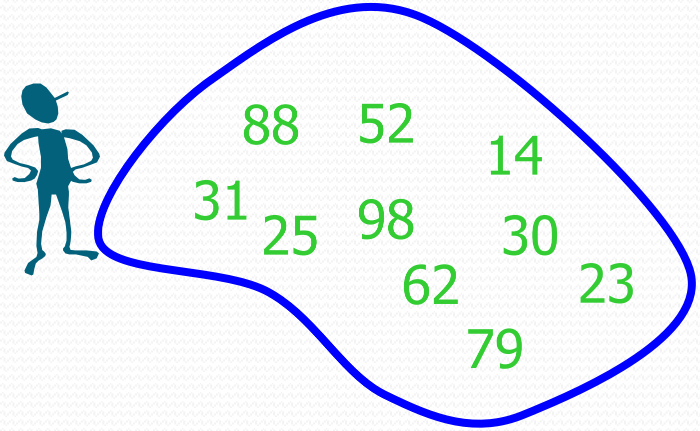
Quick Sort

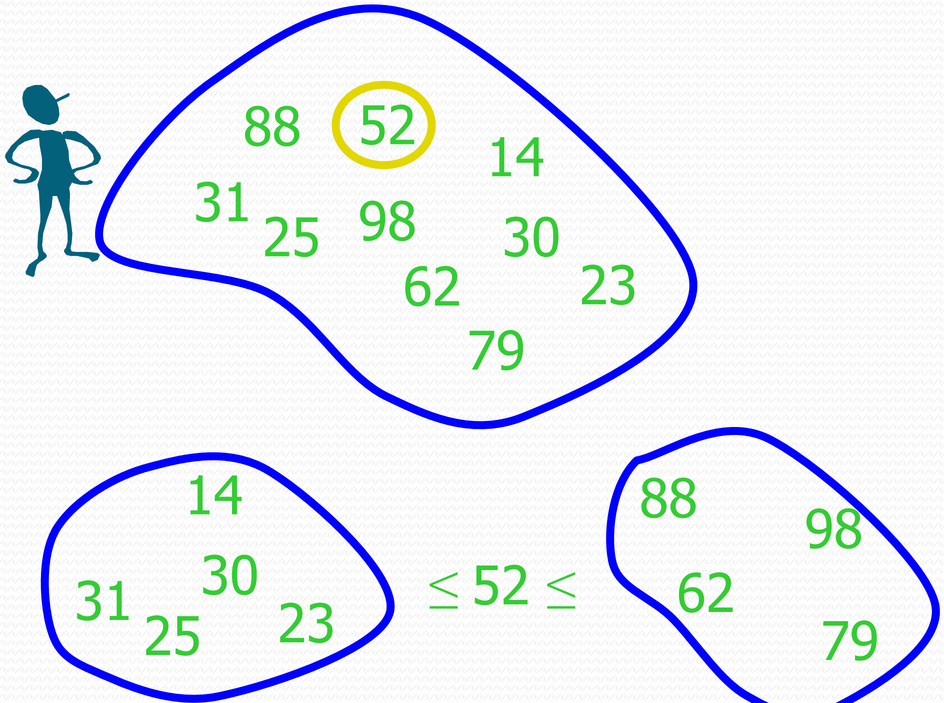
Quick Sort



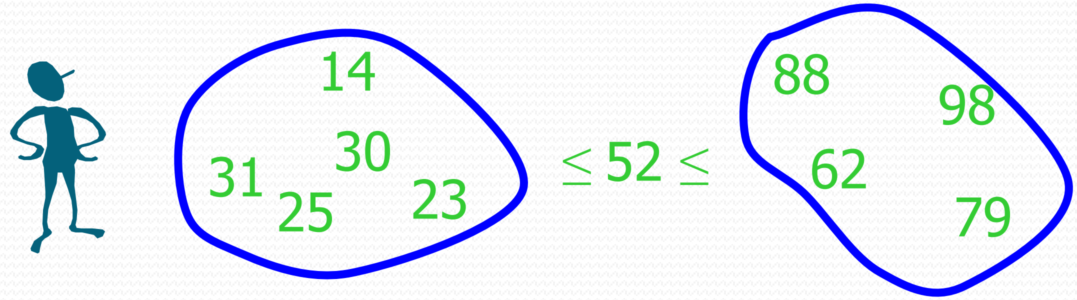
Divide and Conquer

Quick Sort

Partition set into two using randomly chosen pivot



Quick Sort



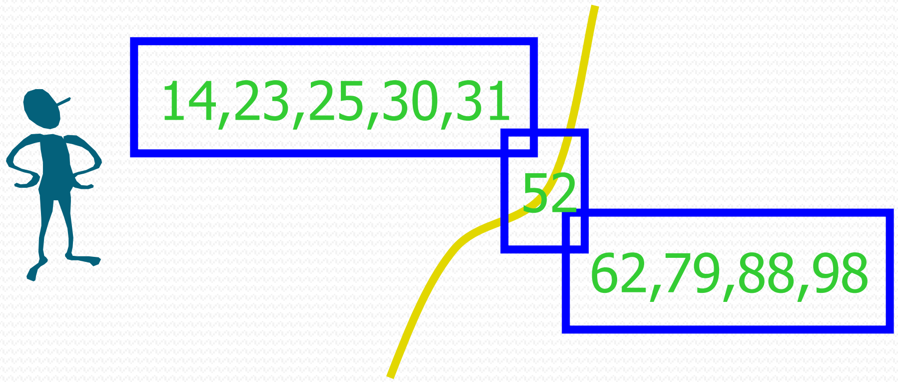
sort the first half.

sort the second half.

14,23,25,30,31

62,79,98,88

Quick Sort



Glue pieces together.

14,23,25,30,31,52,62,79,88,98

Quicksort

- Another divide-and-conquer algorithm:
- *Divide*: $A[p \ldots r]$ is partitioned (rearranged) into two nonempty subarrays $A[p \ldots q\text{-}1]$ and $A[q+1 \ldots r]$ s.t. each element of $A[p \ldots q\text{-}1]$ is less than or equal to each element of $A[q+1 \ldots r]$. Index q is computed here, called **pivot**.
