

Распиши коротко (основная суть) каждый из билетов и пришли в этот чат.

Задача 1. Алгоритм быстрой сортировки QuickSort. Проиллюстрировать работу алгоритма на примере последовательности [31, 21, 24, 29, 29, 31, 11, 27, 24, 27].

Задача 2. Модифицируйте алгоритм QuickSort для сортировки в порядке невозрастания. Проиллюстрировать работу алгоритма на примере последовательности [39, 31, 27, 13, 17, 18, 14, 20, 31, 17].

Задача 3. Чему равно время работы алгоритма QuickSort, когда все элементы массива одинаковы по величине.

Задача 4. Покажите, что если все элементы массива различны и расположены в убывающем порядке, то время работы процедуры QuickSort равно  $\Theta(N^2)$ .

Задача 5. Алгоритм пирамидальной сортировки HeapSort. Проиллюстрировать работу алгоритма на примере последовательности [31, 32, 34, 13, 28, 20, 28, 36, 30, 20].

Задача 6. Алгоритм сортировки подсчётом CountingSort. Проиллюстрировать работу алгоритма на примере последовательности [4, 4, 5, 4, 8, 4, 6, 3, 5, 2, 8, 1, 3, 5, 2].

Задача 7. Алгоритм QFindStatP поиска порядковой статистики за время  $\Theta(N)$  в среднем. Проиллюстрировать работу алгоритма на примере последовательности [36, 39, 37, 25, 12, 12, 11, 32, 29, 10].

Задача 8. Алгоритм QFindStatD поиска порядковой статистики за время  $\Theta(N)$  для случая  $|a_i| \ll N$ . Проиллюстрировать работу алгоритма на примере последовательности [22, 28, 32, 10, 12, 34, 35, 34, 26, 34].

Задача 9. Алгоритм QFindStat5 поиска порядковой статистики за время  $\Theta(N)$  в худшем случае. Проиллюстрировать работу алгоритма на

примере последовательности [38, 28, 32, 17, 32, 20, 29, 25, 15, 14].

Задача 10. Модифицируйте алгоритм HeapSort для сортировки в неубывающем порядке. Проиллюстрировать работу алгоритма на примере последовательности [39, 17, 25, 22, 29, 26, 39, 34, 14, 28].

Задача 11. Проиллюстрируйте работу алгоритма HeapSort на бумаге на примере заданной последовательности чисел. На каждом шаге алгоритма последовательность следует изображать в виде дерева.

Задача 12. Остаётся ли справедливой оценка амортизированной стоимости стековых операций, равная  $O(1)$ , если включить в множество стековых операций операцию MultiPush, помещающую в стек  $k$  элементов?

Задача 13. Покажите, что если бы в пример с  $k$ -битовым счётчиком была включена операция Decrement, стоимость  $n$  операций была бы равной  $\Theta(nk)$ .

Задача 14. Предположим, что над структурой данных выполняется  $n$  операций. Стоимость  $i$ -ой по порядку операции равна  $i$ , если  $i$  — точная степень двойки, и 1 в противном случае. Определите с помощью группового анализа амортизированную стоимость операции.

1

Задача 15. Предположим, что над стеком выполняется последовательность операций; размер стека при этом никогда не превышает  $k$ . После каждых  $k$  операций проводится резервное копирование стека. Присвоив различным стековым операциям амортизированные стоимости, покажите, что стоимость  $n$  стековых операций, включая копирование стека, равна  $O(n)$ .

Задача 16. Предположим, что нам нужно иметь возможность не только увеличивать показания счётчика, но и сбрасывать его (то есть делать так, чтобы значения всех битов были равны 0). Считая, что время проверки или модификации одного бита составляет  $\Theta(1)$ , покажите, как осуществить реализацию счётчика в виде массива

битов, чтобы для выполнения произвольной последовательности из  $n$  операций Increment и Reset над изначально обнулённым счётчиком потребовалось бы время  $O(n)$ .

Указание. Отслеживайте в счётчике самый старший разряд, содержащий единицу.

Задача 17. Чему равна полная стоимость выполнения  $n$  стековых операций Push, Pop и Multipop, если предположить, что в начале стек содержит  $s_0$  объектов, а в конце —  $s_n$  объектов?

Задача 18. Предположим, что изначально показание счётчика не равно нулю, а определяется числом, содержащим в двоичном представлении  $b$  единиц. Покажите, что стоимость выполнения  $n$  операций Increment равна  $O(n)$  при условии, что  $n = \Omega(b)$ . (Не следует полагать, что  $b$  — константа.)

