Q1:

In the case of quicksort, the average case sees the pivot somewhere in the middle of the array, and the worst case sees the pivot always at an end of the array.

```
function quickSort(array, low, high)
    if low < high  O(1)
        // Partition the array and get the pivot index
        pivotIndex = partition(array, low, high)  O(n)

        // Recursively sort the elements before and after partition
        quickSort(array, low, pivotIndex - 1)  T(n/2) (T(n-1) in worst case)
        quickSort(array, pivotIndex + 1, high)  T(n/2) (T(n-1) in worst case)

function partition(array, low, high)
    // Choose the rightmost element as the pivot
    pivot = array[high]  O(1)
    i = low - 1 // Index of smaller element  O(1)

    for j from low to high - 1  O(n)
        if array[j] < pivot  O(1)
            i = i + 1  O(1)
            // Swap if current element is smaller than or equal to pivot
            swap(array[i], array[j])  O(1)

    // Place the pivot in its correct position
    swap(array[i + 1], array[high])  O(1)
    return i + 1 // Return the index of the pivot  O(1)
```

AC: T(n) = 2T(n/2) + O(n)
WC: T(n) = 2T(n-1) + O(n)

AC Recursion Tree:	**WC Recursion Tree:**
T(n) /\ T(n/2) T(n/2) /\ /\ T(n/4) T(n/4) T(n/4) T(n/4) /\ /\ /\ /\ ..... T(1) T(1) T(1) T(1) T(1) T(1) T(1) T(1)	T(n) /\ T(1) T(n-1) /\ T(1) T(n-2) /\ ..... T(1) T(0)
T(n) /\	T(n) /\

T(n/2) T(n/2) /\ /\ T(n/4) T(n/4) T(n/4) T(n/4) /\ /\ /\ /\ ..... O(n) O(n) O(n) O(n) O(n) O(n) O(n) O(n) --- T(n) = logn(n) + n = O(nlogn)	O(n) T(n-1) /\ O(n) T(n-2) /\ ..... O(n) O(n) --- T(n) = n(n) + n = $O(n^2)$
AC Master Theorem: T(n) = 2T(n/2) + n T(n) = aT(n/b) + f(n): a = 2, b = 2, f(n) = n $n^{\log_2 2} = n = f(n)$; T(n) = θ(nlogn)	**WC Master Theorem:** T(n) = 2T(n-1) + n T(n) = aT((n-1)/b) + f(n): a = 2, b = 1, f(n) = n $n^{\log_2 1} = n^2 > f(n)$; T(n) = $θ(n^2)$

Q2:

```
function binarySearchRecursive(array, target, low, high)
    if low > high O(1)
        return -1 // Base case: target not found O(1)

    mid = low + (high - low) / 2 // Calculate the middle index O(1)

    if array[mid] == target O(1)
        return mid // Target found, return the index O(1)
    // Search in the right half
    else if array[mid] < target O(1)
        return binarySearchRecursive(array, target, mid + 1, high) T(n/2)
    // Search in the left half
    else
        return binarySearchRecursive(array, target, low, mid - 1) T(n/2)

// Helper function to start the recursion
function binarySearch(array, target)
    return binarySearchRecursive(array, target, 0, length(array) - 1)
```

T(n) = T(n/2) + O(1)

Recursion Tree:

T(n)
|
T(n/2)
|
T(n/4)
|
.....
T(1)

T(n)
|
T(n/2)
|
T(n/4)
|
.....
O(1)

T(n) = logn(1) + 1 = O(logn)

Master Theorem:

T(n) = T(n/2) + 1

T(n) = aT(n/b) + f(n): a = 1, b = 2, f(n) = 1

$n^{\log_2 1}$ = logn > f(n); T(n) = θ(logn)