Сортировка выбором

2 – Сортировка расческой

3 – Сортировка слиянием

4 – Поразрядная сортировка

После выбора сортировки программа попросит указать порядок, в котором будут выводится элементы массива

0 – по возрастанию

1 – по убыванию

Далее надо будет написать количество элементов и ввести их по порядку или сделать случайную генерацию чисел от -10^6 до 10^6.

В качестве результата на экране дисплея появится:

* количество тактов процесса, в ходе выполнения сортировки
* проверка корректности работы

Также можно будет вывести и сам массив.

По завершению у пользователя будет выбор:

0 – выйти из программы

1 – продолжить работу

# Описание программной реализации

1. Сортировка выбором

Функция представляет из себя для вложенных массива. При каждой итерации внешнего цикла мы присваиваем переменной minI значение текущего индекса. В ходе работы внутреннего цикла присваиваем minI индекс наименьшего элемента в диапазоне от I до n.

CHOOSE(A, n)

1 for i = 0 to n

2 minI = i

3 for j = i + 1 to n

4 if A[j] < A[minI]

5 minI = j

6 swap(a[minI], a[i])

1. Сортировка расческой

Программная реализация включает в себя 2 вложенных списка. While и For соответственно. Вначале задается переменная step равная последнему элементу массива, которая после каждой итерации внешнего цикла делится на factor. А внутренний цикл меняет местами два элемента, если они стоят в порядке убывания.

По выходу из цикла While действует квадратичная сортировка, так как значение factor нельзя точно представить в виде десятичной дроби и присутствует погрешность.

COMB(A, n)

1 step = n - 1

2 while step >= 1

3 for j = 0 to n - step

4 if A[j] > A[j + step]

5 swap(A[j], A[j + step])

6 step = step / factor

7 QUADRATIC SORT(A, n)

1. Сортировка слиянием

В программе реализован рекурсивный алгоритм, построенный по принципу «разделяй и властвуй».

* Делим исходный массив на две примерно равные половины до тех пор пока не будет рассмотрен случай массива из одного элемента, который считается отсортированным.
* Помещаем данные двух массивов во временный массив tmp, в отсортированном виде, используя метод двух указателей.
* Переносим данные из временного хранилища в исходный массив

MERGE\_SORT(A, tmp, l, r)

1 if l<r

2 then mid ← l + (r – l)/2

3 MERGE\_SORT(A, tmp, l, mid)

4 MERGE\_SORT(A, tmp, mid+1, r)

5 MERGE(A, tmp, l, r)

MERGE(A, tmp, l, r)

1 mid = (l+r)/2

2 lp = l; rp = mid + 1

3 k = l

4 while (lp <= mid and rp <= r)

5 if A[lp] > A[rp]

6 tmp[k] = A[rp++]

7 else

8 tmp[k] = A[lp++]

9 k = k + 1

10 for i = lp to mid + 1 do k = k + 1

11 tmp[k] = A[i]

12 for i = rp to r + 1 do k = k + 1

13 tmp[k] = A[i]

14 for i = l to r + 1 do i = i + 1

15 A[i] = tmp[i]

1. Поразрядная сортировка (Radix sort)

Данная сортировка называется Least significant digit (или восходящая) и работает на основе сортировки подсчетом. Основная идея заключается в том, чтобы сортировать элементы массива по отдельным разрядам чисел. В данном случае мы сортируем массив чисел типа float.

Алгоритм начинается с преобразования каждого элемента массива arr в последовательность байтов с помощью указателя pm. Затем выделяется временный массив tmp для записи отсортированных по одному байту элементов.

Затем алгоритм выполняет несколько итераций сортировки по отдельным байтам чисел типа float. На каждой итерации мы делаем сортировку подсчетом для массива count по каждому из 4-х байт, идя от младшего бита к старшему.

Далее совершаем обратный проход по массиву arr. Для каждого элемента алгоритм находит соответствующий элемент массива count и уменьшает его.

Затем алгоритм записывает текущий элемент в массив output, используя уменьшенное значение элемента массива count.

В конце меняем указатели исходного массива и временного.

RADIX\_SORT(A, n)

1 PM = (UNSIGNED CHAR\*)A

2 output[n]; count[256] = {0}

3 for i = 0 to sizeof(float)

4 for I = 0 to n

5 count[PM[i \* sizeof(float) + byte\_num]]++

6 for I = 1 to 256

7 count[i] = count[i] + count[i-1]

8 for I = 0 to n

9 count[PM[i \* sizeof(float) + byte\_num]]--

10 output[count[PM[i\*sizeof(float)+byte\_num]] = arr[i]

11 SWAP\_POINTERS(A, output)

12 PM = (UNSIGNED CHAR\*)A

Минусом подобной реализации является то, что в итоговой последовательности для чисел типа float сначала будут стоять положительные числа в порядке возрастания, затем отрицательные в порядке убывания. Чтобы исправить это, в конце пропишем такой обходной прием:

* Во временный массив сначала записываем отрицательные значения с конца, затем положительные сначала.
* Меняем указатели местами

1 i = n – 1; p = 0

2 while A[i]<=0

3 tmp[p] = A[i]

4 p++

5 i--

6 i = 0

7 while A[i] >= 0

8 tmp[p] = A[i]

9 p++

10 i--

11 SWAP\_POINTERS(A, tmp)

# Подтверждение корректности

Для подтверждения корректности в программе в программе происходит сравнение работы каждой сортировки с встроенной в язык си (будем считать её корректной). Функция проходится по всем значениям массива, отсортированного qsort, и массива, отсортированного одной из четырех выше обозначенных сортировок и сравнивает их значения. Если значения одинаковые, то массив считается отсортированным корректно, иначе – некорректно. Программа выводит s (sorted) и n (not sorted) в соответствующих случаях

В массив вводятся случайные значения от -10^6 до 10^6 (включительно) типа float. Возьмем в качестве размера массива 10^6, так как значения больше будут очень долго работать для квадратичной сортировки.

