* 1. **Checks if a given number x exists in a sorted array arr using binary search. Analyze its time complexity using Big-O notation.**

**Test Case:**

**Example X={ 3,4,6,-9,10,8,9,30} KEY=10**

**Output: Element 10 is found at position 5**

**Example X={ 3,4,6,-9,10,8,9,30} KEY=100**

**Output : Element 100 is not found**

**Aim:**

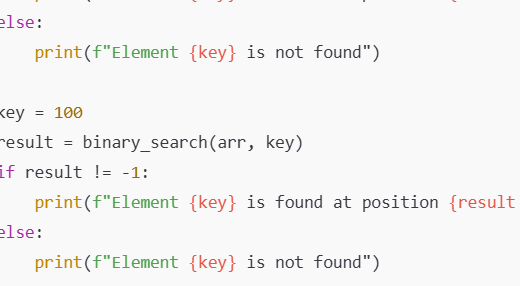
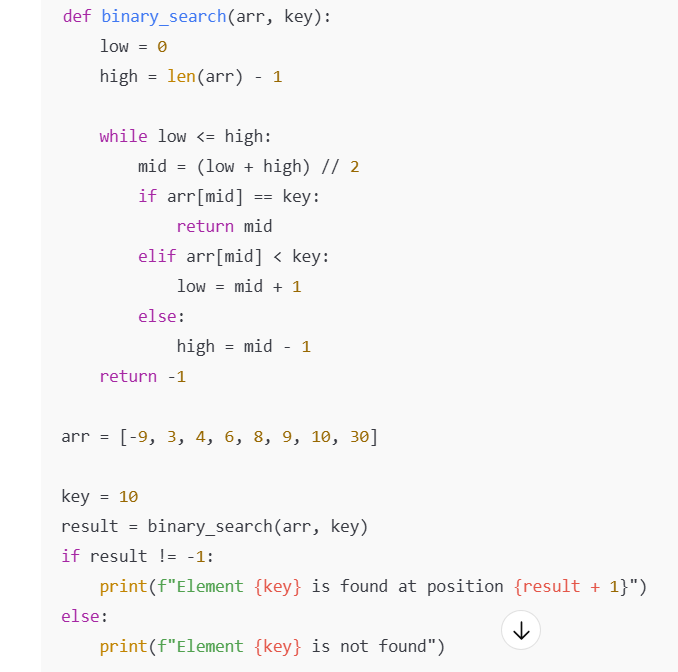
To check if a given number x exists in a sorted array arr using binary search, and analyze its time complexity using Big-O notation.

Algorithm: Binary Search

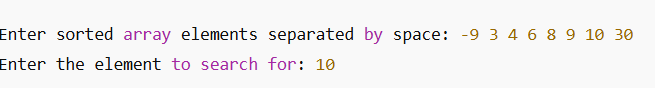
Input: A sorted array arr and a key value key  
Output: The position of key in arr if found, otherwise a message that the element is not found

1. Initialize two pointers: low = 0 and high = len(arr) - 1
2. While low ≤ high, repeat steps 3-6
3. Calculate mid = (low + high) // 2
4. If arr[mid] == key, return mid (position found)
5. If arr[mid] < key, set low = mid + 1 (search in the right half)
6. Else, set high = mid - 1 (search in the left half)
7. If the loop ends without finding the key, return that the element is not found

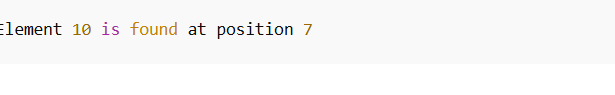
**Python Code:**

****

**Input:**

****

**Output:**

****

**Result : the given number x exists in a sorted array arr using binary search is executed successfully and output is verified**

**Performance analysis:**

**Time Complexity**

* **Best Case: O(1)*O*(1)  
  (When the key is found at the middle element in the first comparison)**
* **Average Case: O(logn)*O*(log*n*)  
  (Because the search space halves each step)**
* **Worst Case: O(logn)*O* (log *n*)  
  (Key found after lognlog*n* divisions or not found at all)**

**Space Complexity**

* **O(1)*O*(1)  
  (Binary search uses only a fixed amount of extra space regardless of input size)**