Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»

(МГТУ им. Н.Э. Баумана)

Дисциплина “Типы и структуры данных”

Лабораторный практикум №2

по теме: «Записи с вариантами, обработка таблиц»

Выполнил студент: __*Клименко Алексей Константинович*_____

фамилия, имя, отчество

Группа: _____*ИУ7-35Б*_____

Проверил, к.п.н.: _____

подпись, дата

Оценка _____ Дата _____

2020 г.

цель работы

Приобрести навыки работы с типом данных «запись» (структура), содержащим вариантную часть (объединение, смесь), и с данными, хранящимися в таблицах, произвести сравнительный анализ реализации алгоритмов сортировки и поиска информации в таблицах, при использовании записей с большим числом полей, и тех же алгоритмов, при использовании таблицы ключей; оценить эффективность программы по времени и по используемому объему памяти при использовании различных структур и эффективность использования различных алгоритмов сортировок.

описание условия задачи

Необходимо создать таблицу, содержащую не менее 40 записей с вариантной частью. Произвести поиск информации по вариантному полю. Упорядочить таблицу, по возрастанию ключей (где ключ – любое невариантное поле по выбору программиста), используя:

а) исходную таблицу

б) массив ключей, двумя алгоритмами сортировки

Оценить эффективность выбранных алгоритмов (по времени и по используемому объему памяти) при различной реализации программы, то есть, в случаях а) и б). Обосновать выбор алгоритмов сортировки. Оценка эффективности должна быть относительной (в %).

указания к выполнению работы

Интерфейс программы должен быть понятен неподготовленному пользователю. При разработке интерфейса программы следует предусмотреть:

- указание формата и диапазона данных при вводе и (или) добавлении записей
- указание операций, производимых программой
- наличие пояснений при выводе результата
- возможность добавления записей в конец таблицы и удаления записи по значению указанного поля
- просмотр отсортированной таблицы ключей при несортированной исходной таблице
- вывод упорядоченной исходной таблицы
- вывод исходной таблицы в упорядоченном виде, используя упорядоченную таблицу ключей
- вывод результатов сравнения эффективности работы программы при обработке данных в исходной таблице и в таблице ключей
- вывод результатов использования различных алгоритмов сортировок

техническое задание

исходные данные

Исходными данными являются данные о записях в таблице, которые программа должна извлекать из текстового файла, имя которого пользователь вводит с клавиатуры.

Текстовый файл с данными имеет следующую структуру: на каждой строке файла располагается информация об одной записи. Наличие пустых строк в файле не допускается, ровно как и наличие информации о двух и более структурах на одной строке.

Строка, описывающая данные о квартире, имеет следующий формат:

<адрес>;<площадь>;<кол-во комнат>;<цена за кв. м>;<тип>;[…]

вместо […] содержится вариативная информация. Если был указан тип **PRIMARY**, то далее ожидается <наличие отделки>. Иначе (если был указан тип **SECONDARY**) – ожидаются следующие поля:

<были животные>;<время постройки>;<кол-во владельцев>;<кол-во жильцов>

При этом, вокруг разделительных символов «;» допускается использовать сколь угодно много пробелов — они опускаются при анализе строки (в том числе и при анализе адреса квартиры).

В строке адреса квартиры не следует использовать символ-разделитель «;», так как это может нарушить работу анализатора.

Везде, где тип данных может быть вещественным числом, допускается вместо этого использовать целые числа.

Для логического типа допустимыми значениями являются только слова «**yes**» и «**no**».

Для указания типа жилья следует использовать слова «**PRIMARY**» - для первичного жилья и «**SECONDARY**» - для вторичного.

результат

Одним из результатов работы программы является количественная информация (представленная в виде графика) с указанием времени, затраченного на обработку исходной таблицы и таблицы ключей двумя алгоритмами сортировки (при этом, не забыть оценить так же время выборки данных из основной таблицы с использованием таблицы ключей), а так же - объем занимаемой при этом оперативной памяти.

описание задачи, реализуемой программой

Программа при запуске запрашивает имя файла с данными, после чего считывает данные из файла и запускает цикл, в котором пользователь может выбрать одну из доступных опций (за одну итерацию цикла). По окончании работы программа записывает результирующие данные в выходной файл, название которого программой запрашивается в начале работы программы.

