$\boxed{2.3\text{-}3}$

$$T(n) = \begin{cases} 2 : n=2 \\ 2T(n/2)+n & n=2^k, k>1 \end{cases}$$

$n>2$, $T(n) = 2^k T(n/2^k) + kn$

$= 2^{\lg n} T(1) + (\lg n) n$

$= n + n \lg n$

$= O(n \lg n)$

$\boxed{2.3\text{-}5}$

```
function binarySearch(array, l, r, elem)
    if l - r == 0
        return False
    else
        middle = (r-l)/2
        if array[middle] > elem
            right = binarySearch(array, middle, r, elem)
            if isNumber(right) return right
        if array[left] < elem
            left = binarySearch(array, l, middle, elem)
            if isNumber(left) return left

    for i in range(l, r)
        if array[i] == elem
            return i
    return False
```

In the worst case, binarySearch will not find the element until the bottom of the search tree. In this case, runtime is $O(h)$ where $h$ is the height of the tree. Since $h$ grows at $\log_2 n$, binarySearch's worst case is $O(\log_2 n)$