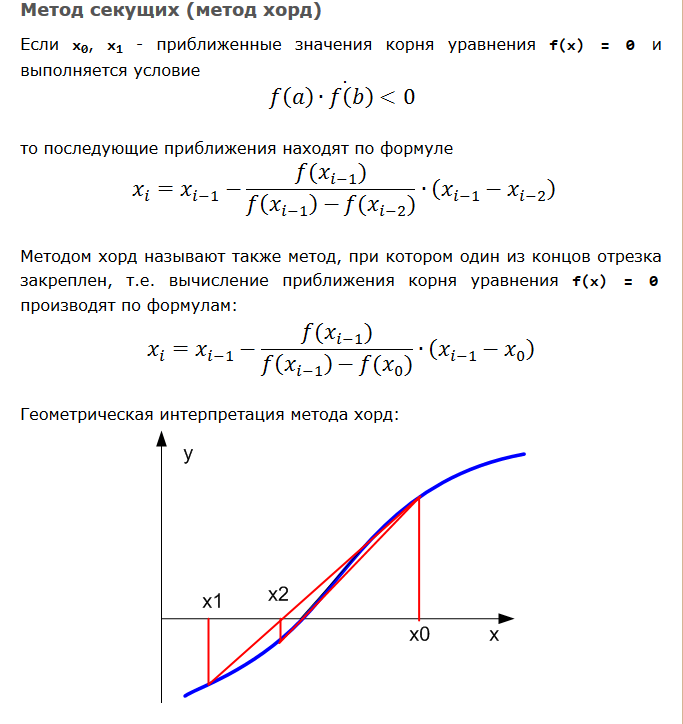
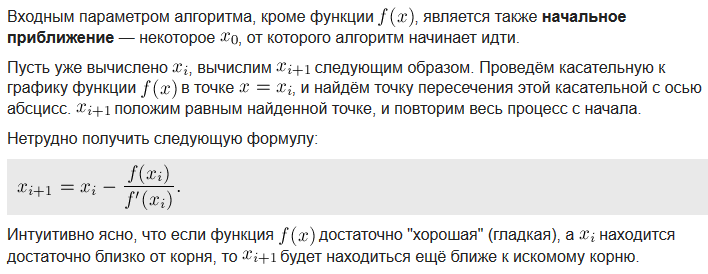
1. *Хеш-таблица* — это ассоциативная структура данных со скоростью доступа O (1). Она состоит из пар вида (ключ, значение) или (объект, индекс в массиве). Она основана на обычном массиве. У неё есть три особенности: можно добавлять пары, искать пары, удалять пары по ключу.  
   Хеш-функция — это функция которая принимает ключ и возвращает его значение. Проще говоря, принимает объект, и возвращает индекс в массиве хеш-таблицы, в котором он должен находиться.   
   Коллизия хеш-таблицы — это ситуация, когда хеш-функция возвращает одно и тоже значение для двух разных ключей.  
   *Открытая адресация* — это способ разрешения коллизии, когда новый ключ добавляется в другой свободный индекс массива. Поиск другой ячейки хеш-таблицы называется пробированием. Недостаток этого способа в целом, это то что при большом количестве совпадений поиск может усложниться до O (n).  
   *Линейное пробирование* — это пробирование по порядку с некоторым фиксированным интервалом. В таком случае ячейки хеш-таблицы могут сгруппироваться в одном месте, и получится обычный массив. Этот недостаток называется кластеризацией.  
   *Квадратичное пробирование* — это пробирование при котором интервал при каждом следующем шаге увеличивается на константу. Такой способ позволяет избежать кластеризации.  
   Двойное хеширование – это способ пробирования когда применяется запасная хеш-функция.   
   *Закрытая адресация* — это способ разрешения коллизий, когда каждому значению в ячейке соответствует список ключей. Тоже имеет склонность усложняться до O (n).
2. *Сортировка*
   1. *Подсчётом*  
      Этот вид сортировки работает только с целыми числами. Скорость O (n2).   
      Сначала находится наибольший элемент массива K. Создаётся вспомогательный массив размером K + 1 (из – за индексации с нуля).  
      После этого мы двигаемся по ячейкам исходного массиву переменной I и увеличиваем значение индекса I во вспомогательном массиве каждый раз на единицу.  
      Потом остается только пройти по вспомогательному массиву и напечатать каждый его индекс столько раз, какое число содержится в его ячейке.
   2. *Поразрядная*Скорость O (n)Создаём столько списков, сколько может быть значений у одного разряда.  
      Разбиваем числа по этим спискам, в зависимости от величины младшего разряда.  
      Соединяем получившиеся списки по порядку.  
      Повторяем это для следующего разряда и так далее пока не пройдём все разряды.
   3. *Блочная*Скорость O (n)  
      Элементы разбиваются по корзинам (блокам) с определённым диапазоном величин в каждой корзине, после чего каждая корзина рекурсивно сортируется снова, или другим алгоритмом сортировки.
   4. *Слиянием*O (n log n)Эта сортировка базируется на нескольких принципах.   