способы обращения к программе

Вызов программы происходит любым возможным способом запуска ***.exe** программ. (Например, через консоль). Для корректной работы программы при её вызове не требуются дополнительные аргументы командной строки.

Программное меню

При запуске, пользователю предлагается ввести имя файла с данными таблицы. После успешного считывания данных из файла, перед пользователем отображается меню, в котором он может выбирать опции по своему желанию.

возможные аварийные ситуации и ошибки пользователя

Наиболее возможными в работе программы будут ошибки, связанные с некорректным вводом пользователя или с содержанием файла. Хотя не исключены возможные сбои из-за невозможности выделения динамической памяти при считывании данных из файла.

В любом случае при дальнейшей невозможности продолжить работу, программа должна оповещать пользователя о случившейся ошибке (или ошибках) и завершать свою работу.

Ошибочный или некорректный ввод пользователя не должен приводить к завершению работы программы. Необходимо либо повторить ввод, либо вернуться в меню.

Структуры данных

Для хранения информации об одной квартире мной используется структура **`flat_t`**, которая содержит в себе в качестве полей: указатель на строку с адресом квартиры, общая площадь комнат, количество комнат, стоимость квадратного метра, тип (первичное жильё или вторичное), далее для первичного типа — с отделкой или без неё, а для вторичного — были ли животные; время постройки, количество предыдущих собственников, количество последних жильцов.

```
You, 6 days ago | 1 author (You)
typedef struct
{
    unsigned int id;
    char address[MAX_ADDRESS_SIZE];
    float area;
    unsigned char rooms_amount;
    float price_per_m2;
    bit_t type;
    You, 6 days ago | 1 author (You)
    union
    {
        bit_t has_trim; // 0 or 1
        bit_t was_pets;
    } type_data;
    time_t build_time;
    unsigned char prev_owners_amount;
    unsigned char prev_lodgers_amount;
} flat_t;
```

```
You, 6 days ago | 1 author (You)
typedef enum
{
    PRIMARY,
    SECONDARY
} flat_type_t;

typedef unsigned char bit_t;
```

```
You, a few seconds ago | 1 author (You)
typedef struct
{
    size_t size;
    flat_t *flats_array;
    flat_t **flat_ptrs;
} flat_table_t;
```

Объединение полей «отделка» и «домашние животные» обосновывается тем, что эти поля имеют равный размер и при любом варианте использования память, выделенная под данное вариантное поле, **будет** использоваться.

При работе с таблицами возможным преимуществом дополнительного массива указателей на структуры будет значительно увеличенная скорость обработки таблицы, однако увеличение скорости обработки оборачивается увеличением используемой памяти, но так как размер указателя на данные не зависит от самого типа данных, на которые он может указывать, это приращение памяти будет не столь серьёзным.

Также стоит заметить, что намеренное отстранение от хранения массива ключей записей и использование вместо них указателей может увеличить область применения программы и алгоритма сортировки в целом: он (алгоритм) больше не привязан к какому-либо отдельному полю и может сортировать таблицу по *любому из возможных невариантных полей*.

Вычислим объём памяти необходимый для хранения одной записи:

sizeof(flat_t) == 4Б + 32Б + … + 4Б == **72Б**

Рассчитаем зависимость занимаемой таблицей памяти при различных количествах записей **N**:

а) **sizeof(flat_table_t)** == 8Б (+ 72Б * N) + 4Б == **(12 + 72N) Б**

б) **sizeof(flat_table_t)** == 8Б (+ 72Б * N) + 8Б (+ 8Б * N) + 4Б ==

(20 + 80N) Б

При $N \to +\infty$размер структуры под пунктом (б) будет **превышать** размер структуры под пунктом (а) **на 14.28%**.

