Одними из самых базовых алгоритмов информатики и программирования являются алгоритмы сортировки. Сортировка -- это упорядочение набора каких-то элементов в соответствии с неким признаком. Пример: упорядочение студентов 144 группы по росту. Стоит отметить, что с точки зрения сортировщика элементы в этом наборе могут быть и неоднородные, важно лишь, чтобы они все обладали заданным признаком (например, к сортируемым студентам можно добавить еще и мебель в аудитории: у шкафа или стола тоже есть геометрические размеры, а значит, можно и сортировать). В компьютерах сортировка используется очень-очень много где. Вы сами неоднократно упорядочивали файлы на жестком диске по размеру или дате создания, письма в почтовом ящике, фоточки в ненужных соцсетях по количеству лайков и все такое.

Одним из самых важных применений сортировки является поиск информации. Представьте себе телефонный справочник, в котором записи находятся в произвольном порядке. Единственный способ найти там что-то -- проглядеть его весь, и по закону Мёрфи нужное имя конечно же окажется на предпоследней странице (или на второй, если вы решите быть самым умным и начать искать с конца). Но если телефонный справочник отсортировать по имени абонента (как и выглядят настоящие телефонные справочники, если кто не в курсе), то поиск сильно упрощается -- открываете на любой странице и понимаете, в какую сторону нужно листать. На этом принципе основан бинарный поиск, при котором в нашем примере справочник открывается ровно посередине и выбирается половина для дальнейшего поиска, после чего все повторяется уже для нее и так далее, пока не будет найден ответ. Очень часто, когда программа много работает с какими-то данными, имеет смысл их предварительно отсортировать (или использовать специализированную структуру данных типа сбалансированного дерева, но об этом мы будем говорить позже).

Сортировать, как вы понимаете, можно все, что угодно. Если сортируемые элементы сложной структуры, то признак, служащий критерием сортировки, называется ключом сортировки (в примере выше с сортировкой студентов ключом был их рост, в случае справочника -- имя абонента). Для простоты мы будем разбирать сегодня сортировку целых чисел в массиве, то есть ключ и само сортируемое значение будут совпадать.

Чем же отличаются алгоритмы сортировки друг от друга? Основной параметр, характеризующий быстродействие алгоритма -- это его вычислительная сложность. Про это будет следующая лекция, но сейчас можно представить себе вычислительную сложность как число неких абстрактных операций (таких, как сравнение или присваивание), которое должен выполнить алгоритм для сортировки заданного массива размера n. Также ряд алгоритмов требуют выделения дополнительной памяти под временное хранение данных. Алгоритмы сортировки, не потребляющие дополнительной памяти, называют *сортировками на месте*.

Сортировок очень много разных. Многие из них применимы в общем случае для произвольных данных, некоторые создаются специально для конкретных задач (то есть “завязаны” на конкретную структуру или природу данных). Так, например, есть сортировки, предназначенные для работы с огромными объемами данных, расположенных где-то на внешних носителях. Это накладывает на алгоритм особенности -- вы не можете сразу видеть все данные сразу, и доступ к произвольным элементам такого набор данных может

быть крайне затратен по времени. Такие алгоритмы называют *внешними сортировками*. Мы же с вами сегодня будем рассматривать более простые варианты алгоритмов, когда программисту доступен одновременно весь набор данных (*внутренние сортировки*).

Теперь давайте разберем несколько самых известных алгоритмов сортировки.

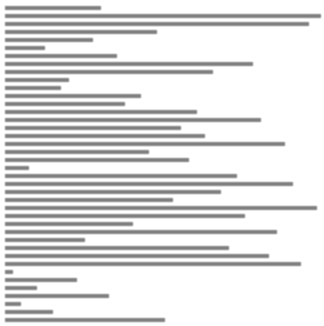
Пузырьковая сортировка

Начнем с самого-самого базового алгоритма -- алгоритма сортировки пузырьком. Для понимания и реализации этот алгоритм — простейший, но эффективен он лишь для небольших массивов. Если кому-то это что-то скажет, сложность алгоритма: $O(n^2)$. Алгоритм считается учебным и практически не применяется вне учебной литературы, вместо него на практике используется что-то более эффективные.

Алгоритм состоит в повторяющихся проходах по сортируемому массиву. За каждый проход элементы последовательно сравниваются попарно и, если порядок в паре неверный, выполняется обмен элементов. Проходы по массиву повторяются до тех пор, пока на очередном проходе не окажется, что обмены больше не нужны, что означает — массив отсортирован. При проходе алгоритма, элемент, стоящий не на своём месте, «всплывает» до нужной позиции как пузырёк в воде, отсюда и название алгоритма.

```
void bubbleSort(int *a, int length)
{
   for (int i = 0; i < length - 1; i++)
   {
      for (int j = 0; j < length - i - 1; j++)
      {
         if (a[j] > a[j + 1])
         {
            int b = a[j];
            a[j] = a[j + 1];
            a[j + 1] = b;
         }
      }
   }
}
```

Пример визуализации работы алгоритма с сайта http://www.sorting-algorithms.com/:



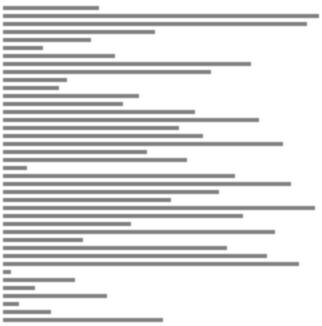
Есть еще ~~абсолютно упоротая~~ забавная визуализация алгоритма в виде танца (смотреть тут).

