Бабушкин А.

1 Обязательные задачи

1.

a) $\log \frac{n!}{100^n} = \log n! - \log 100^n = \Theta(n \log n) - n \log 100 = \Theta(n \log n)$.
b) Если бы было можно, то heapsort работал бы за $n \cdot o(\log n) = o(n \log n)$, а так нельзя.
c) Если бы было можно, то mergesort работал бы за $o(n) \cdot \log n = o(n \log n)$, а так нельзя.

2.

3.

a) Scanline, события открыть/закрыть отрезок. Храним текущую l – левую границу объединения. Когда закрываем отрезок в позиции r, то если после этого события открытых отрезков 0, то прибавляем r - l к ответу. Когда открываем отрезок в позиции x, если до этого открытых 0, то l = x. Работает за $n \log n$, потому что сканлайн.
b) Рассмотрим все отрезки с $l \leq 0$. Понятно, что среди таких нам надо взять хотя бы один. Тогда, очевидно, нам выгоднее всего выбрать тот, у которого r максимально – любой другой не сделает ответ лучше. Все остальные можно выкинуть, они не покроют ничего нужного, что мы ещё не покрыли. Затем рассмотрим те, у которых $l \leq r$, снова выберем с максимальной правой границей. И так до тех пор, пока не покроем M или не кончатся отрезки. Работает за $n \log n$ – сначала сортируем всё по левой границе, потом будет блоками пихать в сет, который сортит их по правой.
Update: Можно не пихать в сет, а просто выбрать отрезок с максимальной правой границей среди всех, что мы рассматриваем на этом шаге. Тогда всё решение работает за $Sort() + O(n)$.
c) Т.к. все отрезки только растут, то если мы можем покрыть нужный отрезок в момент времени t, то и дальше точно сможем, поэтому можно бинарить. Внутри бинаря проверка из пункта b, решение итого за $n \log n \log MAX$. Если r_i не возрастает, то можно за $n \log MAX$, потому что не придётся каждый раз сортить заново отрезки. Если числа в задаче целые (и лучше небольшие), а не вещественные, то проверку можно делать не за $n \log n$, а за снм, потому что каждый отрезок будет сжимать несколько маленьких в один.
А ещё есть не факт что работающее решение за $O(n \log n)$. Рассмотрим отрезок между двумя точками. Нам интересно, когда он "схлопнется". Построим convex hull на точках слева от него и справа, где x-координатой convex hull'а будет координата на прямой, а y-координатой – время, когда самая быстрая из точек расширится до этой координаты. Тогда ответом для точки в отрезке будет максимум из ответа левого convex hull'а и правого, а ответом для отрезка – минимум среди всех точек на нём. То есть ответом для отрезка будет точка, когда левый convex hull и правый пересекаются – это и будет оптимальной точкой схлапывания. Если эти convex hull'ы пересекаются не в нашем текущем

отрезке, то можно просто взять лучший из концов отрезка. Очевидно, что эта точка будет двигаться только вправо, если мы перебираем отрезки слева направо, поэтому её можно поддерживать указателем. Осталось понять, как поддерживать convex hull'ы. Левый очевидно – в него мы только добавляем. Правый сложнее – из него мы удаляем прямые. Можно сделать convex hull с откатами (запоминать, какие прямые убрали), а можно персистентный, потому что это то же самое, что персистентный массив, которые делается за $O(n \log n)$, то есть не испортит нашу асимптотику. Наконец, ответ на всю задачу это максимум из ответов для отрезков.
Update: Время схлопывания считаем именно через convex hull. Пусть у нас есть отрезок, который сначала задан точкой (x_i, r_i). Посмотрим, когда он "накроет" точку $x : x > x_i$. Он это сделает во время $t = \frac{x-x_i}{r_i} = \frac{x}{r_i} - \frac{x_i}{r_i}$, второе слагаемое – константа, а первое – прямая, зависящая от x. Аналогично считаем, когда точку накроет справа. Точка будет накрыта, если её накроют или слева, или справа, а на отрезке худшая точка – та, для которой этот момент наступит позже всего, то есть её одновременно накроют слева и справа $\rightarrow$ это пересечение левого convex hull'а и правого.

4. Посмотрим на элементы сверху от нашего. Они образуют убывающую последовательность, если считать снизу. Мы проскачем наверх какой-то префикс этого пути. Если выписать элементы сверху в массив, то можно этот префикс побинарить, длина массива $O(\log n) \rightarrow O(\log \log n)$ сравнений во время бинаря.

5. То же самое, что и в предыдущем, только мы не знаем сначала путь. Но мы знаем, что если мы пихнём наш элемент вниз, то всегда в меньшего сына, а не в большего. Спутимся до листьев, спускаясь каждый раз в меньшего сына – на сравнение сыновей нужно $\log_2 n$ сравнений, а дальше тот же бинарь.

2 Дополнительные задачи

1. Рассмотрим массив в отсортированном порядке. Мы должны сравнить каждую пару соседних в нём, потому что если не сравним, то не сможем сказать, различны ли они. То есть нам нужны хотя бы эти сравнения. Если мы сможем получить список этих необходимых сравнений за $o(n \log n)$, то по этому списку мы восстановим сортировку, а так нельзя. То есть эти сравнения быстрее чем за $\Omega(n \log n)$ мы не выясним. чтд.
Update: Пусть мы провели все нужные сравнения, то есть для каждой пары элементов, которые соседи в отсорченном массиве, узнали, равны ли они, и записали результаты этих сравнений. Тогда давайте теперь за 0 сравнений восстановим сортировку. У нас есть элемент, для которого не нашлось меньшего – выпишем его первым. Теперь у нас есть элемент, для которого единственным меньшим был уже выписанный. Выпишем его вторым. И т.д. То есть мы сделали $o(n \log n)$ сравнений, но умеем восстанавливать отсорченный массив. Fail.

2.

3. Рассмотрим отрезки в направлении по часовой стрелке. Для каждого отрезка есть один оптимальный – тот, у которого правая граница левее всего, но который справа от нашего. Легко для каждого отрезка найти такой оптимальный, посортировав их. Проведём из каждого отрезка ребро в оптимальный к нему, построив функциональный граф. Теперь на таком графе можно построить двоичные подъёмы и затем для вершины за $\log n$ отвечать, как много отрезков мы можем взять, если начнём от неё. Итого $n \log n$.