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 3823Б1ПМ1-1

Крестьянинов В.С.

**Проверил**:

преподаватель каф. ВВСП,

Волокитин В.Д.

Нижний Новгород

2019

**Содержание**

[Постановка задачи 3](#_Toc26962562)

[Метод решения 4](#_Toc26962563)

[Руководство пользователя](#_Toc26962564) 6

[Описание программной реализации](#_Toc26962565) 7

[Подтверждение корректности](#_Toc26962566) 10

[Результаты экспериментов](#_Toc26962567) 11

[Заключение](#_Toc26962568) 13

[Приложение 1](#_Toc26962569)4

# Постановка задачи

Даётся последовательность чисел типа float. Программа должна отсортировать данную последовательность. Сортировку нужно реализовать 4 видами:

1. Сортировка выбором
2. Сортировка “Шелла”
3. Сортировка слиянием
4. Поразрядная сортировка

Также необходимо проверить на корректность данные алгоритмы и провести тестирование на временную сложность, чтобы провести сравнение данных видов сортировок на основе полученных результатов.

# Метод решения

* Сортировка выбором алгоритм сортировки, который на каждом шаге выбирает минимальный (или максимальный) элемент из неотсортированной части массива и помещает его в начало (или конец) отсортированной части.

Описание алгоритма сортировки выбором:

1. Начинаем с неотсортированного массива.

2. Находим наименьший элемент в неотсортированной части массива и обмениваем его с первым элементом неотсортированной части.

3. Повторяем шаг 2 для оставшейся части массива, исключая уже отсортированные элементы.

4. Повторяем шаги 2 и 3 до тех пор, пока весь массив не будет отсортирован.

* Сортировка Шелла - это усовершенствованный алгоритм сортировки вставками. Он основан на идее сравнения и обмена элементов, находящихся на определенном расстоянии друг от друга. Сортировка Шелла использует последовательность расстояний (шагов), которые сокращаются по мере продвижения сортировки, пока не достигнута сортировка вставками.

Описание алгоритма сортировки Шелла:

1. Задаем последовательность шагов, которая будет использоваться для сортировки. Обычно используется последовательность Кнута: 1, 4, 13, 40, 121 и так далее.

2. Начинаем сортировку, применяя сортировку вставками с выбранным шагом.

3. Уменьшаем шаг и повторяем сортировку вставками с новым шагом до тех пор, пока шаг не станет равным 1.

4. Завершаем сортировку вставками с шагом 1, чтобы окончательно отсортировать массив.

* Сортировка слиянием (Merge sort) - это алгоритм сортировки, который основан на принципе "разделяй и сливай". Он состоит из нескольких шагов:

1. Разделение: Исходный массив делится на две примерно равные части.

2. Рекурсивное слияние: Каждая половина массива рекурсивно сортируется с помощью сортировки слиянием.

3. Слияние: Отсортированные половины массива объединяются в один отсортированный массив.

* Поразрядная сортировка (Radix sort) - это алгоритм сортировки, который основан на разрядной системе счисления. Он сортирует элементы по разрядам, начиная с младшего разряда и двигаясь к старшим разрядам. Этот алгоритм может быть применен к числам, представленным в двоичной, восьмеричной, десятичной или любой другой системе счисления.

Вот подробное описание алгоритма поразрядной сортировки:

1. Определение количества разрядов:

- Находим максимальное число в исходном массиве и определяем количество разрядов этого числа.

- Это количество разрядов будет определять количество проходов, необходимых для полной сортировки массива.

2. Сортировка по разрядам:

- Начиная с младшего разряда и двигаясь к старшим разрядам, выполняем следующие шаги для каждого разряда:

- Создаем 10 "корзин" (0-9), в которые будут помещаться элементы в соответствии с текущим разрядом.

- Распределяем элементы исходного массива по корзинам на основе значения текущего разряда.

- Собираем элементы из корзин в новый массив в порядке их расположения в корзинах.

- Обновляем исходный массив, заменяя его содержимое на отсортированный массив.

