CS102L Data Structures & Algorithms
Lab 6

Department of Computer Science

Dhanani School of Science and Engineering

Habib University

Spring 2024

Contents

Introduction

1.1 Restriction

You are not allowed to use any built in functions other than the append and insert function

1.2 Instructions

- This lab will contribute 1% towards your final grade.
- The deadline for submission of this lab is at the end of lab time.
- The lab must be submitted online via CANVAS. You are required to submit a zip file that contains all the *.py* files.
- The zip file should be named as *Lab_06_aa1234.zip* where *aa1234* will be replaced with your student id.
- **Files that don't follow the appropriate naming convention will not be graded.**
- **Your final grade will comprise of both your submission and your lab performance.**

1.3 Marking scheme

This lab will be marked out of 100.

- 50 Marks are for the completion of the lab.
- 50 Marks are for progress and attendance during the lab.

1.4 Lab Objectives

- To understand and implement binary search in Python
- To use binary search in different exercises

1.5 Late submission policy

There is no late submission policy for the lab.

1.6 Use of AI

Taking help from any AI-based tools such as ChatGPT is strictly prohibited and will be considered plagiarism.

1.7 Viva

Course staff may call any student for Viva to provide an explanation for their submission.

Prerequisites

2.1 Assert Statements

Assert statements are crucial tools in programming used to verify assumptions and test conditions within code. For instance, in Python, an assert statement is typically as follows:

```
assert condition, message
```

For example, in a function calculating the square root, in order to ensure that the input value is non-negative, one might add the following assert statement:

```
assert x >= 0, "x should be a non-negative number"
```

If the condition evaluates to False, an AssertionError is raised, highlighting the issue and its corresponding message.

2.2 Pytest

Pytest is a popular testing framework in Python known for its simplicity and power in facilitating efficient and scalable testing for applications and software. In order to use Pytest, one first needs to install it and create test functions prefixed with *test_* in a Python file. To install Pytest, you can write the following command on the terminal:

```
pip install pytest
```

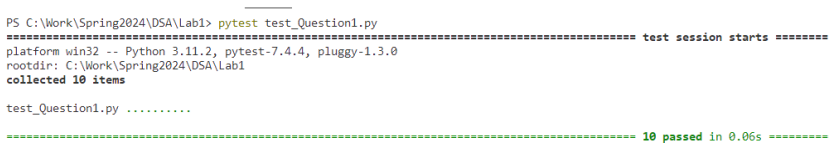
After installing Pytest, one can execute all the tests by typing the following command on the terminal:

```
pytest
```

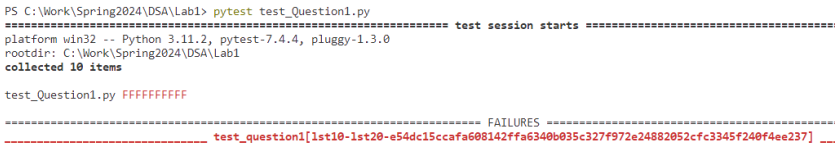
If you want to test any particular file, then you will have to specify the filename as follows:

```
pytest <filename>.py
```

If all the tests pass successfully, then you should receive a similar output.



However, if all the tests fail, then you should receive a similar output.



Exercises

3.1 Binary Search Iterative

Write a function named binary_search_iterative, which takes the following two inputs:

- list – An input list (sorted data)
- item – An item to search in the list

and returns the index of the item using an iterative binary search. It should return -1 if the item is not in the list.

3.1.1 Examples

```
>>> binary_search_iterative( [0, 1, 2, 8, 13, 17, 19, 32,
   42], 3)
-1
>>> binary_search_iterative(  [0, 1, 2, 8, 13, 17, 19, 32,
   42], 17)
5
```

3.1.2 Testing

To test your function, type the following command on the terminal:

```
pytest test_Question1.py
```

3.2 Binary Search Iterative Modified

Write a function named binary_search_iterative_modified, which takes the following two inputs:

- list – An input list (sorted data)
- item – An item to search in the list

and returns the index of the item using an iterative binary search. If the item is not found then it should add the item to the list and return its index.

