**Сортировка массивов**

**Сортировка массива** — это процесс распределения всех элементов [**массива**](https://ravesli.com/urok-74-massivy-chast-1/) в определенном порядке. Очень часто это бывает полезным. Например, в вашем почтовом ящике электронные письма отображаются в зависимости от времени получения; новые письма считаются более релевантными, чем те, которые вы получили полчаса, час, два или день назад; когда вы переходите в свой список контактов, имена обычно находятся в алфавитном порядке, потому что так легче что-то найти. Все эти случаи включают в себя сортировку данных перед их фактическим выводом.

## Как работает сортировка?

Сортировка данных может сделать поиск внутри массива более эффективным не только для людей, но и для компьютеров. Например, рассмотрим случай, когда нам нужно узнать, отображается ли определенное имя в списке имен. Чтобы это узнать, нужно проверить каждый элемент массива на соответствие нашему значению. Поиск в массиве с множеством элементов может оказаться слишком неэффективным (затратным).

Однако, предположим, что наш массив с именами отсортирован в алфавитном порядке. Тогда наш поиск начинается с первой буквы нашего значения и заканчивается буквой, которая идет следующей по алфавиту. В таком случае, если мы дошли до этой буквы и не нашли имя, то точно знаем, что оно не находится в остальной части массива, так как в алфавитном порядке нашу букву мы уже прошли!

Не секрет, что есть алгоритмы поиска внутри отсортированных массивов и получше. Применяя простой алгоритм, мы можем искать определенный элемент в отсортированном массиве, содержащем 1 000 000 элементов, используя всего лишь 20 сравнений! Недостатком, конечно же, является то, что сортировка массива с таким огромным количеством элементов — дело сравнительно затратное, и оно точно не выполняется ради одного поискового запроса.

В некоторых случаях сортировка массива делает поиск ненужным. Например, мы ищем наилучший результат прохождения теста среди студентов. Если массив не отсортирован, то нам придется просмотреть каждый элемент массива, чтобы найти наивысшую оценку. Если же массив отсортирован, то наивысшая оценка будет находиться либо на первой позиции, либо на последней (в зависимости от метода сортировки массива: в порядке возрастания или в порядке убывания), поэтому нам не нужно искать вообще!

Сортировка обычно выполняется путем повторного сравнения пар элементов массива и замены значений, если они отвечают заданным критериям. Порядок, в котором эти элементы сравниваются, зависит от того, какой алгоритм сортировки используется. Критерии определяют, как будет сортироваться массив (например, в порядке возрастания или в порядке убывания).

## **Сортировка массивов методом выбора**

Существует множество способов сортировки массивов. Сортировка массивов методом выбора, пожалуй, самая простая для понимания, хотя и одна из самых медленных.

Для **сортировки массива методом выбора от наименьшего до наибольшего элемента** выполняются следующие шаги:

   Начиная с элемента под индексом 0, ищем в массиве наименьшее значение.

   Найденное значение меняем местами с нулевым элементом.

   Повторяем шаги №1 и №2 уже для следующего индекса в массиве (отсортированный элемент больше не трогаем).

Другими словами, мы ищем наименьший элемент в массиве и перемещаем его на первое место. Затем ищем второй наименьший элемент и перемещаем его уже на второе место после первого наименьшего элемента. Этот процесс продолжается до тех пор, пока в массиве не закончатся неотсортированные элементы.