Также для наглядности на будет количество тактов процессора за время выполнения сортировки, чтобы можно было убедиться в правдоподобности результатов

Результаты проверки:

* Сортировка выбором

Количество тактов процессора – 19322598076

Корректность работы – s

* Сортировка расческой

Количество тактов процессора - 62673677

Корректность работы – s

* Сортировка слиянием

Количество тактов процессора - 72094305

Корректность работы – s

* Поразрядная сортировка

Количество тактов процессора - 8423791

Корректность работы - s

# Результаты экспериментов

Чтобы наглядно показать асимптотику данных алгоритмов (см. стр. 4 «Метод решения»), проведем эксперимент.

Суть его заключается в том, что мы делим количество тактов процессора, в ходе работы сортировки, на значение её асимптотики, и должны таким образом, при достаточно большой длине массива получить константу. Такой подход имеет минусы в виде фоновой работы компьютора, но я попытался, насколько это возможно, минимизировать этот фактор.

* Сортировка вставками

Для значений в диапазоне от 10000 до 100000 значение константы ≈ 1

Рис. 1. График отношения сортировки вставками

* Сортировка расческой

Для значений от 5000000 до 14000000 значение константы ≈ 16

Рис. 2. График отношения сортировки расческой

* Сортировка слиянием

В данном случае значение отношения резко падает, но затем нормализуется на уровне ≈ 14.75

Рис. 3. График отношения сортировки слиянием

* Поразрядная сортировка

Для значений от 1e7 до 1e8 константа ≈ 39

Рис. 4. График отношения поразрядной сортировки

# Заключение

* В отчете написаны принципы работы сортировок, а также конкретная реализация 4 различных сортировок.
* В руководстве для пользователя описано взаимодействие пользователя и программы, а также её возможностей по анализу работы сортировок.
* Была проанализирована работа каждой сортировки, проверка её на корректность и на верность асимптотики, указанной ранее, и поиск её константы для некоторых значений длины массива (пусть и с низкой точностью), с приложением в виде графиков.

# Источники

1. Кормен, Т., Лейзерсон, Ч., Ривест, Р., Штайн, К. // Алгоритмы: построение и анализ / Под ред. И. В. Красикова. — 3-е изд. — М.: Вильямс, 2013.
2. Роберт Седжвик "Фундаментальные алгоритмы на C++"
3. https://ru.wikipedia.org/wiki/Поразрядная\_сортировка
4. https://education.yandex.ru/journal/osnovnye-vidy-sortirovok-i-primery-ikh-realizatsii

# Приложение

void choose(float\* a, int n)

{

int minI;

float t;

for (int i = 0; i < n; i++) {

minI = i;

for (int j = i; j < n; j++) {

if (a[minI] > a[j])

minI = j;

}

t = a[minI];

a[minI] = a[i];

a[i] = t;

}

}

void comb\_sort(float\* a, int n) {

double factor = 1.2473309;

int step = n - 1;

while (step >= 1) {

for (int j = 0; j < n - step; j++) {

if (a[j] > a[j + step]) {

float t = a[j];

a[j] = a[j + step];

a[j + step] = t;

}

}

step = step / factor;

}

int flag = 1;

while (flag) {

flag = 0;

for (int i = 1; i < n; i++) {

if (a[i] < a[i - 1]) {

float t = a[i];

a[i] = a[i - 1];

a[i - 1] = t;

flag = 1;

}

}

}

}

void radixSort(float\* arr, float\* tmp, int n) {

unsigned char\* pm = (unsigned char\*)arr;

int count[256];

for (int byte\_num = 0; byte\_num < sizeof(float); byte\_num++) {

for (int i = 0; i < 256; i++)

count[i] = 0;

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

count[pm[i \* sizeof(float) + byte\_num]]++;

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

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

for (int i = n - 1; i >= 0; i--)

tmp[--count[pm[i \* sizeof(float) + byte\_num]]] = arr[i];

float\* t = arr;

arr = tmp;

tmp = t;

pm = (unsigned char\*)arr;

}

int p = 0;

for (int i = n - 1; arr[i] <= 0; i--)

tmp[p++] = arr[i];

for (int i = 0; arr[i] >= 0; i++)

tmp[p++] = arr[i];

}

void merge(float\* arr, int l, int r, float\* tmp) {

int mid = (l + r) / 2;

int lp = l, rp = mid + 1;

int k = l;

while (lp <= mid && rp <= r) {

if (arr[lp] > arr[rp])

tmp[k++] = arr[rp++];

else

tmp[k++] = arr[lp++];

}

for (; lp <= mid; lp++)

tmp[k++] = arr[lp];

for (; rp <= r; rp++)

tmp[k++] = arr[rp];

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

arr[i] = tmp[i];

}

void mergeSort(float\* arr, float\* tmpdata, int l, int r) {

if (l < r) {

int mid = l + (r - l) / 2;

mergeSort(arr, tmpdata, l, mid);

mergeSort(arr, tmpdata, mid + 1, r);

merge(arr, l, r, tmpdata);

}

}

1. Кормен, с. 53 [↑](#footnote-ref-1)