Национальный исследовательский университет
компьютерных технологий, механики и оптики

Факультет ПИиКТ

Низкоуровневое программирование
Лабораторная работа №1
Вариант 1

Работу выполнил: Зубахин Д. С.
Группа: P33312
Преподаватель: Кореньков Ю. Д.

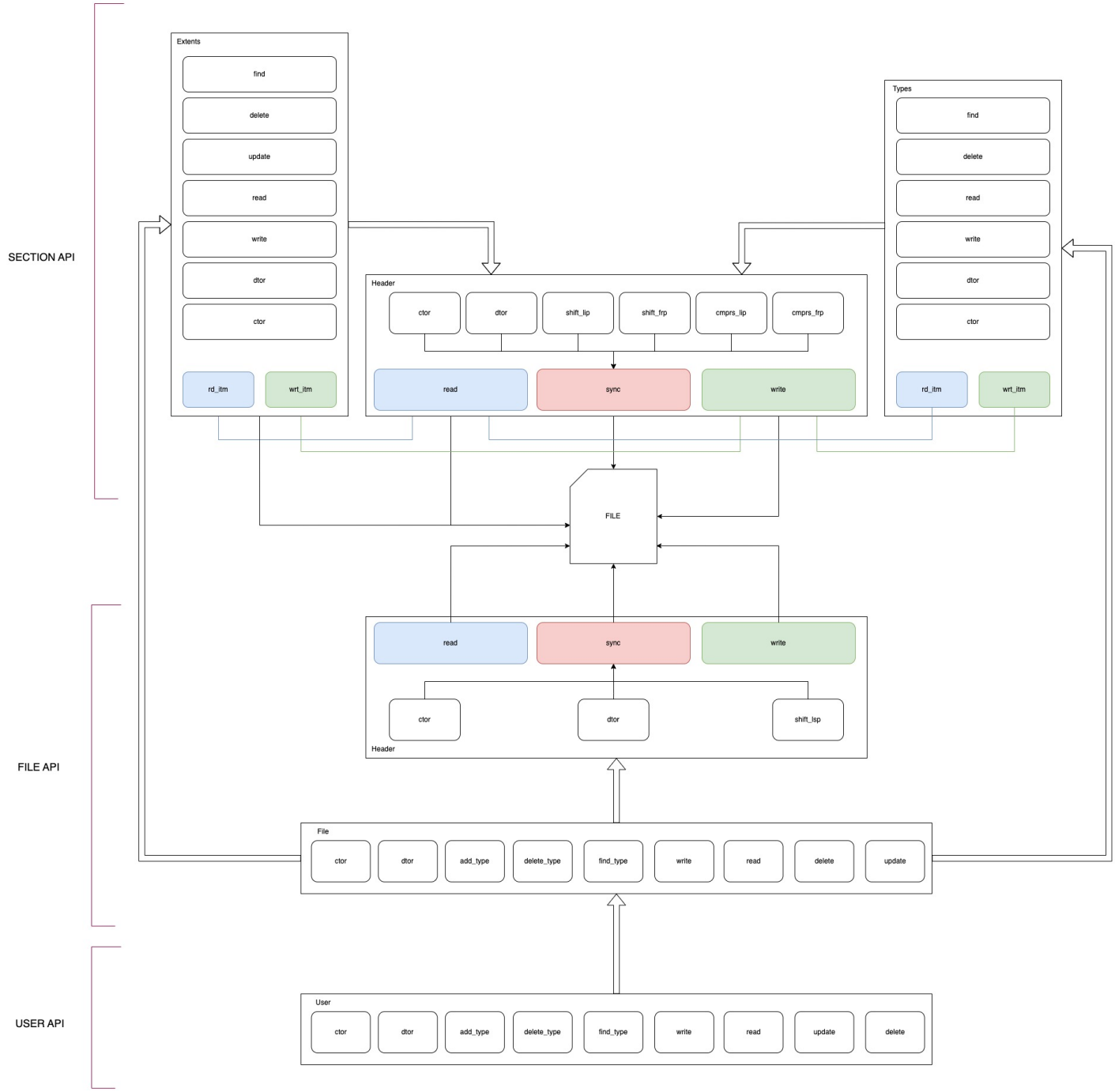
Санкт-Петербург
2022 год

1 Задача

Основная цель лабораторной работы - Создать модуль, реализующий хранение в одном файле данных (выборку, размещение и гранулярное обновление) информации общим объёмом от 10GB соответствующего варианту вида:

