Министерство образования и науки РФ

Федеральное государственное автономное образовательное учреждение высшего образования «Национальный исследовательский Нижегородский государственный университет им. Н.И. Лобачевского» (ННГУ)

Институт информационных технологий, математики и механики

**Отчёт по учебной практике**

«**Методы сортировки**»

Выполнил: студент 381806-3 группы

Зайцев А.Р.

Проверила: к.т.н., доцент каф. МОСТ ИИТММ Кустикова В.Д.

Нижний Новгород

2018

**Содержание**

**Введение**……………………………………………….....3

1. **Постановка задачи**………………………………...4
2. **Руководство пользователя**……………………….5
3. **Руководство программиста**………………………7
4. **Структура программы**……………………………..7
5. **Описание алгоритма**………………………………..8
6. **Сортировка пузырьком**……………………………...8
7. **Сортировка вставками**………………………………9
8. **Сортировка выбором**……………………………….10
9. **Сортировка подсчётом**……………………………...11
10. **Быстрая сортировка**………………………………...12
11. **Сортировка слиянием**………………………………14
12. **Описание функций**………………………………...16

**Заключение**…………………………………………….18

**Литература**……………………………………………..20

**Введение**

Нетрудно догадаться, что людям удобно работать с упорядоченными по каким-либо признакам списками данных. Именно для упорядочивания множеств и списков данных нужны сортировки.

Работа с файлами –неотъемлемый способ взаимодействия человека с операционной системой. Для того чтобы со списком файлов было удобнее работать, его необходимо отсортировать, например, по размеру.

Именно для этого и понадобилась программа «файловый менеджер», с помощью которой можно вывести на экран отсортированный одним из 6 методов сортировки набор файлов с указанием размеров.

**Постановка задачи**

**Задача:** разработать прототип файлового менеджера с функцией показа файлов в заданном каталоге, упорядоченным по возрастанию/убыванию размера.

**Входные данные:** путь до директории, в которой необходимо отсортировать содержимое, и метод сортировки.

**Выходные данные:** отсортированный список имён файлов с указанием размера и время сортировки.

**Руководство пользователя**

На данные момент у меня получилось реализовать сортировку массива несколькими способами сортировки.

1. При запуске программы откроется экран, где будет предложено ввести количество элементов массива.

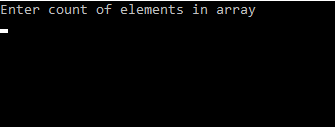


Рис 1. Программа после запуска

1. После того, как мы задали количество (в этом примере это 6), программа попросит ввести вручную каждый элемент массива.

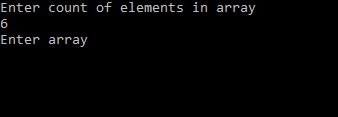


Рис 2. Ввод массива вручную

1. После ввода всех элементов, программа выведет на экран получившийся массив, а также предложит выбрать одну из 6 сортировок.

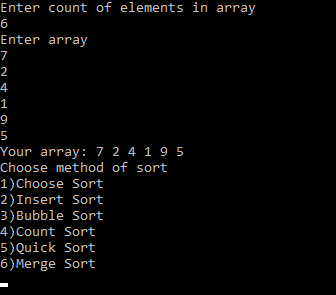


Рис 3. Выбор сортировки

1. После выбора сортировки (в этом примере это 2 – сортировка вставками) программа выведет на экран отсортированный массив

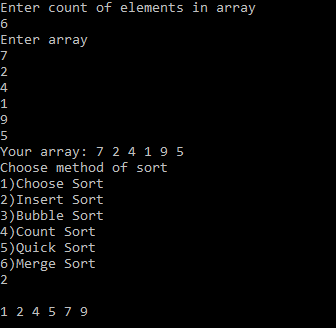


Рис 4. Отсортированный массив

Для выполнения поставленной задачи придётся изменить методы сортировки (списки параметров и само тело).

**Руководство программиста**

**Структура программы**

Весь код программы содержится в файле «File manager.c».

Код программы состоит из отдельных функций, для упрощения написания программы и лучшей наглядности.