3. Повторение сортировки по разрядам:

- Повторяем шаг 2 для каждого следующего разряда, начиная со второго разряда и двигаясь к старшим разрядам.

- После завершения последнего разряда исходный массив будет полностью отсортирован.

# Руководство пользователя

После запуска программы пользователю предоставляется выбор из 4 сортировок: выбором, шелла, слиянием, поразрядная.

Дальше пользователю следует указать в каком порядке будет происходить сортировка (0 - по возрастанию, 1 - по убыванию).

После этого нужно написать количество элементов в массиве

Дальше нужно выбрать между тем, чтобы заполнить массив вручную или случайными числами (в диапазоне от -10^6 до 10^6).

В результате на экране пользователя появится количество тактов процесса, в ходе выполнения сортировки и проверка корректности с помощью quicksort. Еще пользователь может вывести отсортированный массив.

После этого пользователь может выйти из программы или продолжить работу (0 - выйти, 1 -продолжить).

# Описание программной реализации

* Сортировка выбором

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

void choise(float\* arr, float n) {

int i = 0, j = 0;

int minI;

float tmp;

for (i = 0; i < n - 1; i++) {

minI = i;

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

if (arr[minI] > arr[j]) {

minI = j;

}

}

tmp = arr[i];

arr[i] = arr[minI];

arr[minI] = tmp;

* Сортировка Шелла

Задается последовательность. Начинаем сортировку, применяя сортировку вставками с выбранным шагом. Уменьшаем шаг и повторяем сортировку вставками с новым шагом до тех пор, пока шаг не станет равным 1. Завершаем сортировку вставками с шагом 1, чтобы окончательно отсортировать массив.

void shell(float\* a, float n) {

int g, i, j;

float temp;

for (g = n / 2; g != 0; g /= 2) {

for (i = g; i < n; i++) {

temp = a[i];

for (j = i; (j >= g) && (a[j - g] > temp); j -= g) {

a[j] = a[j - g];

}

a[j] = temp;

* Сортировка слиянием

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

Исходный массив делится на две равные (или 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

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);

* Поразрядная сортировка

Данная сортировка называется LSD (Least significant digit) и работает на основе сортировки подсчетом. Основная идея данной сортировки 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.

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]]++;

int temp = count[0];

count[0] = 0;

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

int t = count[i];

count[i] = temp;

temp = t;

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

}

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

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

}

float\* t = arr;

arr = tmp;

tmp = t;

pm = (unsigned char\*)arr;

print\_array(arr, n, 0);

}

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];

# Подтверждение корректности

Для подтверждения корректности в программе в программе происходит сравнение работы каждой сортировки с встроенной в язык си сортировкой. 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 p (perfect) и b (bad) 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.

# Результаты экспериментов

* Выбором

|  |  |
| --- | --- |
| количество элементов | константа |
| 1000 | 1,425945 |
| 5000 | 1,559246 |
| 10000 | 0,991791 |
| 20000 | 1,024962 |
| 50000 | 0,971001 |
| 100000 | 0,959798 |

Таблица 1

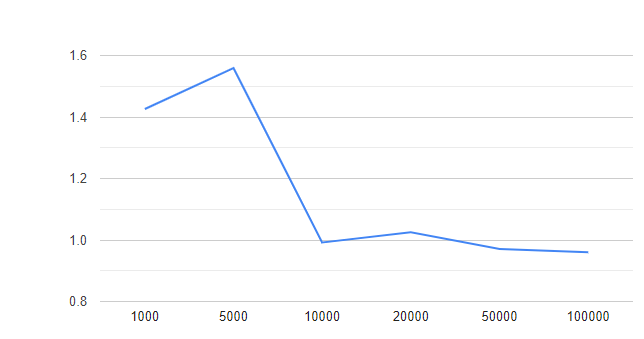


Рисунок 1

* Шелла

|  |  |
| --- | --- |
| количество элементов | константа |
| 1000 | 30,344505 |
| 5000 | 29,109982 |
| 10000 | 30,666029 |
| 20000 | 23,503027 |
| 50000 | 27,59771 |
| 100000 | 30,462143 |

