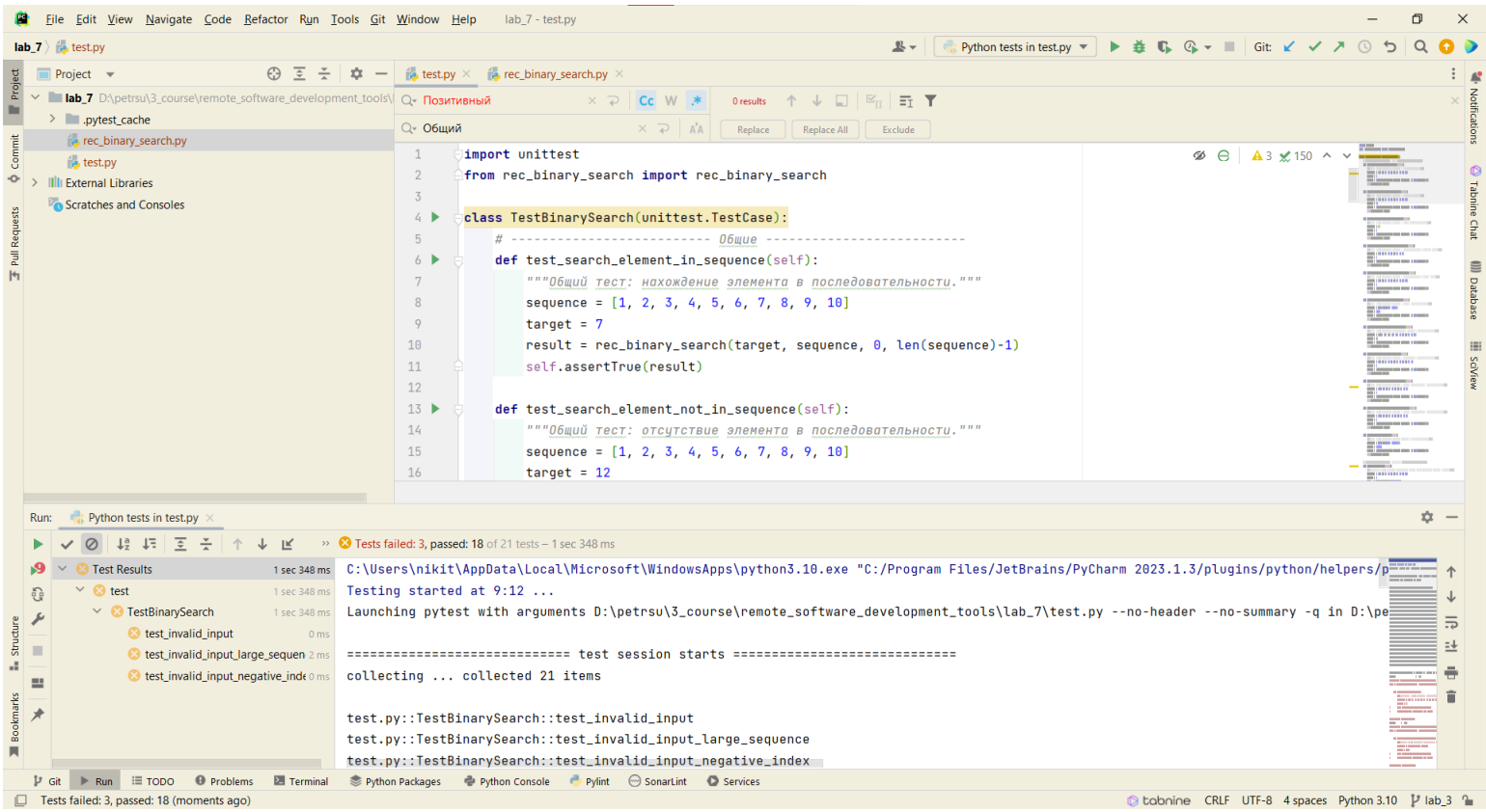
```
# -------------------------- Общие --------------------------
def test_search_element_in_sequence(self):
    """Общий тест: нахождение элемента в последовательности."""