Вот пример работы этого алгоритма в массиве с 5-ю элементами:

{ 30, 50, 20, 10, 40 }

Сначала ищем наименьший элемент, начиная с индекса 0:

{ 30, 50, 20, **10**, 40 }

Затем меняем местами наименьший элемент с элементом под индексом 0:

{ **10**, 50, 20, **30**, 40 }

Теперь, когда первый элемент массива отсортирован, мы его игнорируем. Ищем следующий наименьший элемент, но уже начиная с индекса 1:

{ 10, 50, **20**, 30, 40 }

И меняем его местами с элементом под индексом 1:

{ 10, **20**, **50**, 30, 40 }

Теперь мы игнорируем первые два элемента. Ищем следующий наименьший элемент, начиная с индекса 2:

{ 10, 20, 50, **30**, 40 }

И меняем его местами с элементом под индексом 2:

{ 10, 20, **30**, **50**, 40 }

Ищем следующий наименьший элемент, начиная с индекса 3:

{ 10, 20, 30, 50, **40** }

И меняем его местами с элементом под индексом 3:

{ 10, 20, 30, **40**, **50** }

Ищем следующий наименьший элемент, начиная с индекса 4:

{ 10, 20, 30, 40, **50** }

И меняем его местами с элементом под индексом 4 (выполняется самозамена, т.е. ничего не делаем):

{ 10, 20, 30, 40 **50** }

Готово!

{ 10, 20, 30, 40, 50 }

Обратите внимание, последнее сравнение всегда будет одиночным (т.е. самозамена), что является лишней операцией, поэтому, фактически, мы можем остановить выполнение сортировки перед последним элементом массива.

Пример 1.

**void** BubbleSort(**ref** **int**[] A)

{

**for** (**int** i = 0; i < A.Length; i++)

            {

Int max\_i=i;

**for** (**int** j = 0; j < A.Length - 1; j++)

                {

**if** (A[j] > A[max\_i]

                    max\_i=j;

}

**int** z = A[i];

                 A[i]=A[max\_i];

                 A[max\_i] = z;

             }

}

1. На каждом шаге цикла по i мы знаем, что А [0... i -1 ] уже отсортированы.

2. Индекс наименьшего элемента среди A[i... N-1] будет занесен в max\_i.

3. Если какой-то А[j] окажется больше A[max\_i], надо обновить max\_i — запомнить позицию наименьшего на данный момент элемента.

4. После обмена A[i] и A[max\_i] отсортированы будут уже А[0 ... i].

Внешний цикл сортировки выбором работает с набором данных размером N (всем массивом), а затем размер задействованного набора уменьшается на 1 с каждым оборотом цикла — сначала до К - 1, затем — до N - 2 и так до кон­ца сортировки. Рисунок 1 показывает, что для этого требуется ровно N - 1 обмен.

После N - 1 обменов все элементы оказываются на своих местах, включая последний, A[N - 1], — он не участвовал в поиске минимума, но остался один, и значит, он наибольший. Теперь попробуем посчитать количество сравне­ний — это чуть-чуть сложнее. На рис. 1 сравнений всего 28, что равно сумме чисел от 1 до 7.

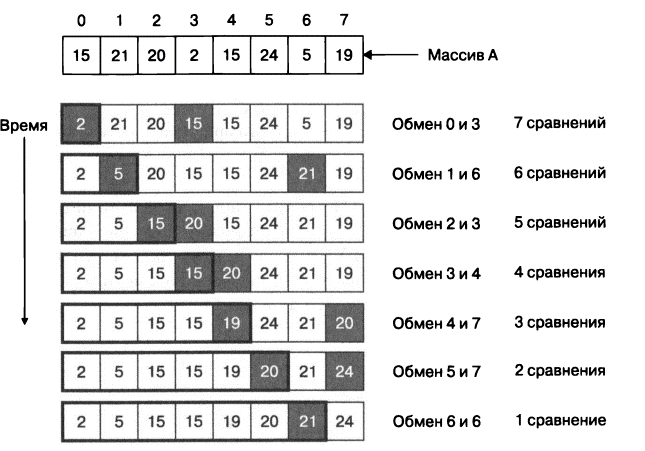


Рис.1. Сортировка произвольного массива выбором

Сумму значений от 1 до К можно вычислить как К х (К + 1) / 2. Такие числа в математике называются треугольными, потому что Кх (К + 1) / 2 предметов можно расположить в виде треугольника. На рис. 2 наглядно представлено само треугольное число 28 и идея, на которой основана формула.

