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

Студент **Паламарчук А.Н.**

Группа **ИУ7-33Б**

Предмет **Типы и структуры данных**

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

Студент _____ **Паламарчук А.Н.**

Преподаватель _____ **Никульшина Т. А.**

Преподаватель _____ **Барышникова М. Ю.**

2023 г.

Условие задачи

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, используя 2 разных алгоритма сортировки (простой, ускоренный). Оценить эффективность этих алгоритмов (по времени и по используемому объему памяти) при различной реализации программы, то есть, в случаях а) и б).

Техническое задание

Ввести список литературы, содержащий фамилию автора, название книги, издательство, количество страниц, вид литературы (1: техническая – отрасль, отечественная, переводная, год издания; 2: художественная – роман, пьеса, поэзия; 3: детская – минимальный возраст, сказки, стихи). Вывести список всех романов указанного автора.

Входные данные

- Команда (число) из меню программы (см. описание алгоритма) – обозначение необходимой операции.

Выходные данные

- Таблица эффективности
- Отсортированная таблица
- Таблица ключей
- Исходная таблица
- Таблица романов искомого автора

Возможные аварийные ситуации

- Некорректный исходный файл
- Ошибка открытия файла
- Отсутствие исходного файла
- Нехватка памяти для работы программы
- Некорректная команда

Описание внутренних структур данных

```
typedef struct
{
    char industry[MAX_LEN_INDUSTRY];
```

```
    size_t national;
    size_t translated;
    int year;
} technical_t;

typedef enum
{
    novel = 1,
    play,
    poetry1
} imaginative_t;

typedef enum {
    tales = 1,
    poetry2
} children_type_t;
typedef struct
{
    int age;
    children_type_t type;
} children_t;

typedef union
{
    technical_t tl;
    imaginative_t il;
    children_t cl;
} literature_t;

typedef enum {
    tl = 1,
    il,
    cl
} item_t;

typedef struct {
    char author[MAX_LEN_AUTHOR];
    char title[MAX_LEN_TITLE];
    char publish[MAX_LEN_PUBLISH];
    int pages;
    item_t type;
    literature_t item;
} library_item_t;
```

Константы

```
#define MAX_SIZE 1000
#define FILE_IN "file_in.txt"
#define COUNT_TESTS 1000
#define MAX_LEN_STR 30
#define MAX_LEN_INDUSTRY 30
#define MAX_LEN_AUTHOR 15
#define MAX_LEN_TITLE 40
#define MAX_LEN_PUBLISH 35
```

```
#define MAX_PAGES 10000
```

Используемые функции

```
unsigned long long cur_ms_gettimeofday();
```

Возвращает текущее время

```
void print_item_key_table(key_table_item_t *table);
```

Печатает элемент таблицы ключей

```
void print_arr_with_key_table(library_item_t *arr, key_table_item_t *key, size_t
size);
```

Печатает массив по таблице ключей

```
void print_literatures(library_item_t *arr, size_t size);
```

Печатает массив

```
void swap_key_table(key_table_item_t *a, key_table_item_t *b);
```

Меняет элементы таблицы ключей

```
void init_key_table(key_table_item_t *dst, library_item_t *src, size_t size);
```

Инициализирует таблицу ключей

```
void sort_lib(library_item_t *arr, size_t size);
```

Сортирует массив

```
void fast_sort_lib(library_item_t *arr, size_t size);
```

Ускоренно сортирует массив

```
void sort_key_table(key_table_item_t *arr, size_t size);
```

Сортирует таблицу ключей

```
void fast_sort_key_table(key_table_item_t *arr, size_t size);
```

Ускоренно сортирует таблицу ключей

```
void print_filter_arr(library_item_t *arr, size_t size, char *search_author);
```

Выводит элементы удовлетворяющие условиям

```
int get_avg_sort_time(library_item_t *arr, size_t size, void (*sort)(library_item_t *arr, size_t size));
```

Считает среднее время сортировки таблицы

```
int get_avg_sort_time_keytable(library_item_t *arr, size_t size, void (*sort_key)(key_table_item_t *arr, size_t size));
```

