МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
имени Н.Э. БАУМАНА

Учебное пособие

И.М.Холин

Методические указания
по выполнению домашнего задания по курсу

«Системное программирование»

МГТУ имени Н.Э. Баумана

МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
имени Н.Э. БАУМАНА

И.М.Холин

Методические указания
по выполнению домашнего задания по курсу

«Системное программирование»

Москва
МГТУ имени Н.Э. Баумана

2012

УДК 681.3.06(075.8)
ББК 32.973-018
И201

И.М.Холин
Методические указания по выполнению домашнего задания по курсу «Системное программирование» / под ред. А.Е.Аверьянихина –
М.: МГТУ им. Н.Э. Баумана, 2012. – 10 с.: ил.

В методических указаниях рассмотрены основные этапы выполнения домашнего задания по курсу «Системное программирования»

Ил. 39. Табл. 5. Библиогр. 7 назв.

УДК 681.3.06(075.8)

АННОТАЦИЯ

В методических указаниях по выполнению домашнего задания будут рассмотрены основные темы курса «Системное программирование» необходимые для выполнения домашнего задания: основы теории алгоритмов, языка С.

Оглавление

Введение

Цель работы – ознакомление с основами анализа сложности алгоритмов, получение навыков ведения практических и теоретических исследований.

Решаемые задачи:

1) разработка алгоритма сортировки (блок-схема, псевдокод, программа на языке С);

2) теоретическая оценка вычислительной сложности алгоритма;

3) практическое исследование вычислительной сложности алгоритма;

4) сравнительный анализ исследований, выводы.

1 Разработка алгоритма программы

1.1 Описание алгоритма

Сортировка вставками — простой алгоритм сортировки. Хотя этот алгоритм сортировки уступает в эффективности более сложным (таким как быстрая сортировка), у него есть ряд преимуществ:

- эффективен на небольших наборах данных, на наборах данных до десятков элементов может оказаться лучшим;
- эффективен на наборах данных, которые уже частично отсортированы;
- это устойчивый алгоритм сортировки (не меняет порядок элементов, которые уже отсортированы);
- может сортировать список по мере его получения;

Минусом же является высокая сложность алгоритма: $O\left(n^2\right)$.

На каждом шаге алгоритма мы выбираем один из элементов входных данных и вставляем его на нужную позицию в уже отсортированном списке, до тех пор, пока набор входных данных не будет исчерпан. Метод выбора очередного элемента из исходного массива произволен; может использоваться практически любой алгоритм выбора. Обычно (и с целью получения устойчивого алгоритма сортировки), элементы вставляются по порядку их появления во входном массиве. Иллюстрация алгоритма представлена на рис. 1.1.

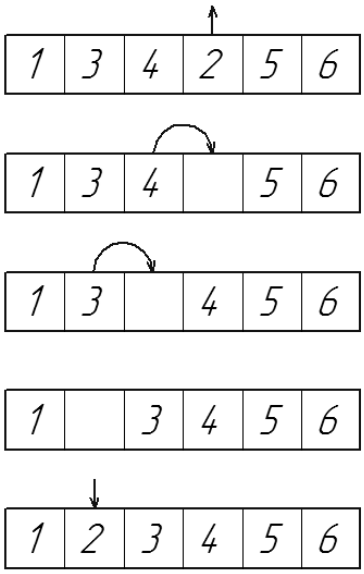

Рис. 1.1 – Иллюстрация алгоритма сортировки массива вставками

1.2 Блок-схема алгоритма

Разработанная блок-схема алгоритма сортировки массива вставками представлена на рис. 1.2.

