**CS469 Data Structure and Algorithms**

**HOS01 Binary Search**

03/29/2023 Reviewed by Christopher Sharp

01/04/2024 Reviewed by Anh Nguyen

09/29/2024 Reviewed by Shahid Khan

School of Technology and Computing (STC) @City University of Seattle (CityU)

**Before You Start**

* The document’s examples are written in Python. If you are unfamiliar with Python, please finish the Python tutorial in Module00 before you start the assignment.
* Some steps are not explained in the tutorial**.** If you are not sure what to do:
  1. Consult the resources listed below.
  2. If you cannot solve the problem after a few tries, ask a TA for help.

**Learning Outcomes**

Students will be able to:

* Understand Binary Search
* Implement Binary Search

**Resources**

* Bhargava, Y, A. (2016). Grokking algorithms GitHub repository. Retrieved from: <https://github.com/egonSchiele/grokking_algorithms>
* Leet Code exploration card: Binary Search. Retrieved from: <https://leetcode.com/explore/learn/card/binary-search/>
* Python Tutor. <https://pythontutor.com/visualize.html>
* Searching sorted list. Retrieved from: <https://www.cs.usfca.edu/~galles/visualization/Search.html>

1. **Binary Search**

Suppose you’re playing a number guessing game. You’re given a number in the range from 1 to 100. Each time you guess, if your answer is equal to the number and your guessing times are less than the 3 times, you win! The host only will tell you whether you win the game or whether your number is bigger than or smaller than the given number. What is the best strategy to minimize the guessing times?

Well, you probably want to try 50 first. If the given number is less than 50, then you know 51 to 100 can’t be the answer. Next, you may try 25, and so on. If you are not lucky, you only need to try times, which are 7 times at most!

The approach is also called binary search.

To use binary search, the given input should be an ordered collection (whether ascending, descending, or rotated sorted). Before starting the searching, the first step is to initialize the lower bound and upper bound. For each step of searching, the algorithm does the following:

1. Get the middle item of lower bound and upper bound
2. Compare the middle item with the target
3. If the middle item is not equal to the target, get rid of the unqualified half

You can visualize the binary search algorithm by using the following website:

<https://www.cs.usfca.edu/~galles/visualization/Search.html>

Application

Description automatically generated with medium confidence

Note: the example array might look different because the array is generated randomly

Type the target number in the left upper corner’s text input. Click the binary search and see the process of binary search.

**Q**: Type the minimal element of the array in the text input and click the binary search button. What do you see? Explain the process and save your explanation as a pdf file in the module folder.

Now, let us implement the algorithm in Python.

* Create a file called **binarySearch.py** in Module01 directory
* Type the following code:

Text

Description automatically generated

Code Explanation:

1. Before searching, we initialize the *low* variable points to the smallest element and *high* variable points to the biggest element of the input array.
2. For each searching step, we compute the middle element index by dividing the sum of low and high by two. Notice that the equation uses a floor division “//” operation to make sure the index is not a float.
3. We get the value of the middle element and assign the value to the variable *guess.*
4. We compare the guess with the target value. If the guess is equal to the target number, return the found index mid.
5. If the guess is less than the target value, we should look at the right part. Thus, we assign the lower bound equal to the mid + 1.
6. If the guess is bigger than the target value, we should look at the left part. Thus, we assign the upper bound equal to the mid – 1.
7. If low > high, it means we didn’t find the target. Break the while loop and return None.

The binary search’s time complexity is O(log n), and the space complexity is O(1).

As you see, the code provides two test cases. The first test case’s input is a numerical list sorted in ascending order, and the second test case’s input is a fruits’ list sorted in alphabet order.

To visualize the searching processes on the two test cases, copy the code into the python tutor website:

<https://pythontutor.com/visualize.html>

**Optional Resource**:

You may notice that this binary search version only can be implemented in an ascending ordered array.

If you want to learn more about the binary search variants and solve some code challenges. You can try the Leet Code Binary Search exploration card: <https://leetcode.com/explore/learn/card/binary-search/>

Because this part is optional, the TA won’t be responsible for answering questions for the challenges on Leet Code. Instead, you can visit the question’s discussion board to find hints and solutions J

**Push Your Work to GitHub**

Open a terminal on visual studio code and make sure you’re in the repository folder. (e.g., “hos01\_[courseName]\_[GitHubUserName]”)

**Type the following command to upload your work**:

>>>> **git add .**

>>>> **git commit -m “Submission for HOS01”**

>>>> **git push origin master**