Ну значит ебать это документик, в который я запишу ВСЁ, что я знаю по алгоритмам.

1) Асимптотика алгоритмов

O() - с помощью символа можно математически описать то, как поведет себя программа в условиях наихудшего сценария при большом количестве входных данных.

Θ (тета) - с помощью этого символа можно математически описать точную оценку алгоритма.

Ω (омега) - с помощью символа можно математически описать то, как поведет себя программа в условиях наилучшего сценария при сравнительно небольшом количестве входных данных. (Например: отсортированный массив в сортировке вставками)

Асимптотика для пунктов 2.1, 2.2, 2.3: в наихудшем O(n^2), в наилучшем O(n) (отсортир. массив).

2.1) Сортировка выбором

Проходим по массиву в поисках максимального (или минимального) элемента. Найденный максимум (или минимум) меняем местами с последним (или первым) элементом. Неотсортированная часть массива уменьшилась на один элемент (не включает последний элемент, куда мы переставили найденный максимум (или минимум)). К этой неотсортированной части применяем те же действия. (Кстати если мы будем одновременно искать и максимум, и минимум и вставлять их, то это будет шейкерная сортировка)

2.2) Сортировка вставками (при использовании бинарного поиска O(nlogn))

Проходим по массиву слева направо и обрабатываем по очереди каждый элемент. Слева от очередного элемента наращиваем отсортированную часть массива, справа по мере процесса потихоньку испаряется неотсортированная. В отсортированной части массива ищется точка вставки для очередного элемента. Сам элемент отправляется в буфер, в результате чего в массиве появляется свободная ячейка — это позволяет сдвинуть элементы и освободить точку вставки.

Дополнение к 2.1 и 2.2)

Главное отличие: в сортировке вставками мы извлекаем из неотсортированной части массива любой элемент и вставляем его на своё место в отсортированной части. В сортировке выбором мы целенаправленно ищем максимальный элемент (или минимальный), которым дополняем отсортированную часть массива. Во вставках мы ищем куда вставить очередной элемент, а в выборе — мы заранее уже знаем в какое место поставим, но при этом требуется найти элемент, этому месту соответствующий.

2.3) Сортировка пузырьком

Просматриваем массив слева-направо и по пути сравниваем соседей. Если мы встретим пару взаимно неотсортированных элементов, то меняем их местами и возвращаемся в самое начало. Снова проходим-проверяем массив, если встретили снова «неправильную» пару соседних элементов, то меняем местами и опять начинаем всё сызнова. Продолжаем до тех пор, пока массив потихоньку-полегоньку не от сортируется.

3.1) Сортировка слиянием

Асимптотика: во всех случаях – O(nlogn).

Таким образом, сортировка слиянием подразумевает разбиение массива поровну до тех пор, пока из одного массива не получится несколько мелких — размером не более двух элементов. Два элемента легко сравнить между собой и упорядочить в зависимости от требования: по возрастанию или убыванию.  
После разбиения следует обратное слияние, при котором в один момент времени (или за проход цикла) выбираются по одному элементу от каждого массива и сравниваются между собой. Наименьший (или наибольший) элемент отправляется в результирующий массив, оставшийся элемент остается актуальным для сравнения с элементом из другого массива на следующем шаге.

3.2) Сортировка быстрая сортировка

Асимптотика: в лучшем и среднем случае – O(nlogn), в худшем O(n^2)

Доп. память: O(logn)

Выбираем опорный элемент. Разбиваем массив на 2 части. Создаём переменные l и r — индексы соответственно начала и конца рассматриваемого подмассива. Увеличиваем l, пока l-й элемент меньше опорного. Уменьшаем r, пока r-й элемент больше опорного. Если l всё ещё меньше r, то меняем l-й и r-й элементы местами, инкрементируем l и декрементируем r. Если l вдруг становится больше r, то прерываем цикл. Повторяем рекурсивно, пока не дойдём до массива из 1 элемента.

4) Нахождение количества инверсий в массиве.

Это почти обычная сортировка слиянием, вся магия скрыта в функции слияния. Когда мы сливаем обе части, мы сравниваем элементы одной (первой, левой) части с элементами другой (правой, второй) части соответственно. И если элемент левой части больше элемента правой части соответственно, то значит это и есть инверсия.  
И так же все оставшиеся элементы левой части тоже будут больше, т.к. левая и правая часть отсортированы. Поэтому количество инверсий нужно увеличить на количество оставшихся элементов + 1 (текущий элемент).