1. Спроектировать структуры данных для представления информации в оперативной памяти
2. Спроектировать физическую организацию данных и архитектуру приложения
3. Спроектировать прикладные интерфейсы модулей
4. Сделать имплементацию интерфейсов посредством TDD
5. Покрыть реализацию модульными тестами
6. Написать стресс тесты для приложения на полученном прикладном интерфейсе

2 Описание работы



Данные хранятся как деревья узлов. Каждый узел содержит ключ свой и значение которое является одним из 4 примитовов. Для объектных типов узлов существуют также схемы. Схемы можно добавить, удалить и прочитать с помощью user api. Если мы проверяем ноду на соответствие типу, то все поля в типе являются required. Остальные поля в типе могут быть или не быть.

Также у нас есть запросы. Запросы состоят из частичек запросов, которые содержат ключ, тип и значение ключа. Таким образом запросы могут быть составными.

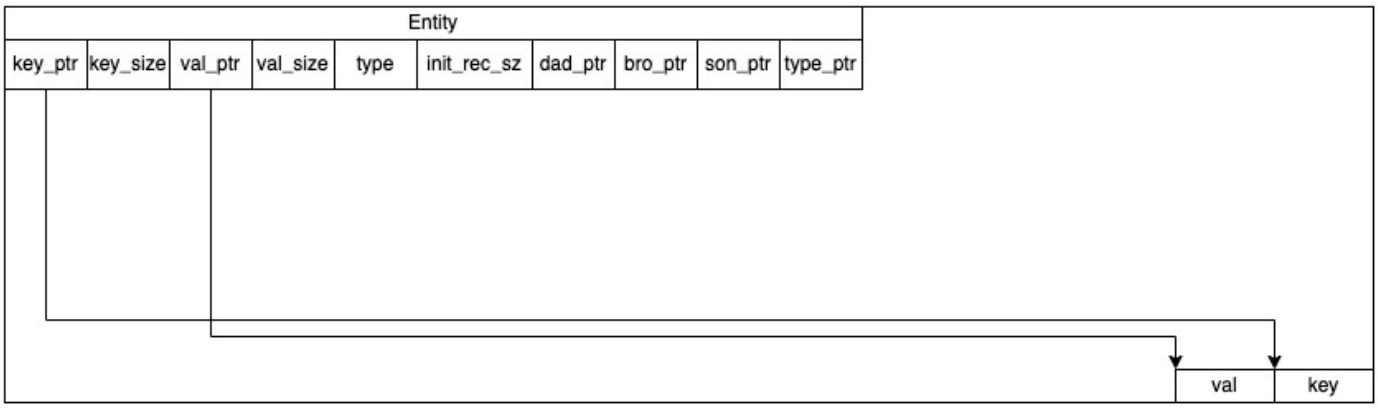
3 Аспекты реализации

Файл для размещения данных использует секции, секции представляют из себя куски файла размером 8192 байта.

