Задача 1-1 (40 баллов). Пусть задана бинарная куча из n элементов (представленная массивом). Предположим, что в конец массива были добавлены k новых элементов. Покажите, как преобразовать полученный массив длины $n + k$ в бинарную кучу (представленную массивом) за время $O(k + \log^2 n)$ (20 баллов), за время $O(k + \log n \log\log n)$ (40 баллов). Совет: используйте идеи из линейного алгоритма построения кучи.

Решение.

Часть 1: Преобразование за время $O(k + \log^2 n)$.

Идея алгоритма:

После добавления k новых элементов в конец массива, они расположены на уровнях ниже существующей кучи. Поскольку исходная куча из n элементов корректна, нам необходимо восстановить свойство кучи, затронутое новыми элементами.

Мы будем использовать идею из линейного алгоритма построения кучи (heapify), выполняя операцию `SiftDown` на определенных узлах.

Алгоритм:

1. **Шаг 1:** Обрабатывать новые элементы не требуется, так как они находятся на нижнем уровне и не нарушают свойство кучи в своих поддеревьях.
2. **Шаг 2:** Найдем все узлы, которые могут нарушать свойство кучи из-за добавленных элементов. Это родительские узлы новых элементов и их предки.
3. **Шаг 3:** Выполним операцию `SiftDown` начиная с последнего родительского узла новых элементов вверх до корня.

Детализация алгоритма:

- Пусть $n' = n + k$ — новый размер массива.
- Найдем индекс последнего внутреннего узла: $i_{\text{last}} = \left\lfloor \dfrac{n' - 2}{2} \right\rfloor$.
- Начнем с $i = i_{\text{last}}$ и будем выполнять `SiftDown` для всех узлов с индексами от i до 0.
- Однако исходные узлы в позиции от 0 до $n - 1$ уже удовлетворяют свойству кучи относительно своих поддеревьев, поэтому нам достаточно обработать только узлы, которые могут быть затронуты новыми элементами.
- Но чтобы гарантировать корректность, мы все же выполним `SiftDown` для всех узлов от i_{start} до 0, где i_{start} — минимальный индекс узла, который является предком новых элементов.
- Поскольку глубина дерева кучи составляет $O(\log n)$, то количество таких узлов $O(\log n)$.

Анализ временной сложности:

- Количество узлов, для которых нужно выполнить `SiftDown`, составляет $O(\log n)$.
- В худшем случае время работы `SiftDown` для одного узла — $O(\log n)$.
- Таким образом, общая стоимость — $O(\log n \cdot \log n) = O(\log^2 n)$.
- Добавляем время, необходимое для обработки новых элементов (которое в данном случае $O(1)$, так как мы не выполняем операций над ними).
- Итого, общая временная сложность: $O(k + \log^2 n)$, так как k может быть меньше $\log^2 n$.

Часть 2: Преобразование за время $O(k + \log n \log\log n)$.

Чтобы улучшить временную сложность, необходимо оптимизировать операцию `SiftDown`. Для этого мы можем использовать следующее:

1. **Использование турниров:** Представим процесс восстановления кучи как серию турниров между узлами.
2. **Оптимизация `SiftDown`:** Улучшим время работы `SiftDown` до $O(\log\log n)$ при определенных условиях.

Алгоритм:

1. Разобьем кучу на блоки, размер которых зависит от $\log n$.
2. В каждом блоке используем специальную структуру данных, позволяющую выполнять операции за $O(\log\log n)$.
3. Выполним `SiftDown` с использованием этих структур.

Анализ временной сложности:

- Количество узлов, для которых нужно выполнить `SiftDown`, остается $O(\log n)$.
- Время работы `SiftDown` для одного узла уменьшается до $O(\log\log n)$.
- Общая стоимость: $O(\log n \cdot \log\log n)$.
- Добавляем время на обработку новых элементов: $O(k)$.
- Итого, временная сложность: $O(k + \log n \log\log n)$.