Задачи **сортировок** – весьма популярный пласт задач в программировании. На вход дается последовательность из n-ого числа записей. Каждая **запись** состоит из ключа сортировки и сопутствующих данных. **Ключ сортировки** – это то значение (как правило, числовое), по которому и происходит сортировка последовательности, а **сопутствующие данные**, как нетрудно догадаться, - это все остальные данные, по сути, в сортировке никак не участвующие. Зачастую запись может не содержать никаких сопутствующих данных.\n

Рассмотрим последовательность из 10 записей:\n

(Москва, Россия, 12 000 000), (Нью-Йорк, США, 8 000 000), (Лондон, Великобритания, 9 000 000), (Париж, Франция, 2 000 000), (Рим, Италия, 3 000 000), (Киев, Украина, 3 000 000), (Пекин, Китай, 21 000 000), (Бразилиа, Бразилия, 2 000 000), (Санкт-Петербург, Россия, 5 000 000), (Рио-де-Жанейро, Бразилия, 6 000 000)\n

В ней каждая запись содержит ключ (в данном случае - численность населения) и сопутствующие данные (в данном случае – название города и название страны, в котором он располагается).\n

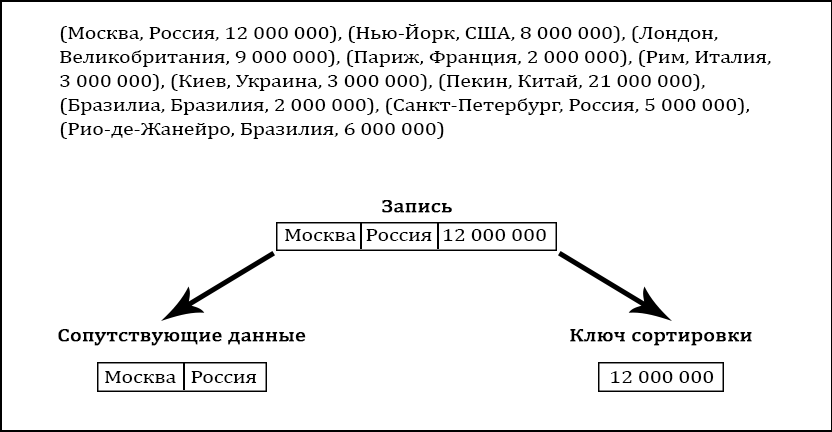
В результате выполнения алгоритма сортировки требуется упорядочить записи в порядке **не убывания** или **не возрастания** (в зависимости от конкретной задачи). \n

Предоставленную на иллюстрации последовательность можно отсортировать несколькими способами (из-за одинаковых ключей в нескольких записях), однако только один способ сортировки будет считаться устойчивым. Алгоритм сортировки считается **устойчивым**, если по его окончанию в последовательности сохраняется относительный порядок элементов с одинаковыми ключами. Так, при использовании устойчивого алгоритма сортировки, записи *(Рим, Италия, 3 000 000)* и *(Киев, Украина, 3 000 000)* с одинаковыми ключами в 3 000 000 населения будут следовать друг за другом строго в том порядке, в каком они были представлены в изначальной последовательности и не в каком другом.\n

Рассмотрим также понятие инверсии. **Инверсия** – это пара ключей с нарушенным порядком следования. Рассмотрим следующую последовательность ключей, которую нужно отсортировать по не убыванию:\n

*{15, 0, 26, 90, 9, 17, 8}* Итоговая последовательность: {0, 8, 9, 15, 17, 26, 90}\n

В ней присутствуют следующие инверсии: (15, 0), (15, 9), (15, 8), (26, 9), (26, 17), (26, 8), (90, 9), (90, 17), (90, 8), (9, 8), (17, 8).

Задачи сортировок – очень популярные задачи как на C#, так и на любом другом языке программирования. Существует огромное множество алгоритмов сортировок, и я выбрал несколько для своей программы. 

Программе дается массив элементов, и ее задача – отсортировать ее по заданному условию. Для простоты обучения и усвоения материала поставим задачу более точно, уточним входные данные.

На вход нам дается массив целых чисел типа int. Этот массив и мы должны отсортировать по возрастанию, то есть первый элемент – наименьший, последний – наибольший, каждый последующий элемент массива должен быть больше предыдущего. Пускай массив будет состоять из 10 случайно сгенерированных элементов от -100 до 100 включительно, чтобы не мозолить глаза большим количеством цифр.

**Сортировка вставками**

**\n\n** Особенности сортировки вставками:\n\n - Сложность в лучшем случае – O(N), в худшем – O(N^2), однако если массив частично упорядочен, то его сложность будет меньше, чем в случае сортировки пузырьком.\n\n - Алгоритм устойчив.\n\n - Как и в алгоритме сортировки пузырьком, для реализации алгоритма не требуется создавать дополнительные массивы, размеры которых зависели бы от размеров начальной последовательности.\n\n - Позволяет отсортировать массив при динамическом добавлении новых элементов. Так называемый, online-алгоритм сортировки, что в определенных ситуациях может быть полезным.