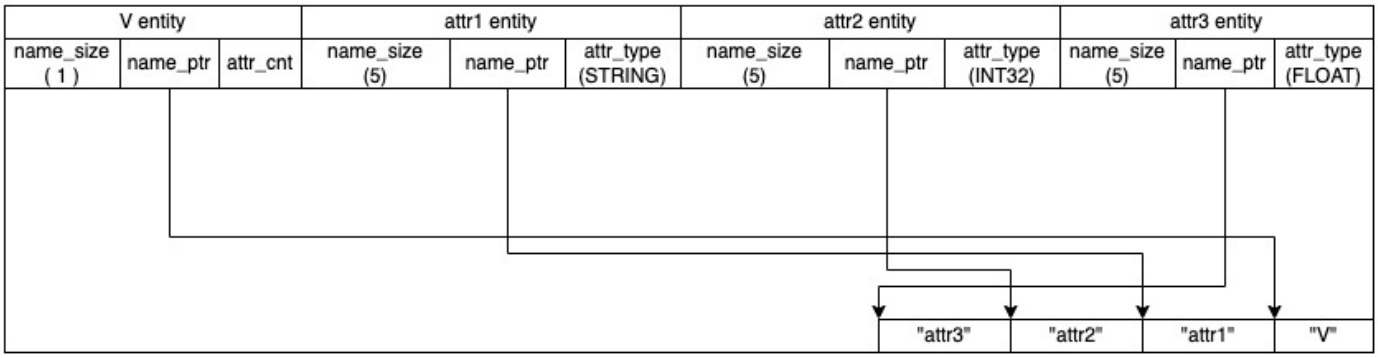
У каждой секции есть header, который хранит метаинформацию для обслуживания.

```
typedef struct Sect_head
{
    size_t free_space;
    Fileoff next_ptr;    // Nullable
    Sectoff lst_itm_ptr; // Pointer to first free cell(after items)
    Sectoff fst_rec_ptr; // Pointer to last free cell(before records)
    Fileoff file_offset; // Section offset from file start
    FILE *filp;
} Sect_head;
```

Размещение нод происходит по следующей логике:



Размещение типов происходит по следующей логике:



Далее мы работаем с этими секциями через file.

У него тоже есть свой header. Который хранит метаинформацию.

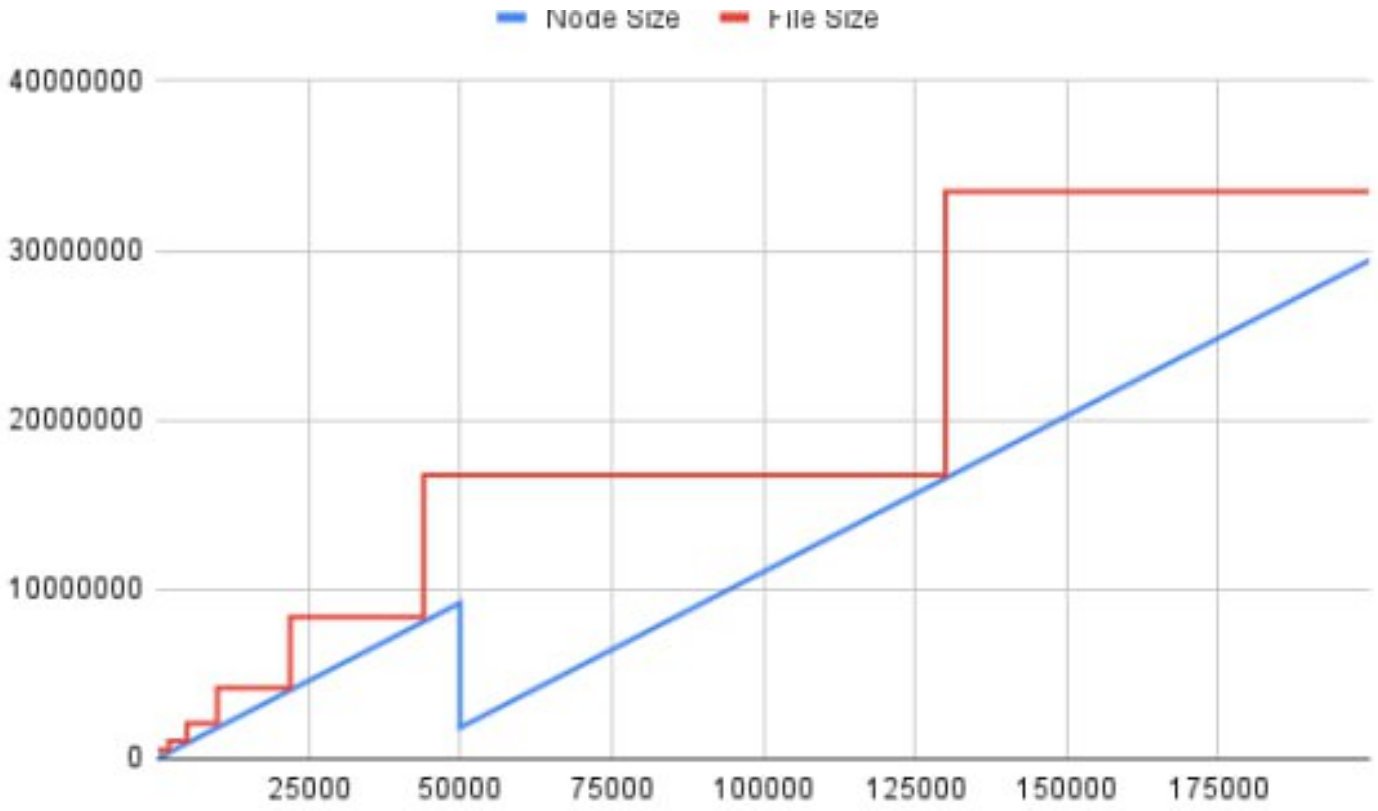
```
typedef struct File_head
{
    Fileoff lst_sect_ptr; // Points first free bit after sects
    FILE *filp;
} File_head;
```

