

1. Оценка сложности алгоритмов.

Таблица:

обозначение	Интуитивное объяснение	Определение
$f(n) \in O(g(n))$ (о большое)	f ограничена сверху функцией g (с точностью до постоянного множителя) асимптотически.	$\exists(C > 0), U : \forall(n \in U)  f(n)  \leq C g(n) $
$f(n) \in \Omega(g(n))$ (омега)	f ограничена снизу функцией g (с точностью до постоянного множителя) асимптотически.	$\exists(C > 0), U : \forall(n \in U) C g(n)  \leq  f(n) $
$f(n) \in \Theta(g(n))$ (тета)	f ограничена снизу и сверху функцией g асимптотически.	$\exists(C > 0), (C' > 0), U : \forall(n \in U) C g(n)  \leq  f(n)  \leq C' g(n) $
$f(n) \in o(g(n))$ (о малое)	g доминирует над f асимптотически	$\forall(C > 0), \exists U : \forall(n \in U)  f(n)  < C g(n) $
$f(n) \in \omega(g(n))$	f доминирует над g асимптотически	$\forall(C > 0), \exists U : \forall(n \in U) C g(n)  <  f(n) $
$f(n) \sim g(n)$	f эквивалентна g асимптотически	$\lim_{n \rightarrow n_0} \frac{f(n)}{g(n)} = 1$

Классификация:

- 1) Линейные – сложность:  $O(n)$ ;
- 2) Полиномиальные – сложность:  $O(n^m)$ , m – константа;
- 3) Экспоненциальные – сложность:  $O(t^{f(n)})$ , t – константа  $> 1$ , а f(n) – полиномиальная функция;
- 4) Суперполиномиальные – сложность:  $O(C^{f(n)})$ , где C – константа, а f(n) – функция, возрастающая быстрее постоянной, но медленнее линейной.

**Модель RAM (Random Access Machine)**

Каждое вычислительное устройство имеет свои особенности, которые могут влиять на длительность вычисления. Обычно при разработке алгоритма не берутся во внимание такие детали, как размер кэша процессора или тип многозадачности, реализуемый операционной системой. Анализ алгоритмов проводят на модели абстрактного вычислителя, называемого *машиной с произвольным доступом к памяти (RAM)*.

Модель состоит из памяти и процессора, которые работают следующим образом:

- память состоит из ячеек, каждая из которых имеет адрес и может хранить один элемент данных;
- каждое обращение к памяти занимает одну единицу времени, независимо от номера адресуемой ячейки;
- количество памяти достаточно для выполнения любого алгоритма;
- процессор выполняет любую элементарную операцию (основные логические и арифметические операции, чтение из памяти, запись в память, вызов подпрограммы и т.п.) за один временной шаг;

- циклы и функции не считаются элементарными операциями.

Несмотря на то, что такая модель далека от реального компьютера, она замечательно подходит для анализа алгоритмов. После того, как алгоритм будет реализован для конкретной ЭВМ, вы можете заняться профилированием и низкоуровневой оптимизацией, но это будет уже оптимизация кода, а не алгоритма.

**МАЖОРИРОВАНИЕ** — означает, что всегда найдется такая константа умножение которой на главный член функции, будет давать функцию которая будет всегда выше исходной функции.

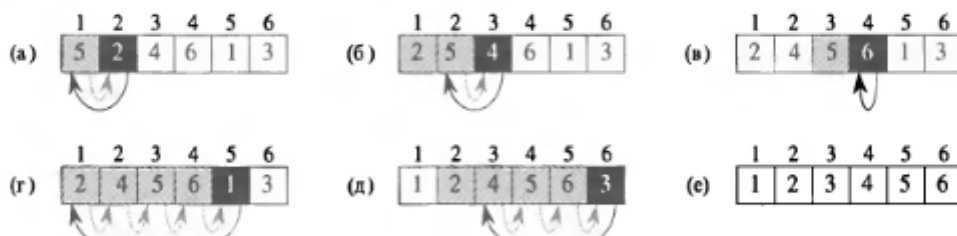
$T(n)$  — время работы алгоритма от размера входных данных.

