Наибольшая возрастающая подпоследовательность

"Найти длину самой большой возрастающей подпоследовательности в массиве."

Есть последовательность: 5, 10, 6, 12, 3, 24, 7, 8

Вот примеры подпоследовательностей:
10, 3, 8
5, 6, 3

А вот примеры возрастающих подпоследовательностей:
5, 6, 7, 8
3, 7, 8

А вот примеры возрастающих подпоследовательностей наибольшей длины:
5, 6, 12, 24
5, 6, 7, 8

Да, максимальных тоже может быть много, нас интересует лишь длина. Здесь она равна 4.

Решение за O(n ^ 2)

Есть последовательность: **5, 10, 6, 12, 3, 24, 7, 8**

Рассмотрим следующую возрастающую подпоследовательность:
5, 6, 12
теперь взглянем на следующее число после последнего элемента в последовательности — это **3**.
Может ли оно быть продолжением нашей последовательности? Нет. Оно меньше чем 12.
А **24** ?
Оно да, оно может.
Соответственно длина нашей последовательности равна теперь **3 + 1**, а последовательность выглядит так:
5, 6, 12, 24
Вот где переиспользование предыдущих вычислений: мы знаем, что у нас есть подпоследовательность 5, 6, 12, которая имеет длину 3 и теперь нам надо как-то легко добавить к ней **24**.

Давайте заведем еще один дополнительный массив (вот оно наше ДП), в котором будем хранить размер возрастающей подпоследовательности для n-го элемента.
Выглядеть это будет так:

	i	j						
numbers	5	10	6	12	3	24	7	8
counts	1	1	1	1	1	1	1	1

Наша задача — заполнить массив counts правильными значениями. Изначально он заполнен единицами, так как каждый элемент сам по себе является минимальной возрастающей подпоследовательностью.

i и j - это индексы итераторов по массиву, которые мы будем использовать. Изменяться они будут с помощью двух циклов, один в другом. i всегда будет меньше чем j.

Сейчас j смотрит на 10 — это наш кандидат в члены последовательностей, которые идут до него. Посмотрим туда, где i, там стоит 5.

10 больше 5 и 1 <= 1, counts[j] <= counts[i]? Да, значит counts[j] = counts[i] + 1,

Теперь таблица выглядит так.

	i	j						
numbers	5	10	6	12	3	24	7	8
counts	1	2	1	1	1	1	1	1

Смещаем j.

