Analysis of Quicksort

Quicksort, like merge sort, applies the divide-and-conquer paradigm introduced in Section 2.3.1. Here is the three-step divide-and-conquer process for sorting a typical subarray $A[p \mathinner{..} r]$:

Divide: Partition (rearrange) the array $A[p \mathinner{..} r]$ into two (possibly empty) subarrays $A[p \mathinner{..} q-1]$ and $A[q+1 \mathinner{..} r]$ such that each element of $A[p \mathinner{..} q-1]$ is less than or equal to $A[q]$, which is, in turn, less than or equal to each element of $A[q+1 \mathinner{..} r]$. Compute the index q as part of this partitioning procedure.

Conquer: Sort the two subarrays $A[p \mathinner{..} q-1]$ and $A[q+1 \mathinner{..} r]$ by recursive calls to quicksort.

Combine: Because the subarrays are already sorted, no work is needed to combine them: the entire array $A[p \mathinner{..} r]$ is now sorted.

```
QUICKSORT(A, p, r)
if p < r
    q = PARTITION(A, p, r)
    QUICKSORT(A, p, q - 1)
    QUICKSORT(A, q + 1, r)
```

To sort an entire array A, the initial call is QUICKSORT(A, 1, $A.length$).

Partitioning the array

The key to the algorithm is the PARTITION procedure, which rearranges the subarray $A[p \, . . \, r]$ in place.

```
PARTITION(A, p, r)
x = A[r]
i = p - 1
for j = p to r - 1
    if A[j] ≤ x
        i = i + 1
        exchange A[i] with A[j]
exchange A[i + 1] with A[r]
return i + 1
```

Worst Case Partitioning

- The running time of quicksort depends on whether the partitioning is **balanced** or not.

∀ $\Theta(n)$ time to partition an array of n elements

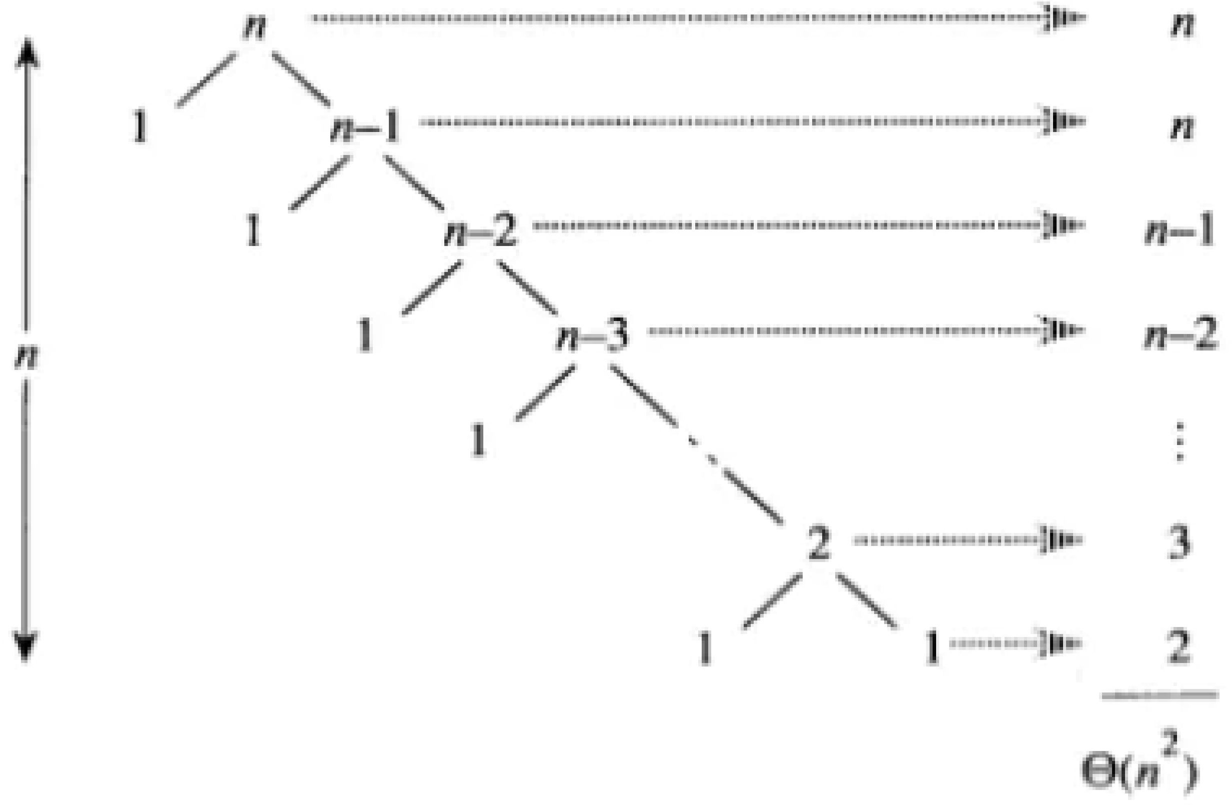
- Let $T(n)$ be the time needed to sort n elements