- *Conquer*: two subarrays are sorted by recursive calls to quicksort.
- *Combine*: unlike merge sort, no work needed since the subarrays are sorted in place already.

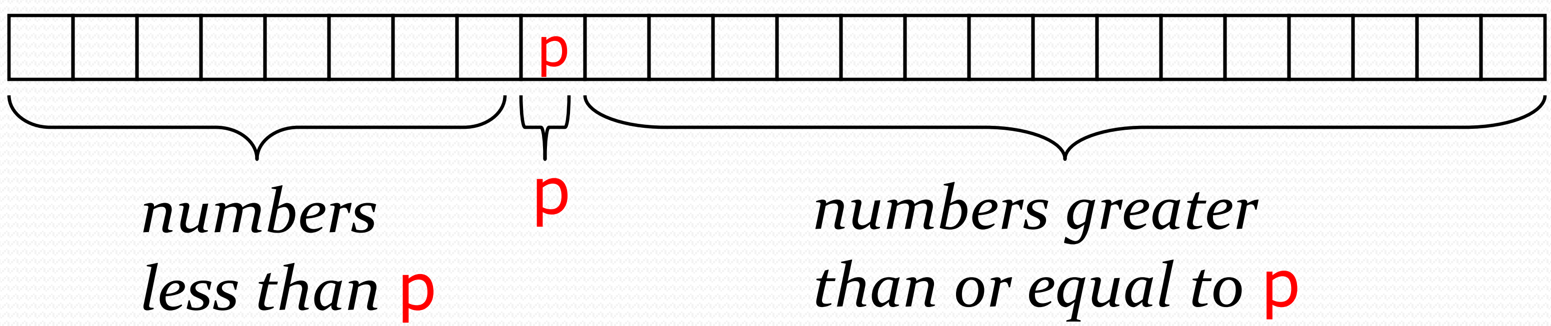
Quicksort

- To sort a[left...right]:

1. if left < right:
    1.1. Partition a[left...right] such that:
        all a[left...p-1] are less than a[p], and
        all a[p+1...right] are >= a[p]
    1.2. Quicksort a[left...p-1]
    1.3. Quicksort a[p+1...right]
2. Terminate

Partitioning

- A key step in the Quick Sort algorithm is partitioning the array
  - We choose some (any) number p in the array to use as a pivot
  - We partition the array into three parts:



Partitioning

- To partition a[left…right]:

