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)

ФАКУЛЬТЕТ «ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ»

КАФЕДРА «ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ ЭВМ И ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ»

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ № 2 ПО ДИСЦИПЛИНЕ:
ТИПЫ И СТРУКТУРЫ ДАННЫХ

Записи с вариантами. Обработка таблиц

Студент **Коротков Б. О.**

Группа **ИУ7-34Б**

Вариант **7**

Название предприятия **НУК ИУ МГТУ им. Н. Э. Баумана**

Студент _____ **Коротков Б. О.**

Преподаватель _____ **Никульшина Т. А.**

Москва, 2024

Описание условия задачи

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.

Техническое задание

1. Входные и выходные данные

Программа получает данные путём чтения ввода пользователя в консоль и указанного пользователем файла.

На вход программы ожидаются следующие данные:

1. Целое число — для выбора действия, которое необходимо выполнить программе.
2. Имя файла — для открытия файла для работы с базой данных или файла для записи результатов измерения эффективности сортировок.
3. Строковые и числовые данные — в зависимости от выбранного пользователем действия.

Имеются следующие ограничения:

- при выборе действия должно вводится целое число в диапазоне от 1 до 2 при начальном выборе действий и от 1 до 13 при основном;
- имя файла не должно превышать 100 символов в длину;
- строковые данные в программе должны не превышать 50 символов в длину, состоять только из латинских букв. Исключение — номер телефона абонента. Он должен состоять из 12 символов и иметь вид „+7**********“, где * - цифра от 0 до 9. При вводе статуса абонента есть только 2 корректных варианта —

„Friend“ и „Partner“;
- числовые данные должны быть в диапазоне от 0 до 2147483647, не содержать знаков „+“ и „-“. Однако при вводе адреса диапазон номера дома составляет от 1 до 1000, при вводе дня рождения введённые числовые данные в совокупности должны образовывать корректную дату с учетом високосного года, год рождения не должен быть меньше 1920 года и не больше 2100 года;
- максимальный размер таблицы не должен превышать 10240 записей.

Выходные данные:

1. Таблица с записями — информацией об абонентах, содержащей:
- фамилию абонента;
- имя абонента;
- номер телефона абонента;
- улицу и номер дома абонента;
- статус абонента;
- день рождения или должность и место работы в зависимости от статуса абонента.
2. Таблица ключей, которая содержит индекс абонента в исходной таблице и его фамилию.
3. Таблица результатов исследования эффективности сортировок, которая содержит:
- количество элементов в таблице;
- время сортировки таблицы пузырьковой сортировкой;
- время сортировки таблицы быстрой сортировкой (quicksort);
- время сортировки таблицы ключей пузырьковой сортировкой;
- время сортировки таблицы ключей быстрой сортировкой.

Данные для конкретного размера располагаются в одной строке таблицы.

2. Описание задачи, реализуемой в программе

Программа выполняет различные действия с массивом структур, представленном в виде таблицы. Данные считываются с файла и изменяются впоследствии пользователем. Основная задача программы — исследование времени работы сортировок при работе с исходной таблицей и таблицей ключей.

3. Способ обращения к программе

Способ обращения к программе происходит через консоль. Пользователь запускает файл app.exe и вводит в консоли необходимые данные.

4. Описание возможных аварийных ситуаций и ошибок пользователя

1. Чтение некорректного статуса абонента. Сопровождается кодом возврата 6 — INCORRECT_STATUS_ERR.

2. Чтение некорректной фамилии абонента. Сопровождается кодом возврата 7 — INCORRECT_SURNAME_ERR.

3. Чтение некорректного имени абонента. Сопровождается кодом возврата 8 — INCORRECT_NAME_ERROR.

4. Чтение некорректного номера телефона абонента. Сопровождается кодом возврата 9 — INCORRECT_NUMBER_ERROR.

5. Чтение некорректной улицы проживания абонента. Сопровождается кодом возврата 10 — INCORRECT_STREET_ERROR.

6. Чтение некорректного номера дома абонента. Сопровождается кодом возврата 11 — INCORRECT_HOUSE_NUMBER_ERROR.

7. Чтение некорректной даты. Сопровождается кодами возврата 12 — INCORRECT_DAY_ERROR, 13 — INCORRECT_MONTH_ERROR и 14 — INCORRECT_YEAR_ERROR для дня, месяца и года соответственно.

8. Чтение некорректной должности абонента. Сопровождается кодом

возврата 15 — INCORRECT_POSITION_ERROR.

