According to the question, there is a need to write code where the user will input an array the way they want and will give a target number that can be or not in the given array. Next, before getting output, the given array will be sorted, and every element of it will be compared with the target number and inserted correctly in the array. Then we will get the output, which will show the index number of that target number.

To do this task, I have used a Binary Search Algorithm.Binary search is defined as a searching algorithm used in a sorted array by repeatedly dividing the search interval in half. The idea of binary search is to use the information that the array is sorted to reduce the time complexity to O(log N).

To apply the binary search algorithm, the data structure must be sorted. But here, the user can also give an unsorted array, so, before applying the binary search algorithm, to sort the input array number I have used the sorted() function. Sorting the array ensures that the binary search algorithm works correctly. The binary search method is then applied to the sorted array by the code after the array has been sorted.When a target is located, the binary search method effectively returns the index; otherwise, it discovers the index where the target would be added in the sorted array.

After dividing the element into left and right portions, the code will choose a midpoint to initiate BSA. The search range is kept valid during the binary search procedure thanks to the while loop. There's a chance the target is in the array's right half if the element at the midpoint is less than the target. The left pointer is changed to mid + 1 in this scenario.It is possible that the target is in the left side of the array if the element at the midway is greater than the goal. The left half of the array is the search range in this instance since the right pointer is updated to mid-1.

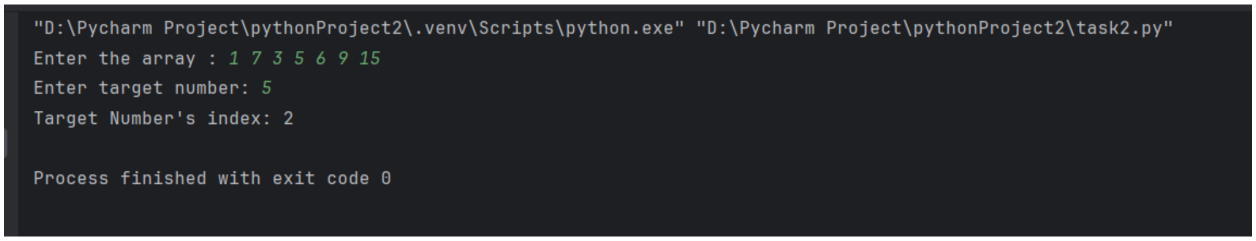
The search_insert function is called with the unsorted array ‘number’ and the target number ‘target’.Since the array is sorted internally within the function, it ensures correct functionality of the binary search algorithm.

In general, users can submit a target number and a sorted or unsorted array of numbers using this code. Before using the binary search algorithm to determine the index where the target would be put, it internally sorts the array.

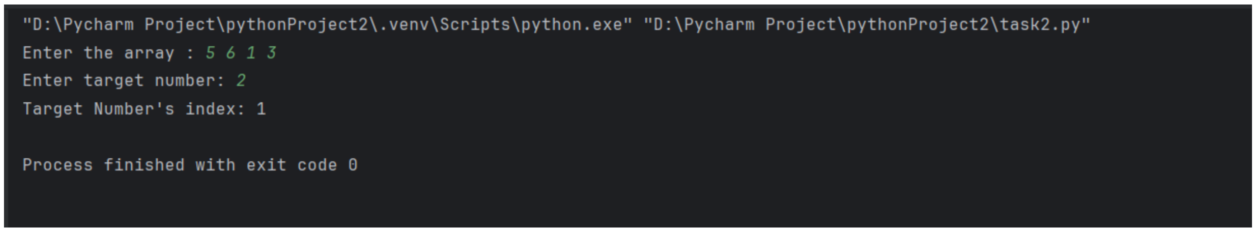
The code is

```
def search_insert(number, target):
    number_sorted = sorted(number)   # Sort the array before applying binary
search
    left, right = 0, len(number_sorted) - 1

    while left <= right:
        mid = (left + right) // 2
        if number_sorted[mid] == target:
            return mid
        elif number_sorted[mid] < target:
            left = mid + 1
        else:
            right = mid - 1

    return left


#user input(array)
number = list(map(int, input("Enter the array : ").split()))

#target number
target = int(input("Enter target number: "))

index = search_insert(number, target)

print("Target Number's index:", index)
```

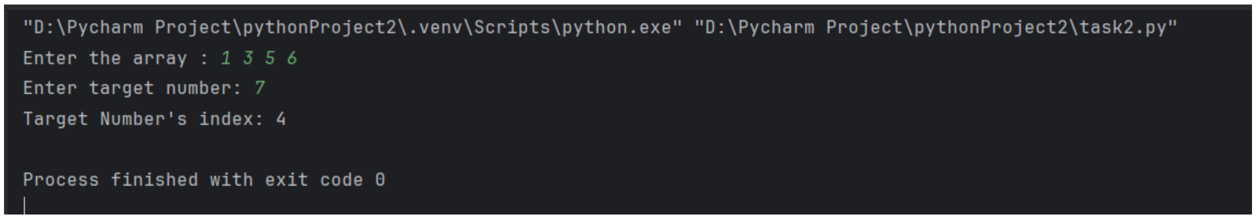
The outputs are:



```
"D:\Pycharm Project\pythonProject2\.venv\Scripts\python.exe" "D:\Pycharm Project\pythonProject2\task2.py"
Enter the array : 1 7 3 5 6 9 15
Enter target number: 5
Target Number's index: 2

Process finished with exit code 0
```



```
"D:\Pycharm Project\pythonProject2\.venv\Scripts\python.exe" "D:\Pycharm Project\pythonProject2\task2.py"
Enter the array : 5 6 1 3
Enter target number: 2
Target Number's index: 1

Process finished with exit code 0
```



```
"D:\Pycharm Project\pythonProject2\.venv\Scripts\python.exe" "D:\Pycharm Project\pythonProject2\task2.py"
Enter the array : 1 3 5 6
Enter target number: 7
Target Number's index: 4

Process finished with exit code 0
```