

Вопрос	Что это	Асимптотика	На чем реализуется	Ограничения	Применимость
Оценка сложности по времени	<p>Это метод, который позволяет оценить время работы написанного кода.</p> <ul style="list-style-type: none"> • 1 операция - 1 пункт сложности • 1 итерация = k операций в итерации - k пунктов сложности • Условие = t логических операций и действий - t пунктов сложности <p>Обозначается $T(n)$</p> <p>Обычно алгоритм оценивают сверху ($O(n)$) и снизу ($\Omega(n)$) (асимптотическая оценка), а также приводят среднюю асимптотику, наблюдаемую в большинстве случаев.</p>	-	-	-	В любой программе
Оценка сложности по памяти	<p>Это метод, который позволяет оценить память, которую использует программа, написанная данным кодом.</p> <ul style="list-style-type: none"> • 1 переменная - 1 пункт сложности • Массив размером n - n пунктов сложности • Строка длиной n - n пунктов сложности <p>Обозначается как $M(n)$</p>	-	-	-	В любой программе
Сортировка вставками	<p>Эта устойчивая сортировка, работающая по следующему принципу (опишем принцип сортировки по возрастанию):</p> <ol style="list-style-type: none"> 1. Пусть массив теперь делится на 2 части: отсортированную (первый элемент) и не отсортированную (остальные элементы), будем хранить конец отсортированной части в переменной k 2. Берется указатель (или индекс) числа (пусть это будет $i = k + 1$), стоящего после отсортированной части. 3. Другим указателем (пусть это будет j) идем по отсортированной части ($j = k$ и до 0), проверяем, значение элемента $m[j] > m[i]$: <ul style="list-style-type: none"> а. если да, то меняем их друг с другом (swap ($m[i]$, $m[j]$)), увеличиваем i на 1 и переходим опять на пункт 2 б. иначе присваиваем $m[j] = m[j - 1]$, уменьшаем j на 1, повторяем пункт 3 	<p>ПО ВРЕМЕНИ</p> <p>Лучший: $O(n)$ Средний: $O(n^2)$ Худший: $O(n^2)$</p> <p>ПО ПАМЯТИ</p> <p>$O(1)$, т.к. мы храним только указатель на конец отсортированной части</p>	Реализуется на массиве исходных данных (например, чисел)	Ограничения могут быть только на объем входных данных	В задачах, где необходимо отсортировать массив из любых элементов, может понадобиться как проверка работы более сложных сортировок

Сортировка слиянием (merge sort)	<p>Это устойчивая (зависит от реализации, но реализация устойчивой и неустойчивой очень схожа) сортировка, работающая по принципу «Разделяй и властвуй» через рекурсию (на спуске делим, на подъеме сортируем и объединяем):</p> <ol style="list-style-type: none"> 1. Берется массив и делится на 2 части 2. Сортируем 1-ю часть и 2-ю часть так: <ul style="list-style-type: none"> • Если длина частей = 1, то они уже отсортированы • Иначе работаем с обеими частями как в пункте 1 3. Объединяем 2 отсортированные части так, чтобы новый массив тоже был отсортирован методом 2-х указателей: указатели i и j указывают на начало 2-х частей, сравниваем $a[i] < b[j]$, если да, то кладем $a[i]$ в новый массив и увеличиваем i на 1, иначе кладем $b[j]$ в новый массив и увеличиваем j на 1. Когда одна из частей закончилась ($i == n$ or $j == m$), кладем оставшиеся числа из другой части в новый массив и возвращаем новый массив как ответ. 	<p>ПО ВРЕМЕНИ $O(n \log(n))$, т.к. мы пройдемся по всем n элементам массива $\log n$ раз (глубина рекурсии)</p> <p>ПО ПАМЯТИ $O(n)$, т.к. нужен только доп. массив</p>	Реализуется на рекуррентном спуске по массиву	Ограничение по глубине рекурсии	
----------------------------------	---	--	---	---------------------------------	--

