Summary

- Отсутствие депенденси и интерфейсов — Для простоты, хотя и в консоль можно встроить инъекции зависимостей (DI) и вообще в REST API без интерфейсов я лично почти ничего не делаю.
- Можно энвов добавить (проще будет запускать из терминала со значениями переменных, не используя правки конфига config.json), но не стал пока что.
- Общие классы и структуры можно упаковать в нугеты (повышается переиспользование кода) — модель сообщения кафки, некоторые расширения из ***Extensions.cs*** и т.д.
- Под Win лучше всего избегать деления файлов на 3 буквы — попадаются защищённые названия — для 1Gb прекрасно подходит 2-х буквенное разделение, для большого объёма (10Гб по заданию)— 4 буквы. Если в словарях будут встречаться спец. символы и т.д., то можно подумать над другой реализацией деления, к примеру, брать int значения символа и делить, к примеру, подчёркиваниями — т.е. для **ABC** будет 65_66_67.sort
- Ввёл расширения файла **.sort**, чтобы постараться избежать популярных расширений, но настраивается в конфиге. Для чего? На случай, если алгоритм не отработает, упадёт и можно было бы распознать, какие файлы создались или на случай, если будет удобно по завершению всех операций просто удалить все файлы *.sort
- 10737418240 == 10 Gb. Файл генерируется чуть больше, по последней строке без обрезания
- Кафка сваливается «Application maximum poll interval (300000ms) exceeded by ...» в теории можно настроить его, либо обрабатывать, но это костыли и так делать не стоит — можно было бы подписываться на приход уведомления и запускать процесс сортировки. Но в рамках данного ТЗ думаю, что я бы на это больше времени потратил. В коде есть место с комментарием и ссылкой на гитхаб с ишью по этому поводу.
- Пока нет тестов — хотел покрыть как минимум сортировку, вручную проверил, что работает на малых файлах. (На больших — Total Commander; notepad++ такие файлы не умеет грузить частями..) А так же в качестве контроля за то, что ничего не потеряно — в конце исполнения сортировщика выводится размер файлов в консоль.
- По бенчмаркам получается, что всё же QuickSort на Win и Linux выигрывает (хотя попадалась статья, что у кого-то QuickSort проигрывал сильно другим алгоритмам), оставил в коде разные сортировки, может будет время у меня ещё побенчмаркать на других примерах (возможно, мой слишком мал). Только переделал его из рекурсии в итеративный, дабы не упереться в пределы рекурсии вызовов. Думал, что без создания копии массива будет быстрее, но копия по скорости выигрывает.
- В коде есть ещё места, где можно оптимизировать, дойдя до низкоуровневых реализаций. В любом случае — мы упрёмся либо в стек, либо в скорость записи на SSD/HDD
- На рабочей машине вся цепочка от генерации до окончания сортировки заняла примерно полчаса. Думаю, что это не так плохо, хотя есть ещё узкие места, в которых я бы провёл исследование на тему оптимизации.

В целом, задание мне понравилось, очень не тривиальное и интересное. Остальное по коду, думаю, будет видно. Так же, почти всё решение было разработано с нуля, кроме алгоритмов сортировок — они у меня были давно из своих внутренних экспериментов и любознательности. В коде сознательно оставлены комментарии и todo с описанием узких мест.

Спасибо за внимание.