Hilde Younce

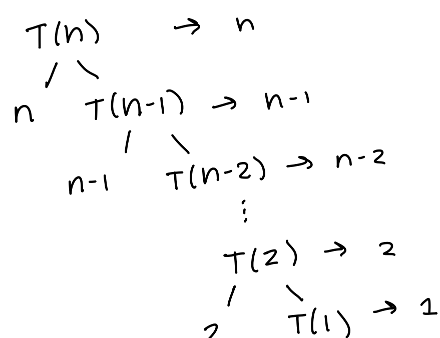
Q 1) Quick Sort:

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

        // Recursively sort the elements before and after partition
        quickSort(array, low, pivotIndex - 1)
        quickSort(array, pivotIndex + 1, high)

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

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

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

$\leftarrow O(n)$

- Recurrence relation (average-case): $T(n) = 2T(\frac{n}{2}) + n$
- Recurrence relation (worst case): $T(n) = T(n-1) + n$

Worst case recursion tree:



$\Rightarrow n + n-1 + n-2 + \cdots + 2 + 1 = \frac{n(n+1)}{2} = \frac{n^2+n}{2} \Rightarrow \boxed{T(n) = O(n^2)}$

Average-Case Recursion Tree:

$$
\begin{array}{ccccl}
 & & T(n) & & \rightarrow n \\
 & T(n/2) & & T(n/2) & \rightarrow n/2 + n/2 = n \\
T(n/4) & T(n/4) & & T(n/4) \quad T(n/4) & \rightarrow n/4 + n/4 + n/4 + n/4 = n \\
 & & \vdots & & \quad \vdots \\
 & & & & \rightarrow n
\end{array}
$$

$$\Rightarrow n(\underbrace{\log n}_{\text{height of tree}} + 1) = n\log n + n \Rightarrow T(n) = \boxed{O(n\log n)}$$

Average Case Master Theorem:

$a = 2$, $b = 2$, $f(n) = n$, $\log_b a = \log_2 2 = 1$

comparing $f(n) = n$ and $n^{\log_b a} = n^1 = n$, they are the same

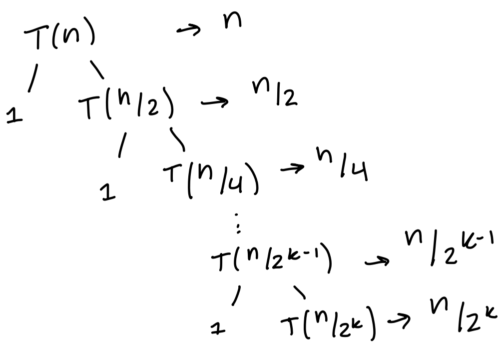
So, $T(n) = O(n^1 \log n) = \boxed{O(n \log n)}$

Q2) Binary Search

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

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

    if array[mid] == target
        return mid // Target found, return the index

    // Search in the right half
    else if array[mid] < target
        return binarySearchRecursive(array, target, mid + 1, high)

    // Search in the left half
    else
        return binarySearchRecursive(array, target, low, mid - 1)

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

- <u>Recurrence relation</u>: $T(n) = T(n/2) + 1$

- <u>Recursion Tree</u>:



$\Rightarrow n + \frac{n}{2} + \frac{n}{4} + \dots + \frac{n}{2^{k-1}} + \frac{n}{2^k}$ where $\frac{n}{2^k} = 1 \Rightarrow k = \log_2 n$

$\Rightarrow T(n) = \boxed{O(\log n)}$

- <u>Master Theorem</u>:

$a = 1$, $b = 2$, $f(n) = 1$, $\log_b a = \log_2 1 = 0$

comparing $f(n) = 1$ with $n^{\log_b a} = n^0 = 1$, they are the same

so, $T(n) = O(n^0 \log n) = \boxed{O(\log n)}$