	i		j					
numbers	5	10	6	12	3	24	7	8
counts	1	2	1	1	1	1	1	1

	i		j					
numbers	5	10	6	12	3	24	7	8
counts	1	2	2	1	1	1	1	1

		i	j					
numbers	5	10	6	12	3	24	7	8
counts	1	2	2	1	1	1	1	1

	i			j				
numbers	5	10	6	12	3	24	7	8
counts	1	2	2	1	1	1	1	1

	i			j				
numbers	5	10	6	12	3	24	7	8
counts	1	2	2	2	1	1	1	1

		i		j				
numbers	5	10	6	12	3	24	7	8
counts	1	2	2	2	1	1	1	1

		i		j				
numbers	5	10	6	12	3	24	7	8
counts	1	2	2	3	1	1	1	1

			i	j				
numbers	5	10	6	12	3	24	7	8
counts	1	2	2	3	1	1	1	1

	i				j			
numbers	5	10	6	12	3	24	7	8
counts	1	2	2	3	1	1	1	1

		i			j			
numbers	5	10	6	12	3	24	7	8
counts	1	2	2	3	1	1	1	1

			i		j			
numbers	5	10	6	12	3	24	7	8
counts	1	2	2	3	1	1	1	1

				i	j			
numbers	5	10	6	12	3	24	7	8
counts	1	2	2	3	1	1	1	1

	i					j		
numbers	5	10	6	12	3	24	7	8
counts	1	2	2	3	1	1	1	1

	i					j		
numbers	5	10	6	12	3	24	7	8
counts	1	2	2	3	1	2	1	1

		i				j		
numbers	5	10	6	12	3	24	7	8
counts	1	2	2	3	1	2	1	1

		i				j		
numbers	5	10	6	12	3	24	7	8
counts	1	2	2	3	1	3	1	1

			i			j		
numbers	5	10	6	12	3	24	7	8
counts	1	2	2	3	1	3	1	1

				i		j		
numbers	5	10	6	12	3	24	7	8
counts	1	2	2	3	1	3	1	1

				i		j		
numbers	5	10	6	12	3	24	7	8
counts	1	2	2	3	1	4	1	1

					i	j		
numbers	5	10	6	12	3	24	7	8
counts	1	2	2	3	1	4	1	1

	i						j	
numbers	5	10	6	12	3	24	7	8
counts	1	2	2	3	1	4	1	1

	i						j	
numbers	5	10	6	12	3	24	7	8
counts	1	2	2	3	1	4	2	1

		i					j	
numbers	5	10	6	12	3	24	7	8
counts	1	2	2	3	1	4	2	1

			i				j	
numbers	5	10	6	12	3	24	7	8
counts	1	2	2	3	1	4	2	1

			i				j	
numbers	5	10	6	12	3	24	7	8
counts	1	2	2	3	1	4	3	1

				i			j	
numbers	5	10	6	12	3	24	7	8
counts	1	2	2	3	1	4	3	1

					i		j	
numbers	5	10	6	12	3	24	7	8
counts	1	2	2	3	1	4	3	1

						i	j	
numbers	5	10	6	12	3	24	7	8
counts	1	2	2	3	1	4	3	1

	i							j
numbers	5	10	6	12	3	24	7	8
counts	1	2	2	3	1	4	3	1

	i							j
numbers	5	10	6	12	3	24	7	8
counts	1	2	2	3	1	4	3	2

		i						j
numbers	5	10	6	12	3	24	7	8
counts	1	2	2	3	1	4	3	2

			i					j
numbers	5	10	6	12	3	24	7	8
counts	1	2	2	3	1	4	3	2

				i				j
numbers	5	10	6	12	3	24	7	8
counts	1	2	2	3	1	4	3	2

					i			j
numbers	5	10	6	12	3	24	7	8
counts	1	2	2	3	1	4	3	2

						i		j
numbers	5	10	6	12	3	24	7	8
counts	1	2	2	3	1	4	3	2

							i	j
numbers	5	10	6	12	3	24	7	8
counts	1	2	2	3	1	4	3	2

							i	j
numbers	5	10	6	12	3	24	7	8
counts	1	2	2	3	1	4	3	4

							i	j
numbers	5	10	6	12	3	24	7	8
counts	1	2	2	3	1	4	3	4

Результат выделен цветом.

Псевдокод:

```
for (int j = 1; j < numbers.count; j++) {
   for (int k = 0; k < j; k++) {
     if (numbers[j] > numbers[k]) {
        if (lengthOfSubsequence[j] <= lengthOfSubsequence[k]) {
              lengthOfSubsequence[j] = lengthOfSubsequence[k] + 1;
                }
            }
        }
    }

int maximum = 0;
    for (int length in lengthOfSubsequence) {
        maximum = MAX(maximum, length);
    }
    return maximum;
}
```

Вы не могли не заметить два вложенных цикла в коде, а там где есть два вложенных цикла проходящих по одному массиву, есть и квадратичная сложность O(n^2), что обычно не есть хорошо.

Чтобы ускорить работу алгоритма нам нужно вспомнить, что такое бинарный поиск.

Бинарный поиск O(log n)

Бинарный поиск работает только на отсортированных массивах. Например, нам нужно найти позицию числа n в отсортированном массиве:
1, 5, 6, 8, 14, 15, 17, 20, 22

Зная что массив отсортирован, мы всегда можем сказать правее или левее определенного числа в массиве искомое число должно находиться.

Мы ищем позицию числа 8 в этом массиве. С какой стороны от середины массива оно будет находиться? 14 — это число в середине массива. 8 < 14 — следовательно 8 левее 14. Теперь нас больше не интересует правая часть массива, и мы можем ее отбросить и повторять ту же самую операцию вновь и вновь пока не наткнемся на 8. Как видите, нам даже не нужно проходить по всем элементам массива, сложность этого алгоритма < O(n) и равна O (log n).

