Сортировка вставками

Слёлин А.В.

17 мая, 2024 г.

Описание алгоритма

Пусть дан массив $A[0..n-1]$, такой что $n = \text{length}[A]$, причём индексация начинается с нуля для облегчения понимания кода.

Рассмотрим случай когда нам нужно отсортировать массив A в неубывающую последовательность; случай, когда нужно отсортировать в невозрастающую последовательность аналогичен.

Сортировка вставками подобна сортировке карт при игре, например, в «Дурака». Вы получаете 6 карт и хотите чтобы эти карты были отсортированы слева направо. Для этого вы с начала берёте карту и ищете для неё место в колоде слева от данной карты. Когда мы найдём такое место, то просто «вставляем» туда данную карту, при этом все карты справа от данной карты сдвигаются в право.

Если мы начнём делать вставку с начала для каждой карты, и будем двигаться в конец, то получим полностью отсортированную последовательность.

Пример

Пусть массив $A = [8, 3, 12, 16, 10, 4, 1, 9, 7, 14]$. i - означает индекс элемента, для которого мы ищем место для вставки, j будет обозначать индекс массива A, который определяет куда будем вставлять элемент с индексом i. (красным цветом обозначается элемент, который мы вставили, а синим цветом все после этого сдвигаемые элементы).

i = 0:

$$A = [8, 3, 12, 16, 10, 4, 1, 9, 7, 14];$$

Индекс элемента, куда вставлять $j = 0$

$$A = [8, 3, 12, 16, 10, 4, 1, 9, 7, 14];$$

$$A = [8, 3, 12, 16, 10, 4, 1, 9, 7, 14];$$

i = 1:

$$A = [8, 3, 12, 16, 10, 4, 1, 9, 7, 14];$$

Индекс элемента, куда вставлять $j = 0$

$$A = [8, 3, 12, 16, 10, 4, 1, 9, 7, 14];$$

$$A = [3, 8, 12, 16, 10, 4, 1, 9, 7, 14];$$

i = 2:

$$A = [3, 8, 12, 16, 10, 4, 1, 9, 7, 14];$$

Индекс элемента, куда вставлять $j = 2$

$$A = [3, 8, 12, 16, 10, 4, 1, 9, 7, 14];$$

$$A = [3, 8, 12, 16, 10, 4, 1, 9, 7, 14];$$

i = 3:

$$A = [3, 8, 12, 16, 10, 4, 1, 9, 7, 14];$$

Индекс элемента, куда вставлять $j = 3$

$$A = [3, 8, 12, 16, 10, 4, 1, 9, 7, 14];$$

$$A = [3, 8, 12, 16, 10, 4, 1, 9, 7, 14];$$

i = 4:

$$A = [3, 8, 12, 16, 10, 4, 1, 9, 7, 14];$$

Индекс элемента, куда вставлять $j = 2$

$$A = [3, 8, 12, 16, 10, 4, 1, 9, 7, 14];$$

$$A = [3, 8, 10, 12, 16, 4, 1, 9, 7, 14];$$

i = 5:

$$A = [3, 8, 10, 12, 16, 4, 1, 9, 7, 14];$$

Индекс элемента, куда вставлять $j = 1$

$$A = [3, 8, 10, 12, 16, 4, 1, 9, 7, 14];$$

$$A = [3, 4, 8, 10, 12, 16, 1, 9, 7, 14];$$

i = 6:

$$A = [3, 4, 8, 10, 12, 16, 1, 9, 7, 14];$$

Индекс элемента, куда вставлять $j = 0$

$$A = [3, 4, 8, 10, 12, 16, 1, 9, 7, 14];$$

$$A = [1, 3, 4, 8, 10, 12, 16, 9, 7, 14];$$

i = 7:

$$A = [1, 3, 4, 8, 10, 12, 16, 9, 7, 14];$$

Индекс элемента, куда вставлять $j = 4$

$$A = [1, 3, 4, 8, 10, 12, 16, 9, 7, 14];$$

$$A = [1, 3, 4, 8, 9, 10, 12, 16, 7, 14];$$

i = 8:

