1 Обязательная часть

1. ***Куча с порядковыми статистиками.***
Описание структуры:

$m + 1$ пара куч. Пара состоит из двух куч с сылками друг на друга: в корне одной минимум, а в корне другой - максимум. В куче с максимум перых m куч должна лежать p_i статистика, а в последней - n статистика. То есть количество элементов в куче - $p_i - p_{i-1}$.
GetStatic(p_i):

p_i статистика по построению будет лежать в корне кучи на максимум из i пары куч. То есть мы можем достать ее за $\mathcal{O}(n)$.
Insert(x):

Бежим по парам куч слева направо. Ищем первую такую кучу, что ее порядковая статистика (корень кучи на макс.) окажется не меньше x. Это будет значить, что x принадлежит промежутку из этой кучи, то есть x адо сюда добавить, а статистику перенести в следующу кучу. Тогда статистику из следующей кучу надо перенести еще на одну вперед. И так далее, пока не достигнем последней кучи. Туда предыдущую статистику можно будет просто добавить. Если оказалось, что x больше любой статистики, то просто добавляем его в последнюю пару куч.

Как мы добавляем элемент a (x или предыдущую статистику). Запоминаем корень кучи p_i на максимум - то, что будем добавлять в слудующую кучу. Ставим на его место a. Делаем SiftDown. Он работает за $\log n$, в результате получаем новую кучу на максимум. Так как кучи из пары ссылаются друг на друга, то мы знаем место, где раньше была p_i в куче на минимум. Так как p_i была больше любого элемента, значит у нее нет детей. Ставим теперь на это место a, делаем SiftUp, работающий за $\log n$.

Таким образом обрабатываем все последующие пары куч. В последней паре ичего не удаляем, просто добавляем наш a. Так как для каждой пары мы работали за $\mathcal{O}(\log n)$, то Insert(x) работает за $\mathcal{O}(m \log n)$.
DelStatistic(p_i).

Бежим по парам куч слева направо. Ищем кучу с нужной статистикой. Из этой кучи надо удалить корень, а на его место поставить минимальный элемент из следующей пары куч. В следующей пары таким образом количество элементов уменьшится, статистику нужно будет обновить. Что добавить? Минимальный элемент из следующей пары. И так далее. В последней же куче ничего добавлять и обновлять не надо, там просто уменьшится количество элементов.

Как работаем с i парой, откуда надо удалить статистику. Берем кучу на максимум. В ее корень ставим корень кучи на минимум из следующей пары. Делаем SiftDown. Теперь куча на минимум. Берем то место, где раньше стояла статистика p_i. На него ставим все тот же корень минимума из следующей пары. Делаем Siftdown.
Как работаем со следующими парами. Пусть мы удалили вершину кучи на минимум из j пары. Теперь надо в эту пару добавить минимальный элемент a из следующей пары. Ставим его в вершину кучи на минимум, делаем SiftDown. В куче на максимум на место, где раньше стоял минимум из пары, тоже ставим a (минимум -> у него не было детей), делаем SiftUp.

Каждый раз делаем $\mathcal{O}(\log n)$ операций, следовательно, работает за $\mathcal{O}(m \log n)$.

2. ***Skew System***

- Если доказать, что для любого $a_k ak-1...a_2a_1$ верно, что $a_k ak-1...a_2a_1 < a_k ak-1...a_2a_1+1$, то мы докажем, что любые два числа в скошенной двоичной задают разные числа в десятичной (то же самое, что нас просят доказать). Докажем этот факт по индукции.
  База: $1=1<2=2<3=10<11=4<5=12$
  Переход: Пусть верно для чисел длины k. Докажем, что верно и для $k+1$.

  Если на конце числа стоял 0, то теперь там будет стоять 1, а остальная часть не изменится. То есть в десятичной системе это число изменится на $1\cdot(2^1-1)-0\cdot(2^1-1)=1$. Как мы видим, оно увеличится. Отлично.

  Если на конце стояла 1, то станет двойка, остальная часть не изменится, а десятичное число увеличится на 1. Супер.

  Теперь что происходит, если на конце стояла 2. Это значит, что теперь в конце будет стоять 0, а следующая цифра увеличится на 1 и станет 1 или 2 (перед 2 по условию не могла стоять 2, то есть перехода точно не будет). На сколько изменится изначальное число? Если после 2 шел 0, то

  $$(1\cdot(2^2-1)+0\cdot(2^1-1))-(0\cdot(2^2-1)+2\cdot(2^1-1))=3-2=1.$$

  То есть оно увеличится. Если же перед 2 шла 1, то

  $$(2\cdot(2^2-1)+1\cdot(2^1-1))-(1\cdot(2^2-1)+2\cdot(2^1-1))=6-5=1.$$

  Опять-таки оно увеличится. Просто великолепно. Получаем, что два разных числа в скошенной двоичной задают разные числа в десятичной, что и требовалось доказать.