Задача 19. Алгоритм Рабина-Карпа для поиска подстроки в строке. Проиллюстрируйте работу алгоритма на примере поиска образца  $ab$  в строке  $aaabcsbaab$ .

Задача 20. Алгоритм поиска подстроки в строке при помощи конечных автоматов. Проиллюстрируйте работу алгоритма на примере поиска образца  $ab$  в строке  $aaabcsbaab$ .

Задача 21. Алгоритм Кнута-Морриса-Пратта поиска подстроки в строке. Проиллюстрируйте работу алгоритма на примере поиска образца  $ab$  в строке  $aaabcsbaab$ .

Задача 22. Предположим, что в образце  $P$  все символы различны. Покажите, как ускорить процедуру NaiveStringMatcher, чтобы время её выполнения при обработке  $n$ -символьного текста  $T$  было равно  $O(n)$ .

Задача 23. Покажите, как обобщить метод Рабина-Карпа, чтобы он позволял решать задачу поиска заданного образца размером  $m \times m$  в символьном массиве размером  $n \times n$ . Образец можно сдвигать по вертикали и по горизонтали, но нельзя вращать.

Задача 24. Сконструируйте автомат поиска подстрок для образца  $P = aabab$  и проиллюстрируйте его работу при обработке текста  $T =$

aaababaabaababab.

Задача 25. Изобразите диаграмму состояний для автомата поиска подстроки, если образец имеет вид ababbabbababbababbabb, а алфавит —  $\Sigma = \{a, b\}$ .

Задача 26. Вычислите префиксную функцию для образца ababbabbabbababbabb.

Задача 27. Покажите, что время работы процедуры KnuthMorrisPratt равно  $\Theta(n)$ .

Задача 28. Алгоритмы поиска и удаления элемента из хэш-таблицы с разрешением коллизий методом линейных проб.

Задача 29. Устройство хэш-функций на основе умножения и хэш-функций на основе деления.

Задача 30. Продемонстрируйте на бумаге происходящее при вставке в хэш-таблицу с разрешением коллизий методом линейных проб набора заданных элементов. Таблица имеет заданное число ячеек  $N$ . Хэш-функция имеет вид  $h(k) = k \bmod N$ .

Задача 31. Пусть  $(u, v)$  — ребро минимального веса в связном графе  $G$ . Покажите, что  $(u, v)$  принадлежит некоторому минимальному остовному дереву графа  $G$ .

2

Задача 32. Алгоритм Крускала может возвращать разные остовные деревья для одного и того же входного графа  $G$  в зависимости от взаимного расположения рёбер с одинаковым весом при сортировке. Покажите, что для любого минимального остовного дерева  $T$  графа  $G$  можно указать способ сортировки рёбер  $G$ , для которого алгоритм Крускала даст минимальное остовное дерево  $T$ .

Задача 33. Приведите простой пример ориентированного графа с отрицательными весами рёбер, для которого алгоритм Дейкстры даёт неправильные ответы. Объясните, почему так происходит.

Задача 34. Алгоритм обхода графа в ширину.

Задача 35. Алгоритм обхода графа в глубину.

Задача 36. Алгоритм Крускала поиска минимального остовного дерева графа.

Задача 37. Алгоритм Беллмана-Форда.

Задача 38. Алгоритм Дейкстры.

Задача 39. Алгоритм сортировки слиянием.

Модификация алгоритма без использования рекурсии. Проиллюстрировать работу алгоритма на примере последовательности [23, 16, 30, 37, 12, 32, 36, 29, 29, 24].

Задача 40. Красно-чёрные деревья. Алгоритм добавления элемента в красно-чёрное дерево.

Задача 41. Красно-чёрные деревья. Алгоритм удаления элемента из красно-чёрного дерева.

Задача 42. B-деревья. Алгоритмы поиска, добавления и удаления элементов.

Задача 43. Сбалансированные деревья. Алгоритм добавления элемента в AVL дерево.

3

Ниже — **краткая основная суть каждого билета** (что это за алгоритм / идея и что в нём главное). Без развёрнутых доказательств, ровно в формате «ответ на билет».

---

## 1. QuickSort

Разделяй-и-властвуй: выбирается опорный элемент, массив делится на  $\leq \text{pivot}$  и  $\geq \text{pivot}$ , рекурсивно сортируется. Среднее время  $\Theta(n \log n)$ , худшее  $\Theta(n^2)$ .

