Tutorial 1, part 2 : Python basics and more

In this second part of the first tutorial, we will dive into a simple problem (element search) and try to optimize our solution using binary search.

1. Exercise

Write a function that takes an ordered list of numbers (a list where the elements are in order from smallest to largest) and another number. The function has to check if the given number is inside the list and returns (then prints) an appropriate boolean (True or False).

There are a number of ways to search for elements in a list, and there is no one correct way to do so. The point of this exercise is to get you thinking about possible ways to search for elements in the list. This exercise might seem silly and easy at first, but the more you dive into this topic, the more difficult it becomes.

The idea of the exercise is simple :

- Write a simple solution first and monitor the time and complexity of it
- Use **binary search** to solve the problem

Binary search :

The word “binary” means there are two choices (in computers, often this is 0 or 1, but it really means any choice between two things). “Search” is to look for something. So the main idea behind binary search is to look for something in a way that gives you a decision tree for where to look, containing two choices.

Example:

Let’s take the list a = [1, 3, 5, 30, 42, 43, 500]. It is an “ordered list”, or a list where the elements in the list go from smaller to larger. Let’s say we want to know whether the element 9 is in the list or not. Here what we can do using binary search :

- Look at the middle element in the list - it is ‘30’. We have ‘9 < 30’, so let us ignore the elements to the right of ‘30’.
- The new list we are looking at is now [1, 3, 5].
- Look at the middle element in this new list - it is 3. We have ‘9 > 3’, so ignore the elements to the left of 3.
- The new list we are looking at is [5]. The list has one element and it is not 9.
- Output False as 9 is not in the list.

What the example shows is that in an ordered list, knowing how the element you are looking for compares to another element in the list splits the list in two - one half where the element can be, and one where it definitely cannot be. **In the case where our list contains millions of elements, knowing that we can cut the “search space” in half is a great increase in efficiency.**

When you are writing the solution using binary search, compare it with your first simple solution in term of time and complexity.
Check this video for more information : https://www.youtube.com/watch?v=zeULw-a7Mw8