9. Чтение некорректной организации абонента. Сопровождается кодом возврата 16 — INCORRECT_ORGANIZATION_ERROR.

10. Переполнение массива. Сопровождается кодом возврата 100 — ARR_OVERFLOW_ERROR.

11. Ошибка при удалении записи. Сопровождается кодом возврата 101 — DELETE_ERROR.

12. Чтение пустого файла. Сопровождается кодом возврата 102 — EMPTY_FILE_ERROR.

13. Ошибка при открытии или создании файла. Сопровождается кодом возврата 200 — INIT_FILE_ERROR.

14. Некорректный ввод номера действия. Сопровождается кодом возврата 401 — INCORRECT_ACTION_ERROR.

Описание внутренних структур данных

Информация об абоненте содержится в программе с помощью структуры person_t:

```
typedef struct
{
    string_t surname;
    string_t name;
    string_t phoneNumber;
    adress_t adress;
    string_t status;
    info_t info;
} person_t;
```

В структуре имеется вариантное поле info_t, которое имеет следующий вид:

```
typedef union
{
    birthday_t birthday;
    work_t work;
} info_t;
```

Адрес абонента хранится в структуре с помощью вспомогательной структуры adress_t:

```
typedef struct
{
    string_t street;
    int houseNumber;
} adress_t;
```

Информация о дне рождения абонента хранится с помощью вспомогательной структуры birthday_t, которая является обёрткой структуры date_t:

```
typedef date_t birthday_t;
typedef struct
{
    int day;
    int month;
```

```
    int year;
} date_t;
```

Информация о месте работы абонента хранится с помощью вспомогательной структуры work_t:

```
typedef struct
{
    string_t position;
    string_t organization;
} work_t;
```

Ключ из таблицы ключей представлен в виде структуры person_key_t, содержащей непосредственно сам ключ и индекс элемента в исходной таблице:

```
typedef struct
{
    string_t key;
    size_t index;
} person_key_t;
```

Чтобы хранить записи об абонентах и ключи, были созданы вспомогательные структуры person_array_t и person_key_array_t:

```
typedef struct
{
    person_t array[MAX_ARRAY_LEN];
    size_t size;
} person_array_t;

typedef struct
{
    person_key_t array[MAX_ARRAY_LEN];
    size_t size;
} person_key_array_t;
```

Представленный способ хранения данных был выбран ввиду своей простоты и эффективности, а использование вариантной части позволяет сэкономить память, занимаемую структурой.

Алгоритм программы

Общий алгоритм программы

1. Инициализация необходимых для работы программы переменных.

2. Ввод пользователем первоначального действия, которое необходимо выполнить программе — открыть/создать файл или завершить свою работу.

3. Ввод пользователем действий, которые необходимо выполнить программе.

Доступные для пользователя действия:

1. Прочитать файл в базу данных.
2. Записать в файл базу данных.
3. Вывести базу данных.
4. Вывести таблицу ключей.
5. Добавить запись в базу данных.
6. Удалить запись из базы данных.
7. Отсортировать базу данных по фамилии абонента.
8. Вывести базу данных по таблице ключей.
9. Отсортировать таблицу ключей.
10. Вывести список друзей, которых надо поздравить с Днём Рождения.
11. Исследование времени работы сортировок.
12. Заполнить таблицу случайными числами.
13. Выйти из программы.

Таблица входных и выходных данных

Ввод	Вывод	Описание
Иванов (при вводе данных)	Некорректные данные пользователя!	Ввод кириллических символов в строковое поле.
Ivanov Ivan +89880000000 (при вводе данных)	Некорректные данные пользователя!	Номер не соответствует корректному.
A (при выборе действия)	Некорректный номер действия!	Ввод некорректного символа при выборе действия.
1 (выбор действия при пустом файле)	Файл пуст!	Чтение данных из пустого файла.
1 (при превышении количества записей в файле)	Произошло переполнение базы данных!	Чтение файла, где количество записей превышает допустимое количество.
1 (при неверном значении поля в файле)	Нарушена структура файла!	Файл содержит структуру с неверным значением поля.
6 (при вводе несуществующего пользователя)	Данный пользователь не найден!	Удаление несуществующего пользователя.
10 (при отсутствии ближайших именинников)	Искомых друзей не найдено!	Отсутствие ближайших именинников.

При корректных входных данных программа выводит ожидаемый пользователем результат.

