Задача 4. Сортировки

В некоторых вариантах данной задачи требуется реализовать сортировки, которые на лекции не были рассмотрены. В этом случае вы должны найти информацию о требуемых сортировках и разобраться самостоятельно (в Интернете по сортировкам полно различной информации, более того, в большинстве случаев есть готовые реализации сортировок, единственное, возможно не на Java, а на других языках).

Если вы берете и адаптируете под свою задачу какую-то готовую реализацию сортировки, вы обязательно должны разобраться, как данная сортировка работает, и быть в состоянии объяснить алгоритм работы преподавателю.

Если в условии задачи что-то непонятно – попросить пояснить преподавателя.

Варианты:

1. (*) Написать программу, которая иллюстрирует работу сортировки пузырьком (Bubble sort) на примере массива целых чисел. Для описания текущего состояния сортировки должен быть описан класс SortState, в котором будут храниться:
   - массив (текущее состояние на данном шаге, естественно, надо делать копию массива, т.к. исходный массив будет меняться в процессе сортировки);
   - переменная цикла i;
   - переменная вложенного цикла j;
   - возможно еще какие-то данные, необходимые для визуализации.

   В модифицированной процедуре сортировки помимо соответствующих действий на каждом шаге необходимо создать экземпляр класса SortState, соответствующий текущему шагу сортировки, и добавлять этот экземпляр в список состояний, который будет возвращаться, т.е. сортировка будет иметь следующую сигнатуру:

   ```
   List<SortState> sort(int[] arr)
   ```

   Далее необходимо реализовать наглядное отображение на форме состояния SortState (возможно с использованием JTable или нарисовать). К таймеру следует привязать обработчик событий, который будет обновлять форму (перерисовывать очередное состояние в процессе сортировки).

   Таким образом, непосредственно сортировка выполняется сразу же, а отображение на форме произошедших событий осуществляется позже с задержками, чтобы пользователь мог оценить, что происходит.

   Предусмотреть режим, когда таймер отключается и очередное состояние отображается по нажатию кнопки (причем, чтобы можно было «проигрывать» события как вперед, так и назад).

2. (*) Предыдущая задача, но для сортировки вставками (Insertion sort).

3. (*) Предыдущая задача, но для сортировки выбором (Selection sort).

4. (*) Предыдущая задача, но для гномьей сортировки (Gnome sort).

5. (*) Реализовать быструю сортировку без явного применения рекурсии с использованием объекта стека (Stack<T>) для хранения интервалов, в которых необходимо сортировать элементы.

6. Написать программу, которая наглядно иллюстрирует эффективность следующих методов сортировки:

– пузырьковая;
– шейкерная (перемешиванием).

Для этого необходимо провести сравнение этих сортировок по количеству сравнений и по количеству обменов (добавить подсчет сравнений и обменов в реализацию сортировок). Для этого построить графики зависимостей данных величин от количества элементов массива (для отображения графиков можно воспользоваться компонентом JFreeChart – http://www.jfree.org/jfreechart/, см. пример к лекции по сортировкам).

7. Написать программу, которая наглядно иллюстрирует эффективность следующих методов сортировки:
   – простыми вставками;
   – бинарными вставками.

   Для этого необходимо провести сравнение этих сортировок по количеству сравнений и по количеству обменов (добавить подсчет сравнений и обменов в реализацию сортировок). Для этого построить графики зависимостей данных величин от количества элементов массива (для отображения графиков можно воспользоваться компонентом JFreeChart – http://www.jfree.org/jfreechart/, см. пример к лекции по сортировкам).

8. Реализовать поразрядную сортировку, сравнить время работы данной сортировки с встроенной в библиотеку языка Java реализацией сортировки (Arrays.sort) для массива примитивных целых чисел (int[]).
   Для этого построить графики зависимостей времени сортировки от количества элементов массива (для отображения графиков можно воспользоваться компонентом JFreeChart – http://www.jfree.org/jfreechart/, см. пример к лекции по сортировкам).

9. Реализовать сортировка Шелла (Shell sort) и сравнить время работы данной сортировки при различных вариантах длин промежутков (https://ru.wikipedia.org/wiki/Сортировка_Шелла).
   Для этого построить графики зависимостей времени сортировки от количества элементов массива (для отображения графиков можно воспользоваться компонентом JFreeChart – http://www.jfree.org/jfreechart/, см. пример к лекции по сортировкам).

