УДК 681.3.06(075)

Маргинальные свойства простых сортировок целочисленных массивов

© А.Ф. Деон, Ю.И. Терентьев

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

Выполнен сравнительный анализ маргинальных скоростных свойств простых сортировок массивов с учетом операций сравнения, сложения, перестановки и запоминания сортируемых элементов в массивах целых чисел.

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

Введение. К простым сортировкам массивов обычно относят сортировки методами выбора, перестановки, парных сравнений, сдвига и вставки [1]. Традиционно их быстродействие оценивают лишь по количеству операций сравнения во время упорядочивания массива. Однако используемые алгоритмы также включают и некоторые другие операции, количество которых различно в указанных методах. Наличие этих операций может повлиять на общую эффективность применяемого метода. В связи с этим необходимо иметь более точные оценки числа выполняемых операций в различных условиях. Особое значение имеют предельные или маргинальные оценки.

Далее выполнен анализ программной реализации семи наиболее популярных методов сортировок. При этом основное внимание уделено двум предельным по количеству операций случаям: минимальному, когда исходный массив уже упорядочен, и максимальному, когда элементы массива расположены в обратном порядке.

Сортировка методом выбора. Метод реализован в рамках функции *SortChoice( )*, в которой позиция текущего упорядоченного элемента заполняется минимальным текущим значением из неупорядоченной части массива:

```
void SortChoice(int a[ ], const int n)
{  int n1 = n – 1;
for (int i = 0; i < n1; i++)
   { int k = i;
      for (int j = i + 1; j < n; j++)
         if (a[j] < a[k]) k = j;
      int r = a[i];
      a[i] = a[k];
      a[k] = r;
      }
 }
```

Интерфейс функции указывает адрес исходного неупорядоченного массива *a*, содержащего *n* элементов. После выполнения функции массив *a* упорядочен по возрастанию.

Скоростные свойства функции *SortChoice( )* определяются количеством выполненных операций в теле функции [2]. Сначала выполняется инструкция *int n1 = n – 1* для определения количества $W_1^{SortChoice}$ итераций главного цикла сортировки. При этом затрачиваются две операции на вычитание и сохранение результата:

$$W_1^{SortChoice} = 2.$$

В заголовке главного цикла *for (int i = 0; i < n1; i++)* до начала итераций настраиваются две операции:

1) установка начального значения *int i = 0*;

2) начальная проверка условия *i < n1*.

Обозначим это количество операций как

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

Кроме того, на каждой итерации в заголовке цикла выполняются одна операция на очередную проверку окончания цикла *i < n1* и две операции на увеличение переменной цикла *i++*:

$$W_{2,2}^{SortChoice} = \sum_{i=0}^{n1-1}(1+2) = 3(n-1).$$

В теле цикла инструкция *int k = i* фиксирует индекс позиции опорного элемента. Количество этих операций

$$W_{2,3}^{SortChoice} = \sum_{i=0}^{n1-1} 1 = n-1.$$

Далее осуществляется поиск текущего минимального элемента под управлением заголовка внутреннего цикла *for (int j = i + 1; j < n; j++)*. До начала итераций этого цикла выполняются установка начального значения *int j = i + 1* — две операции и предварительная проверка условия *j < n* — одна операция:

$$W_{2,4,1}^{SortChoice} = \sum_{i=0}^{n1-1}(2+1) = 3(n-1).$$

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

$$W_{2,4,2}^{SortChoice} = \sum_{i=0}^{n1-1}\sum_{j=i+1}^{n-1}(1+2) = \sum_{i=0}^{n1-1} 3(n-i-1) = \frac{3}{2}n(n-1).$$

В теле внутреннего цикла инструкция условия *if (a[j] < a[k]) k = j* осуществляет поиск текущего минимального значения в неупорядоченной части массива. В минимальном случае затрачиваются только три операции на проверку условия *a[j] < a[k]*:

$$W_{2,4,3\min}^{SortChoice} = \sum_{i=0}^{n1-1}\sum_{j=i+1}^{n-1} 3 = \sum_{i=0}^{n1-1} 3(n-i-1) = \frac{3}{2}n(n-1).$$

В максимальном случае на каждой итерации внутреннего цикла затрачиваются три операции на проверку условия *a[j] < a[k]* и одна операция на присваивание *k = j*:

$$W_{2,4,3\max}^{SortChoice} = \sum_{i=0}^{n1-1}\sum_{j=i+1}^{n-1}(3+1) = \sum_{i=0}^{n1-1}4(n-i-1) = 2n(n-1).$$

