Санкт-Петербургский Национальный Исследовательский Университет ИТМО

Факультет прикладной информатики

**Лабораторная работа №6**

Выполнил:

Сакулин И. М.

Проверил

Мусаев А. А.

Санкт-Петербург,

2024

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 3](#_Toc184137555)

[Задача «187. Repeated DNA Sequences». 4](#_Toc184137556)

[Задача «275. H-Index II» 6](#_Toc184137557)

[Задача «1051. Height Checker» 8](#_Toc184137558)

[Задача «1346. Check If N and Its Double Exist » 10](#_Toc184137559)

[Задача «1455. Check If a Word Occurs As a Prefix of Any Word in a Sentence » 13](#_Toc184137560)

[Задача «1636. Sort Array by Increasing Frequency» 15](#_Toc184137561)

[ЗАКЛЮЧЕНИЕ 17](#_Toc184137562)

ВВЕДЕНИЕ

В данной лабораторной работе требуется взять 6 заданий с leetcode.com по пройденным темам (не более 2 заданий по одной теме) и решить их, попав в лучшие 80% решений по памяти и по скорости, или в лучшие 90% по какому-то одному из критериев. Сохранить скрин с результатами с сайта и описать алгоритмы. Задания можно брать любой сложности.

Цель работы: получить практические навыки в решении задач, соблюдая условия.

Пройденные темы:

* бинарный поиск
* сортировка
* поиск подстрок
* хеширование

Задача «187. Repeated DNA Sequences».

**Темы:** хеширование и поиск подстрок.

**Задание**. Последовательность ДНК состоит из ряда нуклеотидов, сокращенно обозначаемых как "A", "C", "G" и "T".

Например, "ACGAATTCCG" – это последовательность ДНК. При изучении ДНК полезно идентифицировать повторяющиеся последовательности в ДНК.

Задана строка s, представляющая последовательность ДНК, верните все последовательности длиной в 10 букв (подстроки), которые встречаются в молекуле ДНК более одного раза. Вы можете вернуть ответ в любом порядке (рисунок 1).

