Форма 1: разбор

1:

> Мы обсудили алгоритм линейного поиска, работающий за O(n) и алгоритм бинарного поиска, работающий за O(log n). Значит ли это, что на любых входных данных бинарный поиск делает меньше операций, чем линейный поиск? Приведите пример при n = 100, где линейный поиск найдет первую 1 быстрее бинарного

Это, конечно, неправда. Есть разница между worst-case, best-case и average-case сценариями. В асимптотиках мы оцениваем worst-cast, но это не значит, что best-case не может быть моментальным даже для медленных алгоритмов.

В случае бинпоиска, его best-case и worst-case совпадают и соответствуют $\log n$. Асимптотика линейного поиска зависела от положения ответа в массиве. Например, в тесте $[0,\ 1,\ 1,\ 1,\ \ldots]$ линейный поиск найдет первую единицу за два сравнения.

2:

> Мы хотим протестировать, что вы умеете писать бинарный поиск. Считывание массива из n чисел работает за O(n). Получается, если давать тесты на ввод, то бинпоиск не будет "слабым звеном" алгоритма, и даже если вы напишете линейный поиск, то асимптотика не изменится (O(n) на чтение данных через input() + O(n) на линейный поиск = O(n)). Мы планируем дать один массив размера n и попросить на нем сделать q разных поисков, чтобы получить от вас "составную" асимптотику O(n + q log n). Какое нам сделать q, чтобы поиск заметно перевешивал считывание данных? Только так мы сможем замерить, что ваше решение делает поиск быстрее, чем за O(n)

Это задача про сложение асимптотик и балансирование между компонентами. В целом, нам подойдет любой вариант, где $O(q \log n + n) = O(q \log n)$. Для этого слагаемые должны либо совпадать $q = \frac{n}{\log n}$, либо иметь один порядок, но чтобы запросы перевешивали $q = 10 \frac{n}{\log n}$, либо перевешивать на порядок $q = n$. Все три варианта ок, но в практических целях лучше брать один порядок с перевешиванием $q = c \frac{n}{\log n}$. Так мы уже точно отсечем долгий алгоритм, потому что на практике ввод будет сильно быстрее поиска, но при этом соблюдем баланс между слагаемыми.

3:

> Мы придумали на лекции, как делать бинпоиск, если известны и левая и правая границы (то есть область, где мы ищем смену 0 на 1). А что делать, если границы области заранее неизвестны? Например, кто-то загадал натуральное число n, которое может быть каким угодно большим (проигнорируем вопрос того, влезет ли это число в память компьютера), а мы все равно хотим найти границы и сделать бинпоиск вопросами "число n меньше чем x?". Предложите идею, как найти хотя бы одно число, большее n, за O(log(n)) - после этого мы сможем запустить бинпоиск. Вопрос со звездочкой и неправильный ответ / отсутствие не штрафуется.

Идея в том, чтобы удваивать нашу правую границу каждый раз, пока она меньше n. Мы, с одной стороны, не знаем n, поэтому алгоритм может работать очень долго. С другой стороны, поскольку n определено заранее, то в какой-то момент мы его найдем. И в этот момент, поскольку мы начали с `r = 1` и делали шаги с `r *= 2`, мы явно перебирали только степени двойки, сделав $\lceil \log n \rceil$ шагов.

Ну и потом, имея левую и правую границу, можно уже сделать обычный бинпоиск за логарифм