Множетво допустимых значений

Для данной СД для хранения информации о квартирах допустимые значения для полей структуры таковы:

Адрес (address) – строка до 30 символов (без символа «;»)

Площадь комнат (area) – (0.0f, 830.0f]

Количество комнат (rooms_amount) – [1, 10]

Цена за квадратный метр (price_per_m2) – (0.0f, 1.0e+6]

Тип жилья (type) – { PRIMARY, SECONDARY }

Наличие отделки (has_trim) – { 0, 1 }

Были ли домашние животные (was_pets) – { 0, 1 }

Время постройки (build_time) – [1880, 2020]

Кол-во предыдущих владельцев – [0, 100]

Кол-во предыдущих жильцов – [0, 20]

Набор функций

Над структурой, представляющей собой запись в таблице, допустимо совершать следующие действия:

- Инициализация структуры квартиры нулевыми значениями
- Освобождение памяти, выделенное для данной структуры
- Преобразование информации из структуры в строку и наоборот
- Печать данных структуры в файл или на экран
- Считывание данных структуры из файла или консоли

Над таблицей данных *вне зависимости от формы её реализации* разрешено выполнять следующие действия:

- Инициализация таблицы нулевым(-и) указателем(-ями)
- Освобождение памяти, выделенной для данной таблицы
- Сортировка таблицы по одному из предложенных невариантных полей: адрес, площадь комнат, количество комнат
- Добавление новой записи в конец таблицы
- Считывание данных таблицы из файла
- Запись текущих данных таблицы в файл для записи
- Вывод данных таблицы на экран консоли

Описание алгоритма обработки данных

Для реализации сортировки мною будут рассмотрены такие методы как сортировка обменом и сортировка слиянием как представители более медленной и более быстрой сортировок.

Для пункта (а), в котором таблица хранит в себе только массив структур, обе сортировки будут оперировать этим массивом данных непосредственно, т. е. будут производить *обмен элементами на месте*.

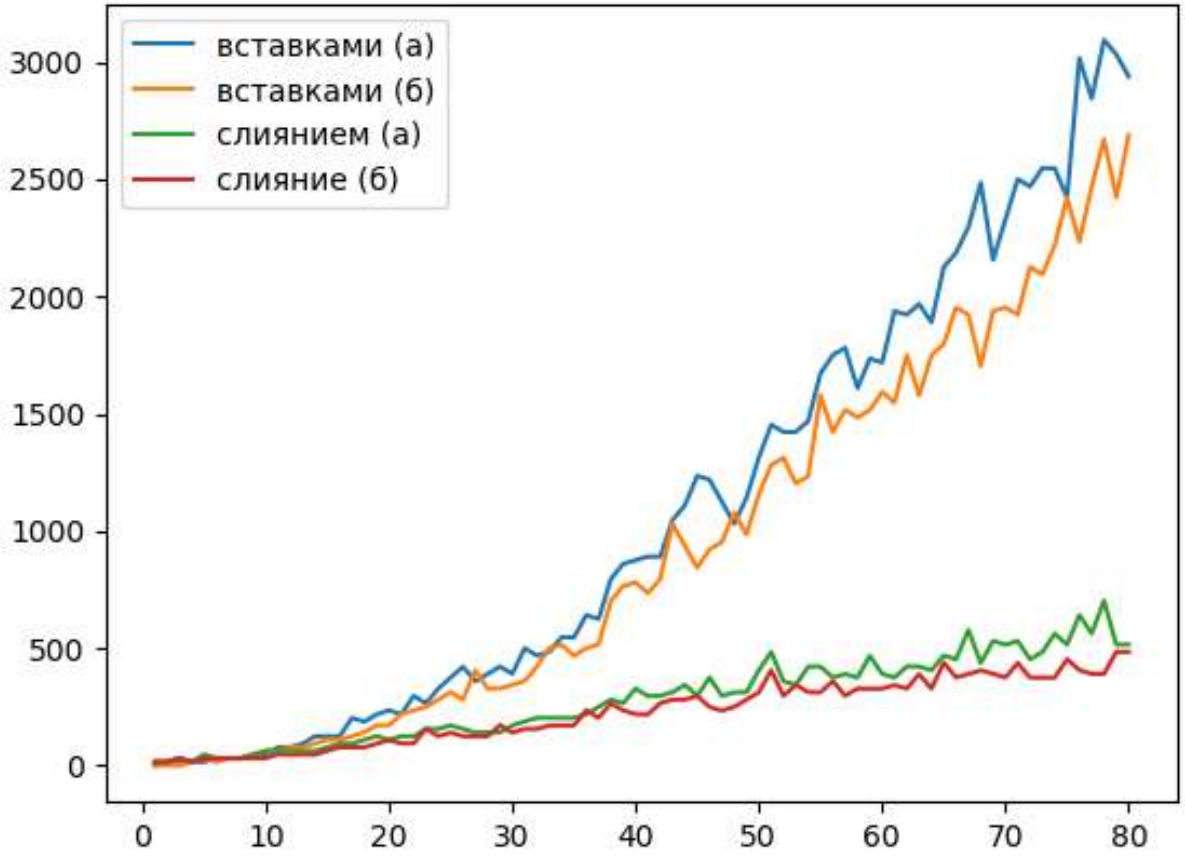
Для пункта (б), в котором помимо массива с данными будет и массив указателей на данные, сортироваться будет массив указателей, а это значит, что объем обрабатываемой при сортировке памяти будет меньше.