5) Нахождение K-ой порядковой статистике (По факту – это бинарный поиск на неотсортированном массиве)

Будем использовать процедуру рассечения массива элементов из алгоритма сортировки QuickSort. Пусть нам надо найти k-ую порядковую статистику, а после рассечения опорный элемент встал на позицию m. Возможно три случая:

**k = m**. Порядковая статистика найдена.

**k < m**. Рекурсивно ищем k-ую статистику в первой части массива.

**k > m**. Рекурсивно ищем (k−m−1)(k−m−1)-ую статистику во второй части массива.

При реализации, однако, вместо рекурсивных вызовов изменяются границы поиска статистики во внешнем цикле. В коде считаем, что процедура **partition** принимает массив и границы отрезка, который будет рассечён (причём правая граница отрезка не включается), и возвращает индекс опорного элемента. Также считается, что массив индексируется с нуля.

6) Двоичный поиск

Асимптотика: O(logn)

Бинарный поиск производится в упорядоченном массиве.  
При бинарном поиске искомый ключ сравнивается с ключом среднего элемента в массиве. Если они равны, то поиск успешен. В противном случае поиск осуществляется аналогично в левой или правой частях массива.

7) Куча, она же Пирамида

Асимптотика:

Нахождение минимума в неубывающей О(1), в невозрастающей О(n).

Извлечения минимума О(logn).

Добавление элемента О(logn).

Изменение элемента О(logn).

Двоичная куча представляет собой полное бинарное дерево, для которого выполняется *основное свойство кучи*: приоритет каждой вершины больше приоритетов её потомков.

8) Пирамидальная сортировка

Асимптотика: O(nlogn)

Постройте max-heap из входных данных.

На данном этапе самый большой элемент хранится в корне кучи. Замените его на последний элемент кучи, а затем уменьшите ее размер на 1. Наконец, преобразуйте полученное дерево в max-heap с новым корнем.

Повторяйте вышеуказанные шаги, пока размер кучи больше 1.

9) Сортировка подсчётом

Асимптотика: О(n + k), где k – кол-во уникальных элементов в массиве.

Подсчитываем сколько раз в массиве встречается каждое значение и заполняем массив подсчитанными элементами в соответствующих количествах. Счётчики для всего диапазона чисел создаются заранее (изначально равны нулю).

Базовый алгоритм. В "чистом" виде не встречается, однако лежит в основе многих сортировок распределением. В частности, сортировка царя Соломона, мелькающая сортировка и бисерная сортировка являются разновидностями данного метода.

10) Цифровая сортировка

Асимптотика: О(n \* k), где k – максимальное кол-во разрядов в числе .

LSD radix sort

Элементы перебираются по порядку и группируются по самому младшему разряду (сначала все, заканчивающиеся на 0, затем заканчивающиеся на 1, ..., заканчивающиеся на 9). Возникает новая последовательность. Затем группируются по следующему разряду с конца, затем по следующему и т.д. пока не будут перебраны все разряды, от младших к старшим.

MSD radix sort

Элементы перегруппировываются по определённому разряду (сначала по самому старшему). Затем разбиваются на подгруппы в зависимости от значения этого разряда: равного 0, равного 1, равного 2, ..., равного 9. Каждая подгруппа обрабатывается отдельно, в ней к следующему разряду рекурсивно применяется radix sort.

MSD реализовывается несколько сложнее чем LSD, но при этом она эффективнее. При ориентации на наименьшие значащие цифры для всех элементов обрабатываются все разряды. А вот в случае наибольших значащих цифр рекурсия продолжается только до той глубины, до которой это необходимо, то есть пока у элементов подгруппы есть различия в определённом разряде.

11.1) Стек

Это структура данных на котором действует правило: первый вошёл, последний вышел.

11.2) Очередь

Это структура данных на котором действует правило: первый вошёл, первым вышел.

12) Хеш-таблица

Хеш-таблица — это контейнер. Мы определяем функцию хеширования, которая по каждому входящему элементу будет определять натуральное число. А уже дальше по этому натуральному числу мы будем класть элемент в (допустим) массив.

Возникает проблема коллизии, или проблемы, когда хеш-функция выдает одинаковое натуральное число для разных элементов. Существует несколько решений данной проблемы: метод цепочек и метод двойного хеширования. В данной статье я постараюсь рассказать о втором методе, как о более красивом и, возможно, более сложном.

При закрытой адресации у нас каждый элемент в хеш-таблицы – это цепочка из элементов у который получился одинаковый хеш.

При открытой адресации у нас есть только один массив – при вставке элемента в занятую ячейку, мы вставляем новый элемент через какой-то шаг.