Быстрая сортировка

[1]

Рассмотрим реализацию одной из самых популярных сортировок на примере языка Java.

⟨*sort.java*⟩≡
```
public static void main(String[] args) {
        ⟨Исходный массив⟩
        ⟨Вызов функции сортировки⟩
        ⟨Вывод результата⟩
}
```

Исходный массив, который необходимо отсортировать.

⟨*Исходный массив*⟩≡
```
int[] testArray = {32, -3, 99, 71, 1, -5, 0, 8, 44};
```

Вызов функции.

⟨*Вызов функции сортировки*⟩≡
```
quickSort(0, testArray.length - 1, testArray);
```

Реализация функции сортировки.

⟨*sort.java*⟩+≡
```
public static void quickSort(int begin, int end, int[] array) {
        ⟨Проверка⟩
        ⟨Опорный элемент⟩
        ⟨Создание переменных⟩
        ⟨Цикл⟩

        ⟨Сортировка слева⟩
        ⟨Сортировка справа⟩
}
```

Проверка на длину исходного массива и на совпадение левой и правой его границы.

⟨*Проверка*⟩≡
```
if (array.length == 0 || end <= begin) {
        return;
}
```

Опорный элемент для быстрой сортировки может быть любым. В данном примере он находится в середине массива для того, чтобы увеличить скорость сортировки.

⟨*Опорный элемент*⟩≡
```
int pivot = array[(begin + end) / 2];
```

[1] Разработал Акимов В.Е.

Создадим переменные для прохождения по элементам массива.

⟨*Создание переменных*⟩≡
```
int b = begin, e = end;
```

Суть алгоритма быстрой сортировки заключается в разбиении массива на две части по опорному элементу и перемещении элементов, меньших опорному справой стороны на левую, а больших с левой на правую (при сортировки по возрастанию). Для этого необходим цикл, заданный ниже.

⟨*Цикл*⟩≡
```
while (b <= e) {
        ⟨Нахождение индекса большего элемента⟩
        ⟨Нахождение индекса меньшего элемента⟩
        ⟨Обмен⟩
}
```

Изначально необходимо найти элемент (его индекс), который больше опорного для переноса его с левой части на правую.

⟨*Нахождение индекса большего элемента*⟩≡
```
while (array[b] < pivot) {
        b++;
}
```

Аналогично для элемента, меньшего опорного.

⟨*Нахождение индекса меньшего элемента*⟩≡
```
while (array[e] > pivot) {
        e--;
}
```

После нахождения этих индексов необходимо поменять местами эти элементы и изменить переменные для дальнейшего прохода по циклу.

⟨*Обмен*⟩≡
```
if (b <= e) {
        int temp = array[b];
        array[b++] = array[e];
        array[e--] = temp;
}
```

Если переменная e будет больше, чем начальная позиция массива, что говорит о том, что с левой стороны относительно опорного элемента имеются элементы, то необходимо рекурсивно вызвать функцию, передав необходимые параметры.

⟨*Сортировка слева*⟩≡
```
if (begin < e) {
        quickSort(begin, e, array);
}
```

Аналогичная проверка для переменной b относительно конечной позиции массива и вызов функции рекурсивно.

⟨*Сортировка справа*⟩≡
```
if (end > b) {
        quickSort(b, end, array);
}
```

Для вывода отсортированного массива воспользуемся `Stream API`.

⟨*Вывод результата*⟩≡
```
Arrays.stream(testArray).forEach(el -> System.out.print(el + " "));
```