Приведём сложность упомянутых алгоритмов:

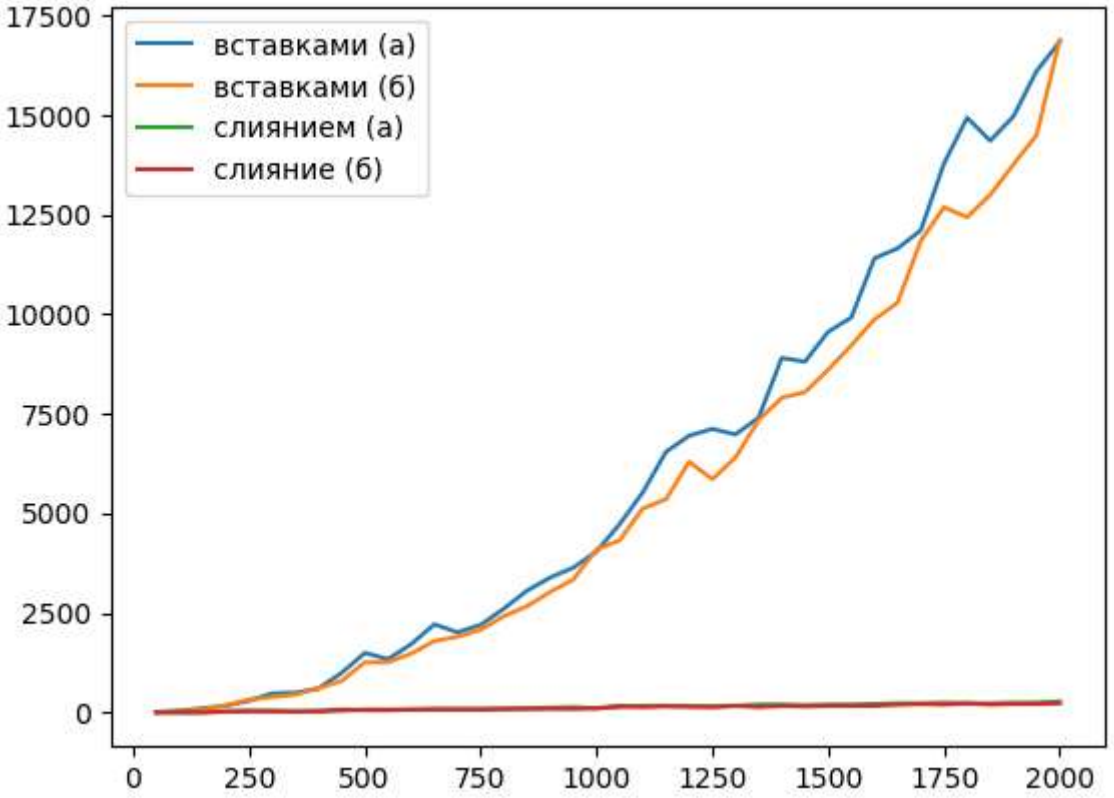
Сортировка вставками: **$O(N^2)$**

Сортировка слиянием: **O(N*log(N))**

Результаты тестирования в виде графика:



Для большего числа элементов:



Расчеты, проведённые в программе MathCAD для наиболее точного расчёта относительной эффективности двух алгоритмов при использовании дополнительной таблицы ключей:

