```
function quickSort (int left, int right, Array arr, int pivot = Default(lastIndex)) {
    if (left - right <= 0)         k0
        return
    else
        pivot = partition(left, right, pivot, arr)   O(n) as the analysis below
        quickSort(left, pivot - 1, arr, pivot)      Since the second half never do anything, here will do everything,
which is n times
        quickSort(pivot + 1, right, arr, pivot)     return immediately, constant k1
    end if
}
```

Conclusion:
partition() => O((n-1) + k2 + k3 + k4) = O(n)
quickSort() => O(n * (k0 + n)) = O(n^2)

so it's n times quickSort() calls with a O(n) runtime complexity partition() in it
Then the whole thing is O(n * n) = O(n^2)

```
function partition(left, right, pivot, arr) {
    leftptr = left
    rightptr = right

    while (true)
        while (leftptr < pivot)    constant 1, left always > pivot
            left++
        while (rightptr > pivot)   n - 1 = O(n)
            right--
        if (leftptr >= rightptr)    k2
            break
        else
            swap leftptr, rightptr  k3
    end while

    swap leftptr, right     k4
    return leftptr
}
```