- Способ, как увеличить на единицу, я описала в предыдущем пункте. Работает за $\mathcal{O}(1)$, так как я в худшем случае работаю с двумя последними цифрами. Как применить это к биномиальной куче. Заметим, что:

  2=10
  20=110
  200=1110
  2(0)xk=(1)x(k+1)0
  1=1
  10=11
  100=111
  1(0)xk=(1)x(k+1)

  То есть если раньше мы строили биномиальную кучу из n элементов, переводя n в двоичную, а затем соединяя все получившиеся биномиальные деревья, то теперь мы переводим n в скошенную, строим биномиальные подкучи, состоящие из биномиальных деревьев, а затем соединям эти подкучи (их корни).

  Как происходит Add. Раньше у нас могло происходить зануление хвоста сколь угодной длины, из-за этого время работы $\mathcal{O}(1)$ было амортизированным. Теперь же мы даже в худшем случае будем работать лишь с двумя последними подкучами, то есть время работы будет $\mathcal{O}(1)$ не амортизированным.

3. ***Weak Heap***

- Минимум в Weak Heap всегда лежит в корне. По определению у корня есть только правое поддерево, а все вершины правого поддерева какой-либо вершины должны быть не больше этой вершины. То есть в корне самая маленькая вершина. Ищем минимум за $\mathcal{O}(1)$.

- Мы можем за единицу узнать (при помощи массива r), являемся ли корнем левого или правого поддерева нашего предка. До тех пор, пока левое - мы не конфликтуем с определнием Weak Heap вне зависимости от нашего значения. Как только мы находим предка, для которого находимся в правом поддереве - проверяем, не конфликтуем ли мы с определением. То есть если этот предок оказался больше меня, то надо поменять местами. Предок же в свою очередь был меньше всех вершин в поддереве кроме меня, то есть новых конфликтов не возникнет. Если я больше предка, то так и оставляю элемент на месте. Так бежим по предкам до самого корня. Сверяемся с ним и заканчиваем пробег.

  Алгоритм работает за максимальное количество предков, которое может быть. А это число не больше высоты кучи. То есть $\mathcal{O}(\log n)$, что и требовалось.

- В корне всегда хранится первый минимум. Уберем корень, посмотрим на оставшуюся кучу. Заметим, что ее минимум всегда будет находится в самой левой ветке кучи. Почему. По определению лишь элементы левого поддерева могут быть меньше вершины. То есть, если идти от корня кучи, то элемент меньше рассматриваемого может находиться только слева от него.

  Как ищем второй минимум. Запускаемся от корня правого поддерева WeakHeap, все время спускаемся по нему налево, попутно реклаксируя ответ. Работать будет за высоту кучи, то есть $\mathcal{O}(\log n)$.

- Смотрим на левую нижнюю вершину. Для удобства пронумеруем вершины сверху вниз от 0 до i (очевидно, что $i = \lfloor \log n \rfloor + 1$). То есть мы движемся от i к 1 вершине. Пусть сейчас рассматриваем j. Если $a[j] < a[0]$ - меняем местами j вершину и корень. Теперь заметим, что все вершины с номерами больше j у нас уже были расположены правильно, то есть они все были больше корня и их поддеревья соответствовали правилам WeakHeap. То есть теперь, когда мы поменяли корень и j вершину, левое поддерево соотвествует определению правого, тогда как про правое мы нчиего не знаем. Ну и поменяем их мстами (делается за единицу заменой числа в массиве r). И так идем до верху.

  Работает за количество рассматриваемых вершин - i, то есть за $\mathcal{O}(\log n)$.

- Add работает при помощи SiftUp - добавляем вершину в конец и делаем для нее SiftUp. В описанном выше SiftUp мы делали сравнений не больше, чем количество наших предков (худший случай - мы в правом поддереве для всех предков). В худшем же случае у нас предков $\lfloor \log n \rfloor + 1$. То есть мы делали максимум столько сравнений. Заметим, что для $n \geq 8$ верно, что $n \leq 2^{\frac{1}{2}n-1}$, то есть $\frac{\log n+1}{n} \leq 0.5$. А что тут тогда оптимизировать. Мы не можем делать отрицательное количество сравнений. Тогда вариант, описанный выше, идеален.