## 2. QuickSort по невозрастанию

То же самое, но при разбиении элементы **больше pivot идут влево**, меньше — вправо. Меняются только знаки сравнений.

## 3. QuickSort при равных элементах

Разбиение не уменьшает размер подзадач  $\rightarrow$  глубина рекурсии  $n \rightarrow$  время  $\Theta(n^2)$ .

## 4. Убывающий массив и QuickSort

При выборе крайнего элемента как pivot каждый раз одна часть пуста  $\rightarrow n$  уровней рекурсии  $\rightarrow \Theta(n^2)$ .

## 5. HeapSort

Строится бинарная куча (heap), затем максимум (или минимум) поочерёдно извлекается. Время  $\Theta(n \log n)$ , память  $O(1)$ .

## 6. CountingSort

Считается количество каждого значения, затем по счётчикам восстанавливается массив. Работает за  $\Theta(n + k)$ , где  $k$  — диапазон значений.

## 7. QFindStatP (QuickSelect, средний случай)

Поиск  $k$ -й порядковой статистики через разбиение как в QuickSort. Среднее время  $\Theta(n)$ .

## 8. QFindStatD (малые $|ai|$ )

Используется идея подсчёта (Counting), так как диапазон значений мал  $\rightarrow \Theta(n)$ .

## 9. QFindStat5 (median of medians)

Детерминированный алгоритм выбора опоры через группы по 5 элементов. Гарантирует  $\Theta(n)$  в худшем случае.

## 10. HeapSort по неубыванию

Используется **max-heap**, максимум переносится в конец массива  $\rightarrow$  итоговый массив возрастает.

## 11. HeapSort «на бумаге»

Каждый шаг показывается как бинарное дерево: построение кучи и восстановление heap-свойства после извлечения.

## 12. MultiPush и амортизированная стоимость

Да, остаётся  $O(1)$ , так как каждый элемент учитывается один раз, суммарная работа линейна.

## 13. $k$ -битовый счётчик с Decrement

Decrement может менять  $k$  битов  $\rightarrow n$  операций дают  $\Theta(nk)$ .

## 14. Амортизированный анализ (групповой)

Редкие дорогие операции ( $i = 2^k$ ) компенсируются множеством дешёвых  $\rightarrow$  амортизированная стоимость  $O(1)$ .

## 15. Стек + резервное копирование

Копирование происходит раз в  $k$  операций  $\rightarrow$  распределяя стоимость, получаем  $O(1)$  амортизированно, всего  $O(n)$ .

## 16. Increment + Reset за $O(n)$

Храним позицию старшего установленного бита  $\rightarrow$  каждый бит меняется  $O(1)$  раз  $\rightarrow$  суммарно  $O(n)$ .

**17. Push / Pop / Multipop**

Полная стоимость = число Push + число Pop  $\leq s_0 + n + s_n \rightarrow \Theta(n)$ .

**18. Счётчик с  $b$  единицами**

Если  $n = \Omega(b)$ , то «дорогие» переносы компенсируются  $\rightarrow$  суммарно  $O(n)$ .

**19. Рабин–Карп**

Хэши подстрок сравниваются за  $O(1)$ , при совпадении — проверка символов.

Среднее  $\Theta(n)$ .

**20. Поиск через конечный автомат**

Автомат по образцу заранее построен, текст читается за один проход  $\rightarrow \Theta(n)$ .

**21. Кнут–Моррис–Пратт (КМП)**

Использует префикс-функцию, чтобы не возвращаться назад по тексту  $\rightarrow \Theta(n)$ .

**22. NaiveStringMatcher при разных символах**

При несовпадении сразу сдвигаемся дальше  $\rightarrow$  каждый символ текста проверяется один раз  $\rightarrow O(n)$ .

**23. 2D Рабин–Карп**

Хэш сначала по строкам, затем по столбцам  $\rightarrow$  поиск  $m \times m$  в  $n \times n$  за  $\Theta(n^2)$ .

**24. Автомат для  $P = aabab$** 

Состояния — длина совпавшего префикса, переходы определяются максимальным суффиксом.

**25. Диаграмма автомата**

Строится по тому же принципу: состояния = длины префиксов, алфавит  $\{a, b\}$ .

**26. Префикс-функция**

Для каждой позиции — длина наибольшего собственного префикса, совпадающего с суффиксом.

**27. Время работы КМП**

Каждый символ текста и образца обрабатывается  $O(1)$  раз  $\rightarrow \Theta(n)$ .