Быстрая сортировка (quick sort)	<p>Это неустойчивая сортировка, работающая по одному из следующих принципов:</p> <p>I. Разбиение Хоара</p> <ol style="list-style-type: none"> 1. На каждой итерации в массиве выбирается опорный элемент из середины, относительно которого сортируется массив (для улучшения асимптотики берется случайное число из массива и меняется с центральным, это одна из возможных оптимизаций кода). 2. Заводится 2 указателя i и j – на начало и конец массива соответственно, при этом считаем, что все элементы, которые находятся слева от опорного ДОЛЖНЫ быть меньше него, а справа ДОЛЖНЫ быть больше него. 3. Дальше повторяем пункты, пока $i \leq j$: <ul style="list-style-type: none"> • Пока элемент по индексу i меньше, чем опорный, увеличиваем i на 1 • Пока элемент по индексу j больше, чем опорный, уменьшаем j на 1 • Если $i \leq j$, то меняем элементы по этим индексам местами, увеличиваем i на 1, уменьшаем j на 1 и возвращаемся на пункт 3 4. Запускаем сортировку заново, но теперь на отрезке массива от L до j и от i до R, где L, R – начало и конец массива соответственно. <p>II. Разбиение Ломута</p> <ol style="list-style-type: none"> 1. На каждой итерации опорным элементом является последний в массиве. 2. Заводится 2 указателя i и j, где i указывает на конец части массива, в которой все элементы меньше или равны опорному (изначально = -1, т.к нет пока такой части), j указывает на начало массива. 3. Пока $j < \text{len}(\text{массив})$: <ul style="list-style-type: none"> • Если элемент по индексу $j \leq$ опорного, то мы добавляем его в отсортированную часть, т.е. увеличиваем индекс i на 1 и меняем элементы на индексах i и j <ul style="list-style-type: none"> • Иначе ничего не делаем • Увеличиваем j на 1 	<p>ПО ВРЕМЕНИ</p> <p>Лучший: $O(n \log n)$ Средний: $O(n \log n)$ Худший: $O(n^2)$</p> <p>ПО ПАМЯТИ</p> <p>Лучший: $O(\log n)$ Средний: $O(\log n)$ Худший: $O(n)$</p>	Реализуется на рекуррентном вызове сортировки на отдельных частях массива	Ограничение по глубине рекурсии	Для быстрой сортировки (как ни странно) массива любых элементов
---------------------------------	---	--	---	---------------------------------	---

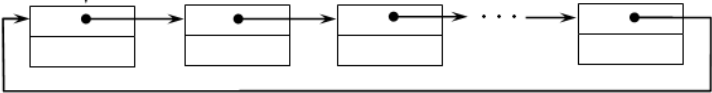
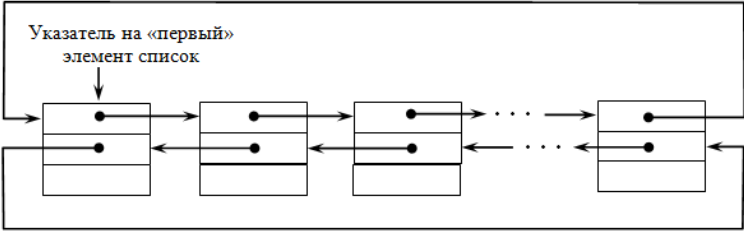
	4. Запускаем сортировку на отрезках от L до i и от i + 1 до R, где L, R – начало и конец массива				
--	--	--	--	--	--