**Описание алгоритма**

**Сортировка пузырьком**

void Bubble\_Sort(int a[], int n)

{

int i, j, tmp;

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

{

for (j = 1; j < n - i; j++)

{

if (a[j - 1] > a[j])

{

tmp = a[j];

a[j] = a[j - 1];

a[j - 1] = tmp;

}

}

}

}

**Описание:** Идея метода состоит в том, чтобы попарно сравнивать соседние элементы. Каждый проход начинается с начала последовательности. Сравнивается первый элемент со вторым: если порядок между ними нарушен, то они меняются местами. Затем сравниваются второй с третьим, третий с четвертым и так далее до конца массива; элементы с неправильным порядком в паре меняются местами. В итоге, после первого прохода, максимальный (или минимальный, в зависимости от вида сортировки: по возрастанию/по убыванию) элемент будет находится на последнем месте в массиве, он как бы “всплывет” наверх. Именно поэтому этот метод называется сортировка пузырьком. На следующем проходе рассматривается последовательность от 1 до N-1, затем от 1 до N-2, и так до конца. После каждого прохода можно делать проверку: выполнялись ли перестановки элементов. Если не выполнялись, то сортировка завершена.

Для понимания и реализации этот алгоритм — простейший, но эффективен он лишь для небольших массивов.

Эта сортировка является устойчивой.

**Сортировка вставками**

void Insert\_Sort(int a[], int n)

{

int i, j, tmp;

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

{

tmp = a[i];

j = i - 1;

while ((j >= 0) && (a[j] > tmp))

{

a[j + 1] = a[j];

a[j] = tmp;

j--;

}

}

}

**Описание:** идея метода состоит в том, что при добавлении нового элемента в уже отсортированный массив его необходимо вставлять на нужное место, вместо того, чтобы ставить его в произвольное, и потом заново сортировать всю последовательность.

Рассмотрим алгоритм метода **сортировки вставками** на примере сортировки по возрастанию. Первый элемент в массиве образует уже отсортированную последовательность. Сравниваем второй элемент с первым. Если порядок между ними нарушен, то первый элемент передвигается на одну позицию вправо. Теперь отсортированный массив состоит из двух элементов.

Далее, в течение каждой итерации, берем следующий элемент (третий, четвертый и т.д.) и сравниваем его поочередно с другими элементами в уже отсортированном списке, **начиная с конца** этого списка. Если порядок между сравниваемыми элементами нарушен, то меняем их местами, если нет, то “вставка” нового элемента закончена, переходим к следующему.

Сортировка вставками эффективна на небольших наборах данных. Рекомендуется использовать этот метод на наборах размером до десятков элементов. Сортировка вставками эффективна на последовательностях с данными, которые уже частично отсортированы.

Эта сортировка является устойчивой.

**Сортировка выбором**

void Choose\_Sort(int a[], int n)

{

int i, j, min, min\_idx;

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

{

min = a[i];

min\_idx = i;

for (j = i + 1; j < n; j++)

{

if (a[j] < min)

{

min = a[j];

min\_idx = j;

}

}

a[min\_idx] = a[i];

a[i] = min;

}

}

**Описание:** идея метода состоит в том, что на первом проходе цикла выбирается минимальный элемент из текущей последовательности и меняется местами с первым элементом последовательности. На следующей итерации цикла поиск минимального элемента осуществляется со второй позиции, после меняется местами найденный минимальный элемент со вторым в списке. Такую процедуру выполняем до конца массива, пока он весь не будет отсортирован.

Эта сортировка может быть как устойчивой, так и неустойчивой.

**Сортировка подсчётом**

void Count\_Sort(int a[], int n)

{

int i, j, idx, k = 0;

k = a[0];

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

{

if (a[i] > k)

{

k = a[i];

}

k++;

}

int \*count = (int \*)malloc(k \* sizeof(int));

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

{

count[i] = 0;

}

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

{

\*(count + a[i]) = \*(count + a[i]) + 1;

count - a[i];

}

idx = 0;

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

{

for (i = 0; i < count[j]; i++)

{

a[idx] = j;

idx++;

}

count - j;

}

}

**Описание:** рассмотрим алгоритм, который можно использовать только для последовательностей, не содержащий одинаковых элементов. Упорядоченная последовательность элементов создается на свободном участке памяти. Идея метода заключается в следующем: в отсортированной последовательности, элемент, занимающий позицию с номером **N+1**, превышает ровно **N** элементов, поэтому в процессе сортировки методом подсчета на каждом **i**-ом проходе мы попарно сравниваем **i**-й элемент со всеми элементами массива. Если установлено, что **mass[i] > mass[j]**, то увеличиваем счетчик **N** на единицу (в начале N = 0). По окончании текущего прохода счетчик N указывает количество элементов, меньших, чем **mass[i]**, поэтому элемент **mass[i]** занимает в отсортированной последовательности позицию **N + 1** (sortedMass[n + 1] = mass[i]).

