**Киевский национальный университет имени Тараса Шевченка**

**Факультет компьютерных наук и кибернетики**

**Алгоритмы и сложность**

**Лабораторный проект № 1**

**Многофазная сортировка слиянием**

**Отчет**

**Подготовил:**

студент группы К-28

Макарчук Иван Иванович

**Киев-2018**

1. **Постановка задачи**

Иногда сортируемый список оказывается настолько велик, что он не помещается целиком в память компьютера. Напомним, что хотя изучаемые нами алгоритмы имеют дело с упорядочиванием ключей, подразумевается, что эти ключи связаны с целыми записями. Во многих случаях длина записи значительно превышает длину ключа. Иногда длина записи очень велика и перестановка двух записей занимает столько времени, что анализ эффективности алгоритма должен учитывать как число сравнений, так и число обменов.

Иногда допустимо объявить массив, размер которого достаточен для размещения всех требуемых данных, хотя размеры этого массива и значительно превышают доступный компьютеру объем памяти. Тогда операционная система пользуется виртуальной памятью, и следует учитывать эффективность ее использования. Однако даже и в этом случае объем обменов между оперативной памятью и дисками может быть значительным. Даже в эффективных алгоритмах сортировки вроде "Быстрой сортировки" соотношение между частями разбиения и длиной блока логической памяти может оказаться таким, что оно приведет к большому числу перезаписываний блоков. Нередко эта проблема не осознается, пока программа не будет реализована на компьютере, причем ее скорость оказывается настолько неудовлетворительной, что приходится применять средства технического анализа для выяснения узких мест. Даже и в этом случае анализ может оказаться безрезультатным, если операционная система не предоставляет инструментов отслеживания обменов в виртуальной памяти.

Для выяснения эффективности алгоритмов сортировки мы подсчитывали число выполняемых ими сравнений. Однако объем работы по чтению или записи на диск блоков виртуальной памяти может значительно превышать трудоемкость логических и арифметических операций. Эта работа выполняется операционной системой, и поэтому у нас нет реальных средств воздействия на ее эффективность.

При другом подходе можно воспользоваться файлами с прямым доступом и заменить непосредственные обращения к массиву операциями поиска в файле для выхода в нужную позицию с последующим чтением блока. В результате размер используемой логической памяти уменьшается, а значит уменьшается неконтролируемая нагрузка на виртуальную память. Объем операций ввода-вывода все равно остается значительным - управляем ли мы им сами или полагаемся на операционную систему.

В результате большинство алгоритмов сортировки на больших объемах данных оказываются непрактичными. Обратимся теперь к другой возможности: пусть у нас есть четыре файла и инструмент их слияния.

1. **Алгоритм выполнения**

Оценим сначала разумное число записей, которые можно хранить в оперативной памяти одновременно. Объявим массив, длина S которого равна этой величине; этот массив будет использоваться на двух этапах сортировки. На первом шаге мы прочитаем S записей и отсортируем их с помощью подходящей внутренней сортировки. Этот набор уже отсортированных записей перепишем в файл A. Затем прочитаем еще S записей, отсортируем их и перепишем в файл B. Этот процесс продолжается, причем отсортированные блоки записей пишутся попеременно то в файл A, то в файл B.

После того, как входной файл полностью разбит на отсортированные отрезки, мы готовы перейти ко второму шагу - слиянию этих отрезков. Каждый из файлов A и B содержит некоторую последовательность отсортированных отрезков, однако, как и в случае сортировки слиянием, мы ничего не можем сказать о порядке записей в двух различных отрезках.

Процесс слияния будет аналогичен функции MergeL из обычной сортировки слиянием, однако теперь вместо того, чтобы переписывать записи в новый массив, мы будем записывать их в новый файл. Поэтому мы начинаем с чтения половинок первых отрезков из файлов A и B. Читаем мы лишь по половине отрезков, поскольку мы уже выяснили, что в памяти может находиться одновременно лишь S записей, а нам нужны записи из обоих файлов. Будем теперь сливать эти половинки отрезков в один отрезок файла C. После того, как одна из половинок закончится, мы прочтем вторую половинку из того же файла. Когда обработка одного из отрезков будет завершена, конец второго отрезка будет переписан в файл C. После того, как слияние первых двух отрезков из файлов A и B будет завершено, следующие два отрезка сливаются в файл D. Этот процесс слияния отрезков продолжается с попеременной записью слитых отрезков в файлы C и D. По завершении мы получаем два файла, разбитых на отсортированные отрезки длины 2S. Затем процесс повторяется, причем отрезки читаются из файлов C и D, а слитые отрезки длины 4S записываются в файлы A и B. Ясно, что в конце концов отрезки сольются в один отсортированный список в одном из файлов.

1. **Особенности:**

Данный метод имеет следующие отличительные особенности:

* допускает использование произвольного числа N вспомогательных последовательностей;
* при слиянии используется (N – 1) вспомогательная последовательность, при этом результат записывается в свободную (N-ю) последовательность;
* слияние производится до тех пор пока все последовательности не пусты (стратегия сливать до опустошения – merge-until-empty);
* требует неравномерного распределения серий по вспомогательным последовательностям.