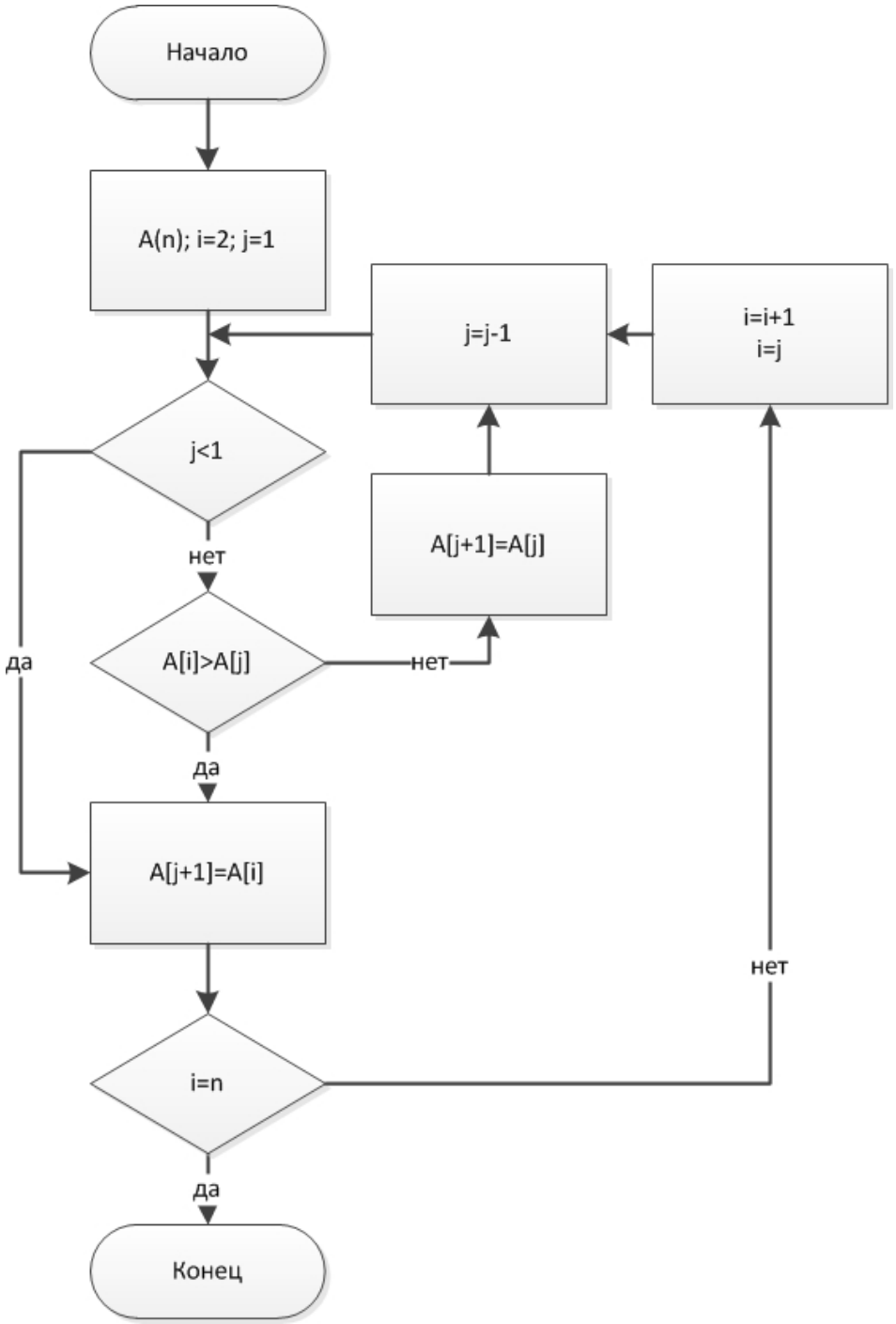

Рис. 1.2 – Блок-схема алгоритма сортировки массива вставками

1.3 Псевдокод алгоритма

Далее представлен псевдокод алгоритма сортировки массива вставками.

```
for i = 2, 3, ..., n:
    key := A[i]
    j := i - 1
    while j >= 1 and A[j] > key:
        A[j + 1] := A[j]
        j := j - 1
    A[j + 1] := key
```

2. Теоретическая оценка сложности алгоритма

Временная оценка каждого шага алгоритма представлена в табл. 2.1.

