Сложные сортировки

Составитель: Рощупкин Александр

Алгоритмы сортировки

* **Сложные методы сортировки**

* Сортировка методом Шелла
* Быстрая сортировка
* Сортировка слиянием

Сортировка Шелла

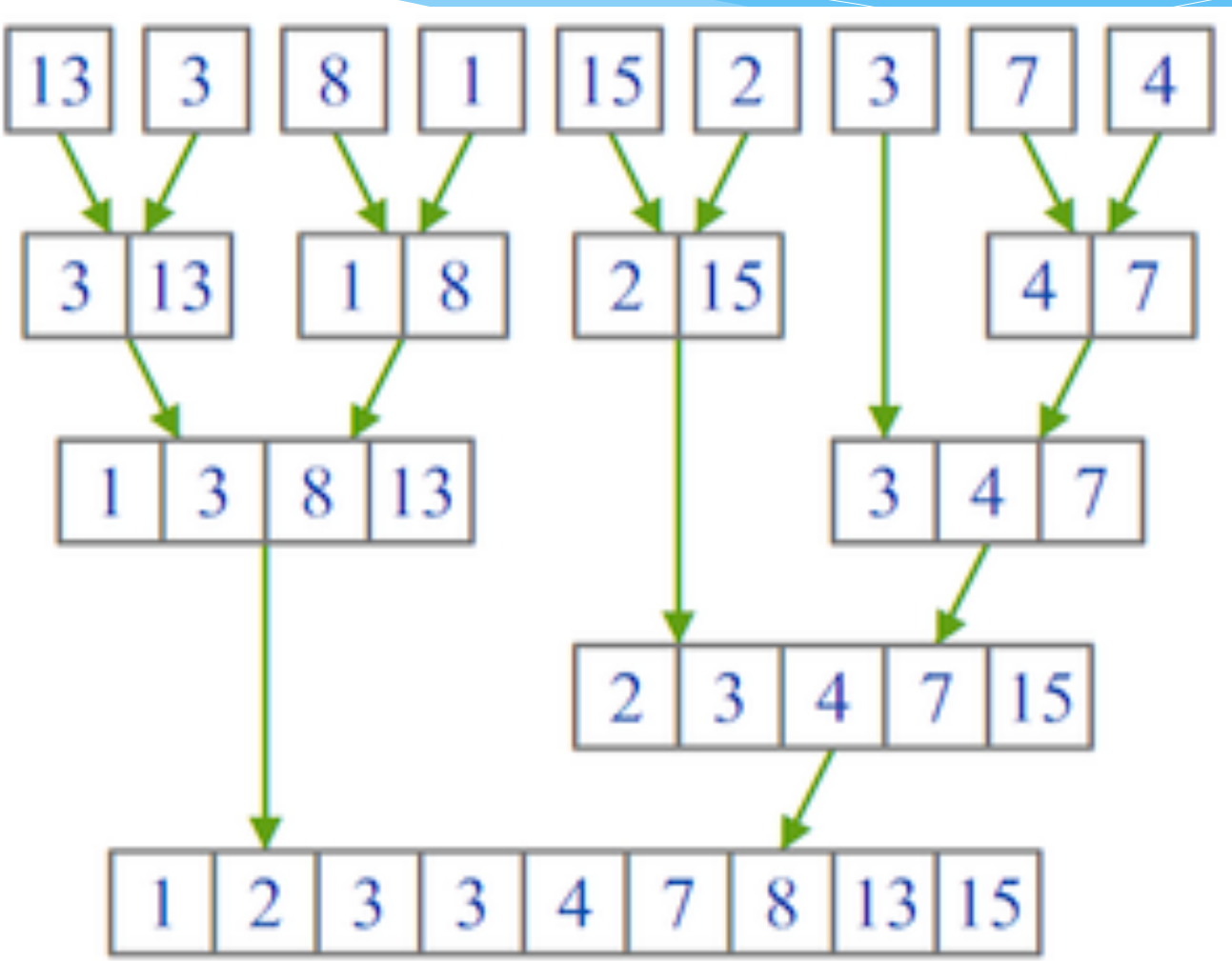
* **Сортировка Шелла** — алгоритм сортировки, являющийся усовершенствованным вариантом сортировки вставками
* Идея метода Шелла состоит в сравнении элементов, стоящих не только рядом, но и на определённом расстоянии друг от друга
* Размер шага выбирается делением исходного массива на 2 и затем уменьшается делением на 2 на каждом проходе

```java
public void shellSort(int[] arr, int num) {
    // Начинаем с самого большого растояния и уменьшаем его на каждом шаге
    for (int gap = num / 2; gap > 0; gap = gap / 2) {
        // выполняем сортировку вставками с заданным шагом
        // начинаем формировать отсортированную часть
        // i - граница отсортированной части
        for (int i = gap; i < num; i++) {
            // цикл просеивания элементов внутри отсортированной части
            // после захвата элемента из неотсортированной части
            for (int j = i - gap; j >= 0; j = j - gap) {
                if (arr[j + gap] >= arr[j])
                    break;
                else {
                    int tmp = arr[j];
                    arr[j] = arr[j + gap];
                    arr[j + gap] = tmp;
                }
            }
        }
    }
}
```

