# Как работет приложение

## Шаг 1

### 1. Запуск шага 1

Чтобы запустить программу из первого шага, необходимо иметь `Node.js`.
Команда:
`node firstTask.js <M>`
где `M` - количество строк
На вход всегда берется файл `demon.txt_Ascii.txt` в месте запуска в кодировке `UTF-8`
Вывод в `random-lines-n-times.txt`
Чтобы получить файл размера в `1MB`, стоит указать примерно `3333` строк

### 2. Алгоритм шага 1

1. Загрузка в память всего входного файла в виде строки
2. Разделение полученной строки по символам `\r` и `\n`
3. Создание файла вывода
4. Получаение случайного числа в районе от 0 до количества строк в входном файле
5. Запись строки из входного файла под случайнымм номером в конец файла вывода

## Шаг 2

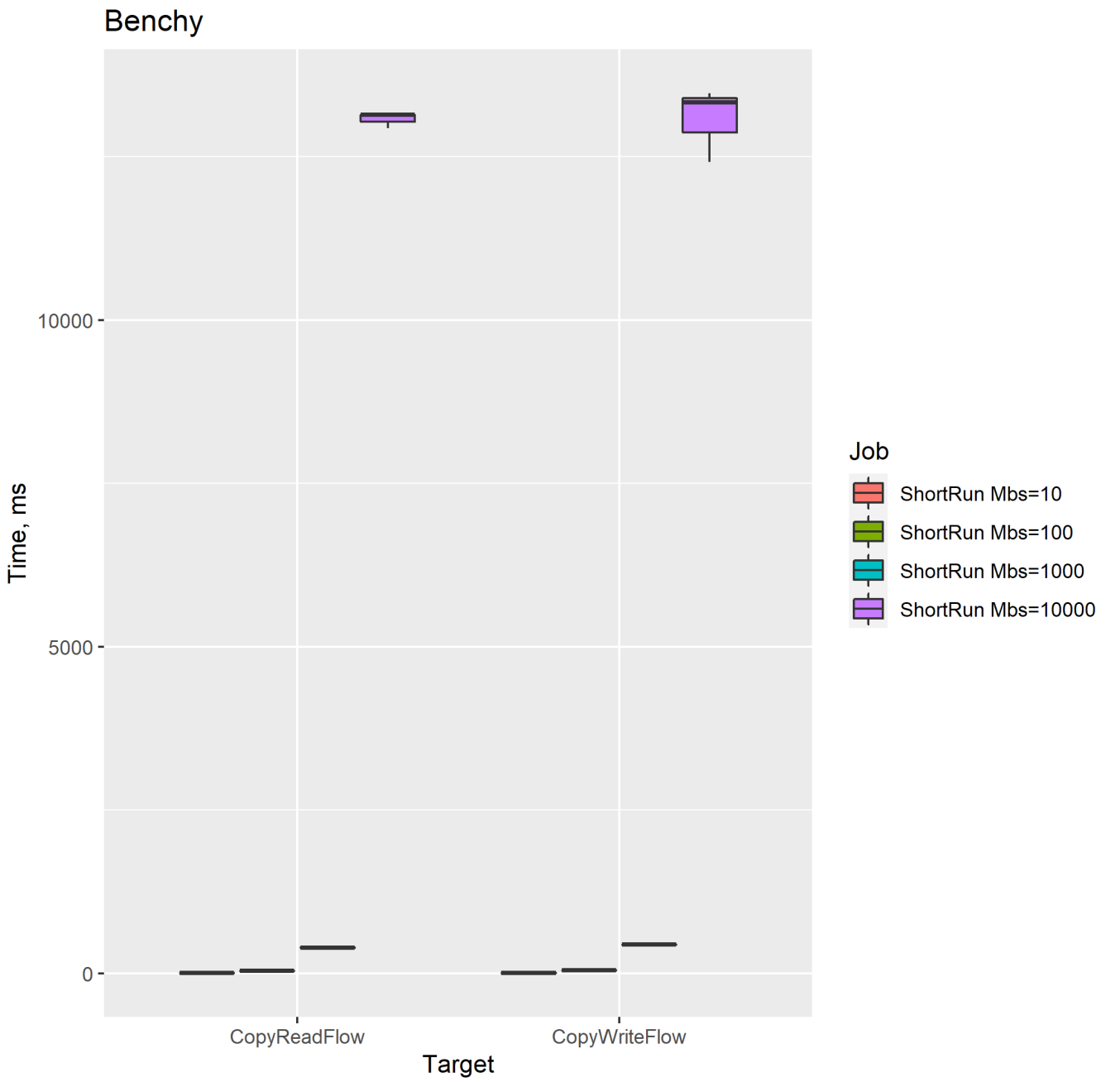
### 1. Запуск шага 2

Чтобы запустить программу из второго шага, необходимо иметь `.NET 5 SDK`
Команда:
`dotnet run [-c Release] -- [-t|--times <TIMES>]`
где `-c Release` - запуск с оптимизациями, `-t|--times <TIMES>` - количество копий входного файла
На вход берется файл `random-lines-n-times.txt` в месте запуска в кодировке `UTF-8`
Вывод в `big-file.txt`\

### 2. Алгоритм шага 2

1. Создать буфер
2. Открыть входной файл
3. Создать выходноый файл с нужным размером
4. Записать в буфер часть входного файла
5. Записать из буфера в выходной файл
6. Повторить шаги 4 - 5, пока входной файл не закончится
7. Повторить шаги 4 - 6, пока выходной файл не закончится

### 3. Скорсть работы приложения из шага 2




*BenchmarkDotNet v0.13.1*

| Method | Mbs | Mean | Error | StdDev | Allocated |
|---|---|---|---|---|---|
| CopyWriteFlow | 10 | 4.377 ms | 0.2053 ms | 0.0113 ms | 1 KB |
| CopyWriteFlow | 100 | 45.440 ms | 3.5592 ms | 0.1951 ms | 1 KB |
| CopyWriteFlow | 1000 | 444.213 ms | 42.5494 ms | 2.3323 ms | 1 KB |
| CopyWriteFlow | 10000 | 13,072.142 ms | 10,361.4465 ms | 567.9458 ms | 1 KB |