Сортировка вставками.

### Инварианты цикла и корректность сортировки вставкой

На рис. 2.2 показано, как этот алгоритм работает с массивом  $A = \langle 5, 2, 4, 6, 1, 3 \rangle$ . Индекс  $j$  указывает “текущую карту”, которая помещается в руку. В начале каждой итерации цикла **for** с индексом  $j$  массив  $A$  состоит из двух частей. Элементы  $A[1..j-1]$  соответствуют отсортированным картам в руке, а элементы  $A[j+1..n]$  — стопке карт, которые пока что остались на столе. Заметим, что элементы  $A[1..j-1]$  изначально также находились в позициях от 1 до  $j-1$ , но в другом порядке, однако теперь они отсортированы. Назовем это свойство элементов  $A[1..j-1]$  **инвариантом цикла** (loop invariant) и сформулируем его еще раз.

В начале каждой итерации цикла **for**, состоящего из строк 1–8, подмассив  $A[1..j-1]$  состоит из элементов, которые изначально находились в  $A[1..j-1]$ , но теперь расположены в отсортированном порядке.



**Рис. 2.2.** Операции процедуры INSERTION-SORT над массивом  $A = \langle 5, 2, 4, 6, 1, 3 \rangle$ . Элементы массива обозначены квадратиками, над которыми находятся индексы, а внутри — значения соответствующих элементов. Части (а)–(д) этого рисунка соответствуют итерациям цикла **for** в строках 1–8 псевдокода. В каждой итерации черный квадратик содержит значение ключа из  $A[j]$ , которое сравнивается со значениями серых квадратиков, расположенных слева от него (строка псевдокода 5). Серыми стрелками указаны те значения массива, которые сдвигаются на одну позицию вправо (строка 6), а черной стрелкой — перемещение ключа (строка 8). В части (е) показано конечное состояние отсортированного массива.

Инварианты цикла позволяют понять, корректно ли работает алгоритм. Необходимо показать, что инварианты циклов обладают следующими тремя свойствами.

**Инициализация.** Они справедливы перед первой итерацией цикла.

**Сохранение.** Если они истинны перед очередной итерацией цикла, то остаются истинны и после нее.

**Завершение.** По завершении цикла инварианты позволяют убедиться в правильности алгоритма.

Если выполняются первые два свойства, инварианты цикла остаются истинными перед каждой очередной итерацией цикла. (Конечно, для доказательства того, что инвариант остается истинным перед каждой итерацией, можно использовать любые другие установленные факты, помимо самого инварианта.) Обратите внимание на сходство с математической индукцией, когда для доказательства определенного свойства для всех элементов упорядоченной последовательности нужно доказать его справедливость для начального элемента этой последовательности, а затем обосновать шаг индукции. В данном случае первой части доказательства соответствует обоснование того, что инвариант цикла выполняется перед первой итерацией, а второй части — доказательство того, что инвариант цикла выполняется после очередной итерации (шаг индукции).

Для наших целей третье свойство, пожалуй, самое важное, так как нам нужно с помощью инварианта цикла продемонстрировать корректность алгоритма. Обычно инвариант цикла используется вместе с условием, заставляющим цикл завершиться. Свойство завершения отличает рассматриваемый нами метод от обычной математической индукции, в которой шаг индукции используется в бесконечных последовательностях. В данном случае по окончании цикла “индукция” останавливается.

Рассмотрим, соблюдаются ли эти свойства для сортировки методом вставки.

**Инициализация.** Начнем с того, что покажем справедливость инварианта цикла перед первой итерацией, т.е. при  $j = 2$ .<sup>1</sup> Таким образом, подмассив  $A[1..j-1]$  состоит только из одного элемента  $A[1]$ , сохраняющего исходное значение. Более того, в этом подмножестве элементы рассортированы (тривиальное утверждение). Все вышесказанное подтверждает, что инвариант цикла соблюдается перед первой итерацией цикла.