**28. Хэш-таблица, линейные пробы**

Поиск и удаление — последовательный просмотр ячеек до ключа или пустой позиции.

**29. Хэш-функции**

- Деление:  $h(k) = k \bmod N$
- Умножение:  $h(k) = \lfloor N \cdot (kA \bmod 1) \rfloor$

Главное — равномерное распределение.

### 30. Вставка при линейных пробах

При коллизии идём вправо (циклически), пока не найдём пустую ячейку.

### 31. Минимальное ребро в MST

По свойству разреза: минимальное ребро любого разреза входит в некоторое MST.

### 32. Разные MST у Крускала

Порядок рёбер одинакового веса можно выбрать так, чтобы получить любое MST.

### 33. Дейкстра с отрицательными рёбрами

Алгоритм «замораживает» расстояния → отрицательные рёбра нарушают корректность.

### 34. BFS

Обход слоями, очередь, находит кратчайшие пути в невзвешенном графе.

### 35. DFS

Обход в глубину, рекурсия или стек, используется для компонент, циклов, топологической сортировки.

### 36. Крускал

Сортировка рёбер + DSU, добавляем рёбра без образования циклов → MST.

### 37. Беллман–Форд

Расслабление всех рёбер  $n-1$  раз, работает с отрицательными весами,  $\Theta(VE)$ .

### 38. Дейкстра

Жадный выбор минимального расстояния, работает только при неотрицательных весах.

### 39. MergeSort без рекурсии

Слияние блоков длины 1, 2, 4, 8... →  $\Theta(n \log n)$ , стабильный.

### 40. Красно-чёрные деревья: вставка

Вставка как в BST + перекраски и повороты для сохранения свойств.



#### 41. Красно-чёрные деревья: удаление

Удаление + исправление «двойной чёрноты» с помощью поворотов и перекрасок.

#### 42. B-деревья

Много ключей в узле, поиск за  $O(\log n)$ , вставка с расщеплением, удаление с заимствованием/слиянием.

#### 43. AVL-дерево: вставка

После вставки проверяется баланс, выполняются LL, RR, LR, RL-повороты.

---

Если нужно:

- упростить ещё сильнее (1–2 предложения)
- или сделать версию “для устного экзамена”
- или отдельно с иллюстрациями шагов

— скажи, сделаю.



Хорошо, но нужно детализировать (всё таки чуть развёрнутее). Необходимо для билетов где есть примеры (где нужно проиллюстрировать работу алгоритма - расписать пример)

Отлично, тогда делаем чуть развёрнуто, но всё ещё «экзаменационно»:

— идея алгоритма

— ключевые шаги

— пошаговый разбор примера там, где он требуется

Ниже — билеты с примерами (1, 2, 5, 6, 7, 8, 9, 10, 11, 19, 20, 21, 24, 25, 26, 39).

Остальные — теоретические, их можно оставить в кратком виде (при необходимости тоже распишу).

---

## 1. QuickSort (пример)

**Идея:**

Выбирается опорный элемент (pivot). Все элементы  $\leq \text{pivot}$  помещаются слева,  $\geq \text{pivot}$  — справа. Затем рекурсивно сортируются подмассивы.

**Пример:**

Массив:

[31, 21, 24, 29, 29, 31, 11, 27, 24, 27]

Возьмём pivot = 31 (первый элемент).

Разбиение:

- $\leq 31$ : [21, 24, 29, 29, 31, 11, 27, 24, 27]
- 31: []

Далее сортируем левую часть, pivot = 21:

- $\leq 21$ : [11]
- 21: [24, 29, 29, 31, 27, 24, 27]

И так далее, пока подмассивы не станут длины 1.

После всех рекурсий получаем отсортированный массив.

---

## 2. QuickSort по невозрастанию

**Идея:**

То же, что QuickSort, но меняем направление сравнений.

**Пример:**

[39, 31, 27, 13, 17, 18, 14, 20, 31, 17]

pivot = 39.

Разбиение:

- $\geq 39$ : []
- $< 39$ : остальные элементы

Далее pivot = 31:

- $\geq 31$ : [31, 31]
- $< 31$ : [27, 13, 17, 18, 14, 20, 17]

После рекурсий итог:

[39, 31, 31, 27, 20, 18, 17, 17, 14, 13]

## 5. HeapSort (пример)

Идея:

1. Построить max-heap
2. Менять корень с последним элементом
3. Восстанавливать heap-свойство

Пример:

[31, 32, 34, 13, 28, 20, 28, 36, 30, 20]

После построения max-heap:

markdown

 Копировать код

```
      36
     /  \
    32   34
   / \  / \
  31 28 20 28
 / \
13 30
```

Шаги:

- 36 ↔ 20 → heapify
- 34 ↔ 13 → heapify
- ...

