УДК 681.3.06

Маргинальные свойства сортировки массивов методом дихотомической вставки

© А.Ф. Деон, Ю.И. Терентьев

МГТУ им. Н.Э. Баумана, Москва, 105005, Россия

Выполнен сравнительный анализ маргинальных скоростных свойств сортировки массивов методами последовательной и дихотомической вставки с учетом операций сравнения, сложения, перестановки и запоминания сортируемых элементов в массивах целых чисел.

Ключевые слова: *сортировка, быстродействие, алгоритм, целые числа.*

Сортировка массива методом последовательной вставки является не самым быстродействующим способом упорядочивания массивов с произвольными значениями [1], однако, когда массив частично упорядочен, применение этого метода может дать более предпочтительные результаты. Для произвольного исходного массива он позволяет воспользоваться дихотомическим поиском в упорядоченной части массива. В этом случае метод сортировки с применением дихотомии для определения места вставки дает лучшие результаты по быстродействию. Необходимо знать точные оценки выполненных операций в различных условиях, среди которых крайние или маргинальные оценки имеют особое значение.

Сортировка методом последовательной вставки. Алгоритм возрастающей сортировки методом последовательной вставки полагает, что упорядоченная часть находится в левой части массива, а неупорядоченная — в правой. Пусть начальный элемент $a[0]$ исходного массива a является единственным элементом упорядоченного подмассива в левой части исходного массива. Это результат простого утверждения, что одноэлементный массив является упорядоченным массивом. Тогда неупорядоченная часть состоит из элементов $a[1]$, $a[2]$, …, $a[n–1]$. Запомним элемент $a[1]$ из начала неупорядоченной части в рабочей переменной $r = a[1]$. Затем осуществим вставку значения r в упорядоченную часть. Это возможно, поскольку элемент $a[1]$ теперь свободен, а его значение скопировано в r. После вставки получаются последовательности r, $a[0]$ или $a[0]$, r. Один из этих вариантов находится теперь в элементах $a[0]$, $a[1]$. После такой итерации исходный массив содержит упорядоченную часть $a[0]$, $a[1]$ и неупорядоченную — $a[2]$, $a[3]$, …, $a[n–1]$. Повторяя подобные действия для неупорядоченного элемента $a[2]$, получаем новые части

исходного массива в виде упорядоченной части $a[0]$, $a[1]$, $a[2]$ и неупорядоченной — $a[3]$, $a[4]$, …, $a[n-1]$. Продолжая итерации для начальных элементов всех неупорядоченных частей, получаем отсортированный массив a. Функция, реализующая метод последовательной вставки, имеет следующий вид:

```
void SortInsertSeq( int a[], const int n )
{
 for( int j = 1; j < n; j++ ) // цикл исходных элементов
 {
 if( a[j-1] <= a[j] ) continue; // вставка не нужна
 for( i = j-1; i >=0; i-- ) // поиск места вставки
 if( a[i] < a[j] ) break; // перед местом вставки
 i++; // место вставки
 int r = a[j]; // запомнить вставляемый элемент
 for( int m = j; m > i; m-- ) a[m] = a[m-1]; // сдвиг
 a[i] = r; // расположить вставляемый элемент
 }
}
```

Скоростные свойства функции *SortInsertSeq()* определяются количеством выполненных операций в теле функции [2]. На первом этапе выполняется внешний цикл для правой неупорядоченной части массива. В заголовке цикла *for*(*int* $j = 1$; $j < n$; j++) до начала итераций настраиваются две операции: 1) установка начального значения *int* $j = 1$; 2) начальная проверка условия $j < n$. Обозначим это количество как

$$W_{1,1} = 2 .$$

Кроме того, на каждой итерации внешнего цикла в заголовке цикла выполняется одна операция на очередную проверку окончания цикла $j < n$ и две операции на увеличение переменной цикла j++:

$$W_{1,2} = \sum_{j=1}^{n-1} \left(1+2\right) = 3\left(n-1\right).$$

Всего на первом этапе выполняется следующее количество операций:

$$W_1 = W_{1,1} + W_{1,2} = 2 + 3\left(n-1\right) = 3n-1 .$$

На втором этапе выполняется проверка на необходимость вставки с помощью инструкции условия *if*($a[j-1] < a[j]$)*continue*. На каждой итерации внешнего цикла по индексу j будет выполнено четыре операции на проверку условия $a[j-1] < a[j]$:

$$W_2 = \sum_{j=1}^{n-1} 4 = 4(n-1).$$

На третьем этапе происходит поиск места вставки во внутреннем цикле для левой упорядоченной части массива. Заголовок внутреннего цикла *for*(*i* = *j*–1; *i* >= 0; *i*– –) контролирует итерации поиска с конца упорядоченной части. Для этого выполняются следующие операции: 1) установка начального значения *i* = *j*–1 с двумя операциями на вычитание и запоминание; 2) одна операция на проверку условия *i* >= 0. Для всех итераций внешнего цикла здесь будет затрачено следующее количество операций:

$$W_{3,1} = \sum_{j=1}^{n-1} (2+1) = 3(n-1).$$

Кроме того, на каждой итерации внутреннего цикла в заголовке цикла выполняется одна операция на очередную проверку окончания цикла *i* >= 0 и две операции на уменьшение переменной цикла *i*– –:

$$W_{3,2} = \sum_{j=1}^{n-1} \sum_{0}^{i=j-1} (1+2) = \sum_{j=1}^{n-1} 3j = 3(1+n-1)\frac{n-1}{2} = \frac{3}{2}n(n-1).$$

Итерации внутреннего цикла ограничены инструкцией условия *if*(*a*[*i*] < *a*[*j*])*break*. В заголовке этой инструкции выполняются три операции на базирование адресов элементов массива и сравнение. Маргинальность задается минимальной и максимальной оценками. Минимальный случай был учтен на предыдущем этапе в инструкции *if(a*[*j*–1] < *a*[*j*] *)continue*, когда значения в массиве упорядочены по возрастанию. Максимальный случай соответствует полному выполнению внутреннего цикла без срабатывания прерывания *break* в инструкции *if(a*[*i*] < *a*[*j*] *)break*, поскольку исходный массив упорядочен наоборот. В максимальном случае во всех итерациях будет выполнено следующее количество операций:

$$W_{3,3} = \sum_{j=1}^{n-1} \sum_{0}^{i=j-1} 3 = \sum_{j=1}^{n-1} 3j = 3(1+n-1)\frac{n-1}{2} = \frac{3}{2}n(n-1).$$

Итак, на третьем этапе будет выполняться максимальное количество операций, если в массиве находятся значения, упорядоченные наоборот:

$$W_3 = W_{3,1} + W_{3,2} + W_{3,3} = 3(n-1) + \frac{3}{2}n(n-1) + \frac{3}{2}n(n-1) = 3n^2 - 3.$$

На четвертом этапе выполняется условная инструкция *i*++, поскольку значение индекса *i* находится на одну позицию левее места вставки:

$$W_4 = \sum_{j=1}^{n-1} 2 = 2(n-1).$$

Пятый этап выполняет вставку. Поскольку нашей целью является оценка маргинальных свойств, то минимальная оценка этого этапа не имеет смысла, так как упорядочивание заканчивается на втором этапе. Максимальная оценка происходит, когда предоставлен массив, упорядоченный наоборот. В инструкции *int r* = *a*[*j*] затрачиваются две операции:

$$W_{5,1} = \sum_{j=1}^{n-1} 2 = 2(n-1).$$

Инструкция второго внутреннего цикла *for*(*int m* = *j; m* > *i; m*– –) *a*[*m*] = *a*[*m*–1] выполняет сдвиг, освобождая место для вставляемого значения. В максимальном случае вставляемое значение всегда будет записываться в нулевой элемент *a*[0]. Перед итерациями сдвига проводится одна операция на установку индекса *int m* = *j* и одна операция *m* > *i* на проверку необходимости выполнения цикла. С учетом итераций внешнего цикла это займет следующее количество операций:

$$W_{5,2} = \sum_{j=1}^{n-1} (1+1) = 2(n-1).$$

В теле цикла сдвига на каждой итерации выполняются: 1) одна операция проверки условия *m* > *i*; 2) две операции на уменьшение индекса *m*– –; 3) четыре операции на копирование значения *a*[*m*] = *a*[*m*–1]. Всего будет выполнено следующее количество операций:

$$W_{5,3} = \sum_{j=1}^{n-1} \sum_{1}^{m=j} (1+2+4) = \sum_{j=1}^{n-1} 7j = 7(1+n-1)\frac{n-1}{2} = \frac{7}{2} n(n-1).$$

Осталось записать саму вставку *a*[*i*] = *r*, которая занимает две операции. С учетом итераций внешнего цикла получаем следующую оценку:

$$W_{5,4} = \sum_{j=1}^{n-1} 2 = 2(n-1).$$

Складывая оценки составных частей пятого этапа, получаем

$$W_5 = W_{5,1} + W_{5,2} + W_{5,3} + W_{5,4} =$$

$$= 2(n-1) + 2(n-1) + \frac{7}{2}n(n-1) + 2(n-1) = \frac{7}{2}n^2 + \frac{5}{2}n - 6.$$