Сортировка подсчетом	<p>Это линейная сортировка (устойчивость зависит от реализации), которая работает по следующему принципу:</p> <p>I. Неустойчивая реализация</p> <ol style="list-style-type: none"> 1. Заводим массив-алфавит, где храним данные в виде: возможное число – его кол-во в исходном массиве (массив алфавит должен быть отсортирован по индексам) 2. Проходимся по начальному массиву и заполняем массив-алфавит 3. Проходимся по массиву алфавиту, и пока значение по i-тому индексу > 0 выводим индекс i и уменьшаем значение по i-тому индексу на 1, затем увеличиваем индекс i на 1 <p>II. Устойчивая реализация</p> <ol style="list-style-type: none"> 1. Заводим массив-алфавит а так же массив, куда будем записывать ответ 2. Проходимся по исходному массиву и заполняем массив алфавит так же, как в неустойчивой реализации 3. Теперь, начиная с $i = 1$ и до конца массива-алфавита, проходимся по массиву-алфавиту и делаем $C[i] = C[i] + C[i - 1]$ – так мы создадим массив, в котором для i-того элемента будет храниться индекс последнего элемента в массиве ответов B 4. Теперь проходимся по начальному массиву C КОНЦА, делаем так: ставим число $A[i]$ из изначального массива на позицию, которая численно равна значению $C[A[i]]$ от этого элемента, т.е. $B[C[A[i]]] = A[i]$ и уменьшаем $C[i]$ на 1 <p>Модификации:</p> <ol style="list-style-type: none"> 1) если не известен диапазон чисел, можно за линию найти минимум и максимум в массиве, тогда числа от минимума до максимума будут составлять диапазон 2) если числа отрицательные, можно завести дельту, на которую сначала будем увеличивать все числа, а при добавлении в ответ уменьшать, чтобы индексы были неотрицательными 	<p>K – размер алфавита, т.е. кол-во всех возможных чисел</p> <p>I-я реализация ПО ВРЕМЕНИ Лучший: $O(n + k)$ Средний: $O(n + k)$ Худший: $O(n + k)$</p> <p>ПО ПАМЯТИ Лучший: $O(k)$ Средний: $O(k)$ Худший: $O(k)$</p> <p>II-я реализация ПО ВРЕМЕНИ Лучший: $O(n)$ Средний: $O(n)$ Худший: $O(n)$</p> <p>ПО ПАМЯТИ Лучший: $O(n + k)$ Средний: $O(n + k)$ Худший: $O(n + k)$</p>	Реализуется на массиве-алфавите, в котором хранится кол-во каждого возможного числа из массива и, если используем 2-ю реализацию, на массиве ответа	Не всегда известен алфавит, например, если надо сортировать какие-то сложные структуры или дробные числа	Если надо очень быстро отсортировать числа, и при этом известен диапазон, откуда они берутся
----------------------	--	--	---	--	--

Цифровая сортировка	<p>Это линейная устойчивая сортировка, которая работает по следующему принципу:</p> <ol style="list-style-type: none"> 1. У каждого числа смотрим на разряд единиц. 2. Используем УСТОЙЧИВУЮ линейную сортировку (если не устойчивая, то сортировка работать не будет, т.к. например на числах 112 и 312 она спокойно сможет ошибиться), опираясь на цифры (алфавит будет от 0 до 9) 3. Повторяем с пункта 1, но теперь смотрим на десятки и так далее, пока не дойдем до самого старшего разряда среди всех чисел включительно <p>Так же данная сортировка может сортировать и строки, тогда алфавитом будет алфавит (простите за тавтологию), а вместо разрядов мы смотрим сначала на последнюю букву, затем на предпоследнюю и так далее</p>	<p>d – максимальное число разрядов, $T(n)$ – время работы линейной сортировки, в нашем случае подсчетом ПО ВРЕМЕНИ</p> <p>Лучший: $O(d \cdot T(n)) = O(d \cdot n)$</p> <p>Средний: $O(d \cdot T(n)) = O(d \cdot n)$</p> <p>Худший: $O(d \cdot T(n)) = O(d \cdot n)$</p> <p>ПО ПАМЯТИ</p> <p>Лучший: $O(n + k)$</p> <p>Средний: $O(n + k)$</p> <p>Худший: $O(n + k)$</p>	Реализуется на устойчивой линейной сортировке по разрядам	Можно сортировать только числа и строки	Быстрая сортировка чисел или строк
---------------------	--	--	---	---	------------------------------------