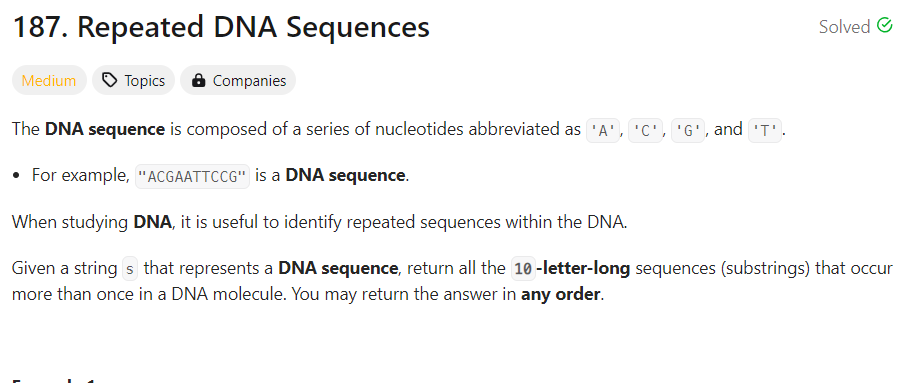


Рисунок 1 – Задача 187

**Результаты** представлены на рисунке 2.

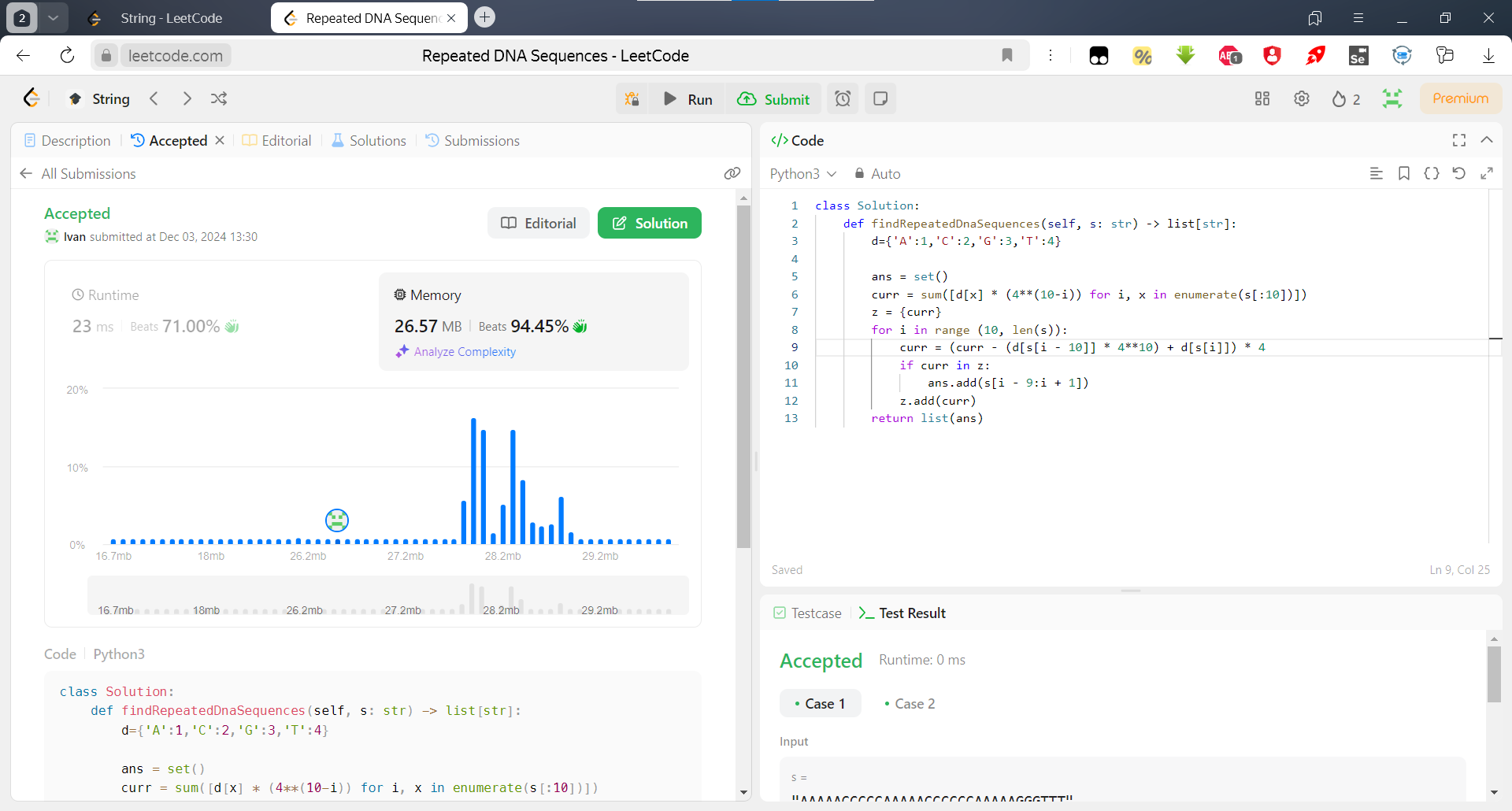


Рисунок 2 – Результаты решения задачи 187

**Решение** изображено на рисунке 3.

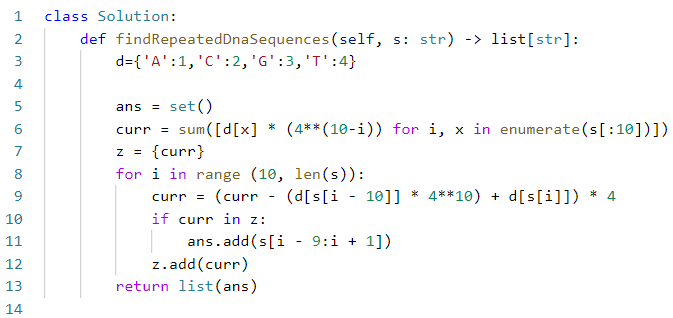


Рисунок 3 – Решение задачи 187

**Описание алгоритма.**

Основной принцип работы алгоритма заключается в том, что мы создаём скользящий hash, который полностью уникален вследствие его строения. Ставим окно в 10 символов, считаем hash подстроки, затем посимвольно сдвигаем окно вправо, параллельно изменяя hash. Для каждой итерации смотрим:

* Если такой hash уже встречался (то есть он есть в z), добавляем строку в ответ
* Если такой hash встретился впервые, добавляем в z

Сложность алгоритма O(n) – линейная, потому что мы 1 раз проходим по строке, а поиск в объекте set почти константная O(1) операция. Пространственная сложность O(n), где n – длина строки.

Задача «275. H-Index II»

**Тема:** бинарный поиск.

**Задание**: Учитывая массив целых чисел citations, где citations[i] - количество цитирований, полученных исследователем за свою i-ю статью, и цитаты отсортированы в порядке возрастания, верните h-индекс исследователя.

Согласно определению индекса Хирша в Википедии: Индекс Хирша определяется как максимальное значение h, такое, что данный исследователь опубликовал не менее h работ, каждая из которых цитировалась не менее h раз.

Вы должны написать алгоритм, который выполняется за логарифмическое время (рисунок 4).