Таблица 2

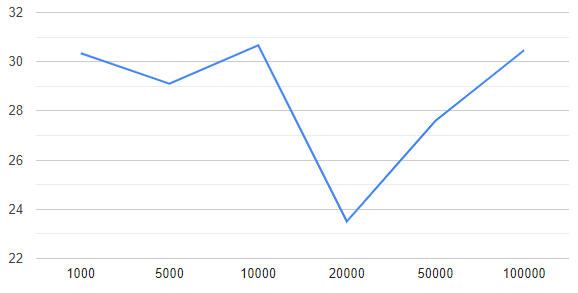


Рисунок 2

* Слияние

|  |  |
| --- | --- |
| количество элементов | константа |
| 1000 | 26,836729 |
| 5000 | 26,190079 |
| 10000 | 26,495045 |
| 20000 | 26,119904 |
| 50000 | 24,792299 |
| 100000 | 22,237293 |

Таблица 3

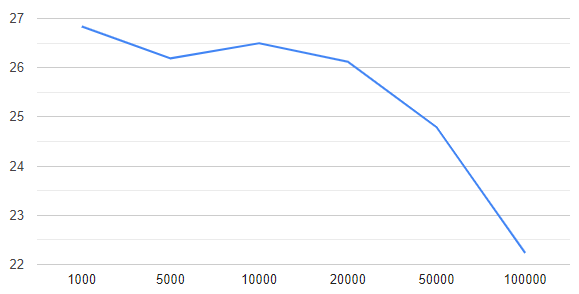


Рисунок 3

* Поразрядная

|  |  |
| --- | --- |
| количество элементов | константа |
| 1000 | 1906677,875 |
| 5000 | 1750043 |
| 10000 | 1734507,875 |
| 20000 | 1620214,625 |
| 50000 | 1567221,875 |
| 100000 | 1540456,125 |

Таблица 4

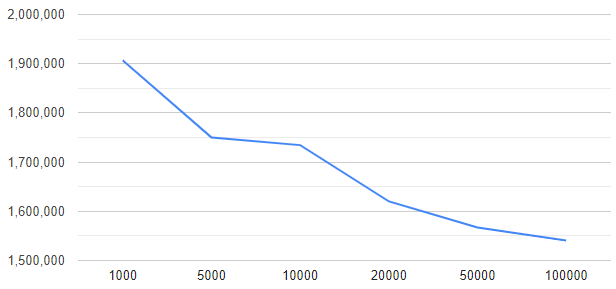


Рисунок 4

# Заключение

Было приведено 4 варианта сортировок, коды были подобно разобраны, также был проведен анализ по времени.

Можно сделать вывод, что для небольших массивов лучше всего использовать такие сортировки как Выбором, Шелла. Для больших массивов лучше использовать поразрядную сортировку и сортировку слиянием

# Приложение

void shell(float\* a, float n) {

int g, i, j;

float temp;

for (g = n / 2; g != 0; g /= 2) {

for (i = g; i < n; i++) {

temp = a[i];

for (j = i; (j >= g) && (a[j - g] > temp); j -= g) {

a[j] = a[j - g];

}

a[j] = temp;

}

}

}

void choise(float\* arr, float n) {

int i = 0, j = 0;

int minI;

float tmp;

for (i = 0; i < n - 1; i++) {

minI = i;

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

if (arr[minI] > arr[j]) {

minI = j;

}

}

tmp = arr[i];

arr[i] = arr[minI];

arr[minI] = tmp;

}

}

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]]++;

int temp = count[0];

count[0] = 0;

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

int t = count[i];

count[i] = temp;

temp = t;

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

}

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

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

}

float\* t = arr;

arr = tmp;

tmp = t;

pm = (unsigned char\*)arr;

print\_array(arr, n, 0);

}

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);

}

}