10. Реализовать программу, которая для произвольного текста (любого размера) построит список из парасочетаний букв (2 буквы, которые стоят в слове друг за другом), упорядочив их в порядке встречаемости от наиболее встречающихся комбинаций к менее встречающимся.

11. Дан набор точек (X, Y) на плоскости. Отобрать N (указывается) точек, наиболее близких к началу координат (точке (0, 0)).
    Для этого необходимо отсортировать точки по критерию близости точки в началу координат, а потом взять N первых точек.
    Воспользоваться двумя методами сортировки – пирамидальной и встроенной (Arrays.sort).

12. Дан список (List<String>) имен файлов (можно считать с какой-то реальной директории). Продемонстрировать сортировку файлов в нескольких вариантах:
    – по расширению, затем по имени файла – по возрастанию (в алфавитном порядке);
    – по расширению, затем по имени файла – по убыванию;
    – по имени файла – по возрастанию;
    – по имени файла – по убыванию.

    Воспользоваться встроенной реализацией сортировок (List.sort), задавая различные критерии сравнения 2-х имен файлов.

13. Используя реализацию Collections.binarySearch(List<T>) сравнить скорость работы данного метода для списков ArrayList<T> и LinkedList<T>.
    Для этого построить графики зависимостей времени поиска случайного элемента списка (для отображения графиков можно воспользоваться компонентом JFreeChart – http://www.jfree.org/jfreechart/, см. пример к лекции по сортировкам).
    Пояснить полученные результаты.

14. Реализовать класс с собственной реализацией двоичного поиска элементов в упорядоченном массиве и списке:

```
// возвращает индекс элемента в массиве со значением value
// или -1, если такого нет
public static int <T extends Comparable<? super T>>
        indexOf(T[] data, T value) { ... }

// возвращает наименьший индекс элемента, строго большего value
// или -1, если такого нет
public static int <T extends Comparable<? super T>>
        indexOfHigher(T[] data, T value) { ... }

// возвращает наибольший индекс элемента, строго меньшего value
// или -1, если такого нет
public static int <T extends Comparable<? super T>>
        indexOfLower(T[] data, T value) { ... }

// и такие же методы для списков (по сути копия кода
// с разницей только в способе обращения к элементам)
public static int <T extends Comparable<? super T>>
        indexOf(List<? super T> data, T value) { ... }
public static int <T extends Comparable<? super T>>
        indexOfHigher(List<? super T> data, T value) { ... }
public static int <T extends Comparable<? super T>>
        indexOfHigher(List<? super T> data, T value) { ... }
```

    Продемострировать работоспособность вашего кода на примере массива чисел, списка строк и списка товаров (товар в данном примере – класс с названием и ценой).

15. Дан массив/список целых чисел размера n, где все числа лежат в диапазоне от 0 до m-1 (m намного меньше, чем n). Реализовать специальную сортировку, которая упорядочит данный массив за O(n + m) (т.к. m много меньше m, то фактически за O(n)).
    Подсказка: Завести новый массив размера m, в нем подсчитать, сколько каждое значение встречается в исходном массиве (в созданном массиве индекс принимаем в качестве значения). Потом в исходный массив просто выпишем по порядку, сколько раз какое значение встречалось.

16. (*) Дан массив/список чисел размера N. Разрешается выполнять только операции обмена двух элементов данного массива. Найти такую последовательность обменов, которая отсортирует числа в исходном массиве/списке. Кол-во обменов должно получиться не более N, сложность алгоритма - O(n*log(n)).
    Подсказка: составляем массив из элементов (значение, первоначальная позиция, конечная позиция), первые два поля заполняем, сортируем по значению. Затем заполняем конченую позицию и сортируем второй раз по начальной позиции. Получили массив, для каждого элемента которого известно, где он должен оказаться, далее - очевидно.

17. Реализовать сортировку расческой. Применить данную сортировку для сортировки строк по следующему критерию:

- вначале по кол-ву гласных в строке (для английского алфавита), впереди должны оказаться строки с большим кол-вом гласных;
- затем по длине строки, впереди более длинные слова.