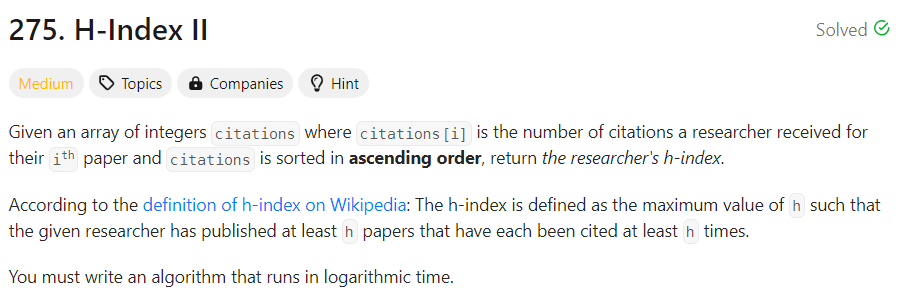


Рисунок 4 – Задача 275

**Результаты** представлены на рисунке 5.

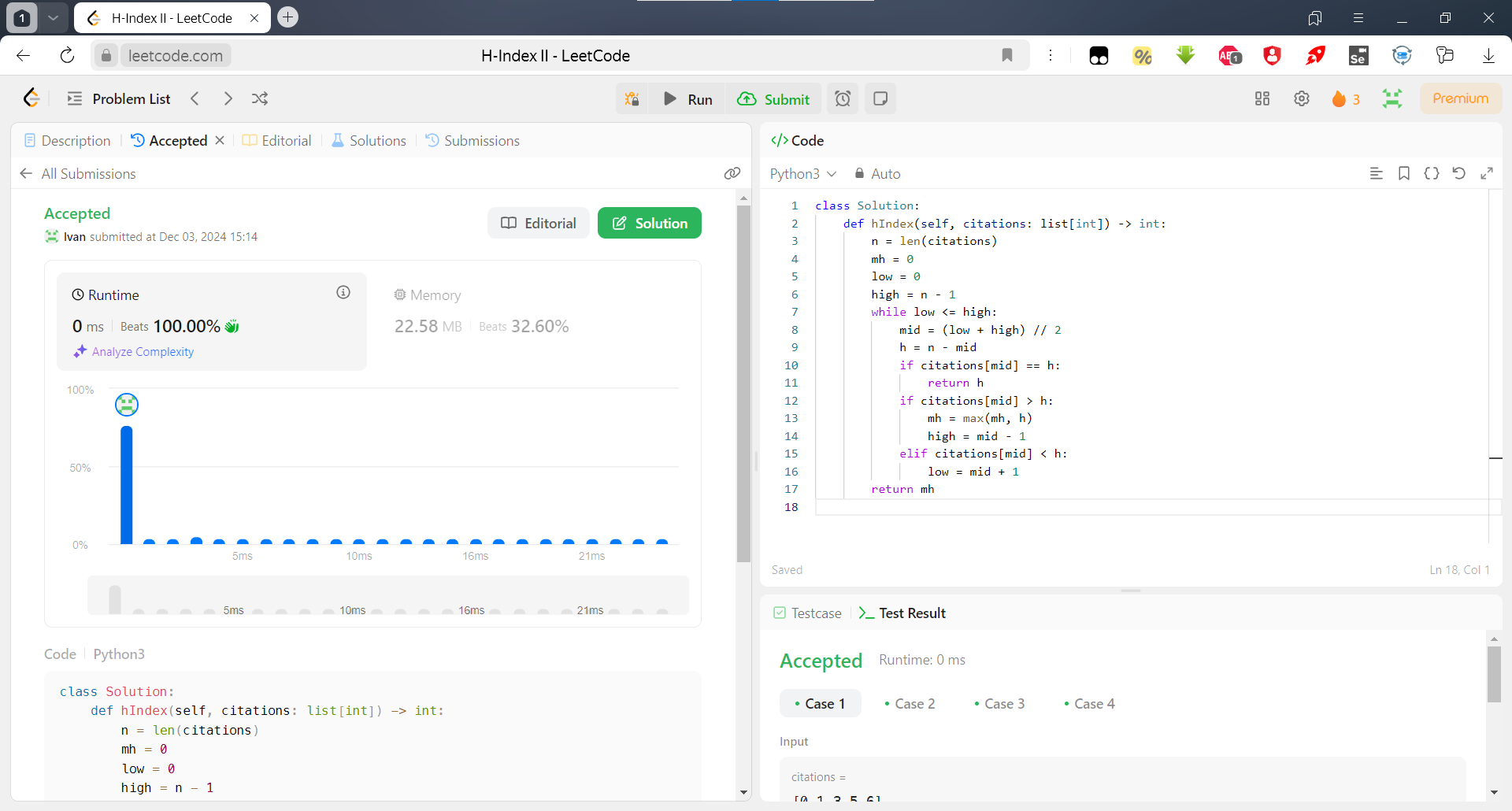


Рисунок 5 – Результаты решения задачи 275

**Решение** изображено на рисунке 6.

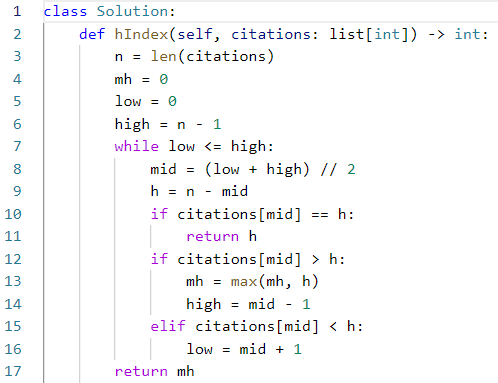


Рисунок 6 – Решения задачи 275

**Описание алгоритма.**

Решение представляет из себя видоизменённый алгоритм бинарного поиск: данные всё также делятся пополам, но условия сдвига границ и выхода изменены, рассмотрим 3 случая:

1. Количество цитат текущей статьи равно количеству статей после него. Тогда мы можем прийти к однозначному выводу, что это и есть h-индекс. Все статьи правее будут иметь по крайней мере столько же цитат. Все цитаты левее будут иметь меньше цитат, чем нужно.
2. Количество цитат больше оставшихся статей, тогда мы определяем больший индекс из всех таких случаев и сдвигаем правую границу.
3. В последнем случае просто сдвигаем левую границу.

Сложность алгоритма логарифмическая, дополнительное пространство – константа.

Задача «1051. Height Checker»

**Тема:** сортировки.

**Задание**: Школа пытается сделать ежегодную фотографию всех учащихся. Учащихся просят выстроиться в одну шеренгу в порядке неубывания роста. Пусть этот порядок будет представлен целочисленным массивом expected, где expected[i] - ожидаемый рост i-го ученика в очереди.

Вам будет предоставлен целочисленный массив heights, представляющий текущий порядок расположения учеников. Каждая высота[i] – это рост i-го ученика в строке (с индексом 0). Возвращает количество индексов, где heights[i] != expected [i] (рисунок 7).