4 Выводы

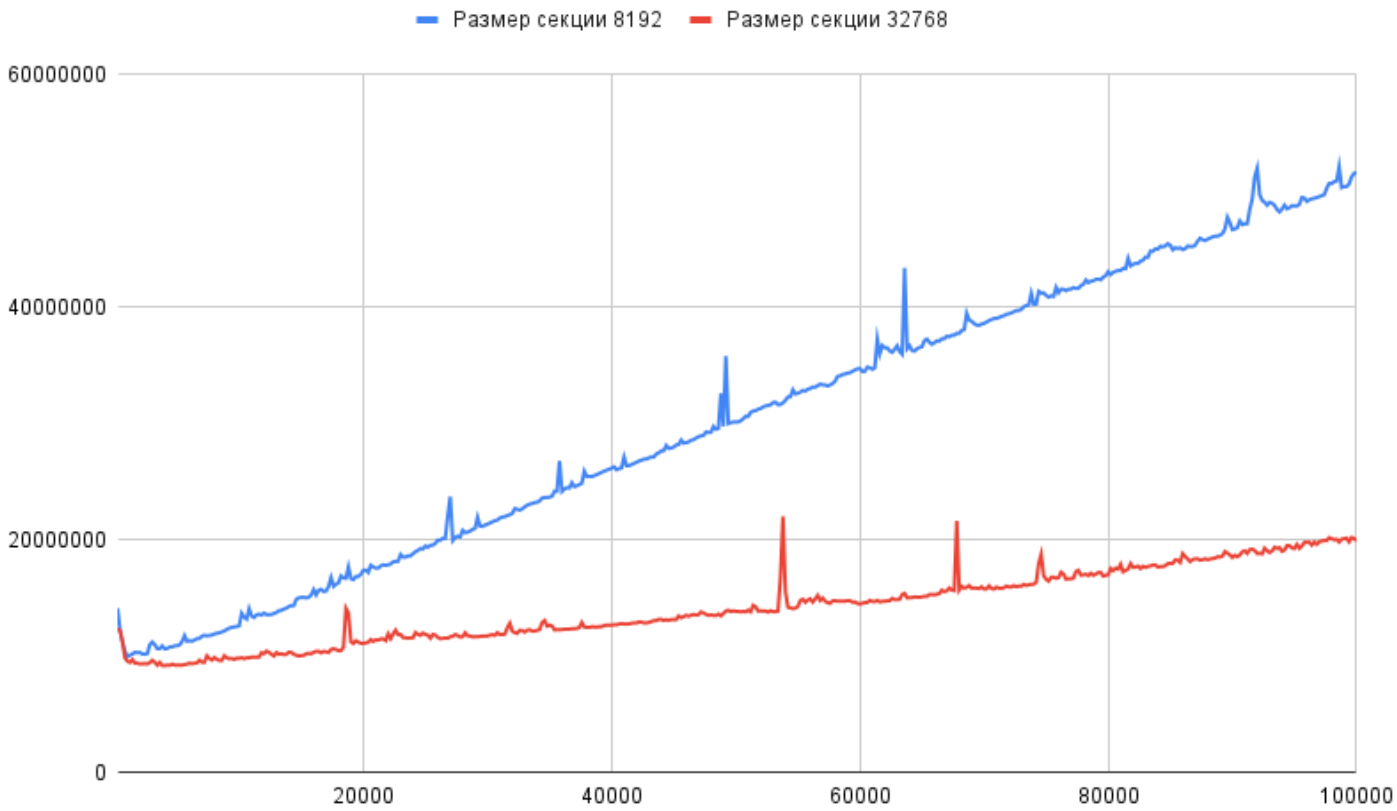
В ходе долгой работы я написал и протестировал субд которая может работать с документным деревом.