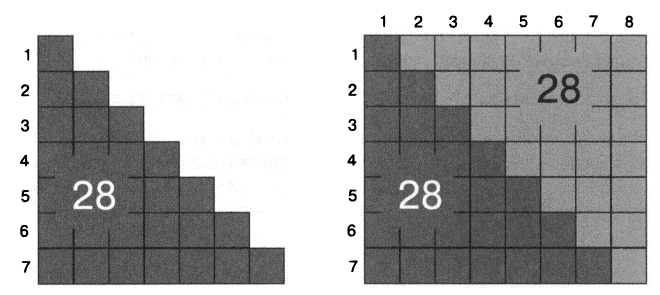


Рис.2. . Наглядное представление формулы треугольного числа: 28 как сумма от 1 до 7

Расположим предметы в виде треугольника из К трок и К элементов в самой длинной строке. Продублируем этот треугольник, перевернем вверх ногами и приложим к исходному. Получим прямоугольник размером К на К+ 1. Выходит, что количество предметов в треугольнике - это ровно половина количества предметов в прямоугольнике. Для треугольника из семи строк двойной прямоугольник будет состоять из 7 х 8 = 56 предметов, значит, в треугольнике их 28. В сортировке К = N - 1 это количество сравнений па первом шаге, необходимое для поиска минимума. Общее число сравнений равно (N - 1) х N / 2.

Пример 2

**int**[] array = new **int**[25];

Array.Sort(array);

Пример 3

**using** System;

**class** Program

{

**public** **static** **void** Main()

    {

**int**[] arr = new **int**[] { 126,548,90,0,21,4,9,6 };

        Array.Sort(arr);

        Array.**ForEach**(arr, Console.WriteLine);

    }

}

## **Сортировка методом пузырька на C#**

***Описание алгоритма:***

Идея данной сортировки заключается в попарном сравнении соседних элементов, начиная с нулевого в массиве. Больший элемент при этом в конце первой итерации оказывается на месте последнего элемента массива, и в следующих итерациях мы его уже не сравниваем его с остальными элементами (то есть у нас будет n-1 сравнений). Затем таким же образом мы находим второй по максимальности элемент и ставим его на предпоследнее место, и т. д. После всех итераций получится, что на месте нулевого элемента окажется элемент с наименьшим числовым значением, а на месте последнего — с наибольшим числовым значением.  Таким образом у нас как бы «всплывают» элементы от большего к меньшему.

*Примечание: можно также реализовать последовательность от меньшего к большему. В коде это лишь замена знака «>» на знак «<» в коде (подробнее в примечании ниже).*

***Пример:***

У нас имеется массив: 7 2 9 4 1 0

Проводим первую итерацию. Мы берем нулевой элемент массива (7) и сравниваем его с соседним:

**7<>2** 9 4 1 0

Так как 7 больше, чем 2, мы меняем эти элементы местами. Получается массив:

2 **7** 9 4 1 0

Далее мы сравниваем 7 и 9.

2 **7<>9** 4 1 0

Число 9 больше, чем 7. Значит 7 остаётся на своём месте. Теперь мы будем сравнивать число 9 с остальными числами и, если 9 будет больше, чем соседние с ним числа, то будет меняться местами с ними.

2 7**9<>4** 1 0

2 7 4 **9<>1** 0

2 7 4 1 **9<>0**

2 7 4 1 0 **9**

Большего числа, чем 9 в нашем массиве не оказалось, поэтому 9 занимает последний элемент в массиве. На этом первая итерация окончена. Теперь мы найдём следующее число, которое по значению будет меньше 9, но больше остальных чисел. Причём нам незачем больше трогать 9, так как оно уже на своём месте и в любом случае у него самое больше числовое значение. Поэтому количество шагов в итерации у нас уменьшается на 1 (n-1).

