# ВОПРОСЫ ДЛЯ САМОКОНТРОЛЯ

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

Амортизационный анализ позволяет оценить среднее время выполнения последовательности операций над структурой данных, даже если некоторые из них выполняются долго. В отличие от классического анализа (где каждая операция оценивается отдельно), амортизационный анализ учитывает, что «дорогие» операции происходят редко и компенсируются множеством «дешёвых».

1. **Назовите и опишите основные методы амортизационного анализа.**
2. Метод агрегирования:

Оценивается общее время выполнения всех операций, затем делится на их количество.

1. Метод учёта:

Каждой операции назначается «стоимость». Если операция выполняется быстрее, чем её стоимость, разница сохраняется как «кредит» для будущих дорогих операций.

1. Метод потенциала:

Вводится функция потенциала, отражающая состояние структуры данных. Амортизированная стоимость равна сумме фактической стоимости и изменения потенциала.

1. **Опишите структуру данных «Динамическая таблица (Dynamic Table)» и покажите, как выполняются операции добавления и удаления элементов.**

Динамическая таблица — это массив, который автоматически расширяется при заполнении и сжимается при сильном уменьшении количества элементов.

**Добавление элемента в динамическую таблицу происходит следующим образом:**

1. Если есть свободное место, новый элемент записывается в свободную ячейку.
2. Если все элементы заняты, выделяется память под новую таблицу большего размера, в которую копируются записи из старой таблицы.
3. После добавления нового элемента старый размер таблицы уменьшается, а новый — увеличивается.

**Удаление элемента из динамической таблицы также имеет особенности, связанные с динамическим распределением памяти:**

1. Если элемент — последнее значение в таблице, его удаление сводится к уменьшению размера структуры на одну ячейку.
2. Если нужно удалить не последнее значение, все элементы правее удаляемого сдвигаются влево на одну позицию, а сама удаляемая запись удаляется
3. **Приведите анализ операций добавления и удаления элементов для структуры данных «Динамическая таблица (Dynamic Table)».**

Анализ операций в динамической таблице:

Добавление: В худшем случае при расширении копирование n элементов занимает O(n) времени. Но если расширение происходит редко (каждые 2^k элементов), амортизированная стоимость добавления — O(1).

Удаление: Аналогично, сжатие таблицы занимает O(n) времени, но амортизированная стоимость удаления — O(1).

Таким образом, амортизационный анализ (методом агрегирования или потенциала) показывает, что обе операции в среднем выполняются за O(1).

1. **Дайте определение абстрактного типа данных «Очередь с приоритетами (Priority Queue)».**

«Очередь с приоритетами» (priority queue) — это структура данных, представляющая собой обычную очередь, элементы которой упорядочены по их приоритету. Она поддерживает следующие операции:

Enqueue(x, p)— вставка элемента xx с приоритетом pp.

Dequeue()— извлечение элемента с наивысшим (или наименьшим) приоритетом.

Increase/DecreaseKey(x, Δp)— изменение приоритета элемента xx.

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

Двоичная пирамида (Binary Heap);

Фибоначчиева куча (Fibonacci Heap;

Биномиальная куча (Binomial Heap;

Сортированный список/массив.

1. **Покажите, как устроена структура данных «Двоичная пирамида (Binary Heap)». Опишите порядок выполнения операций SiftUp и SiftDown. Приведите оценку времени их работы.**

**Двоичная куча (binary heap)** — реализация очереди с приоритетами, использующая корневое дерево. Для неё выполнены три условия:

1. Значение в любой вершине не больше, чем значения её потомков.
2. У любой вершины не более двух сыновей.
3. Слои заполняются последовательно сверху вниз и слева направо, без «дырок».

**Операция siftUp** восстанавливает свойства кучи, если значение изменённого элемента уменьшается. Порядок выполнения:

1. Если элемент больше своего отца, то условие 1 соблюдено для всего дерева, и больше ничего делать не нужно.
2. Иначе элемент меняется местами с отцом.
3. После этого выполняется siftUp для этого отца. Иными словами, слишком маленький элемент всплывает наверх.

**Оценка времени работы операции siftUp**: процедура выполняется за время O(log n)

**Операция siftDown** используется для восстановления свойства кучи с конца: постепенно увеличивается суффикс, на котором выполняется это свойство, с помощью просеивания вниз.

**Оценка времени работы операции siftDown**: в худшем случае новый корень нужно поменять местами с его ребёнком на каждом уровне, пока он не достигнет нижнего уровня кучи. Это означает, что операция удаления имеет сложность времени, связанную с высотой дерева, или O(log n)

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

Алгоритм:

-Заполнить массив элементами в произвольном порядке.

-Выполнить SiftDown для всех внутренних узлов (начиная с n/2 до 0).

def BuildHeap(A)

n = len(A)

for i from (n // 2 - 1) downto 0:

SiftDown(A, i)

1. **Приведите псевдокод операций Enqueue, Dequeue и Increase/DecreaseKey для очереди с приоритетами, реализованной на основе двоичной пирамиды.**

**Псевдокод операции Enqueue**:

1. Добавить элемент в конец массива (пирамиды).
2. Выполнить SiftUp (просеивание вверх) от нового элемента, чтобы восстановить свойство кучи.

**Псевдокод операции Dequeue**:

1. Запомнить корень (максимальный элемент).
2. Переместить последний элемент в корень.
3. Удалить последний элемент из массива.
4. Выполнить SiftDown (просеивание вниз) от корня, чтобы восстановить свойство кучи.

**Псевдокод операции IncreaseKey**:

1. Увеличить приоритет элемента (значение узла).
2. Выполнить SiftUp от этого элемента (так как его приоритет увеличился, он может подняться выше).

**Псевдокод операции DecreaseKey**:

1. Уменьшить приоритет элемента (значение узла).
2. Выполнить SiftDown от этого элемента (так как его приоритет уменьшился, он может опуститься ниже).