Given $a := 1 \quad b := 0 \quad c := 0$

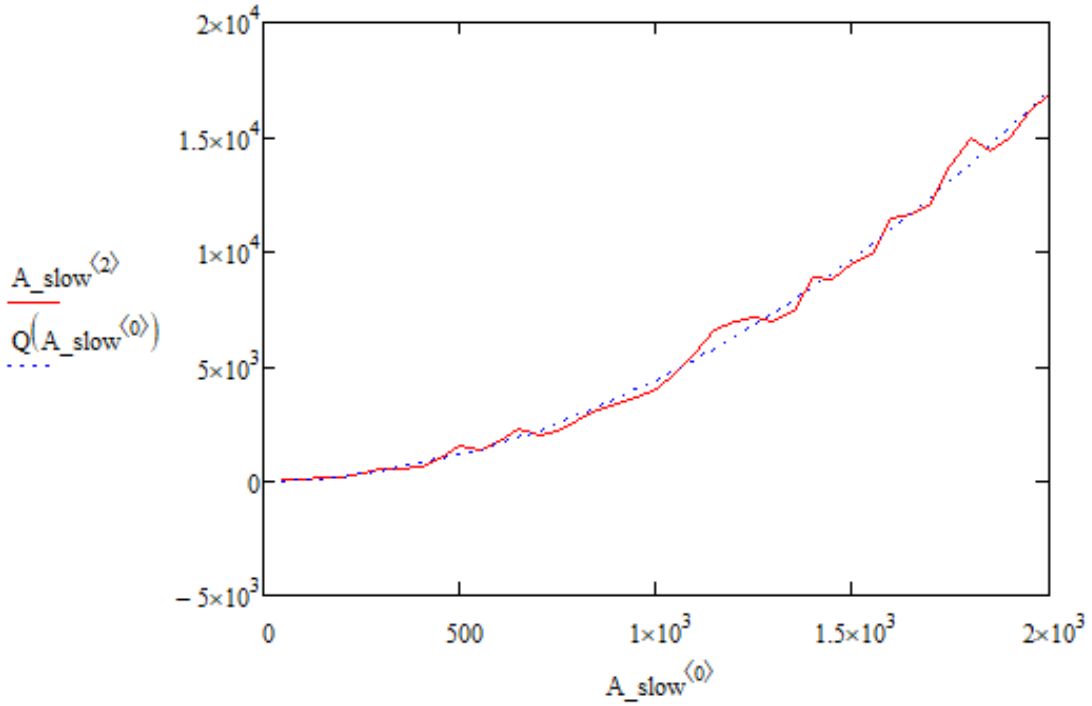
$A\_slow := READPRN("a\_slow.txt")$

$A\_slow^{\langle 2 \rangle} = a \cdot \left(A\_slow^{\langle 0 \rangle}\right)^2 + b \cdot A\_slow^{\langle 0 \rangle} + c$

$\begin{pmatrix} a \\ b \\ c \end{pmatrix} := minerr(a, b, c) = \begin{pmatrix} 4.096 \times 10^{-3} \\ 0.373 \\ -58.425 \end{pmatrix}$