1. Set p = a[left], l = left + 1, r = right;
2. while l < r, do
    2.1. while l < right & a[l] < p, set l = l + 1
    2.2. while r > left & a[r] >= p, set r = r - 1
    2.3. if l < r, swap a[l] and a[r]
3. Set a[left] = a[r], a[r] = p
4. Terminate

Example of partitioning

- choose pivot: 4 3 6 9 2 4 3 1 2 1 8 9 3 5 6
- search: 4 3 6 9 2 4 3 1 2 1 8 9 3 5 6
- swap: 4 3 3 9 2 4 3 1 2 1 8 9 6 5 6
- search: 4 3 3 9 2 4 3 1 2 1 8 9 6 5 6
- swap: 4 3 3 1 2 4 3 1 2 9 8 9 6 5 6
- search: 4 3 3 1 2 4 3 1 2 9 8 9 6 5 6
- swap: 4 3 3 1 2 2 3 1 4 9 8 9 6 5 6
- search: 4 3 3 1 2 2 3 1 4 9 8 9 6 5 6 (left > right)
- swap with pivot: 1 3 3 1 2 2 3 4 4 9 8 9 6 5 6

QuickSort

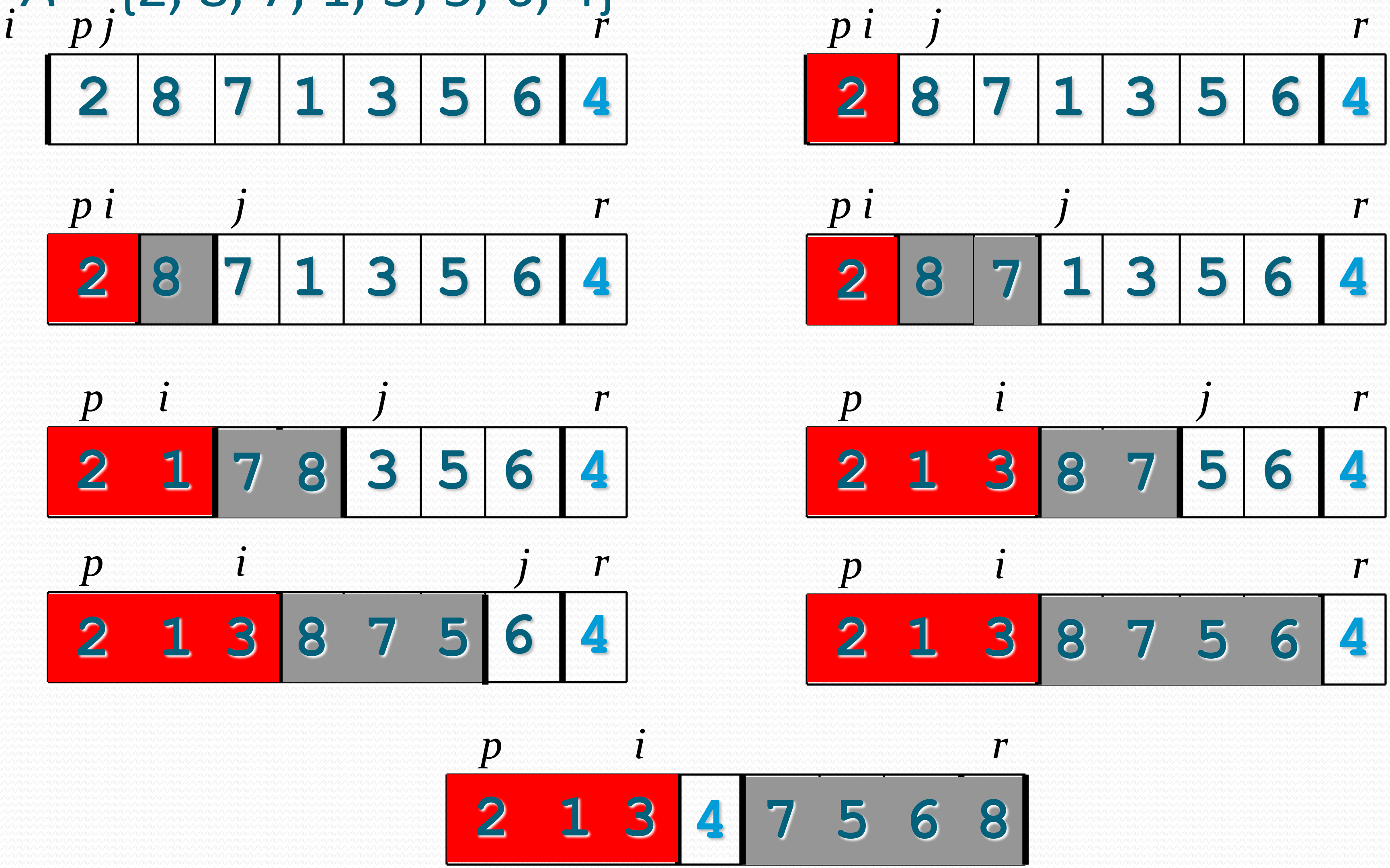
The Pseudo-Code

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

```
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
```

Partition Example

$A = \{2, 8, 7, 1, 3, 5, 6, 4\}$



Recurrence Relation for Quick Sort