Таким образом, рассчитаны составные оценки для всех частей алгоритма. Минимальное количество затрачиваемых операций получается, когда предоставлен уже упорядоченный массив. Обозначим минимальную оценку сортировки методом последовательной вставки в функции *SortInsertSeq()* как

$$W_{\min}^{SortInsertSeq} = W_1 + W_2 = 3n - 1 + 4(n-1) = 7n - 5.$$

Максимальную оценку дает следующее выражение:

$$W_{\max}^{SortInsertSeq} = W_1 + W_2 + W_3 + W_4 + W_5 =$$

$$= 3n - 1 + 4(n-1) + 3n^2 - 3 + 2(n-1) + \frac{7}{2}n^2 + \frac{5}{2}n - 6 = \frac{13}{2}n^2 + \frac{23}{2}n - 16.$$

Общим итогом для сортировки методом последовательной вставки является то, что его применение целесообразно для частично упорядоченных массивов. Такая ситуация наблюдается, когда информационная система поддерживает постоянно упорядоченный массив, в котором редко изменяются значения.

Сортировка методом дихотомической вставки. Алгоритм метода последовательной вставки постепенно увеличивает размер левой упорядоченной части. На каждой итерации в ней происходит поиск места вставки очередного сортируемого элемента из правой части. Поскольку левая часть явно упорядочена, то процесс поиска места вставки можно ускорить, если воспользоваться максимальным по быстродействию методом дихотомии. Ниже представлена функция *SortInsertDich()*, которая повторяет функцию последовательной вставки с заменой внутреннего цикла поиска места вставки на дихотомическую функцию поиска. Для удобства программирования и понимания алгоритма дихотомический поиск вынесен в отдельную функцию *DichPos()*. Для того чтобы в результирующей скомпилированной программе исключить явную операцию вызова функции *DichPos()*, она снабжена модификатором *inline*, который предписывает компилятору вместо вызова функции поставить ее тело.

```
inline int DichPos( const int a[], const int j )
{
 int n1 = 0, n2 = j – 1; // начало и конец массива
 while( n1 <= n2 ) // цикл поиска
 {
```

```
     Int z = ( n1 + n2 ) >> 1; // середина интервала поиска
     if( a[j] == a[z] ) { n1 = z; break; } // число найдено
     if( a[j] < a[z] ) n2 = z – 1; // продолжить поиск слева
     else n1 = z + 1; // продолжить поиск справа
     }
     return n1; // место вставки
    }
    //-----------------------------------------------------
--------
    void SortInsertDich( int a[], const int n )
    {
     for( int j = 1; j < n; j++ ) // цикл исходных элементов
     {
        if( a[j-1] <= a[j] ) continue; // вставка не нужна
     int i = DichPos( a, j ); // место вставки
     int r = a[j]; // запомнить вставляемый элемент
     for( int m = j; m > i; m-- ) a[m] = a[m-1]; // сдвиг
     a[i] = r; // установить текущий максимальный элемент
     }
    }
```

Прежде чем выяснять маргинальные свойства функции *Sort-InsertDich()*, займемся функцией дихотомического поиска *DichPos()*. На первом этапе определим количество операций, необходимых для инструкции *int* $n1 = 0$, $n2 = n - 1$:

$$W_1 = 1 + 2 = 3 .$$

На втором этапе дихотомический поиск контролирует заголовок цикла с предусловием *while(n1 <= n2)*, затрачивая на определение истинности условия $n1 <= n2$ одну операцию на каждой итерации. Количество итераций зависит от расположения информации в элементах массива. Маргинальные оценки дают два варианта. Поиск обнаружил элемент сразу за одну итерацию. Это значит, что элемент поиска находится посредине поступившего массива. Обозначим такое количество операций на втором этапе как

$$W_{2,1\min} = 1 .$$

Максимальное количество итераций определяется логарифмом с округлением в бóльшую сторону (обозначается внешними квадратными скобками):

$$W_{1,2\max} = 1 \cdot]\log_2 n[=]\log_2 n[.$$

В теле цикла определяется индекс элемента, находящегося между граничными элементами *n1* и *n2*. На вычисление этого индекса в инструкции $z = (n1 + n2) >> 1$ затрачиваются три операции. Можно

было бы использовать деление на два, но оно медленное. Поэтому его часто заменяют сдвигом *>>1* на один бит вправо:

$$W_{2,2\min} = 3 \cdot 1 = 3;$$
$$W_{2,2\max} = 3 \cdot]\log_2 n[= 3]\log_2 n[.$$