После окончания внутреннего цикла переменная *k* содержит индекс текущего минимального элемента. Его значение необходимо перенести в опорный элемент по индексу *i*. Для этого инструкция *int r = a[i]* запоминает значение *a[i]* — две операции. Инструкция *a[i] = a[k]* копирует текущее минимальное значение *a[i]* в опорный элемент *a[k]* — три операции. Инструкция *a[k] = r* переносит прежнее значение опорного элемента в освободившийся элемент *a[k]* — две операции. На всех итерациях главного цикла эти три инструкции перестановки значений потребуют следующее количество операций:

$$W_{2,5}^{SortChoice} = \sum_{i=0}^{n1-1}(2+3+2) = 7(n-1).$$

Таким образом, *маргинальные свойства сортировки методом выбора* оцениваются суммированием числа операций на всех этапах:

$$W_{\min}^{SortChoice} = W_1^{SortChoice} + W_{2\min}^{SortChoice} =$$

$$= -10 + 11n + 3n^2;$$

$$W_{\max}^{SortChoice} = W_1^{SortChoice} + W_{2\max}^{SortChoice} =$$

$$= -10 + \frac{21}{2}n + \frac{7}{2}n^2.$$

Сортировка методом перестановки. Ниже представлена функция *SortExchange( )*, в которой позиция текущего упорядоченного элемента заполняется очередным минимальным значением из неупорядоченной части массива:

```
void SortExchange(int a[ ], const int n)
{  int n1 = n - 1;
   for (int i = 0; i < n1; i++)
      for (int j = i + 1; j < n; j++)
         if (a[j] < a[i])
         {  int r = a[i];
            a[i] = a[j];
            a[j] = r;
         }
}
```

Интерфейс функции указывает адрес исходного неупорядоченного массива *a*, содержащего *n* элементов. После выполнения функции массив *a* упорядочен по возрастанию.

В теле функции *SortExchange( )* выполняется инструкция *int n1 = = n – 1*, при этом затрачиваются $W_1^{SortExchange}$ операций:

$$W_1^{SortExchange} = W_1^{SortChoice} = 2.$$

Далее в заголовке главного цикла *for (int i = 0; i < n1; i++)* до начала итераций выполняются $W_{2,1}^{SortExchange}$ операций:

$$W_{2,1}^{SortExchange} = W_{2,1}^{SortChoice} = 2\,.$$

На каждой итерации в заголовке цикла выполняются $W_{2,2}^{SortExchange}$ операций:

$$W_{2,2}^{SortExchange} = W_{2,2}^{SortChoice} = \sum_{i=0}^{n1-1}(1+2) = 3(n-1).$$

Поиск текущего минимального элемента осуществляется под управлением заголовка внутреннего цикла *for (int j = i + 1; j < n; j++)*. До начала итераций этого цикла затрачиваются $W_{2,3,1}^{SortExchange}$ операций:

$$W_{2,3,1}^{SortExchange} = W_{2,4,1}^{SortChoice} = \sum_{i=0}^{n1-1}(2+1) = 3(n-1).$$

Кроме того, на каждой итерации внутреннего цикла в заголовке цикла выполняются $W_{2,3,2}^{SortExchange}$ операций:

$$W_{2,3,2}^{SortExchange} = W_{2,4,2}^{SortChoice} = \sum_{i=0}^{n1-1}\sum_{j=i+1}^{n-1}(1+2) = \frac{3}{2}n(n-1).$$

В теле внутреннего цикла заголовок инструкции условия *if (a[j] < < a[i])* сравнивает текущее минимальное значение *a[i]* с очередным значением элемента *a[j]* в неупорядоченной части массива, при этом выполняются три операции. На всех итерациях внутреннего цикла заголовок инструкции условия затрачивает $W_{2,3,3}^{SortExchange}$ операций:

$$W_{2,3,3}^{SortExchange} = \sum_{i=0}^{n1-1}\sum_{j=i+1}^{n-1} 3 = \sum_{i=0}^{n1-1} 3(n-i-1) = \frac{3}{2}n(n-1).$$

Минимальное количество операций будет тогда, когда элементы в массиве уже упорядочены. В этом случае тело инструкции условия не выполняется.