- $T(0) = T(1) = c$, where c is a constant

- When $n > 1$,
  - $T(n) = T(|\text{left}|) + T(|\text{right}|) + \Theta(n)$

- $T(n)$ is maximum (worst-case) when either $|\text{left}| = 0$ or $|\text{right}| = 0$ following each partitioning

Worst Case Partitioning



Worst Case Partitioning

- **Worst-Case** Performance (unbalanced):
  - $T(n) = T(1) + T(n-1) + \Theta(n)$
    - partitioning takes $\Theta(n)$

  $= [2 + 3 + 4 + \ldots + n-1 + n] + n =$

  $= [\sum_{k=2 \text{ to } n} k] + n = \Theta(n^2)$

$$\sum_{k=1}^{n} k = 1 + 2 + \ldots + n = n(n+1)/2 = \Theta(n^2)$$

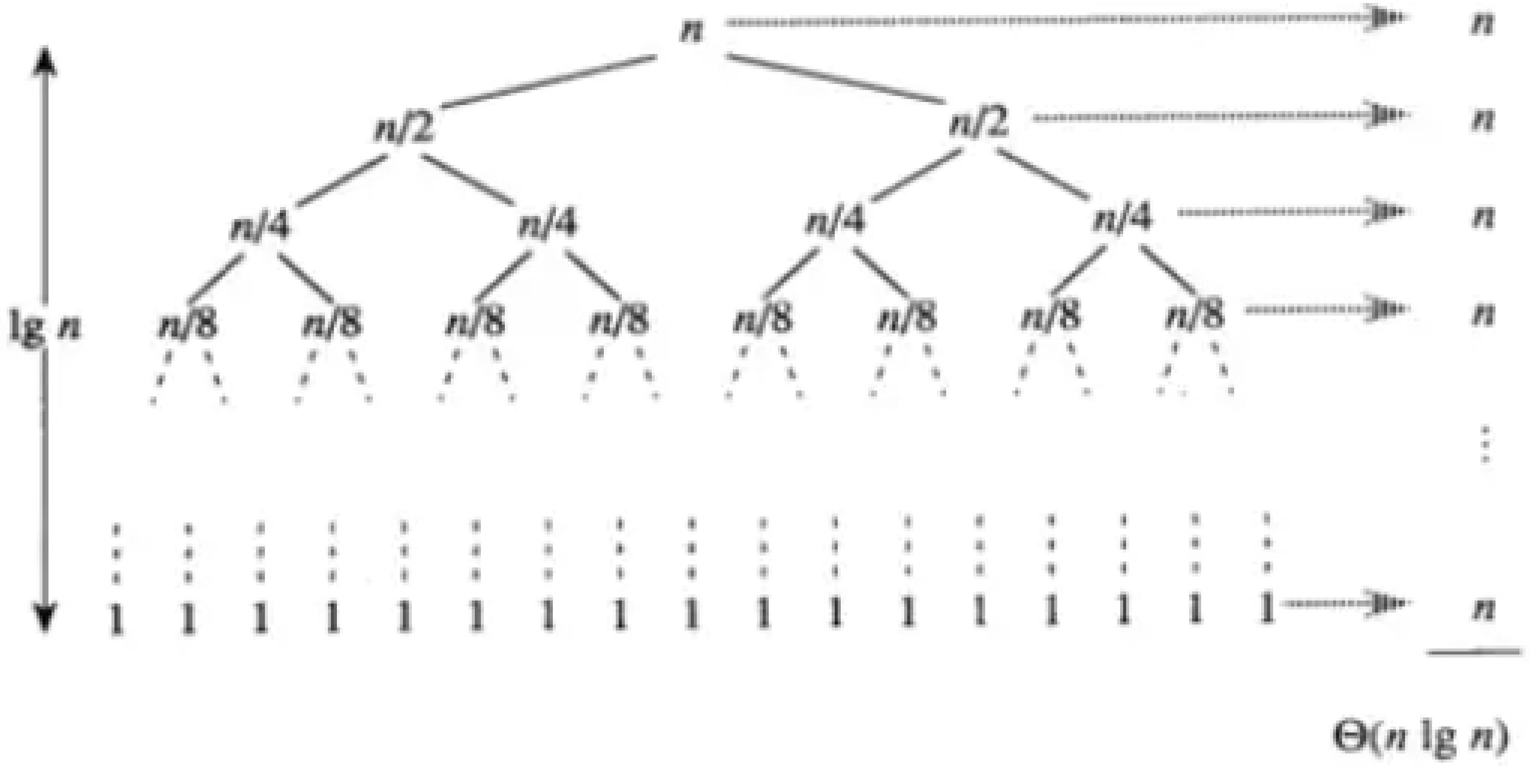
- This occurs when
  - the input is **completely sorted**
- or when
  - the pivot is always the **smallest (largest)** element