Следующая инструкция условия *if*($m == a[z]$) { $n1 = z$; *break;* } проверяет окончание поиска. На выполнение условия $m == a[z]$ затрачиваются две операции. Если условие истинно, то поиск завершен и следует выполнить составную инструкцию {$n1 = z$; *break;*} с одной операцией присваивания. Если условие ложно, то никаких дополнительных действий в инструкции не производится. В маргинальных ситуациях эта инструкция условия проявляет себя следующим образом. Минимальный поиск происходит, когда на первой итерации обнаруживается поисковый элемент:

$$W_{2,3\min} = 1 \cdot (1+1) = 2 .$$

Максимальный поиск — когда произойдут все дихотомические действия, но результата сравнения нет:

$$W_{2,3\max} =]\log_2 n[\cdot 1 =]\log_2 n[.$$

Следующая инструкция альтернативного условия *if*($m < a[z]$) $n2 = z - 1$; *else* $n1 = z + 1$ проясняет, в какой части массива следует продолжать поиск. В любом случае будут выполняться две операции в условии $m < a[z]$ и две альтернативные операции $n2 = z - 1$ или $n1 = z + 1$:

$$W_{2,4} = (2+2) \cdot]\log_2 n[= 4]\log_2 n[.$$

После рассмотрения всех инструкций подведем итог. Минимальный поиск осуществляется, когда сразу происходит совпадение центрального элемента с поисковым элементом. Реально такая ситуация возникает тогда, когда массив содержит одинаковые элементы.

$$W_{\min}^{DichPos} = W_1 + W_{2,1\min} + W_{2,2\min} + W_{2,3\min} = 3 + 1 + 3 + 2 = 9 .$$

Максимальное количество операций затрачивается при выполнении всех итераций дихотомии:

$$W_{\max}^{DichPos} = W_1 + W_{2,1\max} + W_{2,2\max} + W_{2,3\max} + W_{2,4} =$$
$$= 3 +]\log_2 n[+ 3]\log_2 n[+]\log_2 n[+ 4]\log_2 n[= 3 + 9]\log_2 n[.$$

Теперь можно определить количество операций, выполняемых в сортировке с дихотомическим поиском *SortInsertDich()*. На первом

этапе выполняется внешний цикл по правой неупорядоченной части массива. В заголовоке цикла *for*(*int j* = 1*; j < n; j*++) выполняется одна операция на начальную установку индекса *int j = 1* и одна — на предварительную проверку условия *j < n*:

$$W_{1,1} = 1 + 1 = 2.$$

Кроме того, на каждой итерации внешнего цикла в заголовке выполняется одна операция на очередную проверку окончания цикла *j < n* и две операции на увеличение переменной *j*++:

$$W_{1,2} = \sum_{j=1}^{n-1} (1 + 2) = 3(n-1).$$

Всего на первом этапе дихотомической вставки выполняется следующее количество операций:

$$W_1 = W_{2,1} + W_{2,2} = 2 + 3(n-1) = 3n - 1.$$

По аналогии с методом последовательной вставки на втором этапе *if*(*a*[*j* – 1] <= *a*[*j*])*continue* выполняется следующее количество операций:

$$W_2 = \sum_{j=1}^{n-1} 4 = 4(n-1).$$

На третьем этапе с помощью инструкции *int i = DichPos*(*a, j, a*[*j*]) определяется место вставки. Поскольку общее минимальное количество блокирует второй этап, то интерес представляет только максимальное количество операций:

$$W_3 = \sum_{j=1}^{n-1} \left(1 + W_{\max}^{DichPos}\right) = \sum_{j=1}^{n-1} \left(1 + 3 + 9\right] \log_2 n[\right) = \left(4 + 9\right] \log_2 n[\right)(n-1).$$

На четвертом этапе выполняется вставка, рассмотренная ранее в сортировке методом последовательной вставки. В инструкции *int r = a*[*j*] запоминается вставляемое значение, затрачивая следующее количество операций:

$$W_{4,1} = \sum_{j=1}^{n-1} 2 = 2(n-1).$$

Далее проводится сдвиг упорядоченной части массива в инструкции *for*(*int m = j; m > i; m*– –) *a*[*m*] = *a*[*m* – 1]. Перед итерациями цикла произойдет одна операция на установку индекса *int m = j* и одна операция *m > i* на проверку необходимости

выполнения цикла. С учетом итераций внешнего цикла это займет следующее количество операций:

$$W_{4,2} = \sum_{j=1}^{n-1}(1+1) = 2(n-1).$$

Учитывая аналогичный вывод в разделе последовательной вставки, в теле цикла сдвига будет затрачено следующее количество операций:

$$W_{4,3} = \sum_{j=1}^{n-1}\sum_{1}^{m=j}(1+2+4) = \sum_{j=1}^{n-1} 7j = \frac{7}{2}n(n-1).$$