Эта сортировка является устойчивой.

**Быстрая сортировка (Хоара)**

void Quick\_Sort(int a[], int n1, int n2)

{

int m, i, j;

m = (n1 + n2) / 2;

i = n1, j = n2;

Quick\_Split(a, &i, &j, a[m]);

if (i > n1)

{

Quick\_Sort(a, n1, i - 1);

}

if (j < n2)

{

Quick\_Sort(a, j + 1, n2);

}

}

Где Quick\_Split - вспомогательная функция:

void Quick\_Split(int a[], int \*i, int \*j, int p)

{

int tmp;

do{

while (a[(\*i)] < p)

{

(\*i)++;

}

while (a[(\*j)] > p)

{

(\*j)--;

}

if ((\*i) <= (\*j))

{

tmp = a[(\*i)];

a[(\*i)] = a[(\*j)];

a[(\*j)] = tmp;

}

}while ((\*i) < (\*j));

}

**Описание:** это один из самых быстрых известных универсальных алгоритмов сортировки массивов. QuickSort является существенно улучшенным вариантом алгоритма сортировки с помощью прямого обмена (например: «Пузырьковой сортировки»). Любопытный факт: улучшение самого неэффективного прямого метода сортировки дало в результате один из наиболее эффективных улучшенных методов.

Общая идея этого метода состоит в следующем:

1) Выбрать из массива элемент, называемый опорным. Это может быть любой из элементов массива. От выбора опорного элемента не зависит корректность алгоритма, но в отдельных случаях может сильно зависеть его эффективность.

2) Сравнить все остальные элементы с опорным и переставить их в массиве так, чтобы разбить массив на три непрерывных отрезка, следующие друг за другом: «меньшие опорного», «равные» и «большие».

3) Для отрезков «меньших» и «больших» значений выполнить рекурсивно ту же последовательность операций, если длина отрезка больше единицы.

На практике массив обычно делят не на три, а на две части: например, «меньшие опорного» и «равные и большие»; такой подход в общем случае эффективнее, так как упрощает алгоритм разделения.

Эта сортировка является неустойчивой.

**Сортировка слиянием**

void Merge\_Sort(int a[], int left, int right)

{

int mid, i;

if (left + 1 >= right)

{

return;

}

mid = (right + left) / 2;

Merge\_Sort(a, left, mid);

Merge\_Sort(a, mid, right);

Merge(a, left, mid, right);

}

Где Merge – вспомогательная функция:

void Merge(int \*a, int left, int mid, int right)

{

int i, j, k;

i = j = 0;

int \*c = (int \*)malloc((right - left) \* sizeof(int));

while ((left + i < mid) && (mid + j < right))

{

if (a[left + i] < a[mid + j])

{

c[i + j] = a[left + i];

i++;

}

else

{

c[i + j] = a[mid + j];

j++;

}

}

while (left + i < mid)

{

c[i + j] = a[left + i];

i++;

}

while (mid + j < right)

{

c[i + j] = a[mid + j];

j++;

}

for (k = 0; k < i + j; k++)

{

a[left + k] = c[k];

}

free(c);

}

**Описание:** Сортировка слиянием — алгоритм сортировки, который упорядочивает списки (или другие структуры данных, доступ к элементам которых можно получать только последовательно, например — потоки) в определённом порядке. Эта сортировка — хороший пример использования принципа «разделяй и властвуй». Сначала задача разбивается на несколько подзадач меньшего размера. Затем эти задачи решаются с помощью рекурсивного вызова или непосредственно, если их размер достаточно мал. Наконец, их решения комбинируются, и получается решение исходной задачи.