Решение за O (n * log n)

Теперь мы будем проходить по нашему исходному массиву при этом заполняя новый массив, в котором будет храниться возрастающая подпоследовательность. Еще один плюс этого алгоритма: он находит не только длину максимальной возрастающей подпоследовательности, но и саму подпоследовательность.

Как же двоичный поиск поможет нам в заполнении массива подпоследовательности?

С помощью этого алгоритма мы будем искать место для нового элемента в вспомогательном массиве, в котором мы храним для каждой длины подпоследовательности минимальный элемент, на котором она может заканчиваться.

Если элемент больше максимального элемента в массиве, добавляем элемент в конец. Это просто.

Если такой элемент уже существует в массиве, ничего особо не меняется. Это тоже просто.

Что нам нужно рассмотреть, так это случай когда следующий элемент **меньше максимального** в этом массиве. Понятно, что мы не можем его поставить в конец, и он не обязательно вообще должен являться членом именно максимальной последовательности, или наоборот, та подпоследовательность, которую мы имеем сейчас и в которую не входит этот новый элемент, может быть не максимальной.

Сведем рассуждения к рассмотрению 2-х оставшихся случаев.

Рассматриваемый элемент последовательности (x) меньше чем наибольший элемент в массиве (Nmax), но больше чем предпоследний.

Рассматриваемый элемент меньше какого-то элемента в середине массива.

В случае 1 мы просто можем откинуть Nmax в массиве и поставим на его место x. Так как понятно, что если бы последующие элементы были бы больше чем Nmax, то они будут и больше чем x — соответственно мы не потеряем ни одного элемента.

Случай 2: для того чтобы этот случай был нам полезен, мы заведем еще один массив, в котором будем хранить размер подпоследовательности, в которой этот элемент является максимальным. Собственно этим размером и будет являться та позиция в первом вспомогательном массиве для этого элемента, которую мы найдем с помощью двоичного поиска. Когда мы найдем нужную позицию, мы проверим элемент справа от него и заменим на текущий, если текущий меньше (тут действует та же логика как и в первом случае)

Нам нужны:
1) Исходная последовательность
2) Создаем массив, где будем хранить возрастающие элементы для подпоследовательности
3) Создаем массив размеров подпоследовательности, в которой рассматриваемый элемент является максимальным.

numbers	5	10	6	12	3	24	7	8
indexes	0	intMax	intMax	intMax	intMax	intMax	intMax	intMax
minElement ForLength	intMax	intMax	intMax	intMax	intMax	intMax	intMax	intMax

numbers	5	10	6	12	3	24	7	8
indexes	0	1	intMax	intMax	intMax	intMax	intMax	intMax
minElement ForLength	5	10	intMax	intMax	intMax	intMax	intMax	intMax

numbers	5	10	6	12	3	24	7	8
indexes	0	1	1	intMax	intMax	intMax	intMax	intMax
minElement ForLength	5	6	intMax	intMax	intMax	intMax	intMax	intMax

numbers	5	10	6	12	3	24	7	8
indexes	0	1	1	2	intMax	intMax	intMax	intMax
minElement ForLength	5	6	12	intMax	intMax	intMax	intMax	intMax

numbers	5	10	6	12	3	24	7	8
indexes	0	1	1	2	0	intMax	intMax	intMax
minElement ForLength	3	6	12	intMax	intMax	intMax	intMax	intMax

numbers	5	10	6	12	3	24	7	8
indexes	0	1	1	2	0	3	intMax	intMax
minElement ForLength	3	6	12	24	intMax	intMax	intMax	intMax

numbers	5	10	6	12	3	24	7	8
indexes	0	1	1	2	0	3	2	intMax
minElement ForLength	3	6	7	24	intMax	intMax	intMax	intMax

Результат:

numbers	5	10	6	12	3	24	7	8
indexes	0	1	1	2	0	3	2	3
minElement ForLength	3	6	7	8	intMax	intMax	intMax	intMax

Чтобы восстановить подпоследовательность, надо от максимального indexes пройти влево и искать меньший элемент с indexes на 1 меньший текущего