В последней инструкции $a[i] = r$ сортируемое значение запоминается в упорядоченной части массива:

$$W_{4,4} = \sum_{j=1}^{n-1} 2 = 2(n-1).$$

Складывая оценки составных частей четвертого этапа, получаем следующий результат:

$$W_4 = W_{4,1} + W_{4,2} + W_{4,3} + W_{4,4} =$$

$$= 2(n-1) + 2(n-1) + \frac{7}{2}n(n-1) + 2(n-1) = \frac{7}{2}n^2 + \frac{5}{2}n - 6.$$

Все инструкции в функции *SortInsertDich()* рассмотрены. Минимальное количество операций имеет следующую оценку:

$$W_{\min}^{SortInsertDich} = W_1 + W_2 = 3n - 1 + 4(n-1) = 7n - 5.$$

Максимальное количество операций в функции *SortInsertDich()* оценивается следующим образом:

$$W_{\max}^{SortInsertDich} = W_1 + W_2 + W_3 + W_4 =$$

$$= 3n - 1 + 4(n-1) + (4 + 9]\log_2 n[)(n-1) + \frac{7}{2}n^2 + \frac{5}{2}n - 6 =$$

$$= \frac{7}{2}n^2 + 9(n-1)]\log_2 n[+ \frac{17}{2}n - 15.$$

Сравнительная оценка последовательной и дихотомической сортировок. По минимальному количеству операций обе технологии одинаковые:

$$W_{\min}^{SortInsertSeq} = 7n - 5;$$

$$W_{\min}^{SortInsertDich} = 7n - 5.$$

Это ситуации, когда исходный массив уже был упорядоченным, а также, когда массив редко частично обновляется. По максимальному количеству операций метод дихотомической вставки работает быстрее, чем последовательная вставка:

$$W_{\max}^{SortInsertSeq} = \frac{13}{2}n^2 + \frac{23}{2}n - 16;$$

$$W_{\max}^{SortInsertDich} = \frac{7}{2}n^2 + 9\left(n-1\right)]\log_2 n[+\frac{17}{2}n - 15.$$

Непосредственный расчет этих оценок представлен в следующей таблице.

Число элементов n	$W_{\max}^{SortInsertSeq}$	$W_{\max}^{SortInsertDich}$	$\frac{S_{\max}^{SortInsertSeq}}{S_{\max}^{SortInsertDich}}$
4	135	129	1,047
8	493	466	1,058
16	1833	1557	1,177
32	7009	5236	1,339
64	27345	18267	1,497
128	109953	66418	1,655
256	428913	249897	1,716
512	1709809	963232	1,775
1024	6827505	3770775	1,811

Главными коэффициентами в этом сравнении являются коэффициенты при n^2. Быстродействие метода дихотомической сортировки выглядит предпочтительнее. Но эти оценки — для наихудших случаев расположения информации в массиве. Для обеих технологий это массив с обратно упорядоченным расположением значений элементов.

ЛИТЕРАТУРА

[1] Кнут Д.Э. *Искусство программирования. Т. 3. Сортировка и поиск*. Москва, изд-во Вильямс, 2009, 824 с.

[2] Седжвик Р. *Фундаментальные алгоритмы на C++. Анализ, структуры данных, сортировка, поиск*. Санкт-Петербург, ООО «ДиаСофтЮП», 2002, 688 с.

Статья поступила в редакцию 10.06.2013

Ссылку на эту статью просим оформлять следующим образом:

Деон А.Ф., Терентьев Ю.И. Маргинальные свойства сортировки массивов методом дихотомической вставки. *Инженерный журнал: наука и инновации*, 2013, вып. 6. URL: http://engjournal.ru/catalog/it/hidden/769.html

Деон Алексей Федорович родился в 1951 г., окончил МВТУ им. Н.Э. Баумана в 1974 г. Канд. техн. наук, доцент кафедры «Программное обеспечение ЭВМ и информационные технологии». Автор 23 научных трудов, специализируется в области разработки программного обеспечения автоматизированных систем управления предприятием. e-mail: deonalex@mail.ru

Терентьев Юрий Иванович родился в 1948 г.р., окончил МВТУ им. Н.Э. Баумана в 1974 г. Канд. техн. наук, доцент кафедры «Программное обеспечение ЭВМ и информационные технологии». Автор 35 научных трудов, специализируется в области компьютерного моделирования теплофизических процессов в энергетических установках. e-mail: yury_terentev@mail.ru