Если изначально массив упорядочен по убыванию, то на каждой итерации внутреннего цикла выполняется перестановка значений с помощью инструкции *int r = a[i]* — две операции, инструкции *a[i] = = a[j]* — три операции и инструкции *a[j] = r* — две операции; всего семь операций:

$$W_{2,3,4\max}^{SortExchage} = \sum_{i=0}^{n1-1}\sum_{j=i+1}^{n-1} 7 = \sum_{i=0}^{n1-1} 7\left(n-i-1\right) = \frac{7}{2}n\left(n-1\right).$$

Таким образом, *маргинальные свойства сортировки методом перестановки* оцениваются суммированием числа операций на всех этапах:

$$W_{\min}^{SortExchange} = W_1^{SortExchange} + W_{2\min}^{SortExchange} =$$

$$= -2 + 3n + 3n^2;$$

$$W_{\max}^{SortExchange} = W_1^{SortExchange} + W_{2\max}^{SortExchange} =$$

$$= -2 - \frac{1}{2}n + \frac{13}{2}n^2,$$

где $W_{2\min}^{SortExchange}$ и $W_{2\max}^{SortExchange}$ объединяют соответствующие операции на втором этапе.

Сравнивая минимальные результаты сортировки методов выбора и перестановки для уже упорядоченного исходного массива, получаем, что по быстродействию сортировки метод выбора уступает методу перестановки:

$$W_{\min}^{SortChoice} = -10 + 11n + 3n^2 > W_{\min}^{SortExchange} = -2 + 3n + 3n^2.$$

В максимальном случае быстродействие сортировки методом выбора выше, чем быстродействие сортировки методом перестановки ($n > 2$):

$$W_{\max}^{SortChoice} = -10 + \frac{21}{2}n + \frac{7}{2}n^2 < W_{\max}^{SortExchange} = -2 - \frac{1}{2}n + \frac{13}{2}n^2.$$

Сортировка методом парных сравнений. Ниже представлена функция *SortPair( )*, в которой в неупорядоченной части массива сравниваются соседние элементы для выбора меньшего значения:

```
void SortPair(int a[], const int n)
{  int n1 = n - 1;
for (int i = 0; i < n1; i++)
      for (int j = n1; j > i; j--)
         if (a[j] < a[j - 1])
         {  int r = a[j];
            a[j] = a[j - 1];
            a[j - 1] = r;
         }
}
```

Полученное в итоге минимальное значение подсоединяется к упорядоченной части массива. Интерфейс функции указывает адрес исходного неупорядоченного массива *a*, содержащего *n* элементов. После выполнения функции массив *a* упорядочен по возрастанию.

В теле функции *SortPair( )* выполняется инструкция *int n1 = n – 1*, при этом затрачиваются $W_1^{SortPair}$ операций:

$$W_1^{SortPair} = W_1^{SortExchange} = W_1^{SortChoice} = 2.$$

Далее в заголовке главного цикла *for (int i = 0; i < n1; i++)* до начала итераций выполняются $W_{2,1}^{SortPair}$ операций:

$$W_{2,1}^{SortPair} = W_{2,1}^{SortExchange} = W_{2,1}^{SortChoice} = 2.$$

На каждой итерации в заголовке цикла выполняются $W_{2,2}^{SortPair}$ операций:

$$W_{2,2}^{SortPair} = W_{2,2}^{SortExchange} = W_{2,2}^{SortChoice} = \sum_{i=0}^{n1-1}(1+2) = 3(n-1).$$

Поиск текущего минимального элемента осуществляется под управлением заголовка внутреннего цикла *for (int j = n1; j > i; j– –)*. До начала итераций этого цикла выполняются $W_{2,3,1}^{SortPair}$ операций:

$$W_{2,3,1}^{SortPair} = \sum_{i=0}^{n1-1}(1+1) = 2(n-1).$$

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

$$W_{2,3,2}^{SortPair} = \sum_{i=0}^{n1-1}\sum_{j>i}^{j=n-1}(1+2) = \sum_{i=0}^{n1-1}\sum_{z=i+1}^{n-1} 3 = \sum_{i=0}^{n1-1} 3(n-i-1) = \frac{3}{2}n(n-1).$$

В теле внутреннего цикла заголовок инструкции условия *if (a[j] < < a[j – 1])* сравнивает значения пары соседних элементов *a[j] < a[j – 1]* в неупорядоченной части массива — три операции. На всех итерациях внутреннего цикла заголовок инструкции условия затрачивает $W_{2,3,3}^{SortPair}$ операций:

$$W_{2,3,3}^{SortPair} = \sum_{i=0}^{n1-1}\sum_{z>i}^{j=n-1} 3 = \sum_{i=0}^{n1-1}\sum_{j=i+1}^{n-1} 3 = \sum_{i=0}^{n1-1} 3(n-i) = \frac{3}{2}n(n-1).$$

В минимальном случае тело инструкции условия не выполняется.