Стек	<p>Это структура, работающая по принципу First Input Last Output, т.е. элемент, который положили в стек первым, достанут оттуда последним (как пирамидка детская, если уж совсем не понятно)</p> <p>Реализация основных команд стека:</p> <p>1) Добавить элемент: Увеличиваем указатель конца стека на 1 и на это место ставим элемент</p> <p>2) Достать элемент: Сверяем указатель на начало и конец стека, если конец \geq начало, то возвращаем элемент, на который указывает указатель конца стека</p> <p>3) Удалить элемент: Сверяем указатель на начало и конец стека, если конец \geq начало, то уменьшаем указатель конца на 1</p> <p>4) Найти элемент: Достаем значение из стека и удаляем его, пока не найдем нужный элемент и пока размер стека > 0</p>	<p>Достать элемент: $O(1)$ Положить элемент: $O(1)$ Найти элемент: $O(n)$</p>	<p>Реализуется на массиве, где будут храниться элементы, а так же создаются 2 указателя – на начало и конец стека</p>	<p>Ограничен по памяти, т.е. необходимо выделить какое-то фиксированное кол-во памяти при реализации сразу</p>	<p>Для различных задач, например для скобочной последовательности или постфиксной записи</p>
------	--	--	---	--	--

Очередь	<p>Это структура, работающая по принципу First Input First Output, т.е. элемент, который положили в очередь первым, достанут оттуда первым (очередь все знают)</p> <p>Реализация основных команд очереди:</p> <p>1) Добавить элемент: Увеличиваем указатель конца очереди на 1, если указатель выходит за пределы массива, то приравниваем его к нулю, и на это место ставим элемент</p> <p>2) Достать элемент: Сверяем указатель на начало и конец стека, если конец != начало, то возвращаем элемент, на который указывает указатель начала очереди</p> <p>3) Удалить элемент: Сверяем указатель на начало и конец стека, если конец != начало, то увеличиваем указатель начала на 1, если указатель вышел за пределы массива, то приравниваем его к нулю</p> <p>4) Найти элемент: Достаем значение из очереди и удаляем его, пока не найдем нужный элемент и пока размер очереди > 0</p>	<p>Достать элемент: $O(1)$ Положить элемент: $O(1)$ Найти элемент: $O(n)$</p>	<p>Реализуется на массиве, где будут храниться элементы, а так же создаются 2 указателя – на начало и конец очереди</p>	<p>Ограничена по памяти, т.е. необходимо выделить какое-то фиксированное кол-во памяти при реализации сразу</p>	<p>Для решения различных задач</p>
Односвязный список	<p>Это структура, элементы которой могут храниться в любых ячейках памяти, т.е. она динамически выделяет память для элементов</p> <p>Конечный элемент всегда ссылается на что-то, что указывает, что это конец списка (например, на nullptr или на себя же)</p> <p>1) Добавить элемент: Создаем элемент, который указывает на начало списка, и меняем указатель начала списка на этот элемент</p> <p>2) Достать элемент: Вернуть значение, на которое указывает указатель начала списка</p> <p>3) Удалить элемент: Указатель начала меняем на адрес следующего элемента, на который ссылался начальный элемент (т.е. $tail = tail.next$) и удаляем старый элемент</p> <p>4) Поиск элемента как в стеке/очереди</p>	<p>Достать элемент: $O(1)$ Положить элемент: $O(1)$ Найти элемент: $O(n)$</p>	<p>Реализуется на структуре элемента, который хранит в себе значение и указатель на следующий (или предыдущий) элемент, а так же указатель на начало (конец)</p>	<p>-</p>	<p>Для задач, где начальный размер массива неизвестен и нужно динамически выделять память</p>

