**1. Сформулируйте общую идею амортизационного анализа.**

Амортизационный анализ оценивает среднюю стоимость операции в наихудшем случае для последовательности операций над структурой данных. В отличие от анализа отдельных операций, он учитывает, что "дорогие" операции происходят редко, а их стоимость компенсируется множеством "дешевых" операций. Цель — показать, что средняя стоимость операции в последовательности ниже, чем её максимальная стоимость при изолированном выполнении.

**2. Назовите и опишите основные методы амортизационного анализа.**  
  
Основные методы:

* **Метод агрегирования (групповой анализ):**  
  Рассчитывается общая стоимость n операций, затем средняя стоимость одной операции: общая стоимость / n​.
* **Метод бухгалтерского учета (предоплата):**  
  Каждой операции назначается амортизационная стоимость. Если реальная стоимость операции меньше амортизационной, разница сохраняется как "кредит" для покрытия будущих дорогих операций.
* **Метод потенциала:**  
  Вводится функция потенциала ΦΦ, отображающая состояние структуры данных в число. Амортизационная стоимость операции: ci + Φi − Φi−1где ci​ — реальная стоимость, Φi— потенциал после операции.

**3. Опишите структуру данных «Динамическая таблица (Dynamic Table)» и покажите, как выполняются операции добавления и удаления элементов.**  
  
**Структура:** Массив, который автоматически изменяет размер при заполнении/опустошении.

**Добавление:**

* Если есть свободное место, элемент добавляется в конец.
* Если массив заполнен:
* Выделяется новый массив вдвое большего размера.
* Элементы копируются в новый массив.
* Старый массив освобождается.
* Новый элемент добавляется.

**Удаление:**

* Элемент удаляется.
* Если заполненность падает ниже порога (например, 25%):
* Выделяется новый массив вдвое меньшего размера.
* Элементы копируются в новый массив.
* Старый массив освобождается.

**4. Приведите анализ операций добавления и удаления элементов для структуры данных «Динамическая таблица (Dynamic Table)».**

**Добавление:**  
Пусть размер таблицы удваивается при заполнении. Стоимость копирования при увеличении от 1 до 2k: 2k+1−1≈2n

Для n операций вставки общая стоимость: O(n) средняя амортизационная стоимость: O(1).

**Удаление:**  
Аналогично, при уменьшении размера вдвое при заполненности <25%, копирование происходит редко. Амортизационная стоимость удаления: O(1).

**5. Дайте определение абстрактного типа данных «Очередь с приоритетами (Priority Queue)».**  
  
Очередь с приоритетами поддерживает операции:

* Enqueue(x, p): Добавление элемента x с приоритетом p.
* Dequeue(): Извлечение элемента с наивысшим (max-куча) или наинизшим (min-куча) приоритетом.
* Peek(): Получение элемента с экстремальным приоритетом без извлечения.
* IncreasePriority(x, k): Увеличение приоритета элемента x на k (для max-кучи).
* DecreasePriority(x, k): Уменьшение приоритета элемента x на k (для min-кучи).

**6. На основе каких структур данных может быть реализован абстрактный тип данных «Очередь с приоритетами (Priority Queue)»?**

Неупорядоченный массив/список: Enqueue за O(1) Dequeue за O(n)

Упорядоченный массив/список: Enqueue за O(n), Dequeue за O(1)

**Двоичная куча:** Все операции за O(logn).

**7. Покажите, как устроена структура данных «Двоичная пирамида (Binary Heap)». Опишите порядок выполнения операций SiftUp и SiftDown. Приведите оценку времени их работы.**  
  
**Структура:** Полное бинарное дерево, хранящееся в массиве. Свойство:

**Max-куча:** A[i]≥A[2i+1] и A[i]≥A[2i+2]

**Min-куча:** A[i]≤A[2i+1] и A[i]≤A[2i+2]

**SiftUp (просеивание вверх):**

1. Начинаем с элемента i.
2. Сравниваем с родителем p=⌊(i−1)/2⌋
3. Если нарушено свойство кучи (для max: A[i]>A[p] меняем местами.
4. Повторяем для p, пока не восстановится свойство.

**Время:** O(logn).

**SiftDown (просеивание вниз):**

1. Начинаем с элемента i
2. Сравниваем с потомками l=2i+1, r=2i+2
3. Находим экстремального потомка m (для max:  max(A[l],A[r])
4. Если нарушено свойство (для max: A[i]<A[m]), меняем местами.
5. Повторяем для m, пока не восстановится свойство.

**Время:** O(logn)

**8. Опишите, как на основе массива построить двоичную пирамиду за время O(n).**  
Алгоритм:

* Начинаем с последнего **нелистового** элемента (индекс ⌊n/2⌋−1).
* Выполняем SiftDown для всех элементов от ⌊n/2⌋−1 до 0.

**9. Приведите псевдокод операций Enqueue, Dequeue и IncreaseKey для очереди с приоритетами, реализованной на основе двоичной пирамиды.**  
**(max-куча):**