3.2.1 Examples

```
>>> binary_search_iterative_modified([0, 1, 2, 8, 13, 17,
   19, 32, 42], 8)
3
>>> binary_search_iterative_modified([0, 1, 2, 3, 8, 13, 17,
    19, 32, 42], -1)
0
```

3.2.2 Testing

To test your function, type the following command on the terminal:

```
pytest test_Question2.py
```

3.3 Binary Search Recursive

Write a function named binary_search_recursive, which takes the following inputs:

- list – An input list (sorted data)
- item – An item to search in the list
- low – Low index of data list
- high – High index of data list

and returns the index of the item using a recursive binary search. It should return -1 if the item is not in the list.

3.3.1 Examples

```
>>> binary_search_recursive( [0, 1, 2, 8, 13, 17, 19, 32,
   42], 19, 0, 8)
6
>>> binary_search_recursive( [0, 1, 2, 8, 13, 17, 19, 32,
   42], 3, 0, 8)
-1
```

3.3.2 Testing

To test your function, type the following command on the terminal:

```
pytest test_Question3.py
```

3.4 Binary Search Recursive Modified

Write a function named binary_search_recursive_modified, which takes the following inputs:

- list – An input list (sorted data)
- item – An item to search in the list
- low – Low index of data list
- high – High index of data list

and returns the index of the item using a recursive binary search. If the item is not found then it should add the item to the list and return its index.

3.4.1 Examples

```
>>> binary_search_recursive_modified([0, 1, 2, 8, 13, 17,
   19, 32, 42], 3, 0, 8)
3
>>> binary_search_recursive_modified([0, 1, 2, 8, 13, 17,
   19, 32, 42], 17, 0, 8)
5
```

3.4.2 Testing

To test your function, type the following command on the terminal:

```
pytest test_Question4.py
```

3.5 Finding Multiple

Using binary and linear search approaches, write a function named finding_multiple, which takes the following inputs:

- lst – list of items (sorted data)
- item – the item needs to be searched in the lst

and return the list of all the indexes where the item is found in the last. Return an empty list if the item is not found. You must use both binary and linear search simultaneously to solve this problem.

3.5.1 Examples

```
>>> finding_multiple([0, 1, 2, 8, 13, 17, 17, 17 ,17, 19,
   32, 42], 17)
[5, 6, 7, 8]
>>> finding_multiple([0, 1, 2, 8, 13, 17, 17, 17 ,17, 19,
   32, 42], 34)
[]
```

3.5.2 Testing

To test your function, type the following command on the terminal:

```
pytest test_Question5.py
```

3.6 Updating Student Record

Using the binary search approach, write a function named update_record, which takes the following inputs:

- students_records – Nested List of Tuples. Each tuple of the list represents the student's data.
- ID – An ID of a student whose data has to be updated
- record_title – the type of data that has to be updated
- data – A new data which should replace record_title data

and update the record of students' data associated with that specific ID. The function should then return "Record updated".
If ID is given as the data to be updated, then return the message "ID cannot be updated". If the ID is not found in students_records, then return the message "Record not found".
NOTE: The type of record_title input can be "ID", "Email", "Mid1" or "Mid2". Please use the same spelling for these types in your code, because the same have been used in test cases.

3.6.1 Examples

```
student_records = [("aa02822", "ea02822", 80, 65),
 ("ea02822", "ea02822@st.habib.edu.pk", 80, 65),
 ("fa08877", "fa08877@st.habib.edu.pk", 66, 67),
 ("gh04588", "gh04588@st.habib.edu.pk", 33, 50)]
>>> update_record(students_records, "aa02822", "ID", "
   aa02456")
ID cannot be updated
```

```
>>> update_record(students_records, "aa02822", "Email", "
   aa02822@st.habib.edu.pk")
Record updated
>>> update_record(students_records, 'f08877', "Mid2", 50)
Record not found
```

3.6.2 Testing

To test your function, type the following command on the terminal:

```
pytest test_Question6.py
```