    sequence = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
    target = 7
    result = rec_binary_search(target, sequence, 0, len(sequence) - 1)
    self.assertEqual(result, True)

def test_search_element_not_in_sequence(self):
    """Общий тест: отсутствие элемента в последовательности."""
    sequence = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
    target = 12
    result = rec_binary_search(target, sequence, 0, len(sequence)-1)
    self.assertFalse(result)

def test_search_element_in_empty_sequence(self):
    """Общий тест: нахождение элемента в пустой последовательности."""
    sequence = []
    target = 5
    result = rec_binary_search(target, sequence, 0, len(sequence)-1)
    self.assertFalse(result)

def test_search_element_in_odd_length_sequence(self):
    """Общий тест: нахождение элемента в последовательности с нечетной длиной."""
    sequence = [1, 2, 3, 4, 5, 6, 7, 8, 9]
    target = 6
    result = rec_binary_search(target, sequence, 0, len(sequence)-1)
    self.assertTrue(result)

def test_search_element_in_even_length_sequence(self):
    """Общий тест: нахождение элемента в последовательности с четной длиной."""
    sequence = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
    target = 8
    result = rec_binary_search(target, sequence, 0, len(sequence)-1)
    self.assertTrue(result)

def test_search_element_in_large_sequence(self):
    """Общий тест: нахождение элемента в большой последовательности."""
    sequence = list(range(1, 10001))
    target = 7500
    result = rec_binary_search(target, sequence, 0, len(sequence)-1)
    self.assertTrue(result)

def test_search_element_in_negative_sequence(self):
    """Тест поиска отрицательного элемента в последовательности."""
    sequence = [-10, -8, -6, -4, -2, 0, 2, 4, 6, 8, 10]

    # Тестовые случаи
    test_cases = [
        {"target": -6, "expected_result": True, "description": "Целевой элемент в середине последовательности."}
        {"target": -4, "expected_result": True, "description": "Целевой элемент в середине последовательности."}
    ]

    for case in test_cases:
        with self.subTest(description=case["description"]):
            result = rec_binary_search(case["target"], sequence, 0, len(sequence) - 1)
            self.assertEqual(result, case["expected_result"])
def test_search_element_with_duplicates(self):
    """Общий тест: нахождение элемента с повторениями в последовательности."""
    sequence = [1, 2, 3, 3, 4, 5, 5, 6, 7, 8, 9, 10]
    target = 5
    result = rec_binary_search(target, sequence, 0, len(sequence)-1)
    self.assertTrue(result)

def test_search_element_in_reversed_sequence(self):
    """Общий тест: нахождение элемента в обратно отсортированной последовательности."""
    sequence = [10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
    target = 7
    result = rec_binary_search(target, sequence, 0, len(sequence)-1)
    self.assertTrue(result)

def test_search_element_not_in_reversed_sequence(self):
    """Общий тест: отсутствие элемента в обратно отсортированной последовательности."""
    sequence = [10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
    target = 0
    result = rec_binary_search(target, sequence, 0, len(sequence)-1)
    self.assertFalse(result)

def test_search_large_element(self):
    """Общий тест: нахождение большого элемента в последовательности."""
    sequence = list(range(1, 1000001))
    target = 999999
    result = rec_binary_search(target, sequence, 0, len(sequence)-1)
    self.assertTrue(result)
```

```
# -------------------------- Негативный --------------------------
def test_invalid_input(self):
    """Негативный тест: обработка некорректных входных данных (обратные значения индексов)."""
    sequence = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
    target = 7
    with self.assertRaises(ValueError):
        rec_binary_search(target, sequence, 5, 2)

def test_invalid_input_negative_index(self):
    """Негативный тест: обработка некорректных входных данных (отрицательные значения индексов)."""
