**Ответы на вопросы(не знаю, надо ли вообще на них отвечать…)**

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

Амортизационный анализ — это метод [анализа](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B0%D0%BB%D0%B8%D0%B7_%D0%B0%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC%D0%BE%D0%B2) [вычислительной сложности](https://ru.wikipedia.org/wiki/%D0%92%D1%8B%D1%87%D0%B8%D1%81%D0%BB%D0%B8%D1%82%D0%B5%D0%BB%D1%8C%D0%BD%D0%B0%D1%8F_%D1%81%D0%BB%D0%BE%D0%B6%D0%BD%D0%BE%D1%81%D1%82%D1%8C) алгоритма, используемый в случаях, когда время исполнения одного шага алгоритма, умноженное на число шагов, даёт слишком большую оценку на время исполнения всего алгоритма по сравнению с тем, какая она на самом деле

**2)Назовите и опишите основные методы амортизационного анализа.**

**Метод усреднения**

В методе усреднения амортизационная стоимость операций определяется напрямую по формуле, указанной выше: суммарная стоимость всех операций алгоритма делится на их количество.

**Метод потенциала**:

Вводится "потенциальная функция" (Φ), отражающая состояние структуры данных. Амортизированная стоимость: реальная стоимость + ΔΦ.

Пример: Для динамического массива Φ может быть равно 2⋅size−capacity2⋅*size*−*capacity*.

**Метод банкира. Время — деньги**

Пусть каждая единица времени — это монета. Каждой операции выдадим  монет и разрешим тратить . Все свои монеты операции могут раскладывать по структуре данных. Амортизационная стоимость одной операции составит .

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

**Структура:**

Массив, который автоматически изменяет размер при добавлении/удалении элементов.

Параметры: size (текущее количество элементов), capacity (текущая ёмкость).

Операции:

Добавление:

Если место есть (size<capacity*size*<*capacity*), элемент добавляется за O(1)*O*(1).

Если места нет, создаётся новый массив (обычно в 2⋅capacity2⋅*capacity*), элементы копируются (O(n)*O*(*n*)), затем добавляется новый элемент.

Удаление:

Если size≤capacity4*size*≤4*capacity*​, массив уменьшается (обычно до capacity22*capacity*​) для экономии памяти.

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

**Амортизированная стоимость добавления**:

Метод агрегирования: После n*n* добавлений (с удвоениями) общая стоимость O(n)*O*(*n*), средняя — O(1)*O*(1).

Метод потенциала: Φ = 2⋅size−capacity2⋅*size*−*capacity*. При удвоении потенциал становится 0, что компенсирует стоимость копирования.

**Амортизированная стоимость удаления**:

Аналогично, уменьшение ёмкости при size≤capacity4*size*≤4*capacity*​ сохраняет амортизированную стоимость O(1)*O*(1).

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

Абстрактный тип данных, поддерживающий операции:

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

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

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

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

**-Двоичная пирамида (Binary Heap):.**

**-Биномиальная куча**

**-Фибоначчиева куча**

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

**Бинарная пирамида (куча)** — это **полное бинарное дерево**, удовлетворяющее условию:

**Max-Heap**: Для любого узла A[i] выполняется A[i] >= A[2i+1] и A[i] >= A[2i+2].

**Min-Heap**: Для любого узла A[i] выполняется A[i] <= A[2i+1] и A[i] <= A[2i+2].

**Хранится в массиве**, где:

Корень — A[0].

Левый потомок A[i] — A[2i+1].

Правый потомок A[i] — A[2i+2].

Родитель A[i] — A[(i-1)/2].

Операции я реализовывала в **своем коде**, время работы: O(log n).

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

def BuildHeap(A)

n = len(A)

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

SiftDown(A, i)

9)**Приведите псевдокод операций *Enqueue*, Dequeue и *Increase*/*DecreaseKey д*ля очереди с приоритетами, реализованной на основе двоичной пирамиды**.

В моем коде это было реализовано , кроме последнего. Вот Increase/DecreaseKey:

def IncreaseKey(heap, index, new\_priority):

if new\_priority < heap[index].priority:

raise Error("New priority must be larger (для max-heap)")

heap[index].priority = new\_priority

SiftUp(heap, index

def DecreaseKey(heap, index, new\_priority):

if new\_priority > heap[index].priority:

raise Error("New priority must be smaller (для max-heap)")

heap[index].priority = new\_priority

SiftDown(heap, index