Считает среднее время сортировки таблицы ключей

```
void clean_stdin(void);
```

Очищает stdin

```
int fscan_string(char *dst, size_t max_len, FILE* file);
```

Считывает строку

```
int fscan_technical(technical_t *dst, FILE *file);
```

Считывает дополнительные параметры для технической литературы

```
int fscan_imaginative(imaginative_t *dst, FILE *file);
```

Считывает дополнительные параметры для художественной литературы

```
int fscan_children(children_t *dst, FILE *file);
```

Считывает дополнительные параметры для детской литературы

```
int fscan_literature(library_item_t *dst, FILE *file);
```

Считывает элемент таблицы

```
void swap_literature(library_item_t *a, library_item_t *b);
```

Меняет элементы массива

```
void print_line(void);
```

Распечатка линии

```
void print_header(void);
```

Распечатка заголовка

```
void print_literature(library_item_t *src);
```

Печатает элемент таблицы

```
void copy_literature(library_item_t *dst, library_item_t const *src);
```

Копирует элемент таблицы

```
int remove_literature(library_item_t *arr, size_t *size, size_t ind);
```

Удаляет элемент таблицы по индексу

Описание алгоритма
Программа обрабатывает нужную команду:

1 - Получения таблицы производительности

2 - Вывод меню справки

3 - Вывод таблицы произведений

4 - Ввести новое произведение

5 - Удалить произведение по индексу

6 - Вывести таблицу отсортированную с помощью таблицы ключей (оптимизированная сортировка)

7 - Вывести таблицу отсортированную с помощью таблицы ключей (неоптимизированная сортировка)

8 - Вывести отсортированную таблицу (оптимизированная сортировка)

9 - Вывести отсортированную таблицу (неоптимизированная сортировка)

10 - Вывести отсортированную таблицу ключей

11 - Вывести список всех романов указанного автора

0 - Завершить работу программы

Программа считывает данные из file_in.txt, загружает их в таблицу. Реализованы следующие сортировки: пузырьком и выбором. Для того, чтобы отсортировать с помощью таблицы ключей – выделяется динамическая память, она используется для хранения самой таблицы ключей. Для получения соответствующего элемента в массиве записей с помощью элемента таблицы ключей – берётся индекс из таблицы ключей и передаётся в качестве индекса элемента массива.
При замерах производительности время сортировки усредняется для каждого алгоритма и способа.

Производительность

Все измерения проведены в мкс. Количество итераций 100.

N	Sort	Fast sort	Sort(key table)	Fast sort(key table)
100	778	19	33	10
200	3078	72	183	43
300	6932	159	422	108
400	17423	304	778	213
500	24359	477	1207	324
600	37276	686	1773	476
800	53618	1186	3108	826
1000	98244	1917	4875	1315

Кол-во элементов	Сортировка пузырьком с таблицей ключей	Сортировка вставками с таблицей ключей
N	Выигрыш по времени, %	Выигрыш по времени, %
100	96	48
200	94	40
300	95	32
400	96	30
500	95	32
600	95	30
800	95	30
1000	95	31

Для сортировки с помощью таблицы ключей требуется на 10 % памяти больше.

Выводы

Для более быстрой обработки можно использовать метод сортировки, задействующий таблицу ключей, если это позволяет объем памяти.

Использование таблицы ключей позволяет сильно сократить время работы программы при работе с объемными данными.

Использование объединений позволяет уменьшить объем используемой памяти для вариантной части до минимально необходимого.

1. Как выделяется память под вариантную часть записи?

По максимальному размеру возможного варианта.

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

Неопределенное поведение

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

В этом случае вся ответственность лежит на программисте

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

Дополнительная таблица, содержащая ключ соответствующего элемента и его индекс в исходном массиве. Таблица нужна для более быстрой обработки данных.

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

Когда объем обрабатываемых данных недостаточно большой и время их обработки является не существенным.

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

Способы сортировки, которые обмениваю элементы местами минимальное кол-во раз, т.к. на это требуется дополнительное время и память.