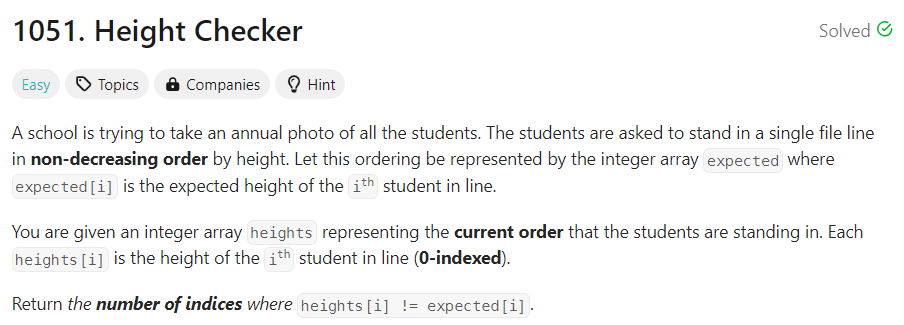


Рисунок 7 – Задача 1051

**Результаты** представлены на рисунке 8.

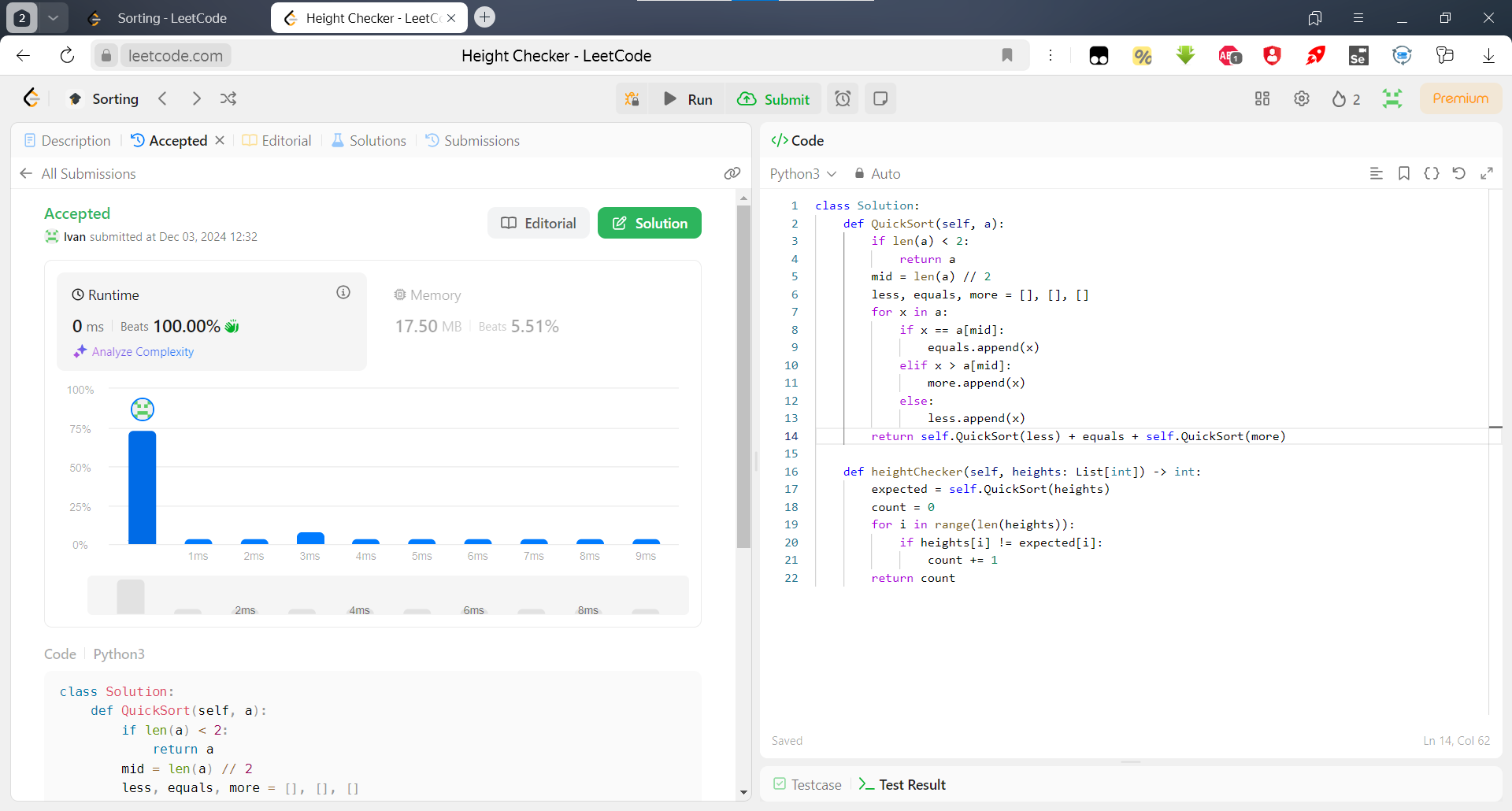


Рисунок 8 – Результаты решения задачи 1051

**Решение** изображено на рисунке 9.

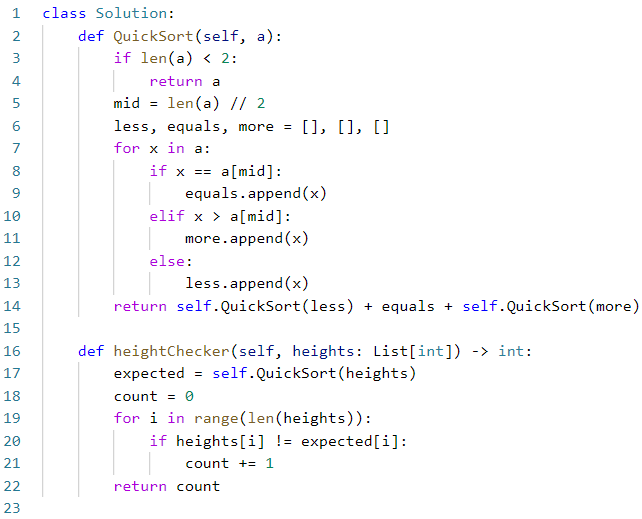


Рисунок 9 – Решение задачи 1051

**Описание алгоритма.**

Алгоритм сравнивает обычные значения с отсортированными. Вся сложность задачи состояла в том, чтобы подобрать оптимальную сортировку, в этом случае подошли встроенная и QuickSort, которая и была использована.

