**Identify the Big-O Notation of the following Blocks of Codes**: (Identify and Discuss the Block of Codes' Structure which complies to the identified Notation)

const sortedInventory =[10, 20, 30, 40, 50, 60, 70];

const binarySearch = (arr, target) => {

let left = 0;

let right = arr.length - 1;

while (left <= right) {

const mid = Math.floor((left + right) / 2);

if(arr[mid] === target) {

return 'Found ${target}';

} else if (arr[mid] < target){

left mid + 1;

} else {

right = mid - 1;

}

}

return '${target} not found';

};

console.log(binarySearch (sortedInventory, 30));

module.exports = { binarySearch };

The **Big-O Notation** for this code is **O(log n)**. This is because, in binary search, the problem size is reduced by half with each iteration of the loop, leading to a logarithmic time complexity.

1. **Initialization**: Setting left and right pointers (**O(1)** time complexity).
2. **While Loop**: The loop continues as long as left <= right. Each iteration of the loop performs:
   * **Calculating mid**: Finding the middle index, which is done in constant time (**O(1)**).
   * **Comparisons**: Checking if the middle element is equal to the target, less than the target, or greater than the target. Each of these comparisons is performed in constant time (**O(1)**).
   * **Adjustments**: Updating the left or right pointers, based on the comparison results, which are also constant time operations (**O(1)**).
3. The overall complexity is determined by the number of times the array is halved, which leads to a logarithmic time complexity (**O(log n)**).

Therefore, the binary search algorithm efficiently narrows down the search space to locate the target value, making it a significantly faster approach compared to linear search for large datasets.