Binary Search Algorithm - DataFlair

About Binary Search Algorithm:

Searching for an element's presence in a list is done using linear search and binary search. Linear search is time and memory expensive, but is the simplest way to search for an element. Binary search, on the other hand, is effective mainly because the dimension of the array gets reduced with each function call. A practical implementation of binary search is autocompletion

Python Binary Search Algorithm:

The objective of this project is to create a simple script for implementing binary search. It can be implemented in two ways: recursive (function calls) and iterative.

Project Prerequisites:

The project uses loops and functions to implement the search function. Hence good knowledge of python loop and function calls is necessary to understand the code flow.

Download Binary Search Algorithm code:

You can download the source code for the Binary Search Algorithm in the given link: Binary Search Python Code

Project File Structure:

Let's have a look at the steps to build the project:

1. Recursive approach
    I. Function definition
    II. Read inputs,sort and call function
2. Iterative approach
    I. Read inputs and sort
    II. Loop for binary search

Let us look at the implementation in detail.

1. Recursive approach

I. Function definition

```
#DataFlair Guide for Binary Search
#RECURSIVE FUNCTION CALL BASED APPROACH
#Function to search element in list
def binary_search(start,end,int_list,target):
  #Condition to check if element is not present
  if start<=end:
     mid = (start+end) // 2

     #Check if mid element is the target element
```

```
        if int_list[mid] == target:
          return mid +1

        #If not, check if lesser than mid element
        #Change range to start to mid-1, since less than mid
        elif target < int_list[mid]:
          return binary_search(start,mid-1,int_list,target)

        #Check if lesser than mid element
        #Change range to mid+1 to end, since greater than mid
        elif target > int_list[mid]:
          return binary_search(mid+1,end,int_list,target)
    else:
        return -1
```

Code Explanation:

- **def binary_search(start,end,int_list,target):** Declare and define the function binary search with parameters: start, end, list of elements and target element
- **start<=end:** This condition is necessary to avoid an out_of_index_error and satisfies the condition when an element is not present in a list.
- **Test conditions:** If the target is the middle element of the list, the position is returned, else it is checked if less than the middle element. Upon satisfying this condition, the function is called with a change in the lower and upper bounds being start and mid-1 respectively. Similarly for the case of the target element being greater than the middle element, the bounds are updated to mid+1 and end.
- **Return value:** The function return position, if element is found and -1 otherwise

II. Read inputs and call function:

```
length = int(input("Enter length of list: "))
int_list = []
#Read elements of list
for i in range(length):
 element =  int(input("Enter element: "))
 int_list.append(element)
#Sort the list
int_list=sorted(int_list)
print(int_list)
#Read target element to be found
target = int(input("Enter target element: "))
position = binary_search(0,length-1,int_list,target)
if position == -1:
```

```
        print('Element not in list')
    else:
        print("Element found at position: "+ str(position))
```

Code Explanation:

- **Inputs:** Read the list length from the user and the elements of the list. Append the elements to the list
- **sorted(int_list):** A prerequisite for binary search is to have a sorted list. Hence using sorted(), we sort the list
- **Function call:** The inputs are passed to the function binary_search. The value returned is printed.

2. Iterative Approach:

I. Read inputs and sort the list:

```
#DataFlair Guide for Binary Search
#ITERATIVE APPROACH
#Read length of list from user
length = int(input("Enter length of list: "))
int_list = []
#Read elements of list
for i in range(length):
 element =  int(input("Enter element: "))
 int_list.append(element)
#Sort the list
int_list=sorted(int_list)
print(int_list)
#Read target element to be found
target = int(input("Enter target element: "))
```

Code explanation:

- **Inputs:** Read the list length from the user and using a for loop, read the elements of the list. Append the elements to the list
- **sorted(int_list):** Sort the list for binary search

II. Loop for binary search

```
#Define variables
start = 0
end = length-1
position = -1

while(start<=end):
 mid = (start+end) // 2
 if int_list[mid] == target:
```

```
            position = mid
            break
        #If not, check if lesser than mid element
        #Change range to start to mid-1, since less than mid
        elif target < int_list[mid]:
            end = mid-1
        #Check if lesser than mid element
        #Change range to mid+1 to end, since greater than mid
        elif target > int_list[mid]:
            start = mid+1

    if position == -1:
        print('Element not in list')
    else:
        print("Element found at position: "+ str(position+1))
```

Code Explanation:

- **Define variables:** Define variables start, end and position. Position is set to -1 initially
- **While loop:** The terminating condition for the while loop is 'start<=end'. Inside the loop, check the target element with the middle element, and update the position variable. If lesser than middle element, update upper bound to middle-1 and in the case of greater than middle element, update start to middle +1
- **Position condition:** If position remains unchanged at -1, it indicates the element is not present in the list. If it is updated, then the position is printed

Project output:

Enter the inputs and view the output:

```
Enter length of list: 5
Enter element: 9
Enter element: 3
Enter element: 6
Enter element: 2
Enter element: 4
[2, 3, 4, 6, 9]
Enter target element: 4
Element found at position: 3
```

Summary

Thus using python, we created a simple Binary Search Algorithm. The project covers loops and function with recursive function calls