Вторая итерация. Мы опять «захватываем» нулевой элемент массива (2) и сравниваем его с соседним.

**2<>7** 4 1 0 **9**

Число 2 меньше, чем 7, поэтому они не будут меняться местами, а на следующем шаге будет сравниваться число 7 со следующим соседним элементом.

2 **7<>4** 1 0 **9**

И следующим:

2 4**7<>1** 0 **9**

И следующим:

2 4 1 **7<>0** **9**

Отлично, мы нашли второй элемент, и он занимает предпоследнее место в массиве. Вторая итерация окончена. Опять уменьшаем шаги итерации на 1, так как нам не надо больше сравнивать оставшиеся числа с 7.

После второй итерации наш массив выглядит так:

2 4 1 0 **7 9**

Третья итерация. Повторяем всё точно так же. Начинаем с нулевого элемента и ищем большее значение среди оставшихся чисел.

**2<>4** 1 0 **7 9**

2 **4<>1** 0 **7 9**

2 1 **4<>0** **7 9**

Итого получается:

2 1 0 **4 7 9**

Третья итерация окончена. Начинаем четвертую и не забываем уменьшить шаги итерации на 1.

**2<>1** 0 **4 7 9**

1 **2<>0** **4 7 9**

1 0 **2 4 7 9**

Осталась последняя итерация и сравнение лишь двух чисел:

**1<>0 2 4 7 9**

Сортировка окончена. Наш упорядоченный массив теперь выглядит вот так:

**0 1 2 4 7 9**

***Код программы:***

Программу будем делать в консоли. Для начала создадим функцию самой сортировки (перед функцией main):

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | static int[] BubbleSort(int[] mas)          {              int temp;              for (int i = 0; i < mas.Length; i++)              {                  for (int j = i + 1; j < mas.Length; j++)                  {                      if (mas[i] > mas[j])                      {                          temp = mas[i];                          mas[i] = mas[j];                          mas[j] = temp;                      }                  }              }              return mas;          } |

Мы создаём функцию BubbleSort. В неё будет передан массив mas, который мы заполним числами для сортировки.

Здесь мы сравниваем, так сказать, предыдущий элемент (i) с последующим (j).

Если элемент массива под номером i будет больше, чем элемент массива под номером j, то меняем элементы местами и продолжаем сравнение дальше, как в алгоритме.

*Примечание: выше обсуждалась возможность менять «направление» сортировки, где меньший элемент будет в конце массива, а больший в начале. Для этого надо лишь поменять строку* if (mas[i] > mas[j])*на*if (mas[i] < mas[j]).

Данная функция возвращает отсортированный массив mas (строка 16).

Теперь разберем основную функцию main:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | static void Main(string[] args)          {              Console.WriteLine("Сколько чисел будем сортировать?");              int N = Convert.ToInt32(Console.ReadLine());              Console.WriteLine("Введите числа для сортировки:");              int[] mas=new int[N];              for (int i = 0; i < mas.Length; i++)              {                  mas[i] = Convert.ToInt32(Console.ReadLine());              }              BubbleSort(mas);              Console.WriteLine("После сортировки:");              for (int i = 0; i < mas.Length; i++)              {                  Console.WriteLine(mas[i]);              }              Console.ReadLine();          } |

В строках 3-10 мы считываем информацию, которую введет нам пользователь. Сначала это будет количество элементов в массиве, затем сами значения этих элементов массива.

В строке 11 мы вызываем функцию сортировки, передаём в неё полученный массив, получаем отсортированный массив.

В остальной части кода мы выводим полученный массив на экран.

Вот так будет выглядеть наша программа в консоли:

[](https://vscode.ru/wp-content/uploads/2015/04/2015-04-01-13-30-11-Skrinshot-e%60krana.png)

*Сортировка методом пузырька на C#*