В ходе тестов я рассмотрел ассимптотики 4 операций:

- **Соотношение размера файла от введеных данных**

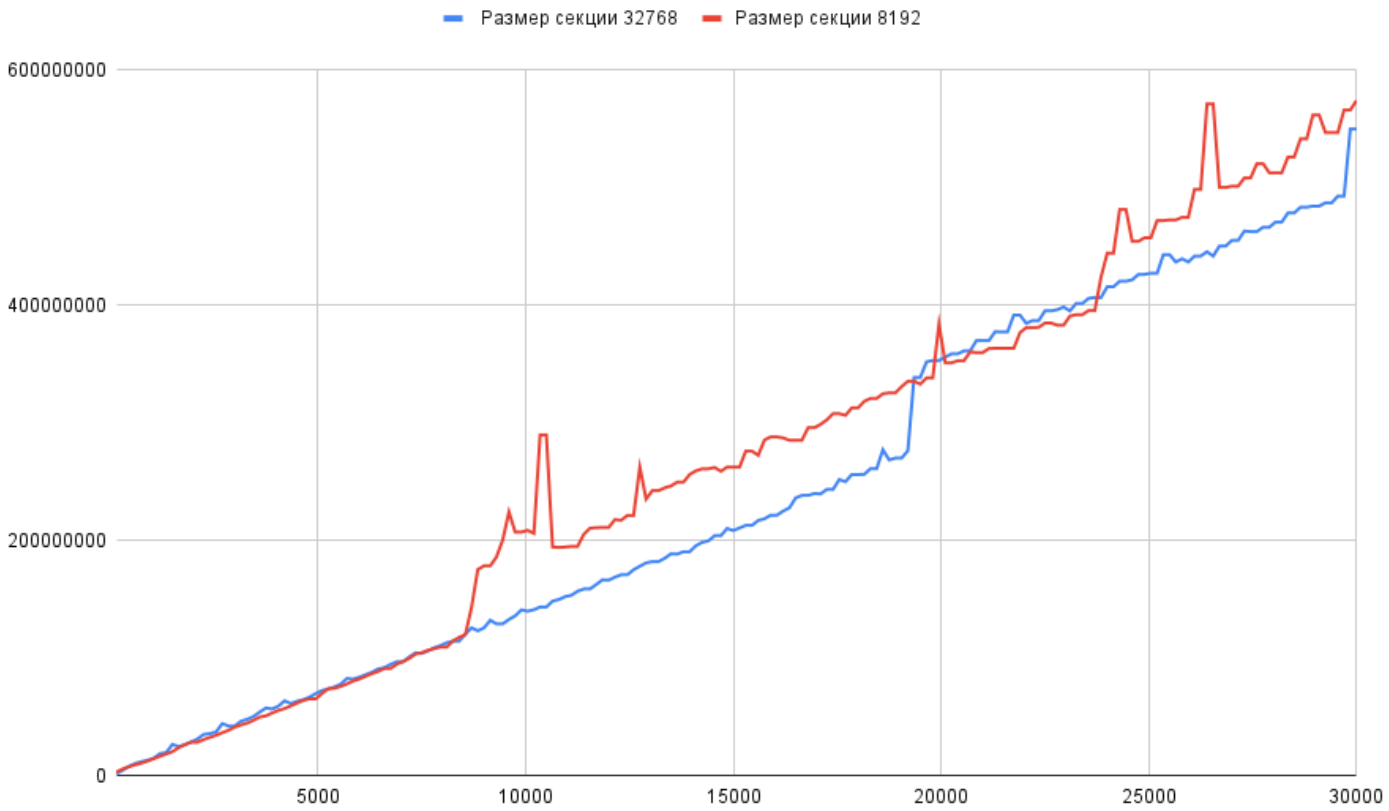


- **Write**