Табл. 2.1 – Временная оценка каждого шага алгоритма

Операция	Кол-во повторений T	Время выполнения C
`for i = 2, 3, ..., n:`	n	C_1
`  key := A[i]`	$n-1$	C_2
`  j := i - 1`	$n-1$	C_2
`  while j >= 1 and A[j] > key:`	$\sum_{2}^{n} t$	C_3
`    A[j + 1] := A[j]`	$\sum_{2}^{n} (t-1)$	C_4
`    j := j - 1`	$\sum_{2}^{n} (t-1)$	C_5
`  A[j + 1] := key`	$n-1$	C_6

По данным из табл. 2.1 рассчитаем сложность алгоритма сортировки вставками:

$$T(n) = C_1 n + C_2(n-1) + C_2(n-1) + C_3\sum_{2}^{n} t + C_4\sum_{2}^{n}(t-1) + C_5\sum_{2}^{n}(t-1) + C_6(n-1)$$

В общем случае t - неизвестно, но для худшего случая (массив отсортирован по убыванию):

$$t = \frac{n(n+1)}{2} - 1.$$

Таким образом, при $n \to \infty$ можно утверждать, что сложность алгоритма, действительно, будет иметь сложность $O\left(n^2\right)$, как уже было сказано в п.1.1.

График квадратичной сложности алгоритма приведен на рис. 1.3.