**Сортировка пузырьком**

“ Сортировка пузырьком – пожалуй, самый интуитивно понятный и простой из всех алгоритмов сортировки.\n Идея сортировки пузырьком: Попарно сравниваем соседние ключи, проходя через весь массив до тех пор, пока тот не будет отсортирован.\n Начиная с 1-го элемента, сравниваем его с предыдущим. Если находим несоответствие – то есть ситуацию, в которой i-й элемент меньше элемента i-1 (и наоборот для сортировки по не возрастанию), - то меняем эти элементы местами и идем дальше, пока не дойдем до конца массива. Дойдя до конца, начинаем сначала – и это повторяется до тех пор, пока массив не будет отсортирован, то есть в течение одного из проходов не один элемент не поменяет своего местоположения (это осуществляется при помощи обычной переменной класса bool, которую мы назначаем true в начале каждой итерации, а при наличии перестановки назначаем false).”

\n\nОсобенности сортировки:\n\n - Легка в понимании и реализации.\n\n - Устойчива.\n\n - Сложность в наилучшем случае – O(N), в наихудшем – O(N2).\n\n - Для реализации алгоритма не требуется создавать дополнительные массивы, требуется всего 3-4 переменные.

**Сортировка Шелла**

И последним алгоритмом сортировки в моей учебной программе будет алгоритм сортировки Шелла. В процессе описания работы алгоритма сортировки Шелла, часто будут встречаться отсылки на алгоритм сортировки пузырьком, так что прежде чем приступать к его изучению, пожалуйста, удостоверьтесь, что понимаете принцип его работы.\n\n Итак, алгоритм сортировки Шелла – это довольно легкий в понимании, но далеко не настолько легкий, скорее даже сложный, в реализации алгоритм сортировки. По сути, сортировка Шелла – это усовершенствованная сортировка пузырьком. Вся разница заключается в том, что если в случае с пузырьком, мы сравниваем между собой лишь соседние элементы последовательности, то в случае сортировки Шелла мы сравниваем элементы, находящиеся друг от друга на определенном расстоянии. Например, на рассмотрим сортировку последовательности с перестановкой местами элементов, находящихся на расстоянии 4.\n\n{12, 9, 7, 6, 5, 4, 3, 1} Количество инверсий – 28.\n Переставим местами элемент номер 0 и элемент номер 4 и получим следующую последовательность: {5, 9, 7, 6, 12, 4, 3, 1} В ней количество инверсий – 21.\n Так, всего за одну перестановку, мы уменьшили число инверсий на 7 (а не на 1, как при сортировке пузырьком).\n\n Пройдя весь массив, мы уменьшаем шаг (расстояние между сравниваемыми элементами), чтобы переставить местами те элементы с неправильным порядком следования, который находятся ближе текущего шага относительно друг друга.\n\n Однако как мы уменьшаем этот шаг? Его надо уменьшать не абы как, а по определенной последовательности. Сам Дональд Шелл, придумавший данный алгоритм, предлагал выбирать шаги в виде убывающей геометрической прогрессии: N/2, N/4, N/8, …, 1. Однако, как позже выяснилось, данная последовательность является далеко не самой эффективной. Мы рассмотрим сортировку Шелла с последовательностью h={1, 4, 13, …}, в которой h(n)= h(n-1)\*3+1, считающуюся одной из самых эффективных. Однако, идеальная последовательность до сих пор не была найдена.

\n\n Особенности сортировки Шелла:\n\n - Сложность алгоритма в лучшем случае – O(N), в худшем – зависит от последовательности (в нашем случае – O(N^3/2), однако ее можно снизить до O(N^4/3)).\n\n - Алгоритм неустойчив, а значит может поменять порядок элементов с одинаковыми ключами.\n\n - Число дополнительных переменных не зависит от размера.\n\n - Алгоритм конкурентен популярным алгоритмам сортировки при не очень высоких N.

1 for j ← 2 to length (A)\n2 do key ← A[j]\n3 // Вставка элемента A[j] в отсортированную последовательность A[1..j-1]\n4 i ← j-1\n5 while i> 0 и A[i] > key\n6 do A[i+1] ← A [i]\n7 i ← i-1\n8 A[i+1] ← кеу

"for(int i=1; i<array.Length; i++)\n{\n for(int j = i; j>0 && array[j - 1]>array[j]; j++)\n {\n int temporaryElement = array[j - 1];\n array[j - 1] = array[j];\n array[j] = temporaryElement;\n }\n}"

1 h ←1\n2 while h < N / 9 do h ← h\*3 + 1\n3 repeat // цикл по сериям\n4 for i ← h... N do {сортировка i-ой серии}\n5 j ← i\n6 tmp ← A[i]\n7 while j <= h и tmp < A[j – h] do //сдвиг\n8 A[j] = A[j – h]\n9 j ← j – h;\n10 A[j] ← tmp\n11 h ← h / 3 {переход к новой серии}\n12 while h>0

1 sorted ← false\n2 while sorted == false do\n3 for i ← 1 to N\n4 if A[i] < A[i – 1] then\n5 tmp ← A[i]\n6 A[i] ← A[i – 1]\n7 A[i – 1] ← tmp\n8 sorted = false;

Поздравляю! Вы успешно прошли нашу учебную программу. Вы можете пройти ее снова, перейдя в меню или на конкретную страницу, либо закрыть ее по нажатии на одноименную кнопку. Спасибо за уделенное нам время, надеемся, оно не было потрачено впустую.