1 Задача 1-1

Без ограничения общности будем считать, что нужно упорядочить ключи по неубыванию. Покажем, что $g(n) = n - 1$.

Оценка снизу:

Для $n = 1$ это очевидно. Иначе предположим, что в корректном дереве есть достижимый лист на глубине меньше $n - 1$. Представим ключи в виде вершин, а сравнения - в виде ребер между вершинами. Поскольку $n \geqslant 2$, а ребер меньше $n - 1$, найдутся две несвязные компоненты, которые могут быть как строго меньше, так и больше друг друга. Получается, что перестановка, которая находится в листе, может как расположить больший элемент после меньшего, так и наоборот. Это противоречит с тем, что дерево корректное и что сортировка идет по неубыванию.

Оценка сверху:

Построим дерево, которое делает $n - 1$ сравнение между каждыми рядом стоящими ключами. Если ключи уже отсортированы, возвращаем тривиальную перестановку. Если же в какой-то момент узнаем, что это не так, реализуем сортировку слиянием с нуля.

2 Задача 1-2

Поскольку число листьев должно быть не меньше количество исходов, которых C_{m+n}^n, то высота дерева не меньше $\log C_{m+n}^n$.

$$
\begin{aligned}
h &\geqslant \log C_{m+n}^n = \log(m+n)! - \log m! - \log n! \\
&\geqslant C_0 + \frac{\log(m+n)}{2} + (m+n)\log(m+n) - \frac{\log m}{2} - m\log m - \frac{\log n}{2} - n\log n \\
&\geqslant C_0 + (m+n)\log(m+n) - (m+n)\log m - \frac{\log n}{2} + n\log m - n\log n \\
&\geqslant C_0 + (m+n)\log(1 + \frac{n}{m}) - \frac{n}{2} + n\log\frac{m}{n} \\
&\geqslant C_0 + m\left(\frac{n}{m} - \frac{1}{2}\left(\frac{n}{m}\right)^2\right) - \frac{n}{2} + n\log\frac{m}{n} \\
&\geqslant C_0 + n\left(1 - \frac{n}{2m}\right) - \frac{n}{2} + n\log\frac{m}{n} \\
&\geqslant C_0 + n\left(1 - \frac{1}{4}\right) - \frac{n}{2} + n\log\frac{m}{n} \\
&\geqslant C_0 + n\log\frac{m}{n}
\end{aligned}
$$

Получаем нижнюю оценку на количество сравнений $\Omega(n\log\frac{m}{n})$

3 Задача 1-3

1. Докажем от противного. То есть предположим, что

$$\forall I, A : \mathsf{E}_I[f_A(I)] > \mathsf{E}_A[f_A(I)]$$

Но тогда

$$\forall I : \mathsf{E}_A\mathsf{E}_I[f_A(I)] > \mathsf{E}_A[f_A(I)]$$

$$\mathsf{E}_A\mathsf{E}_I[f_A(I)] > \mathsf{E}_I\mathsf{E}_A[f_A(I)]$$
$$\mathsf{E}_I\mathsf{E}_A[f_A(I)] > \mathsf{E}_I\mathsf{E}_A[f_A(I)]$$

Противоречие.

Иначе это неравенство можно сформулировать так: найдется алгоритм со сложностью в среднем меньшей, чем рандомизированная сложность.

2. Определим рандомизированный алгоритм сортировки в модели решающих деревьев как алгоритм, который в каждой вершине задает вопрос $x < y$?, где пара (x, y) выбирается случайно из множества пар, таких, что вопрос про них принесет положительное количество бит информации. Если таких пар нет, то мы в листе.

Основываясь на доказанном неравенстве, где множество A - алгоритмы, удовлетворяющие данному определению, а распределение на множестве входов равномерное, получаем оценку на сложность рандомизированного алгоритма. А поскольку для детерминированного алгоритма A нам известна оценка

$$\mathsf{E}_I[f_A(I)] = \Omega(|I| \log |I|) = \Omega(n \log n)$$

4 Задача 1-4

Пусть стек должен поддерживать следующее соотношение:

$$\frac{capacity}{a} \leqslant size \leqslant capacity$$

Тогда при заполнении массива мы выделяем новый массив размера $\sqrt{a} \cdot capacity$, а при опустошении до $\frac{capacity}{a}$, выделяем массив размера $\frac{capacity}{\sqrt{a}}$ и перезаписываем $size$ элементов.

Покажем с помощью функции потенциала, что амортизированные стоимости вставки и удаления будут константными. Пусть функция потенциала имеет вид

$$\varphi(size, capacity) = \begin{cases} \left(\frac{\sqrt{a}}{\sqrt{a}-1}\right)\left(size - \frac{capacity}{\sqrt{a}}\right) & size \geqslant \frac{capacity}{\sqrt{a}} \\ \left(\frac{1}{\sqrt{a}-1}\right)\left(\frac{capacity}{\sqrt{a}} - size\right) & size < \frac{capacity}{\sqrt{a}} \end{cases}$$

Неамортизированные стоимости:

$$time(push) = \begin{cases} 1 & size < capacity \\ 1 + size & size = capacity \end{cases}$$

$$time(pop) = \begin{cases} 1 & size > \frac{capacity}{a} \\ 1 + size & size = \frac{capacity}{a} \end{cases}$$

Амортизированные стоимости:

$$time'(push) =$$

$$= \begin{cases} 1 + \varphi(size + 1, capacity) - \varphi(size, capacity) & size < capacity \\ 1 + size + \varphi(size + 1, \sqrt{a} \cdot capacity) - \varphi(size, capacity) & size = capacity \end{cases}$$

$$= \begin{cases} O(1) & size < capacity \\ O(1) + size - \left(\frac{\sqrt{a}}{\sqrt{a}-1}\right)\left(capacity - \frac{capacity}{\sqrt{a}}\right) & size = capacity \end{cases}$$

$$= O(1)$$

$$time'(pop) =$$

$$= \begin{cases} 1 + \varphi(size - 1, capacity) - \varphi(size, capacity) & size > \frac{capacity}{a} \\ 1 + size + \varphi(size - 1, \frac{capacity}{\sqrt{a}}) - \varphi(size, capacity) & size = \frac{capacity}{a} \end{cases}$$

$$= \begin{cases} O(1) & size > \frac{capacity}{a} \\ O(1) + size + \left(\frac{1}{\sqrt{a}-1}\right)\left(\frac{capacity}{a} - \frac{capacity}{\sqrt{a}}\right) & size = \frac{capacity}{a} \end{cases}$$

$$= O(1)$$

5 Задача 1-5

Реализуем очередь на массиве, зацикливая указатели на начало. Дальше все аналогично предыдущей задаче.

6 Задача 1-6

Пусть инструкция вставки в конец стоит 1 монету. Тогда будем просить за амортизированную операцию вставки 3 монеты.

Пока вектор не заполнится наполовину, будем тратить 1 монету на вставку, остальные выкидывать. Когда достигнем середины, аллоцируем вектор вдвое больший, и будем тратить 1 монету на вставку в изначальный вектор и 2 на копирование в новый. Когда изначальный вектор заполнится, мы перезапишем указатели, деаллоцируем память и будем иметь наполовину заполненный вектор - ситуация, инвариантная начальной.