Бабушкин А.

1 Обязательные задачи

1. Разделяйка. Отсортируем точки по x-координате, потом будем делать следующее: разделим их пополам вертикальной прямой, на нашу текущую прямую спроецируем все точки и добавим эти проекции в ответ, а затем запустимся рекурсивно от левой и правой части. Очевидно, работает $O(n \log n)$, надо доказать корректность. Рассмотрим две точки. Если они обе слева или обе справа от моей прямой, то им сделает хорошо один из рекурсивных вызовов. Если же по разные стороны, то их проекции на мою прямую лежат на границах прямоугольника, построенного на них, то есть как раз есть ещё хотя бы одна точка в этом прямоугольнике, что нам и требовалось. чтд.

2. Раз мы знаем количество запросов заранее, то давайте сделаем $\sqrt{n}$ блоков по $\sqrt{n}$ элементов в каждом. В каждом блоке будем поддерживать минимум, как и минимум во всей куче. Add будет просто добавлять в последний блок (или в новый, если последний полон), Min будет выводить минимум во всей куче, Decrease будет обновлять минимум в блоке и глобальный минимум, Merge обновит глобальный минимум и, возможно, добавит один плохой (т.е. неполный) блок, но его всё ещё можно сделать за O(1), т.к. мы умеем делать Merge списков за O(1). Осталось научиться доставать минимум. Давайте пробежимся по всем блокам, чтобы узнать, где лежит минимум. Потом пробежимся по нужному блоку, достанем минимум из него. Теперь наш блок слишком маленький. Возьмём один элемент из последнего блока и перекинем в текущий. Пересчитаем минимумы в текущем и последнем блоке, а затем пробежкой по всем блокам – и глобальный минимум. Если у нас было больше одного плохого блока, то смёржим их вместе, получив несколько хороших и, возможно, ещё один плохой.
Update: ExtractMin отработает за $n^{3/2}$, т.к. на каждом его вызове будет O(1) пробегов по блокам, которых $\sqrt{n}$, и O(1) пробегов внутри блока, и каждый блок тоже размера $\sqrt{n}$, а всего вызовов не более n.

3. Будем держать m+1 кучу, каждая из которых будем на min и на max. В первой будут элементы, не большие p_1-ой статистики, во второй – не большие p_2-ой и так далее. В последней будут большие p_m-ой. Тогда get будет действительно работать за O(1) – надо только посмотреть на наибольший элемент соотв. кучи. Если надо добавить элемент, то добавляем его в первую кучу, если в ней теперь слишком много элементов, то достаём из неё максимум и кидаем его во вторую, переходим ко второй, если в ней слишком много, то достаём из неё и кидаем в третью и т.д. Каждая операция перекидывания работает за $O(\log n)$, перекидываний не больше, чем куч, т.е. add работает за $O(m \log n)$. Удаление точно так же – удалили элемент, если теперь слишком мало в куче, то взяли из следующей, если в ней слишком мало, то из следуюей за ней и т.д.
Update: Каждая куча – min-max куча, так что мы умеем извлекать и наименьший элемент, и наибольший.

4.

a) Он лежит в корне.

b) Идём от нашего элемента вверх, пока не станем чьим-то правым сыном. Теперь смотрим конфликт с отцом. Если его нет, то всё ок, завершаемся. Если он есть, то меняемся с ним местами и продолжаем SiftUp от своего нового места.

c) Начнём с правого сына корня. Мы должны найти минимум в его поддереве. Справа от него минимума быть не может, т.к. справа от него элементы все элементы не меньше его. Тогда минимум это или он, или минимум в его левом поддереве. Для его левого сына рассуждения точно такие же, поэтому ответ – это просто взять правого сына корня и идти от него влево, пока есть куда, а затем вернуть минимум среди вершин на пройденном пути.

2 Дополнительные задачи

1.