Задача «1346. Check If N and Its Double Exist »

**Тема:** бинарный поиск.

**Задание**: Учитывая массив целых чисел arr, проверьте, существуют ли два индекса i и j, такие, что (рисунок 10):

* i != j
* 0 <= i, j < arr.length
* arr[i] == 2 \* arr[j]

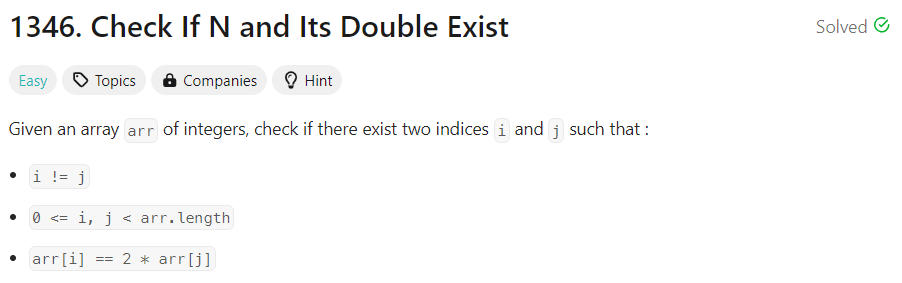


Рисунок 10 – Задачи 1346

**Результаты** представлены на рисунке 11.

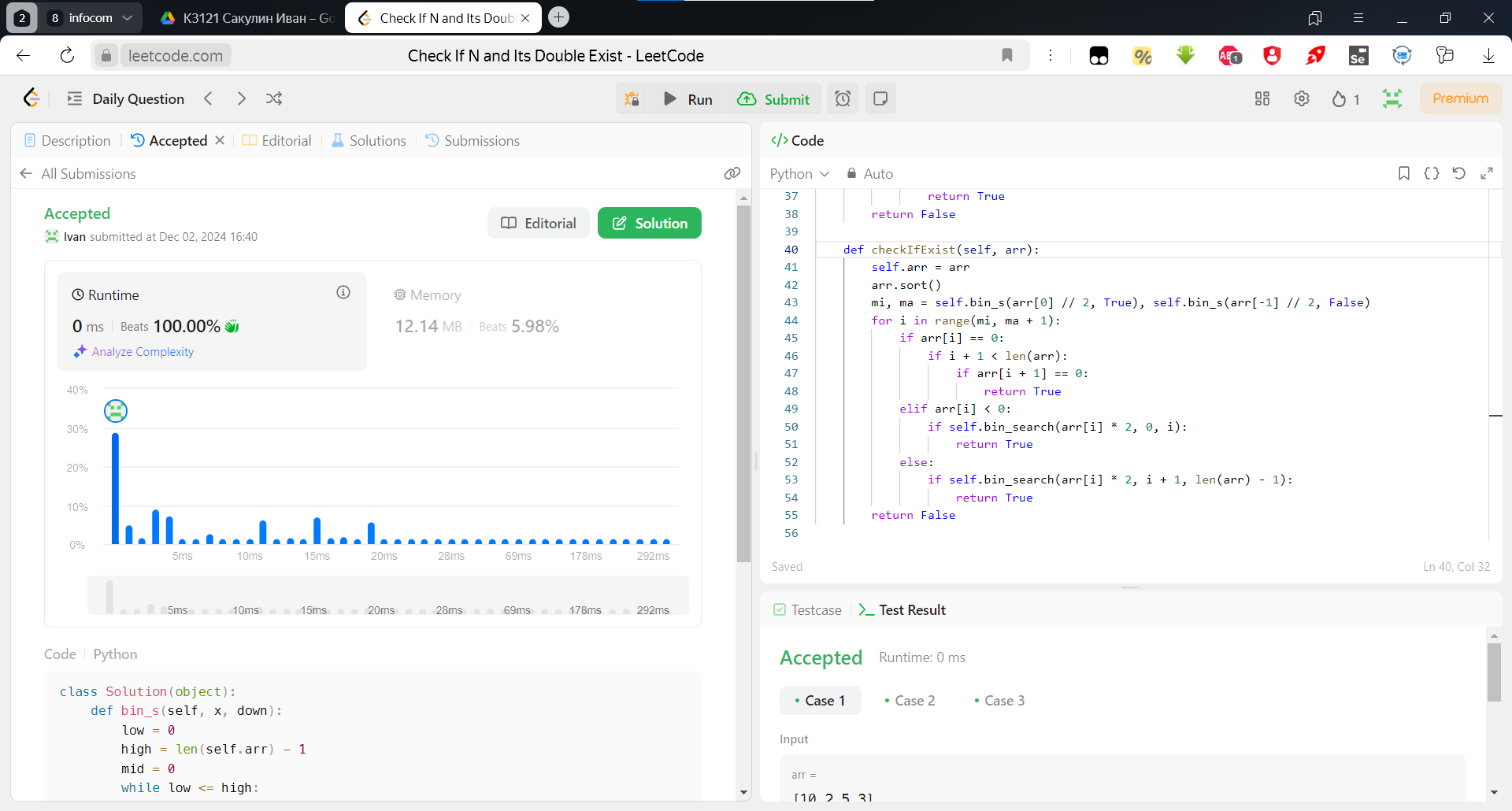


Рисунок 11 – Результаты решения задачи 1346

**Решение** изображено на рисунках 12 и 13.

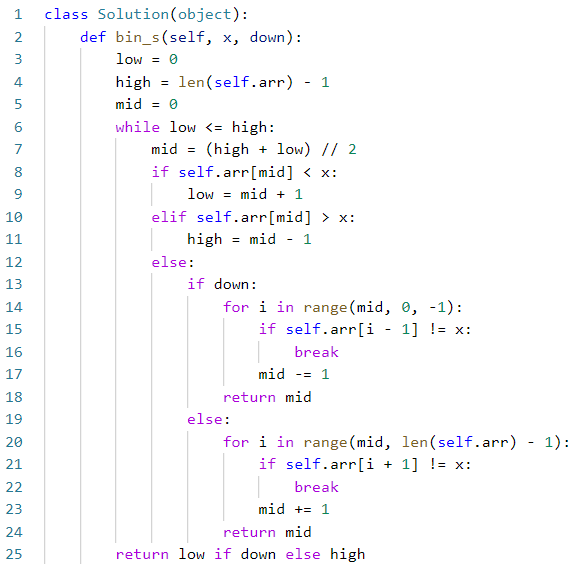


Рисунок 12 – Решение задачи 1346 (1 часть)

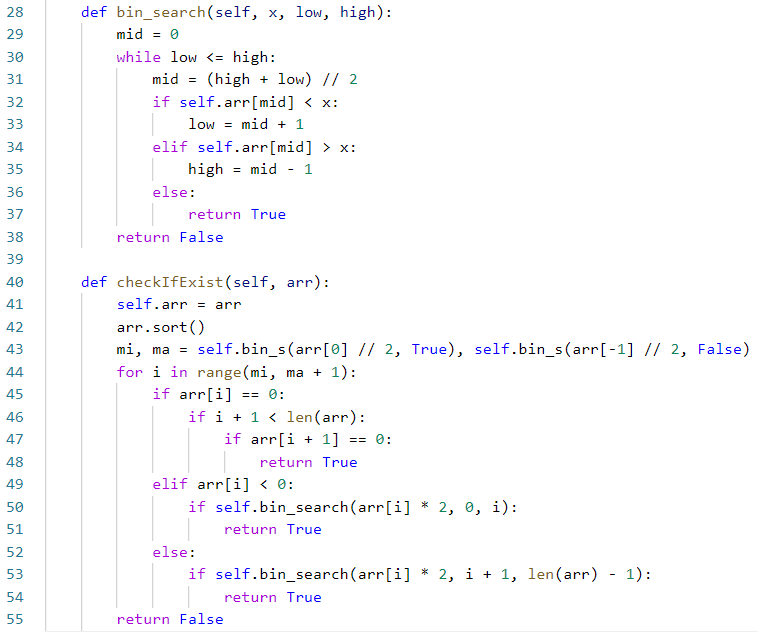


Рисунок 13 – Решение задачи 1346 (2 часть)

**Описание алгоритма.**

Алгоритм основан на многократном применении бинарного поиска. Входящий массив значений сортируется встроенной сортировкой. Затем с помощью бинарного поиска находятся крайние элементы, которые хотя бы в 2 раза меньше минимального и максимального. Поиск находит элемент равный половине максимального (минимального) и сдвигается до его самого правого (левого) вхождения, а иначе берёт самый правый (левый) элемент, подходящий по условию.

После этого мы перебираем все элементы от определённой левой границы до правой. Для них с помощью бинарного поиска проверяется наличие элемента, который ровно в 2 раза больше его, но мы сокращаем пространство поиска: отрицательные ищем от начала массива до элемента, положительные от элемента до конца массива

Сложность получается O(n\*log n), но для нормального распределения выходит, что мы перебираем только половину элементов и бинарный поиск проводим меньше чем над половиной массива. Скорость получается за счёт констант.

Задача «1455. Check If a Word Occurs As a Prefix of Any Word in a Sentence »

**Тема:** поиск подстрок.

**Задание**: Учитывая предложение, состоящее из нескольких слов, разделенных одним пробелом, и поискового слова, проверьте, является ли поисковое слово префиксом какого-либо слова в предложении.

Верните индекс слова в предложении (с индексом 1), где поисковое слово является префиксом этого слова. Если поисковое слово является префиксом более чем одного слова, верните индекс первого слова (минимальный индекс). Если такого слова нет, верните -1.

Префиксом строки s является любая начальная непрерывная подстрока s (рисунок 14).