В максимальном случае на каждой итерации внутреннего цикла осуществляется перестановка значений с помощью инструкции *int r = = a[j]* — две операции, инструкции *a[j] = a[j – 1]* — четыре операции и инструкции *a[j – 1] = r* — три операции; всего девять операций:

$$W_{2,3,4\max}^{SortPair} = \sum_{i=0}^{n1-1}\sum_{j=i+1}^{n-1}(2+4+3) = \sum_{i=0}^{n1-1}\sum_{j=i+1}^{n-1} 9 = \sum_{i=0}^{n1-1} 9(n-i) = \frac{9}{2}n(n-1).$$

Таким образом, *маргинальные свойства сортировки методом парных сравнений* оцениваются суммированием числа операций на всех этапах:

$$W_{\min}^{SortPair} = W_1^{SortPair} + W_{2\min}^{SortPair} = -1 + 2n + 3n^2;$$

$$W_{\max}^{SortPair} = W_1^{SortPair} + W_{2\max}^{SortPair} = -1 - \frac{5}{2}n + \frac{15}{2}n^2.$$

Сравнивая минимальные результаты сортировки методов перестановки и парных сравнений (пузырька) для уже упорядоченного исходного массива, получаем, что быстродействие сортировки методом парных сравнений выше, чем быстродействие сортировки методом перестановки:

$$W_{\min}^{SortPair} = -1 + 2n + 3n^2 < W_{\min}^{SortExchange} = -2 + 3n + 3n^2.$$

В случае когда элементы исходного массива расположены в обратном порядке, быстродействие сортировки методом выбора выше, чем быстродействие сортировки методом парных сравнений ($n > 2$):

$$W_{\max}^{SortPair} = -1 - \frac{5}{2}n + \frac{15}{2}n^2 > W_{\max}^{SortChoice} = -10 + \frac{21}{2}n + \frac{7}{2}n^2.$$

Сортировка модифицированным методом парных сравнений. Ниже представлена функция *SortPairMod( )*, в которой используется алгоритм предыдущей функции *SortPair( )* с добавлением флага:

```
void SortPairMod(int a[], const int n)
{  int n1 = n - 1;
   for (int i = 0; i < n1; i++)
   {  bool flag = true;
      for (int j = n1; j > i; j--)
         if (a[j] < a[j - 1])
         {int r = a[j];
            a[j] = a[j - 1];
            a[j - 1] = r;
            flag = false;
         }
      if (flag) break;
   }
}
```

Флаг указывает, была ли перестановка парных значений на очередной итерации главного цикла. Если перестановки не наблюдались, то массив упорядочен и дальнейшие итерации главного цикла не нужны, сортировка закончена. Это выгодно для частично упорядоченных массивов, которые редко изменяются. Интерфейс функции указывает адрес исходного неупорядоченного массива *a*, содержащего *n* элементов. После выполнения функции массив *a* упорядочен по возрастанию.

При анализе данного метода используются обозначения, принятые для сортировки методом парных сравнений. Инструкция *int n1 = n – 1* занимает две операции:

$$W_1^{SortPairMod} = W_1^{SortPair} = 2.$$

При настройке главного цикла *for (int i = 0; i < n1; i++)* выполняются $W_{2,1}^{SortPairMod}$ операций:

$$W_{2,1}^{SortPairMod} = W_{2,1}^{SortPair} = 2.$$

Кроме того, в максимальном случае на каждой итерации проводится проверка условия *i < n1* — одна операция и увеличивается индекс *i++* — две операции:

$$W_{2,2\max}^{SortPairMod} = \sum_{i=0}^{n1-1}(1+2) = 3(n-1).$$

В теле главного цикла устанавливается флаг *bool flag = true* в предположении, что оставшаяся часть массива будет упорядоченной. Установка флага занимает одну операцию. В минимальном случае главный цикл выполняет только одну итерацию:

$$W_{2,3,1\min}^{SortPairMod} = \sum_{i=0}^{0} 1 = 1.$$

В максимальном случае флаг устанавливается на каждой итерации главного цикла:

$$W_{2,3,1\max}^{SortPairMod} = \sum_{i=0}^{n1-1} 1 = n-1.$$

Поиск текущего минимального элемента осуществляется под управлением заголовка внутреннего цикла *for (int j = n1; j > i; j– –)*. До начала его итераций выполняются установка начального значения *int j = n1* — одна операция и предварительная проверка условия *j > i* — одна операция. В минимальном случае выполняется только одна итерация главного цикла:

$$W_{2,3,2\min}^{SortPairMod} = \sum_{i=0}^{0}(1+1) = 2.$$

В максимальном случае установки начальных значений внутреннего цикла выполняются на всех итерациях главного цикла:

$$W_{2,3,2\max}^{SortPairMod} = \sum_{i=0}^{n1-1}(1+2) = 3(n-1).$$

Кроме того, на каждой итерации внутреннего цикла в заголовке цикла выполняются одна операция на очередную проверку оконча-