    sequence = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
    target = 7
    with self.assertRaises(ValueError):
        rec_binary_search(target, sequence, -2, 5)

def test_search_element_not_in_large_sequence(self):
    """Негативный тест: отсутствие элемента в большой последовательности."""
    sequence = list(range(1, 1000001))
    target = 2000000
    result = rec_binary_search(target, sequence, 0, len(sequence) - 1)
    self.assertFalse(result)

def test_search_element_not_in_negative_sequence(self):
    """Негативный тест: отсутствие положительного элемента в отрицательной последовательности."""
    sequence = [-10, -8, -6, -4, -2, 0, 2, 4, 6, 8, 10]
    target = 5
    result = rec_binary_search(target, sequence, 0, len(sequence) - 1)
    self.assertFalse(result)

def test_search_element_not_with_duplicates(self):
    """Негативный тест: отсутствие элемента без повторений в последовательности с повторениями."""
    sequence = [1, 2, 3, 3, 4, 5, 5, 6, 7, 8, 9, 10]
    target = 11
    result = rec_binary_search(target, sequence, 0, len(sequence) - 1)
    self.assertFalse(result)

def test_search_element_not_in_reversed_sequence(self):
    """Негативный тест: отсутствие элемента в обратно отсортированной последовательности."""
    sequence = [10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
    target = 11
    result = rec_binary_search(target, sequence, 0, len(sequence) - 1)
    self.assertFalse(result)

def test_search_large_element_not_in_sequence(self):
    """Негативный тест: отсутствие большого элемента в большой последовательности."""
    sequence = list(range(1, 1000001))
    target = 2000000
    result = rec_binary_search(target, sequence, 0, len(sequence) - 1)
    self.assertFalse(result)

# -------------------------- Краевые --------------------------
def test_search_element_at_beginning(self):
    """Краевой тест: нахождение элемента в начале последовательности."""
    sequence = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
    target = 1
    result = rec_binary_search(target, sequence, 0, len(sequence)-1)
    self.assertTrue(result)

def test_search_element_at_end(self):
    """Краевой тест: нахождение элемента в конце последовательности."""
    sequence = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
    target = 10
    result = rec_binary_search(target, sequence, 0, len(sequence)-1)
    self.assertTrue(result)

def test_search_element_at_middle(self):
    """Краевой тест: нахождение элемента в середине последовательности."""
    sequence = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
    target = 5
    result = rec_binary_search(target, sequence, 0, len(sequence) - 1)
    self.assertTrue(result)

def test_invalid_input_large_sequence(self):
    """Краевой тест: обработка некорректных входных данных (выход за границы последовательности)."""
    sequence = list(range(1, 10001))
    target = 7500
    with self.assertRaises(IndexError):
        rec_binary_search(target, sequence, 0, len(sequence)+10)
```

Реализован 21 тест.
18 проходят
в трёх ошибки



Материалы

- https://thecleverprogrammer.com/2021/03/06/recursive-binary-search-using-python/

Рекурсивный двоичный поиск с использованием Python

```python
# Рекурсивный двоичный поиск
25 usages
def rec_binary_search(target, sequence, first, last):
    """
    Рекурсивный двоичный поиск в отсортированной последовательности.

    :param target: Искомый элемент
    :param sequence: Отсортированная последовательность
    :param first: Начальный индекс поиска
    :param last: Конечный индекс поиска
    :return: True, если элемент найден, иначе False
    """
    # Добавлено в ходе тестирования
    sequence = sorted(sequence)



    if first > last:
        return False
    else:
        mid = (last + first) // 2
        if sequence[mid] == target:
            return True
        elif target < sequence[mid]:
            return rec_binary_search(target, sequence, first, mid-1)
        else:
            return rec_binary_search(target, sequence, mid + 1, last)
```