18. (*) Реализовать поразрядную сортировку для массива целых (int) чисел. В качестве разряда использовать байты, их которых состоят целые числа. Придумать механизм, чтобы сортировка корректно работала и с положительными и отрицательными числами. Применить даннаю сортировку для сортировки целых чисел (должны быть как положительные, так и отрицательные числа). Проверить корректность работы на случайных примерах с большим кол-вом элементов (> 100000). Сравнить эффективность со встроенной реализацией сортировки (Arrays.sort).

19. (*) Реализовать плавную сортировку (SmoothSort). Сигнатура метода должна быть:
```
public static <T extends Comparable<T>> void sort(T[] data)
```
Ссылки: http://cppalgo.blogspot.com/2010/10/smoothsort.html
Если найдете реализацию на Java – обязательно необходимо разобраться, как сортировка работает и модифицировать, чтобы можно было сортировать любые сравнимые типы (см. сигнатуру метода).

20. (*) Предыдущая задача (со всеми требованиями) но для интроспективной сортировки (Introsort): https://ru.wikipedia.org/wiki/Introsort

21. Дано множество отрезков (a_i, b_i) на прямой. Найти список интервалов этой прямой, которые покрываются одновременно наибольшим кол-вом заданных отрезков. Решение должно иметь сложность O(n*log(n)).
Подсказка: надо составить список из границ отрезков (класс, содержащий значение и признак того, является ли данное значение началом или концом отрезка). Затем этот список упорядочить (вначале по значению, при одинаковом значении концы отрезков должно оказаться раньше начал). Далее проходим по отсортированному списку, начало интервала - +1 к кол-ву интервалов, окончание - -1. Как только кол-во интервалов становится больше уже найденного, сбрасываем результирующий список интервалов и начинаем формировать заново.
Сортировка - O(n*log(n)), проход по списку - O(n), в итоге O(n*log(n) + n) = O(n*log(n)).

22. Модифицировать алгоритм пирамидальной (HeapSort) таким образом, чтобы можно было сортировать элементы массива из заданного диапазона элементов (от from до to, где from – индекс первого элемента диапазона, to – индекс элемента, следующего за последним элементом диапазона. Сигнатура метода должна быть:
```
public static <T extends Comparable<T>> void sort(T[] data, int from, int to)
```

23. Предыдущая задача, но для пузырьковой сортировки.

24. Предыдущая задача, но для сортировки вставками.

25. Предыдущая задача, но для сортировки выбором.

26. Реализовать метод пузырьковой сортировки в варианте, когда в метод передается два массива: первый массив – массив с объектами, которые необходимо отсортировать, а второй массив – массив целых чисел, согласно которым необходимо отсортировать первый массив. Сигнатура метода должна быть:
```
public static <T> void sort(T[] data, int[] orderValues)
```
Сортировка должна работать следующим образом:
{ "green", "blue", "red" }, { 5, 8, 1 } –> { "red", "green", "blue" }, { 1, 5, 8 }

Очевидно, что все сравнения надо выполнять над числами из второго массива, а любые обмены делать одновременно в двух массивах.

27. Предыдущая задача, но для сортировки вставками.

28. Предыдущая задача, но для сортировки выбором.

29. Предыдущая задача, но для быстрой сортировки (QuickSort).

30. Предыдущая задача, но для пирамидальной сортировки (HeapSort).

31. Отсортировать студентов так, чтобы сначала сортировка шла по курсу, затем по группе и только потом по фамилии (для этого необходимо соответствующим образом реализовать интерфейс Comparable<Student> для класса Student).

32. Представим себе массив, в котором некоторые элементы запрещено менять. Необходимо модифицировать метод пузырьковой сортировки таким образом, чтобы он отсортировал по возрастанию все оставшиеся (которые можно менять) элементы массива. Например (красным отмечены цифры, которые нельзя менять):
{ 7, 10, 3, 8, 7, 2, 1, 9, 5, 7 } –> { 1, 10, 3, 7, 7, 2, 7, 8, 5, 9 }
В метод сортировки должно передаваться два массива, один массив – с числами, второй массив такого же размера – с логическими значениями, указывающими на элементы позицию которых нельзя менять, сигнатура метода:
```
public static <T extends Comparable<T>> void sort(T[] data, boolean[] fixed)
```
Дополнительные массивы использовать запрещено.

33. Предыдущая задача, но модифицировать нужно алгоритм сортировки вставками.

34. Предыдущая задача, но модифицировать нужно алгоритм сортировки выбором.

35. (*) Предыдущая задача, но модифицировать нужно алгоритм быстрой сортировки.

36.