ния цикла $j > i$ и две операции на уменьшение переменной цикла j– –. В минимальном случае, когда массив уже упорядочен, выполняется только одна итерация главного цикла:

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

Если исходный массив упорядочен по убыванию, то затрачивается максимальное количество операций на $j > i$ и j– –:

$$W_{2,3,3\max}^{SortPairMod} = \sum_{i=0}^{n1-1}\sum_{j>i}^{j=n-1}(1+2) = \sum_{i=0}^{n1-1}\sum_{z=i+1}^{n-1} 3 = \sum_{i=0}^{n1-1} 3(n-i) = \frac{3}{2}n(n-1).$$

В теле внутреннего цикла заголовок инструкции условия *if (a[j] < < a[j – 1])* сравнивает значения пары соседних элементов *a[j] < a[j – 1]* в неупорядоченной части массива — три операции. На всех итерациях внутреннего цикла заголовок инструкции условия затрачивает $W_{2,3,4\min}^{SortPairMod}$ и $W_{2,3,4\max}^{SortPairMod}$ операций:

$$W_{2,3,4\min}^{SortPairMod} = \sum_{i=0}^{0}\sum_{j>i}^{j=n-1} 3 = \sum_{z=i+1}^{n-1} 3 = 3(n-1);$$

$$W_{2,3,4\max}^{SortPairMod} = \sum_{i=0}^{n1-1}\sum_{j>i}^{j=n-1} 3 = \sum_{i=0}^{n1-1}\sum_{z=i+1}^{n-1} 3 = \sum_{i=0}^{n1-1} 3(n-i) = \frac{3}{2}n(n-1).$$

В минимальном случае тело инструкции условия не выполняется.

Если элементы исходного массива расположены в обратном порядке, то на каждой итерации внутреннего цикла выполняется перестановка значений с помощью инструкции *int r = a[j]* — две операции, инструкции *a[j] = a[j – 1]* — четыре операции и инструкции *a[j – 1] = r* — три операции; всего девять операций:

$$W_{2,3,5\max}^{SortPairMod} = \sum_{i=0}^{n1-1}\sum_{j=i+1}^{n-1}(2+4+3) = \sum_{i=0}^{n1-1}\sum_{j=i+1}^{n-1} 9 = \sum_{i=0}^{n1-1} 9(n-i) = \frac{9}{2}n(n-1).$$

В конце тела главного цикла выполняется проверка флага в инструкции условия *if (flag) break* — одна операция. В минимальном случае сортировку далее проводить не следует:

$$W_{2,4\min}^{SortPairMod} = \sum_{i=0}^{0} 1 = 1.$$

В максимальном случае проверка флага выполняется на каждой итерации главного цикла:

$$W_{2,4\max}^{SortPairMod} = \sum_{i=0}^{n1-1} 1 = n-1.$$

Таким образом, *маргинальные свойства сортировки модифицированным методом парных сравнений (пузырька)* оцениваются сум-

мированием соответствующего числа операций на всех этапах. В минимальном случае число операций

$$W_{\min}^{SortPairMod} = W_1^{SortPairMod} + W_{2\min}^{SortPairMod} =$$

$$= 2 + 6n,$$

т. е. для уже упорядоченного массива количество операций при сортировке данным методом пропорционально только длине массива. Проверке подвергаются соседние элементы только один раз.

Максимальный случай дает следующий результат:

$$W_{\max}^{SortPairMod} = W_1^{SortPairMod} + W_{2\max}^{SortPairMod} =$$

$$= -4 + 2n + 6n^2.$$

Сравнивая минимальные результаты сортировки обычного и модифицированного методов парных сравнений, получаем, что быстродействие сортировки модифицированным методом парных сравнений превосходит быстродействие сортировки обычным методом парных сравнений:

$$W_{\min}^{SortPair} = -1 + 2n + 3n^2 > W_{\min}^{SortPairMod} = 2 + 6n.$$

В максимальном случае быстродействие сортировки методом выбора выше, чем быстродействие сортировки модифицированным методом парных сравнений ($n > 2$):

$$W_{\max}^{SortPairMod} = -4 + 2n + 6n^2 > W_{\max}^{SortChoice} = -10 + \frac{21}{2}n + \frac{7}{2}n^2.$$

Сортировка методом сдвига. Ниже представлена функция *SortShift( )*, в которой позиция текущего упорядоченного элемента заполняется очередным минимальным значением из неупорядоченной части массива, при этом происходит сдвиг неупорядоченных элементов до места очередного минимального элемента:

```
void SortShift(int a[], const int n)
{  int n1 = n - 1;
   for (int i = 0; i < n1; i++)
   {  int k = i;
      for (int j = i + 1; j < n; j++)
         if (a[j] < a[k]) k = j;
      int r = a[k];
      for (int m = k; m > i; m--) a[m] = a[m - 1];
      a[i] = r;
   }
}
```

Интерфейс функции указывает адрес исходного неупорядоченного массива *a*, содержащего *n* элементов. После выполнения функции массив *a* упорядочен по возрастанию.