$$A = [1, 3, 4, 8, 9, 10, 12, 16, 7, 14];$$

Индекс элемента, куда вставлять $j = 4$

$$A = [1, 3, 4, 8, 9, 10, 12, 16, 7, 14];$$

$$A = [1, 3, 4, 7, 8, 9, 10, 12, 16, 14];$$

i = 9:

$$A = [1, 3, 4, 7, 8, 9, 10, 12, 16, 14];$$

Индекс элемента, куда вставлять $j = 8$

$$A = [1, 3, 4, 7, 8, 9, 10, 12, 16, 14];$$

$$A = [1, 3, 4, 7, 8, 9, 10, 12, 14, 16];$$

Массив A полностью отсортирован. Для реализации невозрастающей последовательности следует просто поменять знак сравнения.

Код сортировки

Листинг 1: insertionSort

```
1      public static void insertionSort(int[] A, boolean reverse){
2          for (int i = 1; i < A.length; ++i) {
3              int j = i - 1, key = A[i];
4              while (j >= 0 && (reverse ? A[j] < key : A[j] > key)
                   )
5                  A[j + 1] = A[j--];
6
7              A[++j] = key;
8          }
9      }
```

Исходный код

Представим весь код Main.java для тестирования алгоритма.

Листинг 2: Main

```
1  public class Main {
2      public static void print(int[] A){
3          for(int arg : A){
4              System.out.print(arg + " ");
5          }
6          System.out.println();
7      }
8
9      public static void insertionSort(int[] A, boolean reverse){
10         for (int i = 1; i < A.length; ++i) {
11             int j = i - 1, key = A[i];
```

```
12              while (j >= 0 && (reverse ? A[j] < key : A[j] > key)
                    )
13                  A[j + 1] = A[j--];
14
15              A[++j] = key;
16          }
17      }
18
19      public static void main(String[] args) {
20          int[] A = {8, 3, 12, 16, 10, 4, 1, 9, 7, 14};
21          boolean reverse = false;
22
23          System.out.println("Before:");
24          print(A);
25
26          insertionSort(A, reverse);
27
28          System.out.println("After:");
29          print(A);
30      }
31  }
```

Сложность

Во-первых, заметим, что мы не используем дополнительную память, не считая некоторых переменных, поэтому сразу можно сказать, что пространственная сложность сортировки вставками состовляет $O(1)$.

Рассмотрим лучший случай - на входе у нас был сразу отсортированный масив. В таком случае нам придётся выполнить цикл for, но для каждой итерации цикл while не будет выполняться. То есть один раз мы прошлись линейно по массиву. А это значит, что временная сложность сортировки вставками в лучшем случае равна $\Omega(n)$.

Рассмотрим средний случай - на входе обычный массив (несортированный). В этом случае будет работать и цикл for, и цикл while. И там, и там, мы будем проходиться линейно по массиву, следовательно временная сложность сортировки вставками в среднем случае равна $\Theta(n^2)$.

Рассмотрим худший случай - на входе отсортированный в обратную сторону массив. В таком случае нам придётся честно проделать два цикла, поэтому временная сложность сортировки вставками в худшем случае равна $O(n^2)$. Коэффициенты, скрывающиеся за О-большим больше, чем в среднем случае.

Комментарии

Соритровка вставками достаточно проста в понимании и не использует дополнительной памяти, но работает, как видно из таблицы, очень медленно для

Временная сложность			Пространственная сложность
Худший	Средний	Лучший	Худший
$O(n^2)$	$\Theta(n^2)$	$\Omega(n)$	$O(1)$

Таблица 1: **Резюме**

алгоритмов сортировки, поэтому стоит сказать, что данный алгоритм можно использовать только для учебных целей.

Также стоит заметить, что сортировка вставками полностью идентична по асимптотике пузырьковой сортировке.