Сортировка вставками

Чуть более продвинутый алгоритм. На каждом шаге мы выбираем один из элементов входных данных и вставляем его на нужную позицию в уже отсортированном массиве до тех пор, пока набор входных данных не будет исчерпан. Метод выбора очередного элемента из исходного массива произволен, может использоваться практически любой алгоритм выбора. Обычно элементы вставляются по порядку их появления во входном массиве. Приведенный ниже алгоритм использует именно эту стратегию выбора.

```
void insertionSort(int arr[], int length)
{
    for (int i = 1; i < length; i++)
    {
        int j = i;
        while (j > 0 && arr[j - 1] > arr[j]) {
            int tmp = arr[j];
            arr[j] = arr[j - 1];
            arr[j - 1] = tmp;
            j--;
        }
    }
}
```

Пример визуализации:



Адские пляски: http://www.youtube.com/watch?v=ROalU379l3U.

Хотя этот алгоритм сортировки уступает в эффективности более сложным (таким как быстрая сортировка), у него есть ряд преимуществ:

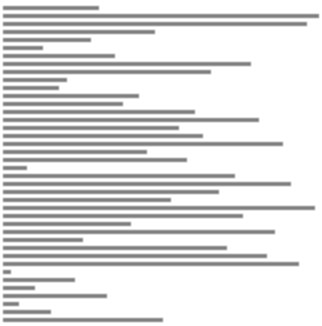
- эффективен на небольших наборах данных, на наборах данных до десятков элементов может оказаться лучшим;
- эффективен на наборах данных, которые уже частично отсортированы;
- это устойчивый алгоритм сортировки (не меняет порядок элементов, которые уже отсортированы);
- может сортировать список по мере его получения.

Минусом же является высокая сложность алгоритма: $O(n^2)$.

Сортировка Шелла

Сортировка Шелла — алгоритм сортировки, являющийся усовершенствованным вариантом сортировки вставками. Идея метода Шелла состоит в сравнении элементов, стоящих не только рядом, но и на определённом расстоянии друг от друга. Иными словами — это сортировка вставками с предварительными «грубыми» проходами.

При сортировке Шелла сначала сравниваются и сортируются между собой значения, отстоящие один от другого на некотором расстоянии d. После этого процедура повторяется для некоторых меньших значений d, а завершается сортировка Шелла упорядочиванием элементов при d = 1 (то есть, обычной сортировкой вставками). Эффективность сортировки Шелла в определённых случаях обеспечивается тем, что элементы «быстрее» встают на свои места (в простых методах сортировки, например, пузырьковой, каждая перестановка двух элементов уменьшает количество инверсий в списке максимум на 1, а при сортировке Шелла это число может быть больше).



Танец: http://www.youtube.com/watch?v=CmPA7zE8mx0.

Сортировка выбором

Шаги алгоритма:

- находим минимальное значение в текущем списке (неотсортированной части);
- производим обмен этого значения со значением на первой неотсортированной позиции;
- сортируем хвост списка, исключив из рассмотрения уже отсортированные элементы.

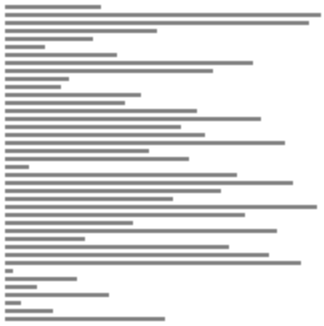
```
void selectionSort(char* a, int size)
{
   for (int i = 0; i < size - 1; i++)
   {
      int minIndex = i;
      for (int j = i + 1; j < size; j++)
      {
         if (a[j] < a[minIndex])
            minIndex = j;
      }

      if (minIndex != i)
         swap(a[i], a[minIndex]);
   }
}
```

Сортировка вставкой делает меньше всего действий с элементами массива по сравнению со всеми остальными сортировками вообще, к тому же может применяться к данным по мере их поступления. Зато сортировка выбором требует наименьшего числа

копирований данных.

Часто спрашивают, когда какую сортировку применять. Бывают случаи, когда “дорогой” является операция копирования, а бывает когда операция сравнения. Выбор надо делать в соответствии с этими соображениями. Пример из реальной жизни — перестановка шкафов в магазине мебели. Померить линейкой их легко (операция сравнения), а таскать туда-сюда — сложно (операция копирования).

Визуализация:



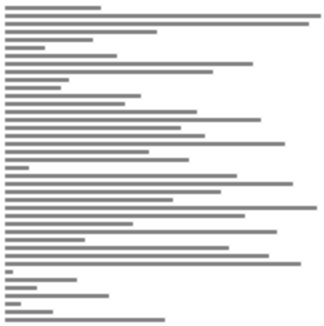
Дергающиеся цыгане: http://www.youtube.com/watch?v=Ns4TPTC8whw.

Быстрая сортировка

Один из быстрых известных универсальных алгоритмов сортировки массивов (в среднем O(n log n) обменов при упорядочении n элементов).

Шаги такие: выбираем некоторое значение v в качестве "опорного элемента", относительно которого переупорядочиваются все элементы массива: меньшие - в начало, большие либо равные - в конец. Получим в итоге какой-то индекс j, такой, что все элементы от 1 до j-1 будут меньше этому опорному, а все от j+1 и дальше -- больше и равные опорному. Повторяем алгоритм для этих двух кусков до тех пор, пока каждый из кусков не станет равным одному элементу. Более подробно алгоритм можно почитать, например, в википедии.

Выглядит это все как-то так:



Есть еще неплохая визуализация этого алгоритма тут: http://sorting.at/.

Ну и безумные венгры, куда без них: http://www.youtube.com/watch?v=ywWBy6J5gz8.