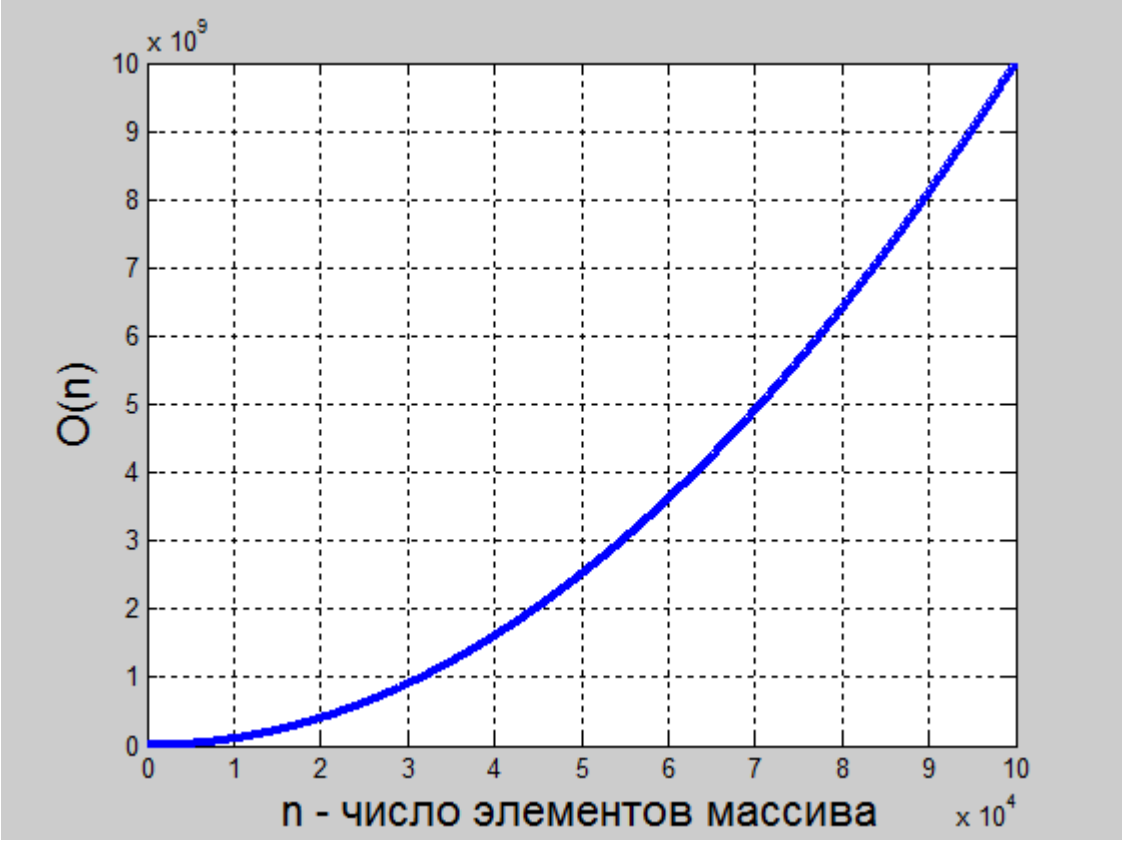

Рис. 2.1 – Рост сложности алгоритма в зависимости от кол-ва элементов массива

3 Программа на языке С

Разработаем код программы на языке С реализующей алгоритм сортировки массива вставками. Листинг представлен ниже.

```
/* заголовочный файл стандартной библиотеки языка С */
#include <stdio.h>
/* заголовочный файл стандартной библиотеки языка программирования
С, содержащий типы и функции для работы с датой и временем */
#include <time.h>
/* основная функция main() */
int(main(){
/* объявление счетчиков */
    int i;
    int j;
/* объявление переменной, хранящей текущий элемент массива */
    int mem;
/* объявление переменной кол-ва элементов массива */
    int n;
/* вывод приветсвия */
    printf("\r\n");
    printf("Hellow!\r\n");
/* приглашение ввести кол-во элементов массива */
    printf("Insert a number of elements: ");
/* чтение кол-ва элементов с клавиатуры */
    scanf("%d", &n);
    printf("\r\n");
/* объявление массива */
    int mass[n];
/* генерация и вывод неотсортированного массива */
    printf("Unsorted massive:\r\n");
    for(i = 0; i < n; i++){
/* заполняем массив случайными числами от 0 до 999 */
        mass[i] = rand()%1000;
/* вывод текущего элемента массива */
        printf("%d", mass[i]);
        if(i != n-1){
            printf("-");
        }
    }
/*   объявление переменной начального значения счетчика и
присваевание ей текущего значения счета */
    int t_start = clock();
/* сортируем массив, берем каждый элемент, начиная со второго */
    for(i = 1; i < n; i++){
```

```
/* заносим текущий элемент в переменную памяти */
        mem = mass[i];
/* выбир элемента слева от текущего */
        j = i - 1;
/* поочередно идем по каждому элементу слева от текущего до тех пор
пока не найдем элемент меньше чем текущий или не дойдем до конца */
        while(j >= 0 && mass[j] > mem){
/* сдвиг элемента вправо */
            mass[j + 1] = mass[j];
/* декремент счетчика сдвигаемых элементов */
            j--;
        }
/* перенос текущего элемента на новое место */
        mass[j + 1] = mem;
    }
/* объявление переменной конечного значения счетчика и присваевание
ей текущего значения счета */
    int t_end = clock();
/* генерация и вывод отсортированного массива */
    printf("Unsorted massive:\r\n");
    for(i = 0; i < n; i++){
/* вывод текущего элемент массива */
        printf("%d", mass[i]);
        if(i != n-1){
            printf("-");
        }
    }
/* вывод количества затраченного на сортировку времени */
    printf("\r\n");
    printf("\r\n");
    printf("Time passed: ");
/* вычисление, перевод в секунды, и вывод затраченных на сортировку
тактов системного счетчика */
    printf("%f"((float)(t_end - t_start)) / CLOCKS_PER_SEC);
    printf(" sec.");
    printf("\r\n");
}
```

4 Практическое исследование вычислительной сложности алгоритма

Для практического исследования вычислительной сложности рассматриваемого алгоритма сортировки массива вставками в код программы был добавлен счетчик времени, которое тратится на сортировку.

Время выполнения в зависимости от кол-ва элементов сортируемого массива представлено в табл. 4.1, а график зависимости на рис. 4.1.