**Сохранение.** Далее обоснуем второе свойство: покажем, что инвариант цикла сохраняется после каждой итерации. Выражаясь неформально, можно сказать, что в теле внешнего цикла **for** происходит сдвиг элементов  $A[j-1]$ ,  $A[j-2]$ ,

---

<sup>1</sup>Если рассматривается цикл **for**, момент времени, когда проверяется справедливость инварианта цикла перед первой итерацией, наступает сразу после начального присваивания значения индексу цикла, непосредственно перед первой проверкой в заголовочной инструкции цикла. В процедуре INSERTION-SORT это момент, когда переменной  $j$  присвоено значение 2, но еще не выполнена проверка неравенства  $j \leq A.length$ .

$A[j - 3], \dots$  на одну позицию вправо до тех пор, пока не освободится подходящее место для элемента  $A[j]$  (строки 4–7), куда и вставляется значение  $A[j]$  (строка 8). Подмассив  $A[1 \dots j]$  после этого состоит из элементов, изначально находившихся в  $A[1 \dots j]$ , но в отсортированном порядке. Следующее за этим увеличение  $j$  для следующей итерации цикла **for** сохраняет инвариант цикла.

При более формальном подходе к рассмотрению второго свойства потребовалось бы сформулировать и обосновать инвариант для внутреннего цикла **while** в строках 5–7. Однако на данном этапе мы предпочитаем не вдаваться в такие формальные подробности, поэтому будем довольствоваться неформальным анализом, чтобы показать, что для внешнего цикла соблюдается второе свойство инварианта цикла.

**Завершение.** Наконец посмотрим, что происходит по завершении работы цикла. Условие, приводящее к завершению цикла **for**, —  $j > A.length = n$ . Поскольку каждая итерация цикла увеличивает  $j$  на 1, мы должны в этот момент иметь  $j = n + 1$ . Подставив в формулировку инварианта цикла вместо  $j$  значение  $n + 1$ , получим, что подмассив  $A[1 \dots n]$  состоит из элементов, изначально находившихся в  $A[1 \dots n]$ , но расположенных в отсортированном порядке. Заметим, что подмассив  $A[1 \dots n]$  и есть сам массив  $A$ , так что весь массив отсортирован, а следовательно, алгоритм корректен.


Метод инвариантов циклов будет применяться далее в данной главе, а также в последующих главах книги.

**Инвариант цикла** — в программировании — логическое выражение, истинное после каждого прохода тела **цикла** (после выполнения фиксированного оператора) и перед началом выполнения **цикла**, зависящее от переменных, изменяющихся в теле **цикла**. **Инварианты** используются в теории верификации программ для доказательства правильности результата, полученного циклическим алгоритмом.

(то что нашел в лекции)

Инвариант цикла – истинное утверждение, которое сохраняется перед и после каждой итерации цикла алгоритма.

4. Теорема о связи.  $\Theta, \Omega, O$

$$T(n) = \Theta(g(n)) \Leftrightarrow \begin{cases} T(n) = O(g(n)) \\ T(n) = \Omega(g(n)) \end{cases}$$


## Элементарные структуры данных.

Динамическое множество – поддерживает операции добавления и удаления элементов

### 1. Стек

LIFO – last in, first out: последний добавленный при удалении будет первым.

Операции:

- 1) Проверка на пустоту
- 2) Добавление элемента
- 3) Снять элемент со стека

### 2. Очередь

FIFO – first in, first out: первый добавленный при удалении будет первым

Операции:

- 1) Добавление элемента в очередь
- 2) Извлечь элемент из очереди

### 3. Связный список

Динамическое множество, элементы которого расположены в линейном порядке, определенном указателями.

Поля:

key – ключевое значение

next – указатель на следующий элемент

prev – указатель на предыдущий элемент

head – указатель на первый элемент списка

Типы списков:

- 1) Односвязный список
- 2) Двусвязный список
- 3) Циклический список

Двоичный поиск в отсортированном массиве.