1) массив рекурсивно разбивается пополам, и каждая из половин делиться до тех пор, пока размер очередного подмассива не станет равным единице;

2) далее выполняется операция алгоритма, называемая слиянием. Два единичных массива сливаются в общий результирующий массив, при этом из каждого выбирается меньший элемент (сортировка по возрастанию) и записывается в свободную левую ячейку результирующего массива. После чего из двух результирующих массивов собирается третий общий отсортированный массив, и так далее. В случае если один из массивов закончиться, элементы другого дописываются в собираемый массив;

3) в конце операции слияния, элементы перезаписываются из результирующего массива в исходный.

Эта сортировка является устойчивой.

**Описание функций**

void Input\_a(int a[], int n)

**Назначение:** ввод элементов массива.

**Входные параметры:** a[] – массив, n - количество элементов в массиве.

**Выходные данные:** ручной ввод n элементов массива.

void Output\_a(int a[], int n)

**Назначение:** вывод элементов массива.

**Входные параметры:** a[] – массив, n – количество элементов в массиве.

**Выходные данные:** показ элементов массива в строчку на экране.

void Choose\_Sort(int a[], int n)

**Назначение:** выполнение сортировки выбором для указанного массива.

**Входные параметры:** a[] – массив, n – количество элементов в массиве.

**Выходные данные:** сортировка элементов данного массива.

void Insert\_Sort(int a[], int n)

**Назначение:** выполнение сортировки вставками для указанного массива.

**Входные параметры:** a[] – массив, n – количество элементов в массиве.

**Выходные данные:** сортировка элементов данного массива.

void Bubble\_Sort(int a[], int n)

**Назначение:** выполнение сортировки пузырьком для указанного массива.

**Входные параметры:** a[] – массив, n – количество элементов в массиве.

**Выходные данные:** сортировка элементов данного массива.

void Count\_Sort(int a[], int n)

**Назначение:** выполнение сортировки подсчётом для указанного массива.

**Входные параметры:** a[] – массив, n – количество элементов в массиве.

**Выходные данные:** сортировка элементов данного массива.

void Quick\_Sort(int a[], int n1, int n2)

**Назначение:** выполнение быстрой сортировки для указанного массива.

**Входные параметры:** a[] – массив; n1, n2 - индексы крайних элементов сортируемого фрагмента массива.

**Выходные данные:** сортировка элементов данного массива.

void Quick\_Split(int a[], int \*i, int \*j, int p)

**Назначение:** выполнение необходимых действий для быстрой сортировки.

**Входные параметры:** a[] – массив, \*i – счётчик i с указателем, \*j - счётчик j с указателем, p - переменная, которая будет заменена массивом с определённым индексом.

**Выходные данные:** указатель на упорядочивающий массив индексов.

void Merge(int \*a, int left, int mid, int right)

**Назначение:** выполнение необходимых действий для сортировки слиянием.

**Входные параметры:** left, mid, right – индексы левого, центрального и правого элементов сортируемого фрагмента массива.

void Merge\_Sort(int a[], int left, int right)

**Назначение:** выполнение сортировки слиянием для указанного массива.

**Входные параметры:** a[] – массив; left, right – индексы левого и правого элементов сортируемого фрагмента массива.

**Выходные данные:** сортировка элементов данного массива.

void main()

**Назначение:** основная функция (точка входа).

**Входные параметры:** отсутствуют.

scanf("%d", &(a[i]))

**Назначение:** ввод числа.

**Входные параметры:** &(a[i]) – адрес элемента массива, "%d" – формат целочисленной переменной.

**Выходные данные:** наше введённое число займёт место под индексом i в массиве.

printf("%d ", a[i])

**Назначение:** печать числа на экране.

**Входные параметры:** a[i] – массив с индексом i; "%d" – формат целочисленной переменной.

**Выходные данные:** a[i].

**Заключение**

В ходе выполнения практической работы «Методы сортировки» был разработан прототип файлового менеджера с функцией показа файлов в заданном каталоге, упорядоченных по размеру. Для сортировки было использовано 6 различных методов ( 1)пузырьковая сортировка 2)сортировка вставками 3)сортировка выбором 4)сортировка подсчётом 5)быстрая сортировка 6)сортировка слиянием). В таблице ниже приведена оценка сложности каждого метода сортировки.