Табл. 4.1 – Время выполнения сортировки в зависимости от кол-ва элементов

Количество элементов массива	**Время выполнения сортировки, с**
1	0
10	0
100	0
1000	0
2000	0,01
3000	0,02
4000	0,03
5000	0,04
10000	0,18
20000	0,73
30000	1,66
40000	2,95
50000	4,69
75000	10,38
100000	18,7

Примечание: подсчет времени выполнения сортировки при n<2000 невозможен из-за того, что выполняется сортировка очень быстро и вывести столь малую величину не позволяет системная библиотек time.h.

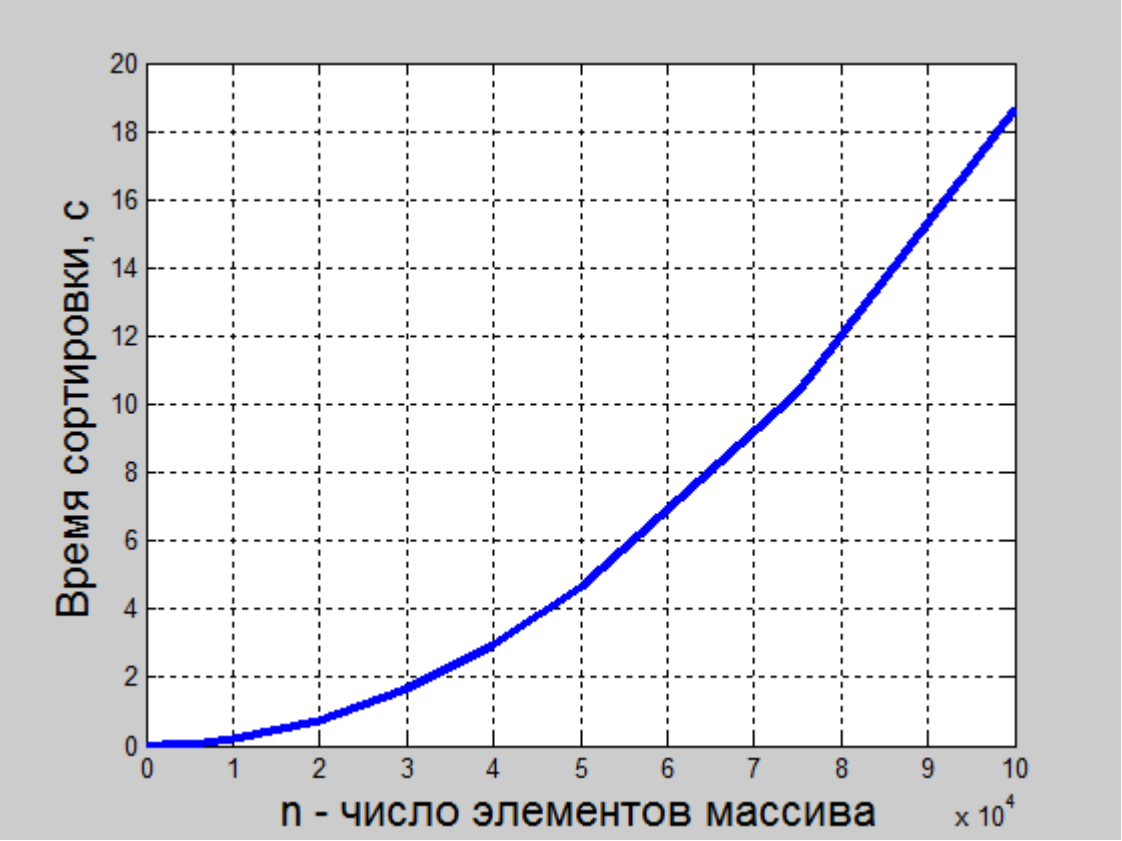

Рис. 4.1 – Рост времени сортировки в зависимости от кол-ва элементов массива

Выводы

Из рис. 2.1 и 4.1 видно, что кривая, построенная по эмпирическим данным (рис. 4.1) практически полностью повторяет параболу, построенную, исходя из теории алгоритмов (рис. 2.1). Таким образом, можно утверждать, что алгоритм сортировки массива вставками имеет сложность $O\left(n^2\right)$, что критично для большого числа элементов массива, но приемлемо при малом.