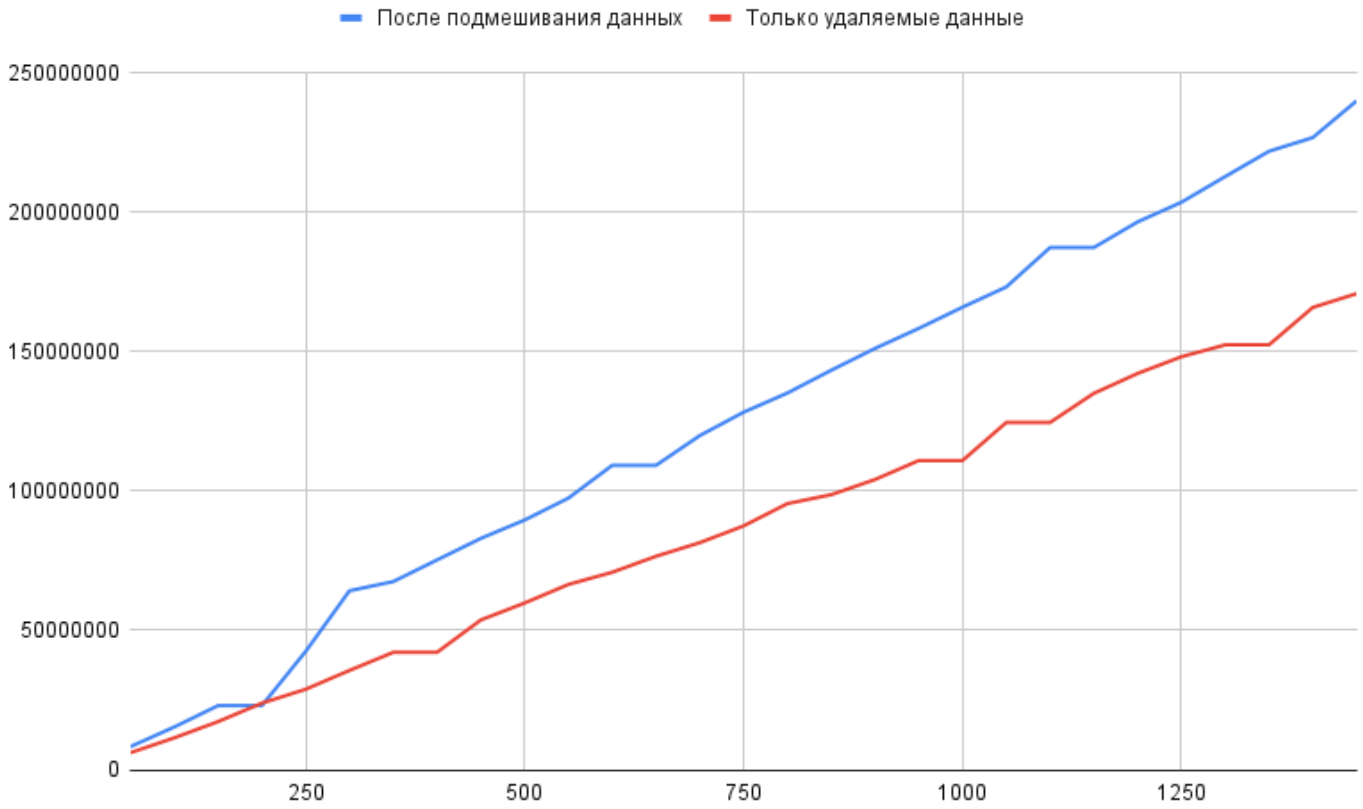
Как видно по графикам, при увеличении размера секции мы видим уменьшение затрачиваемого времени. Следовательно мы вставляем за линию от кол-ва секций.