В теле функции *SortShift( )* выполняется инструкция *int n1 = n – 1*, при этом затрачиваются $W_1^{SortShift}$ операций:

$$W_1^{SortShift} = W_1^{SortPair} = 2.$$

Далее в настройке заголовка главного цикла *for (int i = 0; i < n1; i++)* выполняются $W_{2,1}^{SortShift}$ операций:

$$W_{2,1}^{SortShift} = W_{2,1}^{SortPair} = 2.$$

Кроме того, на каждой итерации в заголовке цикла выполняются $W_{2,2}^{SortShift}$ операций:

$$W_{2,2}^{SortShift} = W_{2,2}^{SortPair} = \sum_{i=0}^{n1-1}(1+2) = 3(n-1).$$

В теле цикла инструкция *int k = i* фиксирует индекс позиции опорного элемента. Количество этих операций

$$W_{2,3}^{SortShift} = \sum_{i=0}^{n1-1} 1 = n-1.$$

Далее осуществляется поиск текущего минимального элемента под управлением заголовка первого внутреннего цикла *for (int j = i + 1; j < n; j++)*. До начала итераций этого цикла выполняются установка начального значения *int j = i + 1* — две операции и предварительная проверка условия *j < n* — одна операция:

$$W_{2,4,1}^{SortShift} = \sum_{i=0}^{n1-1}(2+1) = 3(n-1).$$

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

$$W_{2,4,2}^{SortShift} = \sum_{i=0}^{n1-1}\sum_{j=i+1}^{n-1}(1+2) = \sum_{i=0}^{n1-1} 3(n-i-1) = \frac{3}{2}n(n-1).$$

В теле первого внутреннего цикла заголовок инструкции условия *if (a[j] < a[k])* сравнивает текущее минимальное значение *a[k]* с очередным значением элемента *a[j]* в неупорядоченной части массива — три операции. На всех итерациях первого внутреннего цикла заголовок инструкции условия затрачивает $W_{2,4,3}^{SortShift}$ операций:

$$W_{2,4,3}^{SortShift} = \sum_{i=0}^{n1-1}\sum_{j=i+1}^{n-1} 3 = \sum_{i=0}^{n1-1} 3(n-i-1) = \frac{3}{2}n(n-1).$$

В минимальном случае тело инструкции условия не выполняется.

В максимальном случае на каждой итерации внутреннего цикла затрачивается одна операция на присваивание $k = j$:

$$W_{2,4,4\max}^{SortChoice} = \sum_{i=0}^{n1-1}\sum_{j=i+1}^{n-1}(3+1) = \sum_{i=0}^{n1-1}4(n-i-1) = 2n(n-1).$$

Первый внутренний цикл закончен. Найденное текущее минимальное значение запоминается в рабочей переменной с помощью инструкции *int r = a[k]* — две операции:

$$W_{2,5}^{SortShift} = \sum_{i=0}^{n1-1}2 = 2(n-1).$$

Второй внутренний цикл выполняет сдвиг значений от позиции опорного элемента до позиции элемента с текущим минимумом *a[k]*. Сдвиг проводится под управлением заголовка *for (int m = k; m > i; m– –)*. До итераций этого цикла выполняются установка начального значения *int m = k* — одна операция и проверка условия $m > i$ — одна операция:

$$W_{2,6,1}^{SortShift} = \sum_{i=0}^{n1-1}(1+1) = 2(n-1).$$

Кроме того, на каждой итерации внутреннего цикла в заголовке цикла выполняются одна операция на очередную проверку окончания цикла $m > i$ и две операции на уменьшение переменной цикла *m– –*. В минимальном случае предварительная проверка $m > i$ блокирует выполнение тела второго внутреннего цикла, т. е. сдвиг не выполняется. В максимальном случае итерации выполняются для каждого элемента из неупорядоченной части, т. е. сдвиг проводится вправо на один элемент по длине всей неупорядоченной части:

$$W_{2,6,2\max}^{SortShift} = \sum_{i=0}^{n1-1}\sum_{m>i}^{m=n-1}(1+2) = \sum_{i=0}^{n1-1}\sum_{z=i+1}^{n-1}3 = \sum_{i=0}^{n1-1}3(n-i-1) =$$

$$= \frac{3}{2}n(n-1).$$

В теле второго внутреннего цикла выполняется сдвиг с помощью инструкции *a[m] = a[m – 1]* — четыре операции. В минимальном случае этого не происходит, поскольку тело цикла блокирует условие заголовка. В максимальном случае сдвиг выполняется на каждой итерации:

$$W_{2,6,3\max}^{SortShift} = \sum_{i=0}^{n1-1}\sum_{m>i}^{m=n-1}4 = \sum_{i=0}^{n1-1}\sum_{z=i+1}^{n-1}4 = \sum_{i=0}^{n1-1}4(n-i-1) =$$

$$= 2n(n-1).$$

Второй внутренний цикл завершен.

Последняя инструкция *a[i] = r* главного цикла — две операции — устанавливает на место промежуточный минимум:

$$W_{2,7}^{SortShift} = \sum_{i=0}^{n1-1} 2 = 2(n-1).$$

Таким образом, *маргинальные свойства сортировки методом сдвига* оцениваются суммированием числа операций на всех этапах:

$$W_{\min}^{SortShift} = W_1^{SortShift} + W_{2\min}^{SortShift} = -9 + 10n + 3n^2;$$

$$W_{\max}^{SortShift} = W_1^{SortShift} + W_{2\max}^{SortShift} = -9 + \frac{13}{2}n + \frac{13}{2}n^2.$$

Сравнивая минимальные результаты сортировки, полученные методом сдвига и модифицированным методом парных сравнений для уже упорядоченного исходного массива, видим, что быстродействие сортировки модифицированным методом парных сравнений превосходит быстродействие сортировки методом сдвига:

$$W_{\min}^{SortShift} = -9 + 10n + 3n^2 > W_{\min}^{SortPairMod} = 2 + 6n.$$

В случае когда исходный массив упорядочен по убыванию, быстродействие сортировки методом сдвига ниже, чем быстродействие сортировки методом выбора:

$$W_{\max}^{SortShift} = -9 + \frac{13}{2}n + \frac{13}{2}n^2 > W_{\max}^{SortChoice} = -10 + \frac{21}{2}n + \frac{7}{2}n^2.$$

Сортировка методом последовательной вставки. В алгоритме возрастающей сортировки методом последовательной вставки предусмотрено, что упорядоченная часть находится в левой части массива, а неупорядоченная — в правой. Начальный элемент из неупорядоченной части вставляется на соответствующее место в упорядоченной части. Продолжая итерации для начальных элементов всех неупорядоченных частей, получаем отсортированный массив *a*.

В работе [3] исследованы свойства этого метода сортировки. Обозначим минимальную оценку сортировки методом последовательной вставки как

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

Максимальная оценка этого метода

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

Сортировку методом последовательной вставки целесообразно применять для *частично упорядоченных массивов*. Такая ситуация наблюдается в случае, когда информационная система поддерживает постоянно упорядоченный массив, в котором редко изменяются значения.

Сравнивая минимальные результаты сортировки методом последовательной вставки и модифицированным методом парных сравнений для уже упорядоченного исходного массива, получаем, что быстродействие сортировки модифицированным методом парных сравнений превосходит быстродействие сортировки методом последовательной вставки ($n > 7$):

$$W_{\min}^{SortInsertSeq} = -5 + 7n > W_{\min}^{SortPairMod} = 2 + 6n.$$

Если исходный массив упорядочен в обратном направлении, быстродействие сортировки методом выбора выше, чем быстродействие сортировки методом последовательной вставки:

$$W_{\max}^{SortChoice} = -10 + \frac{21}{2}n + \frac{7}{2}n^2 < W_{\max}^{SortInsertSeq} = -16 + \frac{23}{2}n + \frac{13}{2}n^2.$$

Сортировка методом дихотомической вставки. В предыдущем алгоритме метода последовательной вставки предусмотрено постепенное увеличение размера левой упорядоченной части. На каждой итерации в этой части 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 [3].

Минимальная оценка этого метода сортировки [3]

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

а максимальная оценка определяется выражением [3]

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

Открытые квадратные скобки обозначают округление до ближайшего большего целого числа.