Двусвязный список	<p>Это структура, которая схожа с односвязным списком, но каждый элемент хранит еще и предыдущий элемент</p> <p>Все операции схожи с односвязным списком, но их можно разворачивать (например, добавить в конец или взять из начала и т.д.)</p>	<p>Достать элемент: $O(1)$</p> <p>Положить элемент: $O(1)$</p> <p>Найти элемент: $O(n)$</p>	<p>Реализуется на структуре элемента, который хранит в себе значение и указатель на следующий и предыдущий элемент, а так же указатель на начало и конец</p>	-	<p>Может понадобиться для реализации двухсторонней очереди, а так же для реализации стека и очереди при помощи всего одной структуры</p>
Циклический список	<p>Это либо односвязный список, конец которого ссылается на начало, либо двусвязный список, следующий элемент конца и предыдущий элемент начала которого равны ссылке на начало и конец соответственно</p> <p>Операции схожи с односвязным/двусвязным списками с учетом того, что надо всегда при добавлении/удалении элемента сохранять цикличность списка</p> <p>На односвязном списке:</p> <p>Указатель на «первый» элемент списка</p>  <p>На двусвязном списке:</p> 	<p>Достать элемент: $O(1)$</p> <p>Положить элемент: $O(1)$</p> <p>Найти элемент: $O(n)$</p> <p>Сдвинуть циклически массив на x: $O(1)$</p>	<p>Реализуется на односвязном или двусвязном списке, конец и начало которых связаны</p>	-	<p>Например, задача на циклический сдвиг, или если надо решать задачу с числами, которые стоят по кругу</p>
Стек на списках	<p>Это реализация структуры стека на односвязном списке, по факту односвязный список – это и есть стек</p>	<p>Достать элемент: $O(1)$</p> <p>Положить элемент: $O(1)$</p> <p>Найти элемент: $O(n)$</p>	<p>Реализуется на односвязном списке</p>	-	<p>То же, что и у обычного стека, но память динамическая</p>

Очередь на списках	Это реализация структуры очереди на двухсвязном списке, но так же удаление и получение элемента происходит с последним, а не с начальным элементом	Достать элемент: $O(1)$ Положить элемент: $O(1)$ Найти элемент: $O(n)$	Реализуется на двухсвязном списке	-	То же, что и у обычной очереди, но память динамическая
Бинарный поиск	<p>Это метод поиска элемента V в отсортированном по какому-то критерию массиве (нам пока важно знать, что массив просто отсортирован, до бин поиска по ответу мы еще не дошли), реализация:</p> <ol style="list-style-type: none"> 1. Берем начало отрезка массива L и R (изначально 0 и $n - 1$ соответственно), повторяем все пункты пока $R > L + 1$ 2. Теперь находим элемент M (M – индекс среднего элемента), стоящий в середине отрезка, т.е. $M = (L + R) / 2$ 3. Теперь в зависимости от того, что мы хотим, сравниваем значение, стоящее на середине отрезка, и данный элемент V: <ul style="list-style-type: none"> • Если мы ищем первое вхождение элемента в массиве (например, в массиве $1, 2, 3, 3, 3, 3, 4$ первое вхождение элемента 3 на позиции 2, т.к. индексация с нуля, а последнее вхождение на позиции 6), то если значение среднего элемента СТРОГО МЕНЬШЕ, чем V, то делаем $L = M$, иначе $R = M$ • Если мы ищем последнее вхождение элемента в массиве, то если значение среднего элемента МЕНЬШЕ ИЛИ РАВНО, чем V, то делаем $L = M$, иначе $R = M$ 4. Повторяем все то же самое, начиная с пункта 2 5. Возвращаем индекс найденного элемента или -1 если не нашли его 	Поиск элемента: $O(\log n)$, потому что при каждой новой итерации мы работаем с массивом, меньшим предыдущего ровно в 2 раза \rightarrow кол-во вызовов бин поиска = $\log n$	Реализуется на списке и указателях на начало и конец отрезка	Можно применять ТОЛЬКО на отсортированном массиве	Поиск первого и последнего элемента в массиве, проверка на нахождение элемента в массиве и т.д.