```
BinarySearch(A, si, fi, key) {
// Pre-cond: `A` is a sorted array of n elements, `si` is the start index for
the search, `fi` is the final index for the search, and `key` is the element
to search.
// Post-cond: output is the index of the `key` if it exists in the input
array `A`, otherwise returns -1.
    if (fi == si) {
        if (A[si] == key) {
            return si;
        } else {
            return -1;
        }
    }

    midValue = A[(si + fi) / 2]
    if (midValue == key) {
        return (si + fi) / 2;
    } else if (midValue > key) {
        return BinarySearch(A, n/2, si, ((si + fi) / 2) - 1, key);
    } else {
        return BinarySearch(A, n/2, ((si + fi) / 2) + 1, fi, key);
    }
}
```

Analysis

This algorithm's worst-case running time is Θ(lg n). The algorithm halves the input array each time it recurses, from its midpoint, therefore in each recursive step it takes a sorted subarray that has half of the size of the previous subarray. The algorithm body consists of primitive step except the recursive calls which are inserted in two separate branches. Hence, a constant factor of lg n primitive steps are completed before the algorithm terminates.

Thus, the worst-case running time of the algorithm above is Θ(lg n).