Оценка эффективности

В программе реализовано два алгоритма сортировки и два способа сортировки данных. В совокупности имеем четыре разных варианта сортировки:

1. Сортировка исходной таблицы пузырьковой сортировкой.
2. Сортировка исходной таблицы быстрой сортировкой.
3. Сортировка таблицы ключей пузырьковой сортировкой.
4. Сортировка таблицы ключей быстрой сортировкой.

Чтобы оценить эффективность каждого способа, в программе реализована

функция замера времени выполнения сортировки базы данных этими способами. Результат выводится в отдельный текстовый файл, который впоследствии можно обработать сторонней программой.

Функция заполняет файл перед каждой сортировкой случайными корректными записями и проводит замеры для каждого количества записей 10 раз для уменьшения неточности. Замеры проводятся для количества записей в диапазоне от 10 до 9910 с шагом 100.

Результаты, полученные в ходе исследования, представлены в виде графиков ниже:

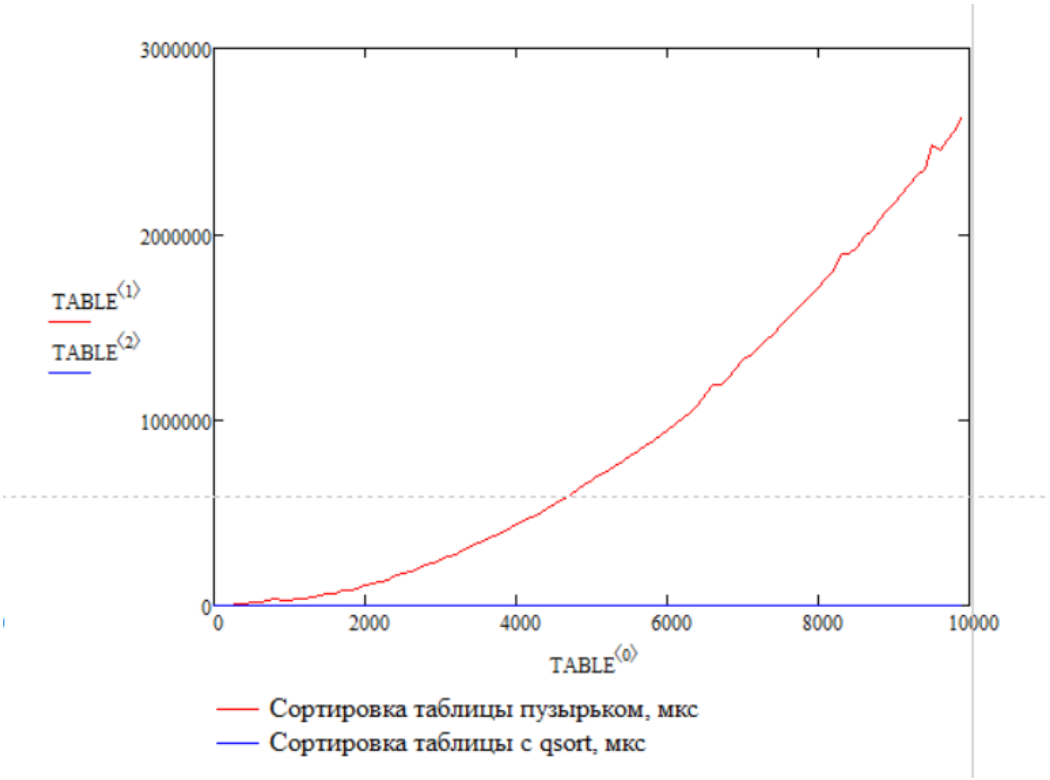

Рисунок 1: Сравнение сортировки исходной таблицы пузырьком и quicksort

На графике выше мы видим, что время выполнения сортировки с помощью быстрой сортировки ничтожно мало по сравнению с сортировкой пузырьком. Поэтому следующий график будет показывать те же замеры, но с логарифмической шкалой по оси Y (ось Y — время выполнения сортировки, ось X — количество записей).

