# Быстрые методы сортировки массивов

Основное требование к методам сортировки массивов – экономное использование памяти. Т.е. перестановку элементов нужно выполнять на одно месте (обмен с использованием вспомогательной переменной или сдвиг). Тогда в качестве меры эффективности сортировки можно выбрать количество необходимых сравнений ключей (С) и пересылок элементов (М).

Рассмотренные ранее простые методы сортировок (простое включение, простой выбор и простой обмен) требуют порядка n2 сравнений ключей.

* Простые методы хорошо подходят для объяснения свойств большинства принципов сортировок.
* Просты для понимания и коротки.
* Хотя сложные методы требуют меньшего числа операций, сами операции более сложные, поэтому на небольших массивах простые методы будут работать быстрее.

## Шейкер сортировка (улучшенная сортировка методом простого обмена)

**Улучшения:**

* Меняются направления проходов по массиву.
* Запоминаются номера последних элементов, между которыми произошел обмен.

static void ShakerSort(int[] mas)

{

int right = mas.Length - 1;

int tek=0;

int direction = 1;// справа налево

for (int left = 1; left < right; )

{

if (direction == 1)

{

direction = -direction;//меняем направление

for ( int i = right; i >=left; i--)

{

if (mas[i] < mas[i - 1])

{

int buf = mas[i];

mas[i] = mas[i - 1];

mas[i-1] = buf;

tek = i;

}

}

left=tek;

}

else

{

direction = -direction;//меняем направление

for (int j = left; j<right; j++)

{

if (mas[j] >mas[j + 1])

{

int buf = mas[j];

mas[j] = mas[j + 1];

mas[j+1] = buf;

tek = j;

}

}

right=tek;

}

}

}

1 проход по массиву: справа налево

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 44 | 55 | 12 | 42 | 94 | 18 | 06 | 67 |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |

2 проход по массиву слева направо

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |

3 проход по массиву справа налево

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |

4 проход по массиву слева направо

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |

5 проход по массиву слева направо

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |

## Сортировка Шелла (на базе метода простого включения)

На первом проходе будем сравнивать и менять местами с помощью сортировки простым включением элементы, которые находятся на расстоянии 4 друг от друга.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **44** | 55 | *12* | 42 | **94** | 18 | *06* | 67 |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |

В результате получим:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **44** | 18 | *06* | 42 | **94** | 55 | *12* | 67 |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |

На втором проходе будем сравнивать и менять местами с помощью сортировки простым включением элементы, которые находятся на расстоянии 2 друг от друга.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **44** | 18 | **06** | 42 | **94** | 55 | **12** | 67 |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |

В результате получим:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **06** | 18 | **12** | 42 | **44** | 55 | **94** | 67 |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |

На третьем проходе будем сравнивать и менять местами с помощью сортировки простым включением элементы, которые находятся на расстоянии 1 друг от друга.

В результате получим:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 06 | 12 | 18 | 42 | 44 | 55 | 67 | 94 |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |

Таким образом, на каждом шаге либо сортируется мало элементов, либо они ужу все хорошо упорядочены и требуется мало перестановок.

Каждый проход будет использовать результаты предыдущего прохода, т.к. каждая i-я сортировка объединяет 2 группы элементов, отсортированные предыдущей сортировкой.

Применима любая последовательность расстояний между элементами, главное, чтобы последнее расстояние было равно 1.

Каждая i-я сортировка программируется как сортировка простыми включениями.

static void ShellSort(int[]mas)

{

const int t=4;

int[]step=new int[t]{9,5,3,1}; //расстояния для сортировки

int j;

for(int m=0; m<t;m++)

{

int k=step[m];//шаг соpтировки

for(int i=k; i<mas.Length;i++)

{

int buf=mas[i];

j=i-k;

while(j>=0&&buf<mas[j])

{

mas[j+k]=mas[j];

j-=k;

}

mas[j+k]=buf;

}

}

}

Затраты на сортировку Шелла пропорциональны n (с\*n1,2).

## Сортировка Хоара

Основывается на методе выбора. Для наибольшей эффективности требуется выполнять обмен между элементами, которые находятся друг от друга на большом расстоянии.

Рассмотрим следующий алгоритм (разделение). Выберем произвольный элемент X и будем двигаться слева направо до тех пор пока не найдам элемент больше Х, затем будем двигаться справа налево, до тех пор, пока не найдем элемент меньше, чем Х. Поменяем местами найденные элементы. Этот процесс повторим до тех пор, пока оба просмотра не встретятся примерно посередине массива. В результате выполнения этого алгоритма массив разделится относительно выбранного элемента на 2 части: в одной будут элементы меньше Х, в другой больше Х.

Теперь применим полученный алгоритм к левой, а потом и к правой части массива. В качестве элемента, относительно которого будут меняться другие элементы, будем брать средний элемент массива или подмассива.

static void sort(int left, int right, int[] mas)

{

int i = left, j = right;//встали на границы

int buf = mas[(left + right) / 2];//элемент, относительно //которого выполняется разделение

do

{

while (mas[i] < buf) i++;

while (mas[j] > buf) j--;

if (i <= j) //обмен

{

int temp = mas[i];

mas[i] = mas[j];

mas[j] = temp;

i++; j--;

}

} while (i <= j);

if (left < j) sort(left, j,mas);

if (i < right) sort(i, right,mas);

}

Эффективность сортировки определяется выбором элемента Х, относительно которого выполняется обмен. В общем случае затраты пропорциональны n ln n (c\* n\* ln(n)).

## Сортировка слиянием

Используется, если надо получить отсортированный массив из уже имеющихся отсортированных массивов.

static void MergeSort( int [] A, int [] B, out int[]C)

{

int lenC=A.Length+B.Length;

C=new int[lenC];

int first=0;

int second=0;

int third=0;

while(first<A.Length&&second<B.Length)

{

if(A[first]<B[second])

{

C[third]=A[first];

first++;third++;

}

else

{

C[third]=B[second];

second++;third++;

}

}

if(first==A.Length)

{

C[third]=B[second];

second++;third++;

}

if(second==B.Length)

{

C[third]=A[first];

first++;third++;

}

}