- **Read**



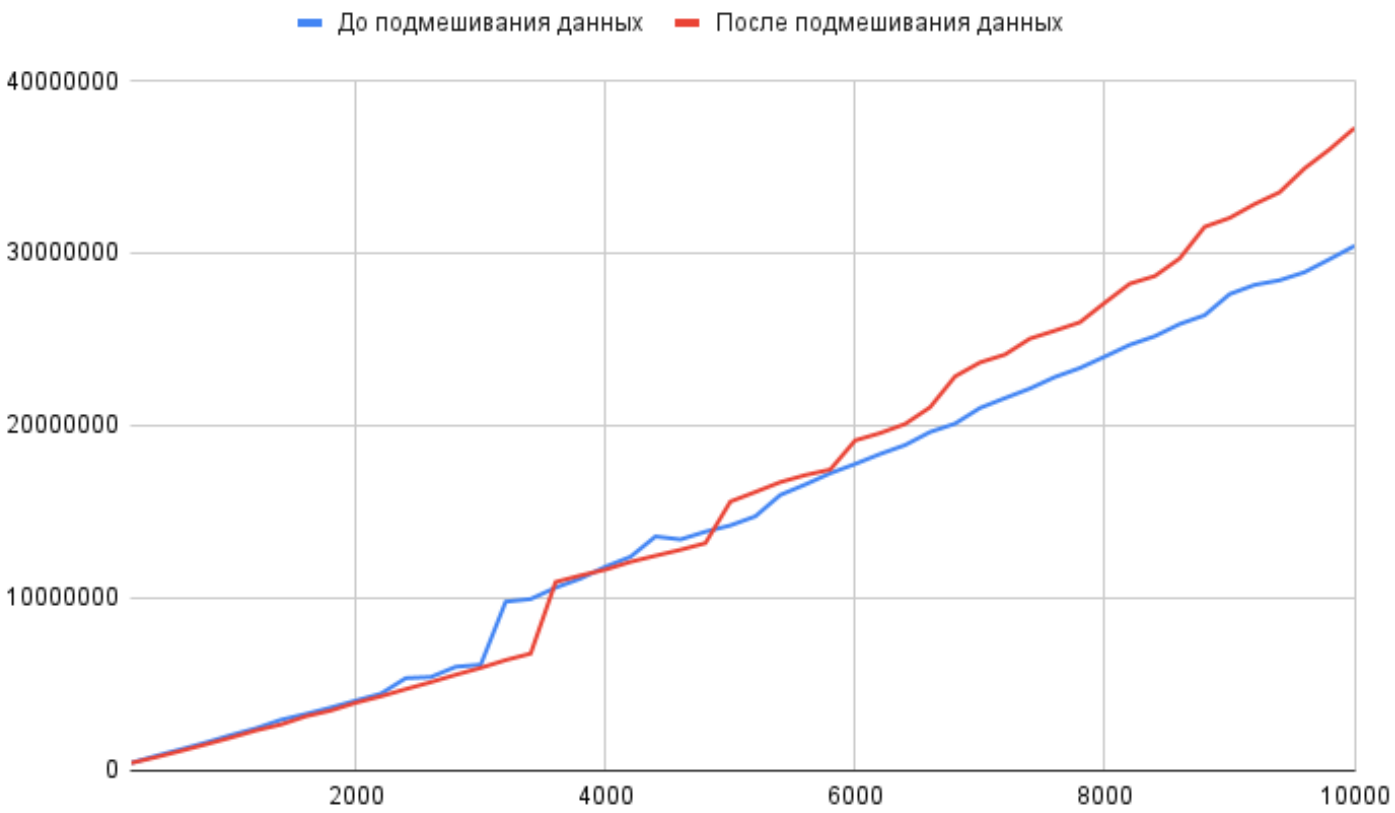
Как видно по графикам, при увеличении размера секции мы **не** видим уменьшение затрачиваемого времени. Следовательно мы читаем за линию от кол-ва элементов.

- **Delete**



Как видно по графикам, при подмешивании не удаляемых элементов время удаления увеличивается. Следовательно мы удаляем за линию от кол-ва затронутых элементов.

- **Update**



Как видно по графикам, при подмешивании не обновляемых элементов время обновления увеличивается. Следовательно мы обновляем за линию от кол-ва затронутых элементов.