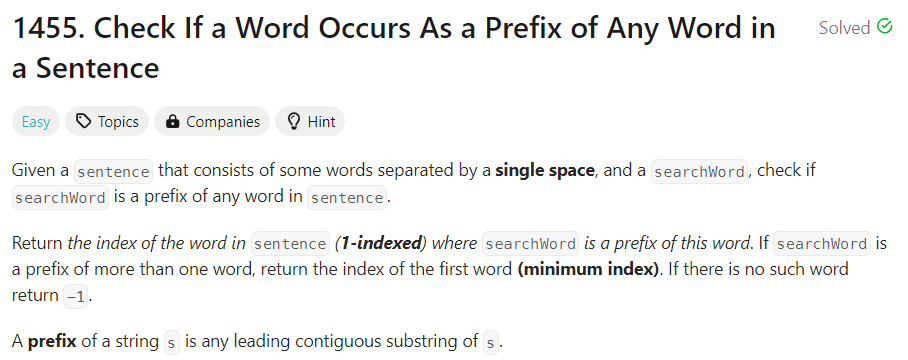


Рисунок 14 – Задача 1455

**Результаты** представлены на рисунке 15.

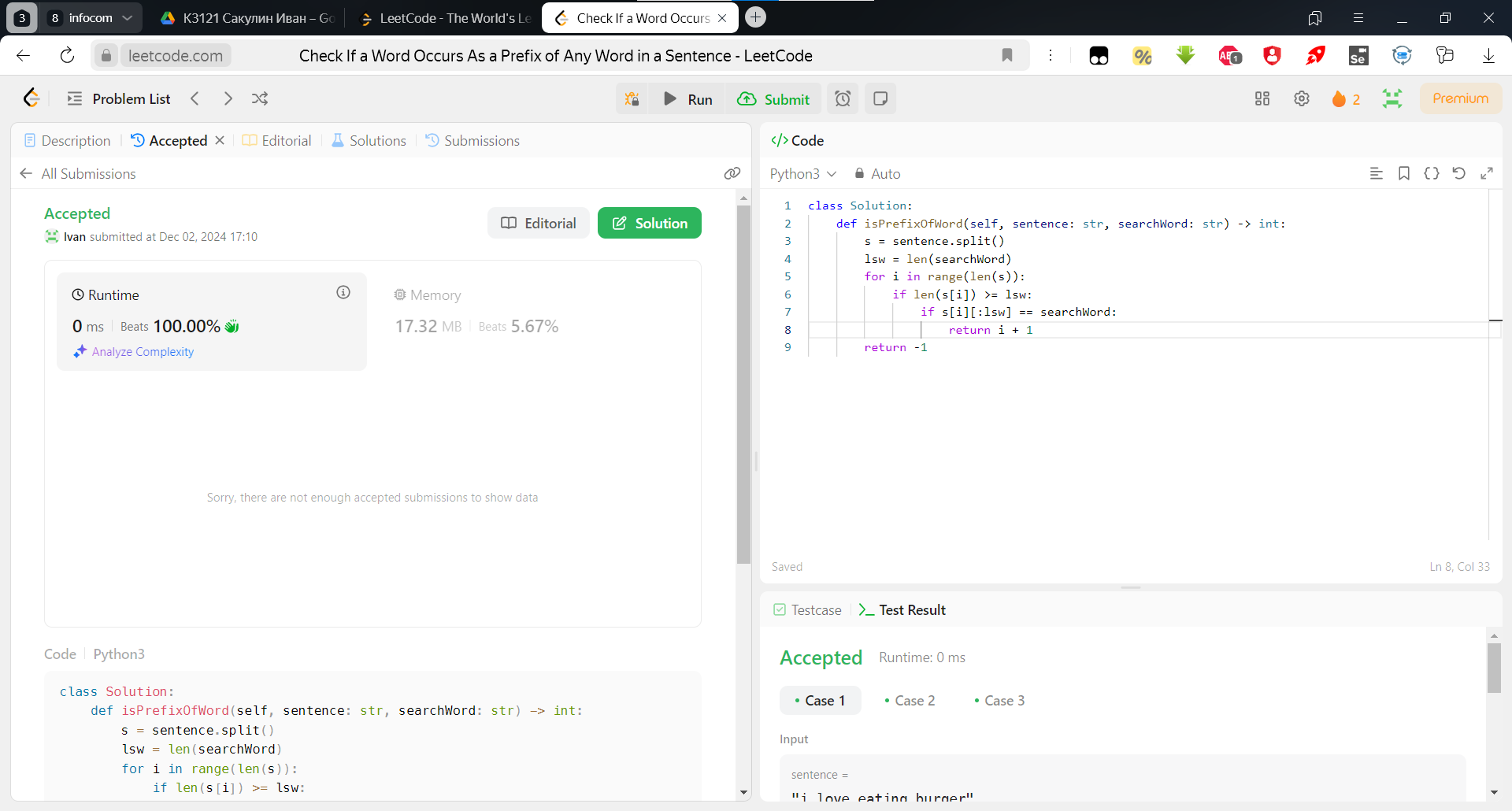


Рисунок 15 – Результаты решения задачи 1455

**Решение** изображено на рисунке 16.

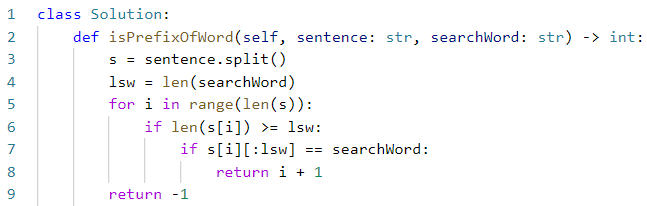


Рисунок 16 – Решение задачи 1455

**Описание алгоритма.**

Решение перебором: разделим предложение по пробелам и проверим совпадение префиксов.

Задача «1636. Sort Array by Increasing Frequency»

**Тема:** сортировки.

**Задание**: Учитывая массив целых чисел nums, отсортируйте массив в порядке возрастания в зависимости от частоты значений. Если несколько значений имеют одинаковую частоту, отсортируйте их в порядке убывания.