$Q(N) := a \cdot N^2 + b \cdot N + c$

$rate\_slow\_a := a$



Given

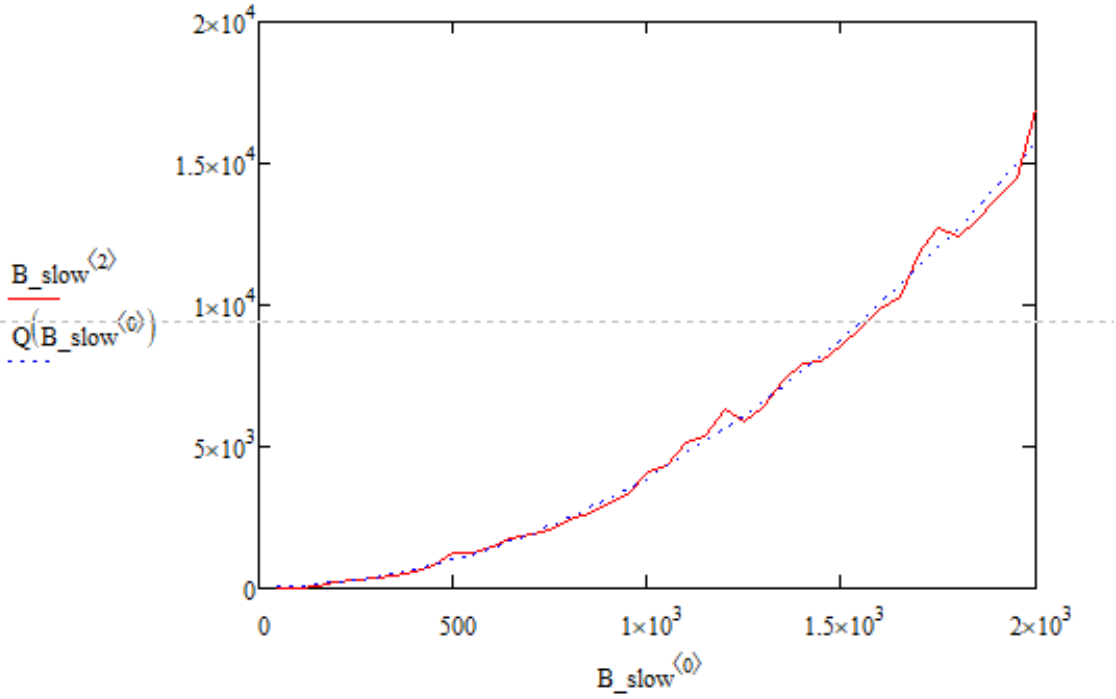
$B\_slow^{\langle 2 \rangle} = a \cdot \left(B\_slow^{\langle 0 \rangle}\right)^2 + b \cdot B\_slow^{\langle 0 \rangle} + c$

$\begin{pmatrix} a \\ b \\ c \end{pmatrix} := \text{minerr}(a, b, c) = \begin{pmatrix} 3.971 \times 10^{-3} \\ -0.097 \\ 55.435 \end{pmatrix}$

$B\_slow := \text{READPRN}(\text{"b\_slow.txt"})$

$Q(N) := a \cdot N^2 + b \cdot N + c$

$rate\_slow\_b := a$



Относительная эффективность по времени двух медленных сортировок:

$f := \frac{rate\_slow\_a - rate\_slow\_b}{rate\_slow\_b} = 0.031$ Составила 3.1%

Given $a := 1$

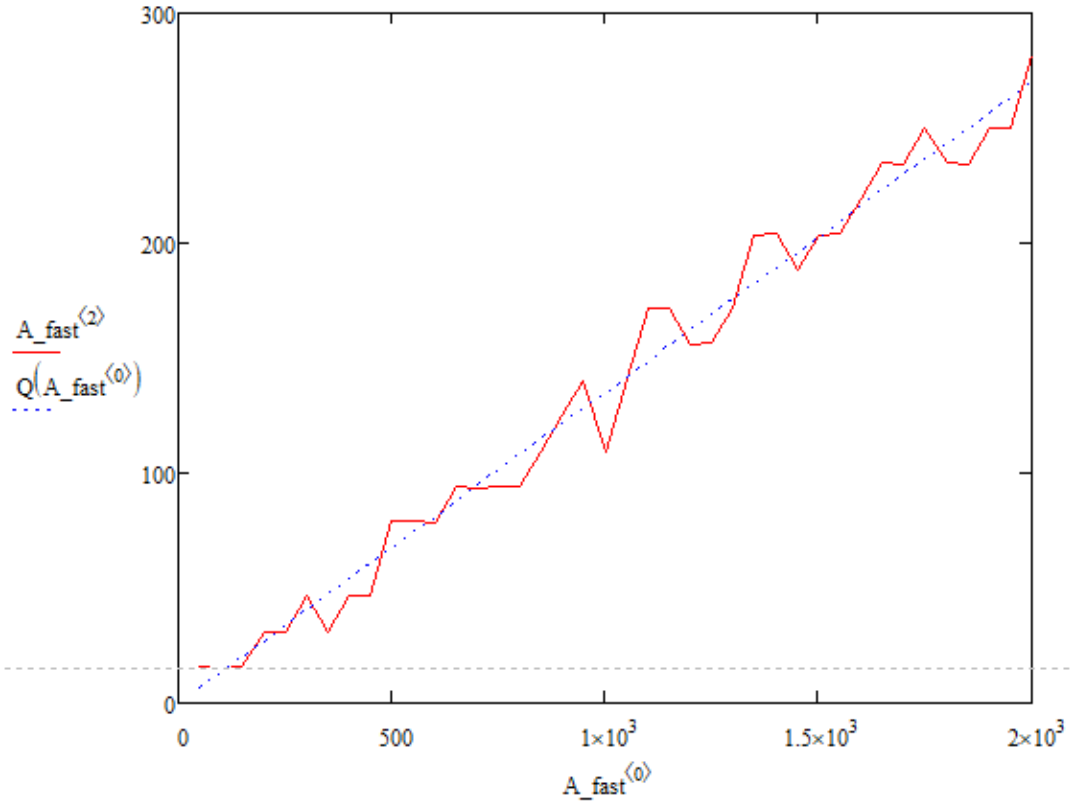
$A\_fast^{\langle 2 \rangle} = a \cdot \left(A\_fast^{\langle 0 \rangle}\right)$