Best Case Partition

- When the partitioning procedure produces two regions of size ***n*/2**, we get the a **balanced** partition with **best case** performance:
  - $T(n) = 2T(n/2) + \Theta(n) = \Theta(n \lg n)$

- **Average** complexity is also $\Theta(n \lg n)$

Best Case Partitioning



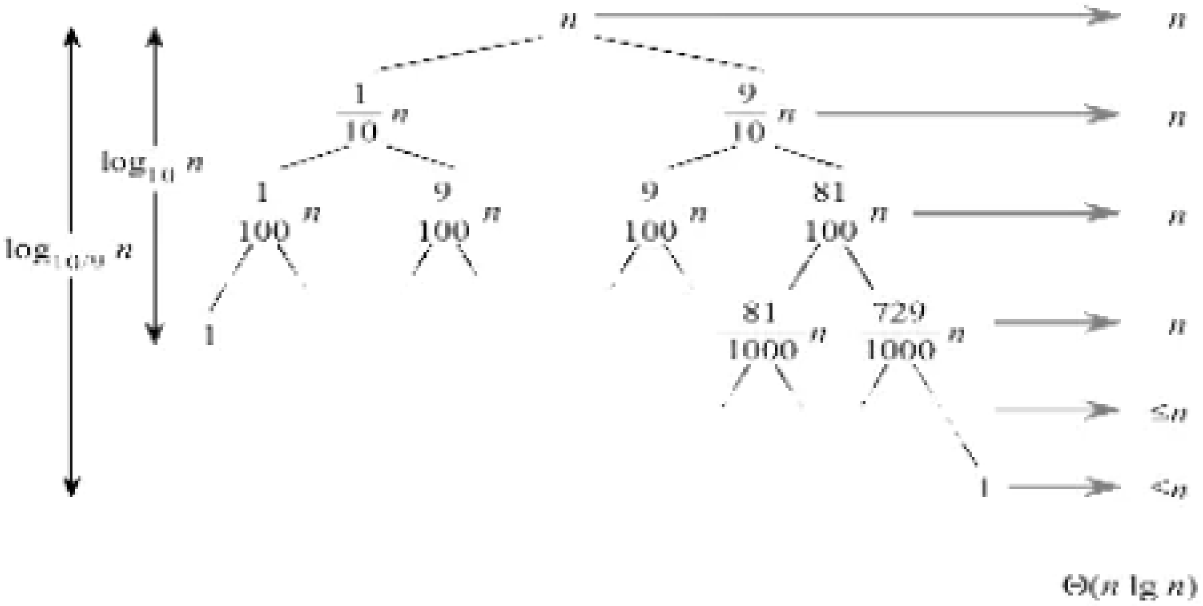
Average Case

- Assuming random input, average-case running time is much closer to $\Theta(n \lg n)$ than $\Theta(n^2)$
- First, a more intuitive explanation/example:
  - Suppose that **partition()** always produces a 9-to-1 **proportional** split. This looks quite unbalanced!
  - The recurrence is thus:

    $T(n) = T(9n/10) + \mathrm{T}(n/10) + \Theta(n) = \Theta(n \lg n)$?

Average Case

$$T(n) = T(n/10) + T(9n/10) + \Theta(n) = \Theta(n\log n)!$$



$$\log_2 n = \log_{10} n / \log_{10} 2$$

Average Case

- Every level of the tree has cost cn, until a boundary condition is reached at depth $\log_{10} n = \Theta(\lg n)$, and then the levels have cost at most cn.
- The recursion terminates at depth $\log_{10/9} n = \Theta(\lg n)$.
- The total cost of quicksort is therefore $O(n \lg n)$.

Average Case

- What happens if we bad-split root node, then good-split the resulting size (n-1) node?
  - We end up with three subarrays, size
    - 1, (n-1)/2, (n-1)/2
  - Combined cost of splits = n + n-1 = 2n -1 = $\Theta(n)$