|  |  |  |
| --- | --- | --- |
| Методы сортировки | Средняя сложность алгоритма | |
| Время выполнения | Память |
| Пузырьковая сортировка | O(n2) | O(1) |
| Сортировка вставками | O(n2) | O(1) |
| Сортировка выбором | O(n2) | O(1) |
| Сортировка подсчетом | O(n) | max(n) – min(n) + 1 |
| Быстрая сортировка | O(n·log2 n) | O(1) |
| Сортировка слиянием | O(n·log2 n) | O(n) |

Таблица 1. Оценка сложности методов сортировки

Также ниже находятся данные о зависимости времени выполнения сортировки от количества элементов в массиве практически для всех наших методов сортировки и некоторых других.

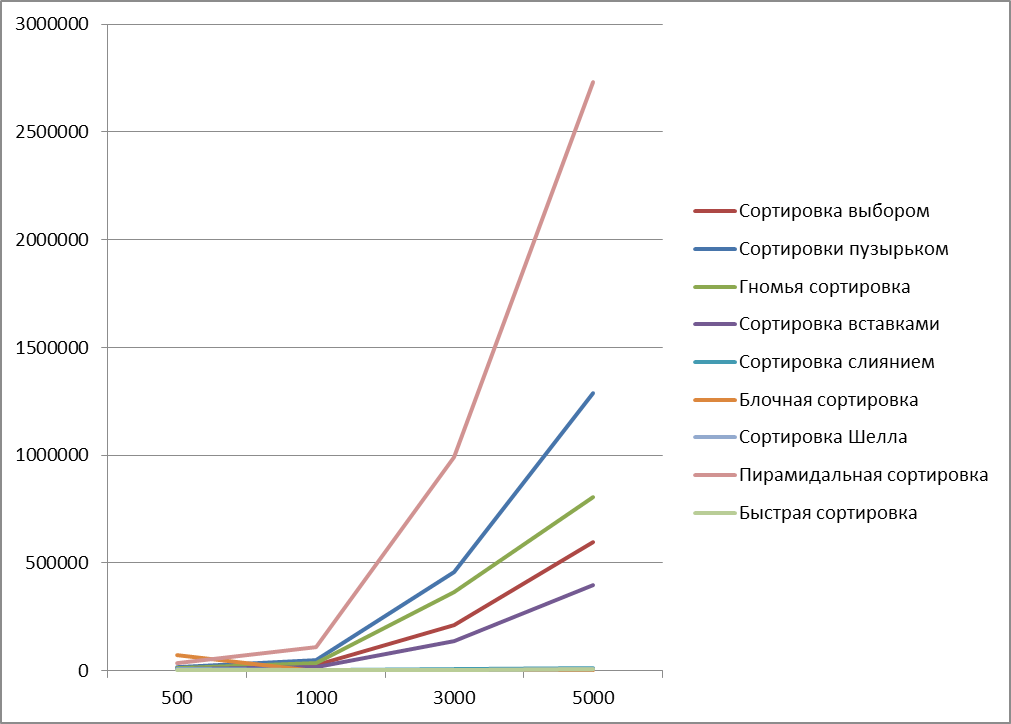


Рис 5. Зависимость времени выполнения сортировки от количества элементов

На графиках ниже приведена зависимость времени работы сортировки от количества элементов для разных оценок. По вертикали отображается время, а по горизонтали количество элементов.

Диаграмма 1. Зав-ть времени работы сортировки от кол-ва эл-тов для разных оценок

**Литература**

**Источники:**

<https://vscode.ru/prog-lessons/sortirovka-puzyirkom-na-si.html>

<https://vscode.ru/prog-lessons/sortirovka-vstavkami-na-si.html>

<https://vscode.ru/prog-lessons/sortirovka-metodom-vyibora-na-yazyike-si.html>

<https://vscode.ru/prog-lessons/sortirovka-metodom-podscheta-na-si.html>

<https://ru.wikipedia.org/wiki/Быстрая_сортировка>

<https://forkettle.ru/vidioteka/programmirovanie-i-set/algoritmy-i-struktury-dannykh/108-sortirovka-i-poisk-dlya-chajnikov/1015-sortirovka-sliyaniem-merge-sort>

<https://studfiles.net/preview/4652654/>

<https://studfiles.net/preview/4241651/page:3/>