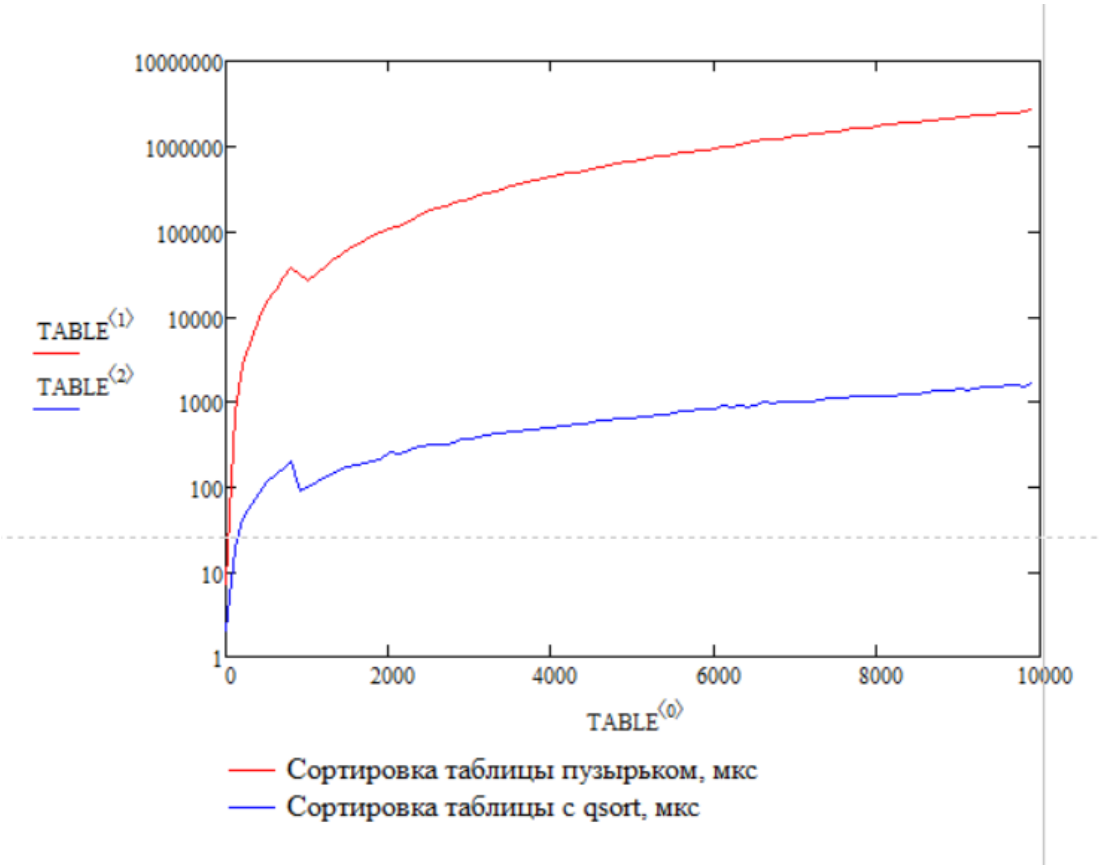

Рисунок 2: Сортировка пузырьком и quicksort с логарифмическим масштабом

На этом графике лучше видно время работы быстрой сортировки.