$a := \text{minerr}(a) = 0.135$

$A\_fast := \text{READPRN}(\text{"a\_fast.txt"})$

$Q(N) := a \cdot (N)$

$rate\_fast\_a := a$



Given

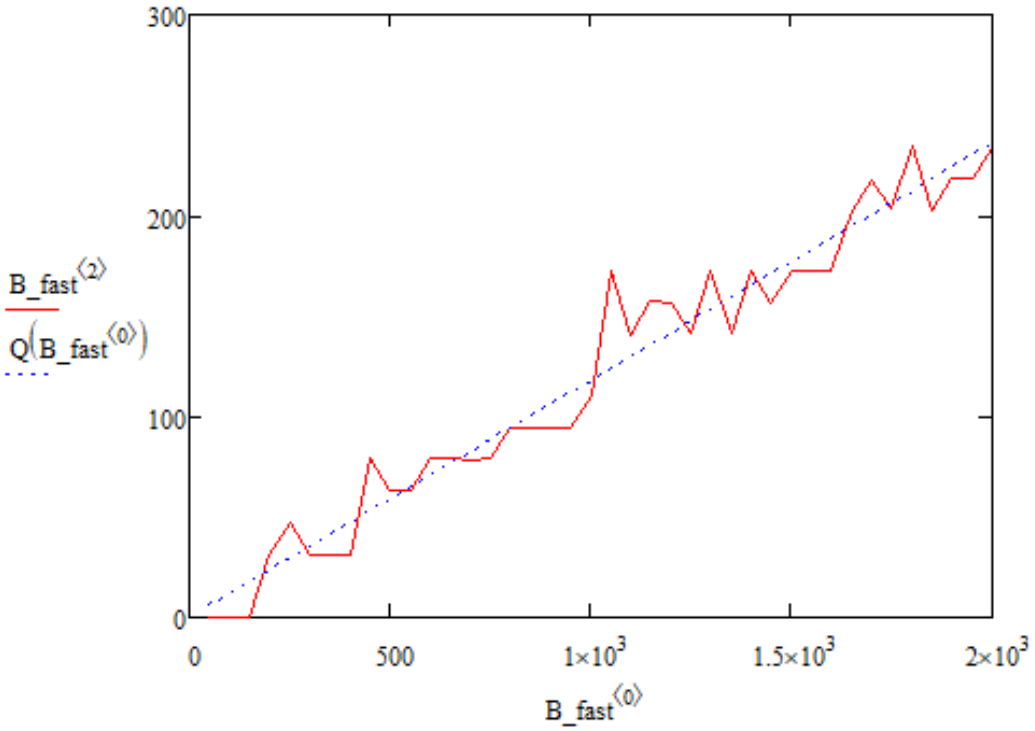
$B\_fast^{\langle 2 \rangle} = a \cdot B\_fast^{\langle 0 \rangle}$

B_fast := READPRN("b_fast.txt")

a := minerr(a) = 0.118

Q(N) := a·N

rate_fast_b := a



Относительная эффективность по времени двух быстрых сортировок:

$f := \frac{rate\_fast\_a - rate\_fast\_b}{rate\_fast\_b} = 0.145$ Составила 14.5%