Сравнивая результаты сортировки, получаем

$$W_{\min}^{SortPairMod} = 2 + 6n < W_{\min}^{SortInsertDich} = -5 + 7n;$$

$$W_{\max}^{SortChoice} = -10 + \frac{21}{2}n + \frac{7}{2}n^2 > W_{\max}^{SortInsertDich} =$$

$$= -15 + 9\left(n - 1\right)\rceil \log_2 n\lceil + \frac{17}{2}n + \frac{7}{2}n^2.$$

Выводы. Формулы для оценки количества операций всех рассмотренных сортировок в предельных условиях сведены в таблицу. Главными коэффициентами в сравнениях по быстродействию являются коэффициенты при n^2.

Метод сортировки	Количество операций	
	минимальное	максимальное
Метод выбора	$-10+11n+3n^2$	$-10+\frac{21}{2}n+\frac{7}{2}n^2$
Метод перестановки	$-2+3n+3n^2$	$-2-\frac{1}{2}n+\frac{13}{2}n^2$
Метод парных сравнений	$-1+2n+3n^2$	$-1-\frac{5}{2}n+\frac{15}{2}n^2$
Модифицированный метод парных сравнений	$2+6n$	$-4+2n+6n^2$
Метод сдвига	$-9+10n+3n^2$	$-9+\frac{13}{2}n+\frac{13}{2}n^2$
Метод последовательной вставки	$-5+7n$	$-16+\frac{23}{2}n+\frac{13}{2}n^2$
Метод дихотомической вставки	$-5+7n$	$-15+9\left(n-1\right)\left]\log_2 n\right[+\frac{17}{2}n+\frac{7}{2}n^2$

В наилучшем случае, когда исходный массив уже упорядочен, т. е. для частично упорядоченных редко сортируемых массивов, предпочтительными являются *модифицированный метод парных сравнений* и *методы последовательной* и *дихотомической вставки*.

В наихудшем случае, когда исходный массив упорядочен по убыванию, предпочтительнее *метод выбора* и *метод дихотомической вставки*.

ЛИТЕРАТУРА

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

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

[3] Деон А.Ф., Терентьев Ю.И. Маргинальные свойства сортировки массивов методом дихотомической вставки. *Инженерный журнал: наука и инновации,* 2013, вып. 6 (18). URL: http:/engjournal.ru/catalog/it/hidden/769.html.

Статья поступила в редакцию 06.11.2014

Ссылку на эту статью просим оформлять следующим образом:

Деон А.Ф., Терентьев Ю.И. Маргинальные свойства простых сортировок целочисленных массивов. *Инженерный журнал: наука и инновации*, 2014, вып. 11.

URL: http://engjournal.ru/catalog/it/hidden/1309.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» МГТУ им. Н.Э. Баумана. Автор 32 научных трудов. Специализируется в области разработки программного обеспечения автоматизированных систем управления предприятием. e-mail: deonalex@mail.ru

Терентьев Юрий Иванович родился в 1948 г., окончил МВТУ им. Н.Э. Баумана в 1974 г. Канд. техн. наук, доцент кафедры «Программное обеспечение ЭВМ и информационные технологии» МГТУ им. Н.Э. Баумана. Автор 40 научных трудов. Специализируется в области компьютерного моделирования теплофизических процессов в энергетических установках. e-mail: yury_terentev@mail.ru

The marginal properties of simple sorting for integer arrays

© A.F. Deon, Yu.I. Terentiev

Bauman Moscow State Technical University, Moscow, 105005, Russia

The article presents a comparative analysis of marginal speed properties which was carried out for simple sorting elements in integer arrays. We take into account the comparison, addition, transposition and saving operation.

Keywords: *sorting, speed, algorithm, integer numbers.*

REFERENCES

[1] Knuth D. E. *The Art of Computer Programming*. *Vol. 3. Sorting and Searching*, 2009, 824 p.

[2] Sedgewick R. *Algorithms in C++. Parts 1–4. Fundamentals, Data Structures, Sorting, Searching*, 2002, 688 p.

[3] Deon A.F., Terentiev Yu.I. *Inzhenernyi zhurnal: nauka i innovatsii - Engineering Journal: Science and Innovation*, 2013, no. 6(18). Available at: http:/engjournal.ru/catalog/it/hidden/769.html.

Deon A.F. (b. 1951) graduated from Bauman Moscow Higher Technical School in 1974. Ph.D., Assoc. Professor of the Software and Information Technologies Department of Bauman Moscow State Technical University. Author of 32 scientific works. The area of interest is the software development for computer control systems.
e-mail: deonalex@mail.ru

Terentiev Ju.I. (b. 1948) graduated from Bauman Moscow Higher Technical School in 1974. Ph.D., Assoc. Professor of the Software and Information Technologies Department of Bauman Moscow State Technical University. Author of 40 scientific works. The area of interest is the software development for computer simulation of physical warm processes. e-mail: yury_terentev@mail.ru