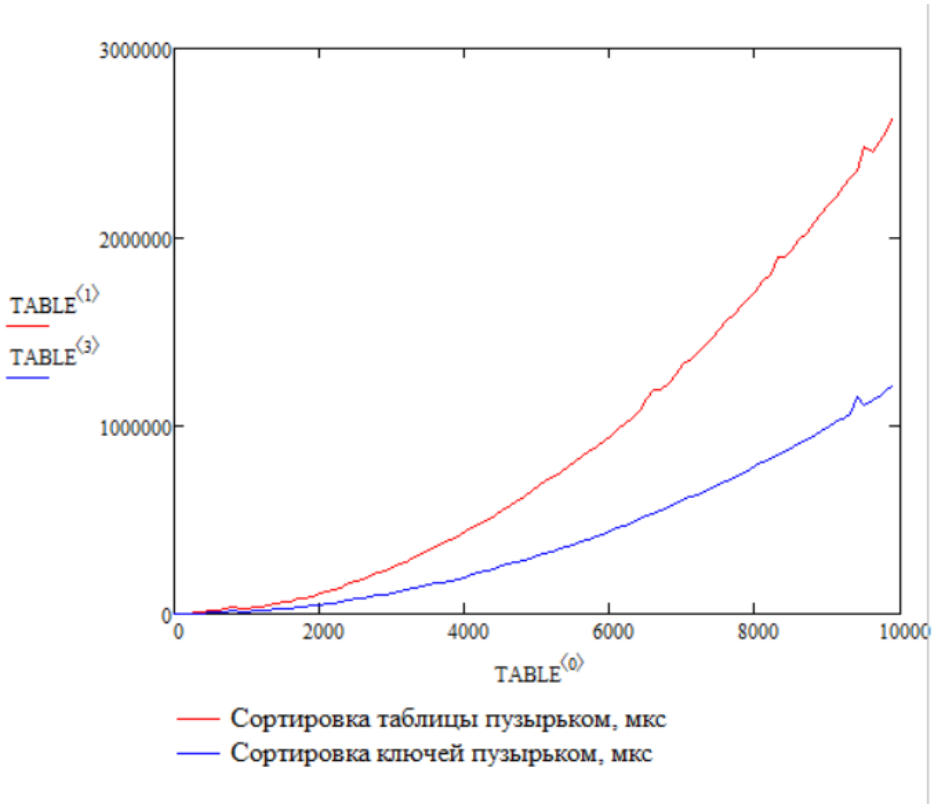

Рисунок 3: Сравнение сортировки исходной таблицы и таблицы ключей

На графике видно, что сортировка таблицы ключей гораздо эффективнее,

чем сортировка исходной таблицы.

График, содержащий все замеры:

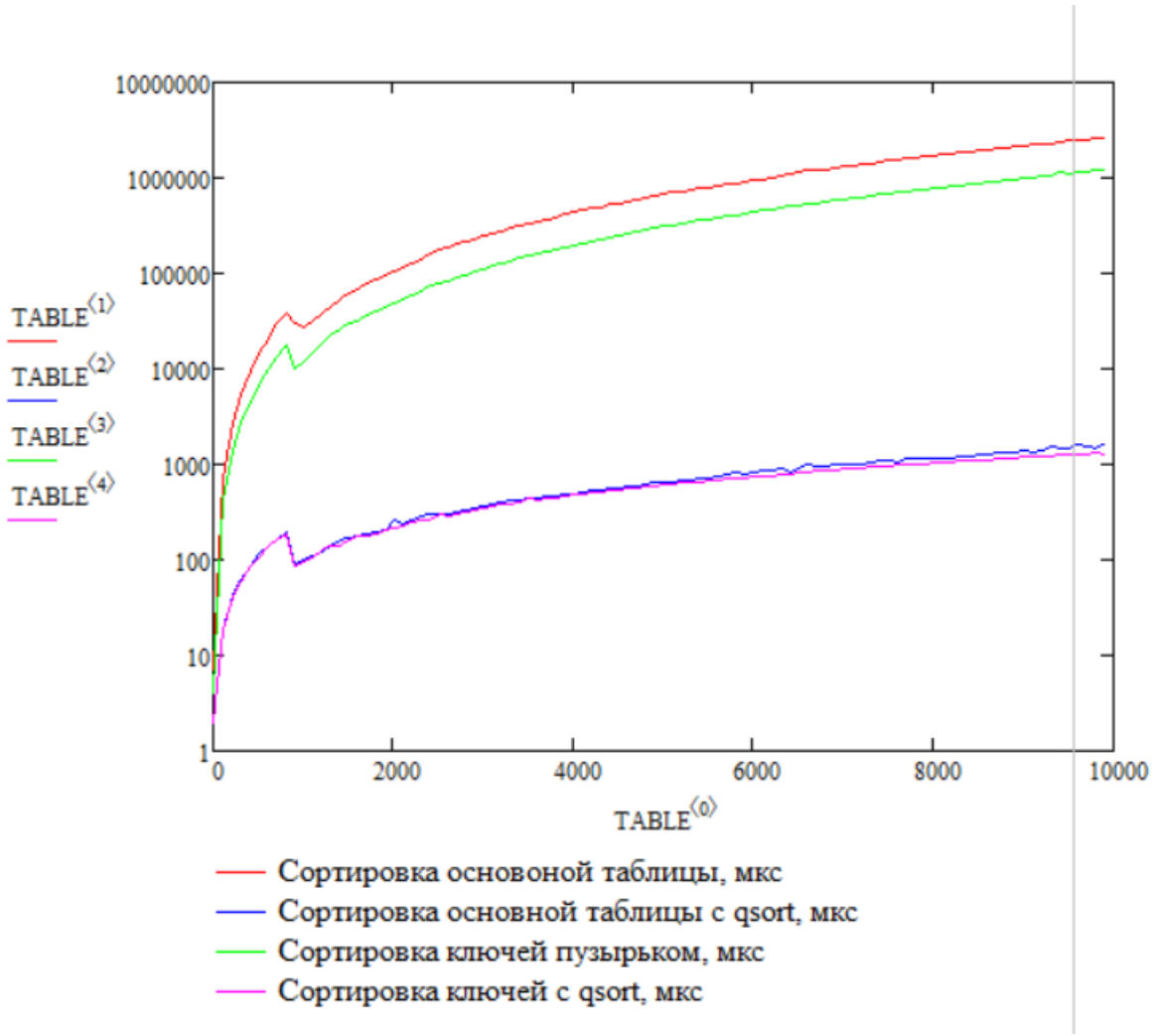

Рисунок 4: Общий график

Часть таблицы полученных значений:

Кол-во элементов	**Bubblesort исходной таблицы, мкс**	**Quicksort исходной таблицы, мкс**	**Bubblesort таблицы ключей, мкс**	**Quicksort таблицы ключей, мкс**
5710	874999	940	412490	754
6610	1140731	1023	518372	839
7610	1544346	1906	711067	988
9410	2370032	2175	1074422	1269

Сравнение по времени:

Кол-во элементов	**Сортировка таблицы ключей против сортировки исходной при пузырьковой сортировке**	**Сортировка таблицы ключей против сортировки исходной при quicksort**
5710	На 52% быстрее	На 25% быстрее
6610	На 55% быстрее	На 18% быстрее
7610	На 54% быстрее	На 48% быстрее
9410	На 55% быстрее	На 41% быстрее

Объём занимаемой памяти при использовании каждого метода:

1. Пузырьковая сортировка без таблицы ключей:

- память под массив записей.

Доп. память: нет.

2. Пузырьковая сортировка с таблицей ключей:

- память под массив записей;
- память под массив ключей.

Доп. память: ~20%.

3. Быстрая сортировка без таблицы ключей:

- память под массив записей;

Доп. память: нет.

4. Быстрая сортировка с таблицей ключей:

- память под массив записей;
- память под массив ключей.

Доп. память: ~20%.

Вывод

Выбор метода сортировки любых данных прежде всего зависит от поставленной программисту задачи. Если стоит цель использовать как можно меньше памяти, то целесообразно использовать алгоритмы, не потребляющие дополнительную память, однако в общем случае такие алгоритмы работают гораздо медленнее. Если на первом месте стоит скорость, то необходимо пожертвовать памятью для более быстрого алгоритма. В дополнение к этому, можно создать таблицу ключей, по которым будет вестись сортировка, чтобы ускорить процесс еще сильнее. Однако такой прирост будет наблюдаться только при большом количестве сортируемой информации, иначе разница будет незначительна.

При использовании быстрой сортировки алгоритм показывает разницу во времени выполнения между сортировкой исходной таблицы и таблицы ключей только при большом количестве элементов, иначе использование таблицы ключей нецелесообразно.

Ответы на контрольные вопросы

1. Как выделяется память под вариативную часть записи?

Память под вариативную часть записи обычно выделяется с использованием указателей или динамических массивов. В основной структуре данных определен указатель, который указывает на область памяти, где хранятся вариативные данные. Размер этой памяти равен размеру самого затратного по памяти полю.

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

Если в вариативную часть введены данные, несоответствующие ожидаемым, это может привести к некорректной работе программы. Например, если ожидается ввод числа, а пользователь вводит текст, это может вызвать ошибку или некорректное поведение программы. Поэтому важно

предусмотреть проверки и обработку некорректного ввода.

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

Ответственность за правильность выполнения операций с вариативной частью записи лежит на программисте. Программист должен уделять внимание правильному выделению и освобождению памяти, а также обработке данных в вариативной части, чтобы избежать утечек памяти и ошибок в программе.

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

Таблица ключей представляет собой структуру данных, которая используется для ускорения поиска, сортировки и доступа к данным в основной таблице. Она содержит ссылки (индексы, указатели) на записи в основной таблице. Таблица ключей полезна, когда требуется быстрый доступ к данным, отсортированным по определенному критерию.

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

Эффективность обработки данных в самой таблице или с использованием таблицы ключей зависит от конкретной задачи. Если часто выполняются операции поиска и сортировки данных, то использование таблицы ключей может значительно ускорить выполнение программы. Однако, если требуется постоянный доступ и изменение данных, то работа с данными в основной таблице может быть более эффективной.

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

Выбор способа сортировки зависит от размера данных и требований к производительности. Быстрая сортировка, например, может быть предпочтительнее для больших объемов данных, так как она имеет лучшую

временную сложность. Однако, она требует дополнительной памяти для стека вызовов. Медленная сортировка, такая как сортировка пузырьком, может быть менее эффективной, но более простой в реализации и не требует дополнительной памяти. Выбор зависит от конкретной задачи и компромиссов между временем выполнения и потребляемой памятью.