Сортировка слиянимем

* Алгоритм рекурсивно делит массив пополам до тех пор, пока не размер каждого массива не будет равен 1
* А массив из одного элемента считается упорядоченным и после этого начинается процедура слияния упорядоченных массивов

Пример рекурсивного алгоритма



Пример слияния

```
private static void merge(int[] subset, int low, int mid, int high) {
    // длинна сортируемого подмассива
    int n = high-low+1;
    // временный массив в который переносятся элементы в сортированном порядке
    int[] temp = new int[n];

    // i и j – индексы для массивов a1 и a2 соответственно, которые указывают на
    // текущие элементы на каждом шаге и образуют тот самый буфер
    int i = low, j = mid + 1;
    int k = 0;
    // перебираем до тех пор пока не дойдём до концов обоих массивов
    while (i <= mid || j <= high) {
        // вышли за границу первого: переносим остаток второго массива
        if (i > mid)
            temp[k++] = subset[j++];
        // вышли за границу второго: переносим остаток первого массива
        else if (j > high)
            temp[k++] = subset[i++];
        // если элемент первого массива меньше то переносим их
        else if (subset[i] < subset[j])
            temp[k++] = subset[i++];
        // если элемент второго массива меньше то переносим их
        else
            temp[k++] = subset[j++];
    }
    // переносим элементы из отсортированного временного массива в изначальный
    for (j = 0; j < n; j++) {
        subset[low + j] = temp[j];
    }
}
```

Сортировка слиянием

```
public static void mergeSort(int[] elements, int low, int high) {
    if (low < high) {
        int mid = (low + high) / 2;
        mergeSort(elements, low, mid); // рекурсивно сортируем первую половину
        mergeSort(elements, mid + 1, high); // рекурсивно сортируем вторую половину
        merge(elements, low, mid, high); // объединяем две отсортированные части
    }
}
```

Быстрая сортировка

* Выбрать из массива элемент, называемый опорным. Это может быть любой из элементов массива, обычно средний
* Сравнить все остальные элементы с опорным и переставить их в массиве так, чтобы разбить массив на три непрерывных отрезка, следующих друг за другом: «элементы меньшие опорного», «равные» и «большие»
* Для отрезков «меньших» и «больших» значений выполнить рекурсивно ту же последовательность операций, если длина отрезка больше единицы

Частичное упорядочивание

```
private static void partialSort(int[] vector) {
    // значение опорного элемента равно среднему элементу
    int pivot = vector[vector.length / 2];
    int left = 0;
    int right = vector.length - 1;
    while (left < right) {
        // ищем кандидата на обмен слева
        // выбираем элемент больший или равный опорному
        // пропускаем эелемент меньший
        while (vector[left] < pivot) {
            left++;
        }
        // ищем кандидата на обмен справа
        // выбираем элемент меньший опорного
        while (vector[right] >= pivot) {
            right--;
        }
        // меняем кандидатов если они есть с обоих сторон
        if (left < right) {
            int tmp = vector[left];
            vector[left] = vector[right];
            vector[right] = tmp;
        }
    }
}
```

Недостатки быстрой сортировки

* Временная сложность алгоритма быстрой сортировки в лучшем случае и в среднем случае составляет O(n log n).
* В некоторых случаях (например, на малых массивах) алгоритм быстрой сортировки обладает не лучшей производительностью, деградирует до квадратичной сложности.
* Поэтому имеет смысл вместо него применять другие алгоритмы, которые в общем случае проигрывают, но в конкретных случаях могут дать выигрыш

Стандартная сортировка

* Стандартная сортировка в Java для массивов примитивов – быстрая сортировка если элементы «хорошо упорядочены», иначе используется сортировка слиянием
* Представлена классом DualPivotQuickSort
* Т.к. алгоритм быстрой сортировки на малых массивах обладает плохой производительностью, то на массивах, размером до 47 эффективнее использовать сортировку вставками и подсчётом
* Для массивов объектов, стандартной сортировкой является сортировка слиянием
* Сортировка списков преобразует в массив и затем сортирует

Лабораторная работа

* Сравнить быстродействие сортировок
  * Быстрой
  * Слиянием
  * Выбором
  * Вставками
  * Подсчётом
* Для разного количества элементов: 10,100,1000
* Для разного набора данных: отсортированный в обратном направлении, частично отсортированный, случайные данные
* Замеры делать с помощью засечек времени System.currentTimeMillis()