разряде имеют цифру 0, отправляются в 0-й стек, а номера которых в k-м разряде имеют цифру 1, отправляются в 1-й стек. После роспуска вытягиваются сначала вагоны из 0-го стека, а затем из 1-го. На этом k-й шаг сортировки заканчивается.

Посмотрим на примере решения задачи 4,а), как работает наш алгоритм (рис.6). Из рисунка видно, что для сортировки потребовалось сделать 3 шага ($8 = 2^3$) и $6 = 2\times 3$ вытягиваний. Теперь понятно, что нужно делать, если у сортировочной горки 3 стека — нужно записывать номера вагонов в 3-ичной системе счисления и сортировать по разрядам 3-ичной системы.

Чтобы освоиться теперь с алгоритмом поразрядной сортировки, решите с его помощью задачи 3,а)–6,а). Совпадают ли эти решения с вашими?

Основные утверждения

Приведем одно очевидное утверждение, с которого начинаются решения наших задач.

Лемма 1. *Если два вагона попадают в один стек, то порядок между ними не меняется.*

Теорема 1. *Алгоритм поразрядной сортировки решает задачу 9.*

Вспомним, как мы сравниваем два числа $a = a_s a_{s-1}...a_0$ и $b = b_s b_{s-1}...b_0$. Мы всегда будем считать, что их записи выровнены по разрядам (в крайнем случае можно приписать нули). $a < b$ тогда и только тогда, когда $a_s = b_s$, $a_{s-1} = b_{s-1}$, ..., $a_{l+1} = b_{l+1}$, а $a_l < b_l$.

Посмотрим, что происходит с вагонами a и b при поразрядной сортировке. Первые $(l - 1)$ шагов вагоны a, b как-то шатаются по стекам. Абсолютно не важно, как. На l-м шаге вагон a попадает в стек с номером a_l, а вагон b попадает в стек с номером b_l. Так как $a_l < b_l$, то вагон a будет вытянут на этом шаге перед вагоном b и окажется левее b. Но во всех старших, чем l, разрядах у a и b стоят одинаковые цифры, поэтому во все последующие роспуски вагоны a и b будут попадать в одни и те же стеки. Значит, по лемме 1 их порядок не изменится. Следовательно, в конце сортировки вагон a будет находиться левее b. А так как это верно для любой пары вагонов, то весь состав будет упорядочен.

Заметим, что точно так же упорядочены слова в словаре. Такой порядок называется *лексикографическим*.

Необходимое для записи числа n число разрядов в k-ичной системе счисления ограничено сверху числом $\lceil \log_k n \rceil$, значит, и число шагов в поразрядной сортировке состава из n вагонов ограничено сверху $\lceil \log_k n \rceil$, где $\lceil x \rceil$ — наименьшее целое число, не меньшее чем x.

Определение 1. Множество 1, 2,... ..., $n - 1$, n в перепутанном порядке $(\sigma_1, \sigma_2, \sigma_3, ..., \sigma_n)$ называется *перестановкой*.

Вообще-то, перестановкой в математике называется взаимно-однозначное отображение $\sigma: \{1, 2, ..., n\} \to \{1, 2, ..., n\}$, но в нашей задаче можно не различать перестановку σ и результат ее действия.

Назовем множество элементов $\{\sigma_{k_1}, \sigma_{k_2}, \sigma_{k_3}, ..., \sigma_{k_s}\}$ перестановки $\sigma = \{\sigma_1, \sigma_2, \sigma_3, ..., \sigma_n\}$ *обратной цепью*, если $\sigma_{k_i} > \sigma_{k_{i+1}}$, $k_i < k_{i+1}$. Обратная цепь, состоящая из наибольшего числа элементов, называется *максимальной*. Обозначим длину максимальной обратной цепи $f(\sigma)$.

Из принципа Дирихле следует, что длина максимальной обратной цепи $f(\sigma)$ за один шаг может уменьшиться не более чем в k раз. Действительно, распуская состав в k стеков, мы делим максимальную обратную цепь на k обратных цепей, поэтому длина максимальной обратной цепи в получившейся перестановке не меньше $\dfrac{f(\sigma)}{k}$.

Это дает оценку снизу, а так как наибольшей максимальной обратной цепью обладает перестановка $(n, n-1, ..., 2, 1)$ и ее длина равна n, то эту перестановку нельзя отсортировать за меньшее, чем $\lceil \log_k n \rceil$, число шагов.

Теперь возникает вопрос, что делать с каждой конкретной перестановкой. Например, состав уже отсортирован, т.е. идет в порядке 1, 2, 3, ..., $n - 1$, n. Было бы глупо его сортировать.

Упражнение 1. Посмотрите, что произойдет с этим составом, если к нему применить поразрядную сортировку, например, при $k = 2$?

Опять выручает лемма 1. Из нее следует уже менее очевидное утверждение.

Лемма 2. *Пусть в составе $(i+1)$-й вагон расположен правее i-го. Тогда для любого алгоритма (A), сортирующего состав, существует алгоритм (B), который не «хуже первого» (это означает, что новый алгоритм делает не больше шагов и не больше вытягиваний) и который при сортировке все время отправляет i-й и $(i+1)$-й вагоны в один стек.*

Действительно, изменим алгоритм (A): пусть в алгоритме (B) все вагоны попадают в те же стеки, что и в алгоритме (A), за исключением $(i+1)$-го — $(i+1)$-й вагон всегда распускается в тот же самый стек, что и i-й вагон в алгоритме (A). Ясно, что число вагонов между i-м и $(i+1)$-м вагонами не увеличивается. Пусть в самом начале между i-м и $(i+1)$-м вагонами стоял вагон с номером j. Если бы в алгоритме (A) j-й вагон все время попадал в тот же стек, что и i-й, то в конце мы получили бы неотсортированный состав. А так как состав в конце работы алгоритма (A) отсортирован, это означает, что в какой-то момент вагоны i и j попадают в разные стеки, и число вагонов между i-м и $(i+1)$-м уменьшается. А так как это верно для любого вагона, находящегося в начале между i-м и $(i+1)$-м вагонами, то в конце работы алгоритма (B) они будут стоять рядом и в нужном порядке. Это завершает доказательство леммы 2.

Следовательно, если в составе $(i+1)$-й вагон расположен правее i-го, то оба эти вагона каждый раз можно отправлять в один стек. А это, в свою очередь, означает, что эти вагоны не нужно различать и, следовательно, им нужно дать одинаковые номера.

Определение 2. Назовем каждую максимальную последовательность i, $i+1$, $i+2$, ..., $i+m$, в которой каждый следующий элемент стоит в перестановке правее предыдущего, *блоком*. Ясно, что элементам в блоке нужно давать один и тот же номер. Назовем эту операцию *перенумерацией*.

Пример 2. В составе из задачи 6 (5, 7, 1, 3, 2, 6, 4) 4 блока (1, 2; 3, 4; 5, 6 и 7), после перенумерации он будет выглядеть так: 2, 3, 0, 1, 0, 2, 1. А как выглядят после перенумерации составы из задач 7 и 8?

Лемма 3. *Пусть алгоритм (A) сортирует состав из n вагонов, идущих в обратном порядке. Тогда этот алгоритм так же (т.е. за такое же*