Верните отсортированный массив (рисунок 17).

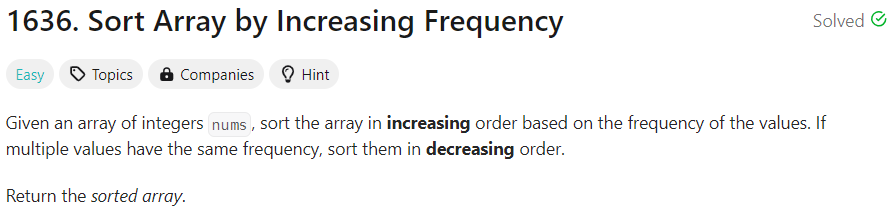


Рисунок 17 – Задача 1636

**Результаты** представлены на рисунке 18.

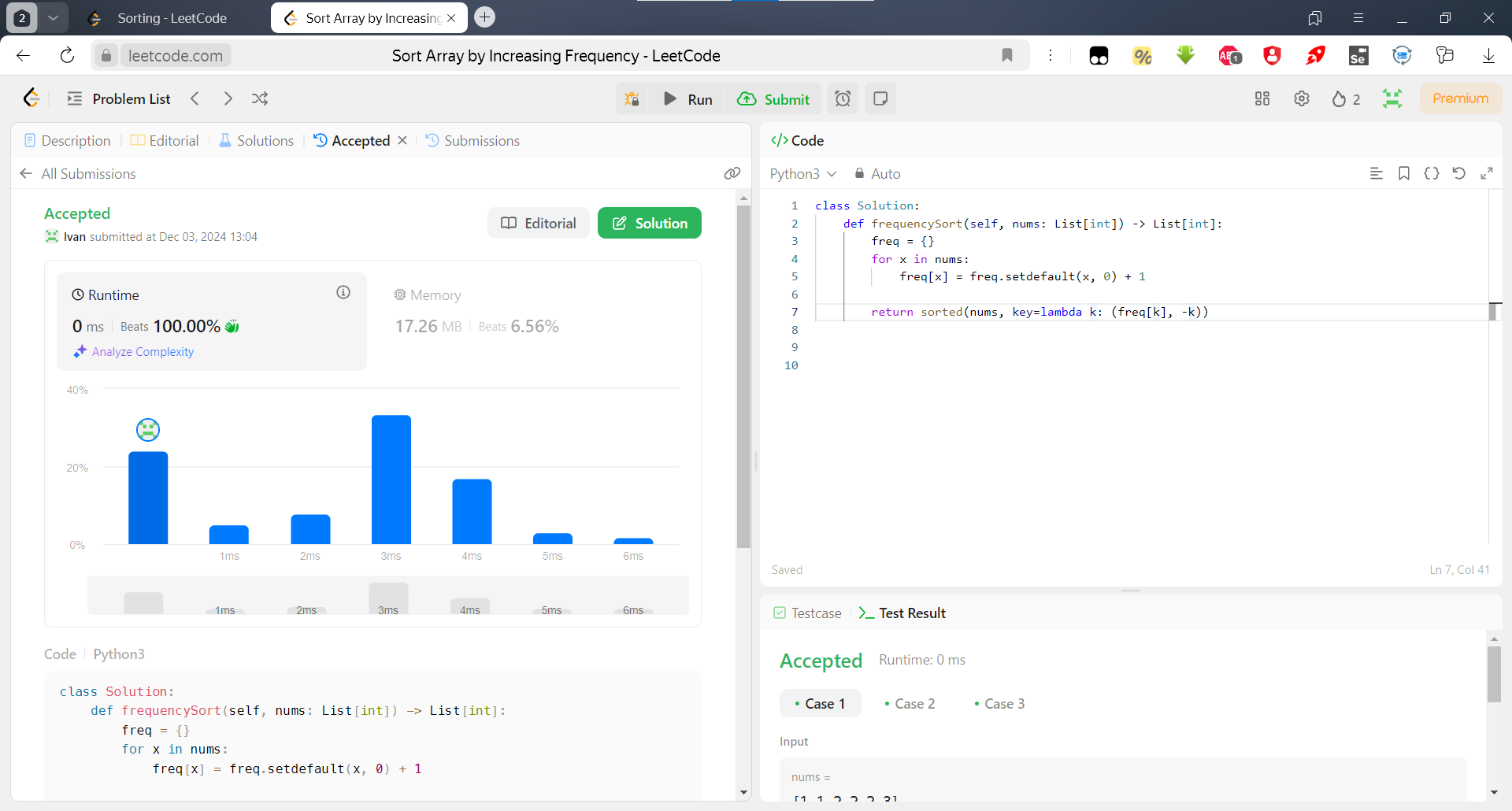


Рисунок 18 – Результаты решения задачи 1636

**Решение** изображено на рисунке 19.

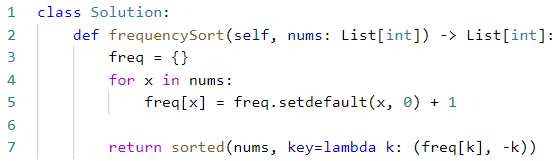


Рисунок 19 – Решение задачи 1636

**Описание алгоритма.**

Для решения был использован словарь. Сначала каждому элементу рассчитываются количество вхождений, затем встроенной сортировкой сортируем значения по возрастанию их частоты, затем по убыванию значений.

ЗАКЛЮЧЕНИЕ

В ходе решения задач были применены изученные алгоритмы. Были решены 6 заданий по установленным требованиям.