      Каждый массив состоящий из одного элемента является уже отсортированным.   
      Изначально исходный массив разбивается на массивы состоящие из одного элемента.  
      Затем происходит склеивание уже отсортированных массивов по двое, до тех пор, пока они вновь не станут одним массивом.  
      Особенность в том, что при склеивании массивы остаются отсортированными.  
      Это делается так – когда объединяются два массива в один, они начинают сравниваться с первых элементов, поочередно к последним.  
      Таким образом, сортируется весь исходный массив.
3. *Куча* — это дерево у которого значение каждой вершины больше значений потомков. Бывает минимальная куча, у которой значение каждой вершины меньше значений потомков.  
   *Пирамидальная сортировка*  
   Скорость O (n log n)  
   Сначала строится максимальная бинарная куча из массива. Для этого дерево может быть представлено в виде массива, где Array [0] корневой элемент, а потомки каждого элемента определяются так: Array [2i+1] и Array [2i+2].  
   После этого в результирующий список из кучи забирается корневой элемент. Дерево перестраивается опять. Это повторяется до тех пор, пока из дерева по порядку не уйдут все элементы. Так и получается отсортированный список.
4. *Бинарные деревья* — это структура данных где каждый элемент хранит в себе какое-то значение и два указателя на элементы того же типа что и он сам. Значение правого элемента должно быть больше, а левого меньше. O (n) худший случай скорости доступа.  
   *Добавление элемента* в бинарное дерево производится так:  
   Если дерево не пусто, то мы выбираем следующий узел: правый, если элемент больше корня, левый, если меньше. И рекурсивно повторяем с ними эту же процедуру, пока не найдём свободное место.  
   *Удаление элемента* из бинарного дерева по ключу производится так:  
   Сравниваем значение с текущим и двигаемся либовправо, либо влево, пока ключ не совпадёт с текущим элементом.  
   Если нет детей, то обнуляем ссылку на этот элемент.  
   Если есть один ребенок, то ставим его на место отца.  
   Если есть два ребенка, то находим у самого правого потомка левый элемент и заменяем им удаляемый.  
   *Обход дерева* это обход всех его элементов в определённом порядке. К – Корень. Л – Левый. П – Правый. КЛП (префиксная или ещё, польская запись), ЛПК (инфиксная запись), ПКЛ, ЛКП (постфиксная или ещё, обратная польская нотация).  
   Процедуру ЛКП можно написать рекурсивно.  
   *Разбор обратной польской записи в цикле пока есть элементы:*Взять очередной элемент;  
   Если это число, добавить его в стек;  
   Если это знак операции, то взять из стека два операнда; выполнить операцию и записать результат в стек.
5. *AVL-дерево* это двоичное дерево у которого для каждого элемента высота левого поддерева отличается от высоты правого не больше чем на единицу.
7. *B - деревья*
8. *Поиск кратчайшего пути от каждой вершины до каждой:  
   Алгоритм Дейкстры*
9. ***Задача коммивояжера метод ветвей и границ***
10. ***Задача на минимум суммы***
11. ***Задача о наибольшем паросочетании***
12. *Множества*
    1. *Перебор подмножеств*Заведем вектор **B**, состоящий из 4-ёх чисел, каждое из которых может принимать значение 0 или 1.   
       Будем считать, что значение 1 указывает на то, что соответствующий по номеру компонент исходного множества включается в множество, а значение 0 указывает на то, что элемент не включается.
    2. *Перебор размещений из N по K*1) Генерируем все сочетания по K  
       2) Для каждого сочетания генерируем перестановки
    3. *Перебор сочетаний*Выберем наименьшие **M** из **N** чисел и выпишем их в порядке возрастания - **1 2 3** - это начальное сочетание.

Наибольшие **M** из **N** чисел (**3 4 5**), выписанные в порядке возрастания, последнее сочетание.  
Двигаемся от 123 до 345 с участием только цифр 1 2 3 4 5

1. *Решение уравнения f(x) = 0*
   1. *Бинарный поиск*Выберем такие границы, где значение функции точно больше и точно меньше 0. Выберем значение в середине этого отрезка. Если оно меньше, чем заданное, то сместим левую границу в середину отрезка. В противном случае сместим правую границу. Далее повторим процесс сужения границ. Остановимся, когда рассматриваемый отрезок станет меньше заданной погрешности *ε*.
   2. *Метод хорд*
   3. *Метод касательных*
2. Алгоритмы сжатия
   1. *Хаффмана*Составим список кодируемых символов, при этом будем рассматривать один символ как дерево, состоящее из одного элемента c весом, равным частоте появления символа в строке.  
      Из списка выберем два узла с наименьшим весом.  
      Сформируем новый узел с весом, равным сумме весов выбранных узлов, и присоединим к нему два выбранных узла в качестве детей.  
      Добавим к списку только что сформированный узел вместо двух объединенных узлов.  
      Если в списке больше одного узла, то повторим пункты со второго по пятый.
   2. *RLE* (количество, символ). Оптимизация до *(*количество неповторяющихся, символы)
   3. LZW  
      