В конце получаем отсортированный массив.

## 6. CountingSort (пример)

Идея:

Считаем количество каждого значения, затем восстанавливаем массив.

Пример:

[4, 4, 5, 4, 8, 4, 6, 3, 5, 2, 8, 1, 3, 5, 2]

Счётчики:

 Копировать код

1 → 1  
2 → 2  
3 → 2  
4 → 4  
5 → 3  
6 → 1  
8 → 2

Восстановление:

[1, 2, 2, 3, 3, 4, 4, 4, 4, 5, 5, 5, 6, 8, 8]

---

## 7. QFindStatP (QuickSelect)

**Идея:**

Ищем  $k$ -й элемент без полной сортировки.

**Пример:**

[36, 39, 37, 25, 12, 12, 11, 32, 29, 10]

Ищем медиану ( $k = 5$ ).

pivot = 36

$\leq 36$ : [25, 12, 12, 11, 32, 29, 10] (7 элементов)

Так как  $7 > 5 \rightarrow$  ищем в левой части.

Процесс повторяется, пока не найдём нужный элемент.

---

## 8. QFindStatD (малый диапазон)

**Идея:**

Если  $|a_i| \ll N$ , используем подсчёт.

**Пример:**

[22, 28, 32, 10, 12, 34, 35, 34, 26, 34]

Считаем частоты, затем идём по значениям, накапливая количество элементов, пока не дойдём до  $k$ -й позиции.

---

## 9. QFindStat5 (median of medians)

**Идея:**

Группируем по 5 элементов, берём медианы, рекурсивно находим медиану медиан — это pivot.

**Пример:**

[38, 28, 32, 17, 32 | 20, 29, 25, 15, 14]

Медианы групп:

- [17, 28, 32, 32, 38] → 32
- [14, 15, 20, 25, 29] → 20

Медиана медиан = 32.

Используется как опора →  $\Theta(n)$  в худшем случае.

---

## 10. HeapSort по неубыванию

**Идея:**

Используем max-heap, каждый раз максимальный элемент отправляется в конец.

**Пример:**

[39, 17, 25, 22, 29, 26, 39, 34, 14, 28]

После всех извлечений:

[14, 17, 22, 25, 26, 28, 29, 34, 39, 39]

---

## 11. HeapSort с деревьями

На каждом шаге:

- массив изображается как полное бинарное дерево
- выполняется swap и heapify

Используется для наглядного понимания heap-свойства.

---

## 19. Рабин–Карп

**Пример:**

Текст: aaabcsbaab

Образец: ab

Хэш(ab) = X

Хэши подстрок текста:

- aa ≠ X
- aa ≠ X
- ab = X → проверка → совпадение

## 20. Конечные автоматы

Идея:

Состояние — длина совпавшего префикса.

Для ab :

- 0 → ничего не совпало
- 1 → a
- 2 → ab (найдено)

Текст читается за один проход.

## 21. КМП

Префикс-функция для ab :

CSS

 Копировать код

a b

0 0

При несовпадении не возвращаемся по тексту, а используем π-функцию.

## 24. Автомат для P = aabab

Состояния: 0...5

Каждое состояние — длина совпавшего префикса.

Переходы строятся по максимальному суффиксу.

## 25. Диаграмма автомата

Для длинного образца:

- количество состояний =  $|P| + 1$
  - из каждого состояния по  $a$  и  $b$  есть переход
- 

## 26. Префикс-функция

Для `ababbabbabababbabb`

На каждом шаге считаем длину наибольшего совпадающего префикса и суффикса.

---

## 39. MergeSort без рекурсии

Пример:

[23, 16, 30, 37, 12, 32, 36, 29, 29, 24]

Шаги:

- длина блоков 1 → сортировка пар
- длина 2 → слияние
- длина 4 → слияние
- ...

Итог — отсортированный массив за  $\Theta(n \log n)$ .

---

Если хочешь, дальше могу:

- 📌 оформить как шпаргалку на 1–2 страницы
- 📌 сделать версию “как отвечать у доски”
- 📌 выделить формулы и ключевые фразы для заучивания

Скажи, в каком формате дальше 👍

