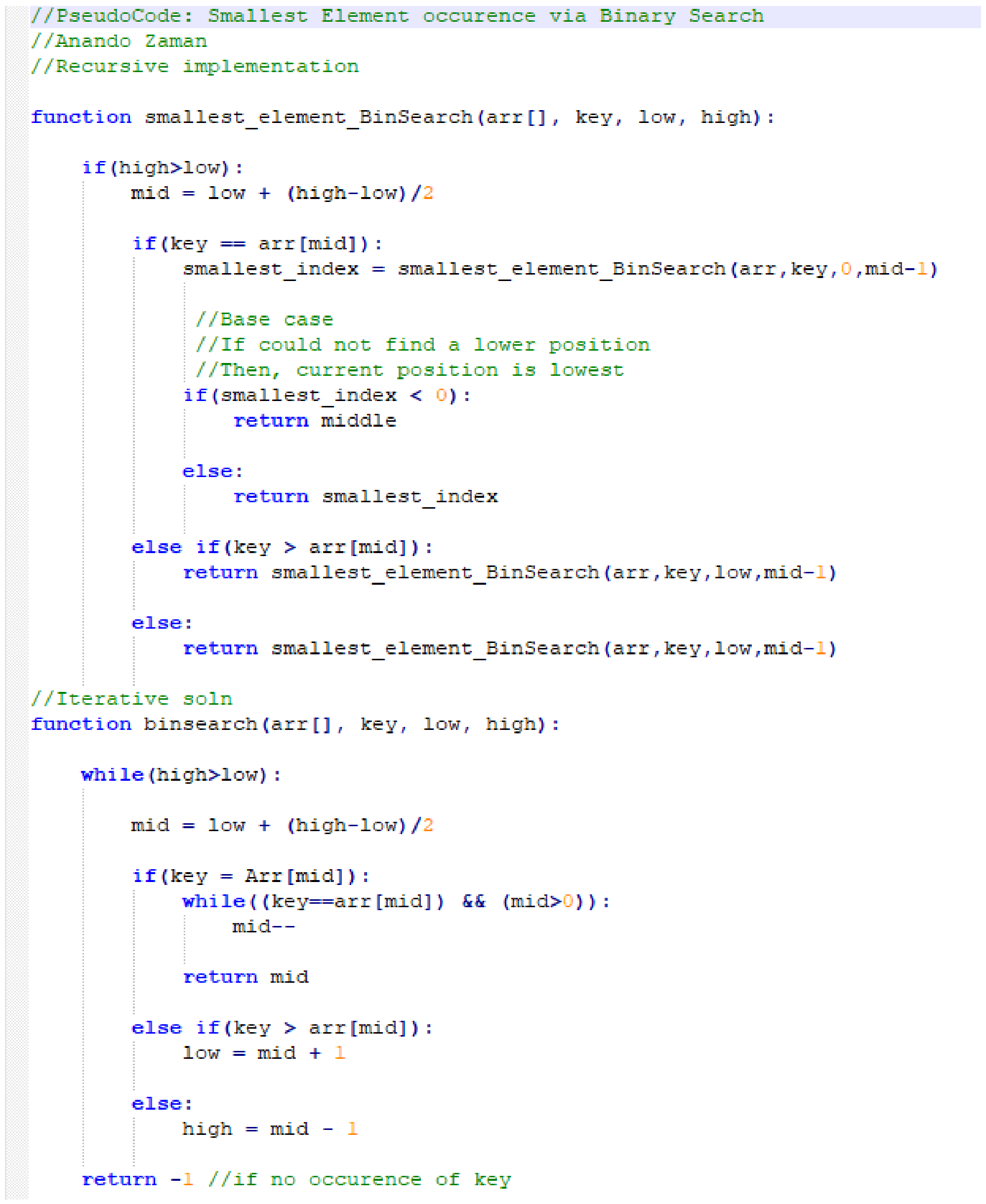
Exercise 1.4.10: Modify binary search so that it always returns the element with the smallest index that matches the search element (and still guarantees *logarithmic* running time).

Solution: Recall Binary search works by picking a midpoint value and then recursively checking each half(left or right) depending on the key we are looking for. RHS are elements that are larger than the key while LHS are elements that are smaller than the key. Binary Search only works for sorted arrays. To solve the above problem, continue binary search on LHS until that number is the last instance. Eliminate RHS each time since that half consists of keys that are larger.

Another solution would be to check left side of array until you find last instance. However, this becomes $O(N)$ runtime and we want $O(nlogn)$.



```
//PseudoCode: Smallest Element occurence via Binary Search
//Anando Zaman
//Recursive implementation

function smallest_element_BinSearch(arr[], key, low, high):

    if(high>low):
        mid = low + (high-low)/2

        if(key == arr[mid]):
            smallest_index = smallest_element_BinSearch(arr,key,0,mid-1)

             //Base case
             //If could not find a lower position
             //Then, current position is lowest
            if(smallest_index < 0):
                return middle

            else:
                return smallest_index

        else if(key > arr[mid]):
            return smallest_element_BinSearch(arr,key,low,mid-1)

        else:
            return smallest_element_BinSearch(arr,key,low,mid-1)

//Iterative soln
function binsearch(arr[], key, low, high):

    while(high>low):

        mid = low + (high-low)/2

        if(key = Arr[mid]):
            while((key==arr[mid]) && (mid>0)):
                mid--

            return mid

        else if(key > arr[mid]):
            low = mid + 1

        else:
            high = mid - 1

    return -1 //if no occurence of key
```