набор тестов

№	Описание теста	Входные данные	Выходные данные
1	Вывод таблицы	`in.txt` `3`	Форматированная таблица с данными
2	Ошибка при чтении файла	`Null`	Сообщение об ошибке. Завершение работы
3	Неверный выбор опции меню	`in.txt` `a`	Отображение сообщения об ошибке и повторный запуск меню
4	Добавление новой записи	`in.txt` `1` *`<flat data>`* `3`	Добавление записи в конец таблицы

5	Ошибки при добавлении новой записи	in.txt 1 *<invalid data>*	Прерывание операции, возврат в меню
6	Удаление записи	in.txt 5 *<flat id>* 3	Удаление записи с указанным ID из таблицы
7	Ошибки при удалении записи	in.txt 5 *<not flat id>*	Отображение сообщения об ошибке и переход в меню
8	Сортировка таблицы по ключу	in.txt 2 *<1,2,3,4>* *<1,2,3>* *<0,1>*	Сортировка таблицы с последующим её отображением
9	Ошибки при сортировке таблицы	in.txt 2 *<invalid vals>*	Отображение сообщения об ошибке и переход в меню
10	Поиск записей по условию	in.txt 4 *<min price>* *<max price>*	Вывод на экран всех подходящих под условия поиска записей или сообщения, что подходящих записей нет
11	Неверное условие при поиске записей	in.txt 4 *<wrong prices>*	Вывод на экран сообщения об ошибке и переход в меню

Выводы по проделанной работе

По окончании работы мне удалось на практике сравнить эффективность двух различных алгоритмов сортировки, а также подтвердить практически их асимптотическую сложность, рассчитанную теоритически.

Судя по полученным результатам, сортировать таблицу с применением дополнительных массивов, оказывается немного более эффективным решением с точки зрения времени выполнения (приблизительно на **14.5%** для алгоритма сортировки слиянием). Однако уменьшение скорости оборачивается увеличением необходимого для работы объема памяти (приблизительно на **14.28%**).

Таким образом, при разработке программ необходимо выбирать структуры данных и алгоритмы по их обработке основываясь на имеющихся ресурсах для того, чтобы сделать разрабатываемый продукт наиболее эффективным.

Контрольные вопросы

1. *Как выделяется память под вариантную часть записи?*

Память под вариантную часть записи выделяется единым блоком, который по своему объему может уместить максимальный тип из используемых. При этом остальные типы используют ту же область памяти,

из-за чего могут быть логические ошибки при неверном интерпретировании имеющихся в вариантой части данных.

2. Что будет, если в вариантную часть ввести данные, несоответствующие описанным?

В лучшем случае произойдет ошибка компиляции. В худшем — введённые данные будут неправильно интерпретироваться в дальнейшем и в какой-то момент приведут к более серьёзным последствиям.

3. Кто должен следить за правильностью выполнения операций с вариантной частью записи?

За правильностью выполнения операций с вариантной частью должен следить сам программист. Для облегчения отслеживания текущей интерпретации данных допускается использование дополнительного флагового поля, которое будет показывать, какой тип данных используется в данный момент. (Конечно же, это поле не должно храниться внутри вариантного поля).

4. Что представляет собой таблица ключей, зачем она нужна?

Таблица ключей представляет собой массив из упрощенных моделей обычных записей, которые включают в себя минимально возможную информацию для однозначного сопоставления их с исходными записями.

Таблица ключей нужна для сокращения времени работы с исходной таблицей при необходимости частой модификации структуры таблицы, но не самих записей в ней. Например, такой модификацией можно считать сортировку записей, вставка новой записи с сохранением упорядоченности таблицы.

5. В каких случаях эффективнее обрабатывать данные в самой таблице, а когда – использовать таблицу ключей?

В случаях, когда память является более весомым критерием эффективности, следует обрабатывать данные непосредственно на месте, а когда на первом месте стоит время, то конечно стоит использовать таблицу ключей.

Также, если в самой таблице не очень много данных, и они не часто обрабатываются, то перебарщивать с оптимизацией не нужно — в большинстве случаев прирост производительности будет неоправданным (если вообще будет).

6. Какие способы сортировки предпочтительнее для обработки таблиц и почему?

Для обработки таблиц предпочтительнее использовать способы сортировки не требующие большого количества проходов по всему объему данных, так как таблицы зачастую хранят довольно большие объемы информации и такие «обходы» могут очень дорого обойтись, когда речь зайдёт об эффективности алгоритмов сортировки.