- We are dividing the array into two parts based on pivot.
- T(0)=T(1)=c
- T(n) =T(left)+T(right)+n

Time required to sort left sub-array

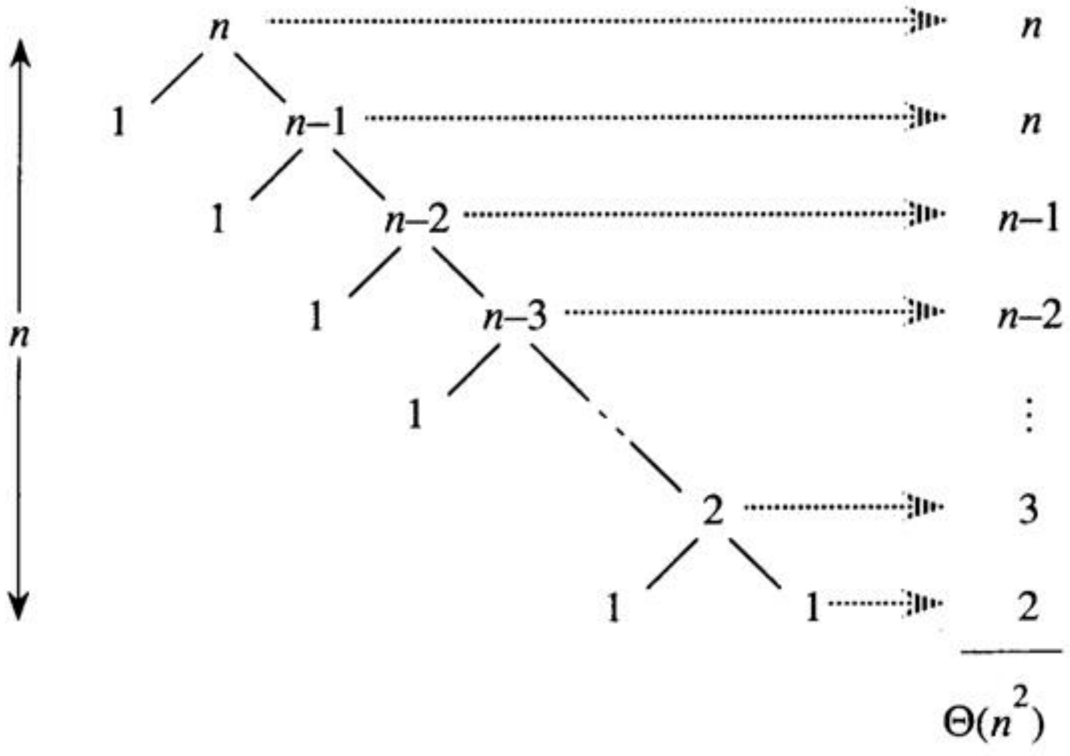
Time required to sort right sub-array

Time required for partitioning n elements

Worst Case

- The running time of quick sort 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



A recursion tree for QUICKSORT in which the PARTITION procedure always puts only a single element on one side of the partition (the worst case). The resulting running time is $\Theta(n^2)$.

Worst Case Partitioning

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

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

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

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

- 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:

$$T(n) = \begin{cases} c & \text{if } n < 2 \\ 2T(n/2) + n & \text{if } n \geq 2 \end{cases}$$

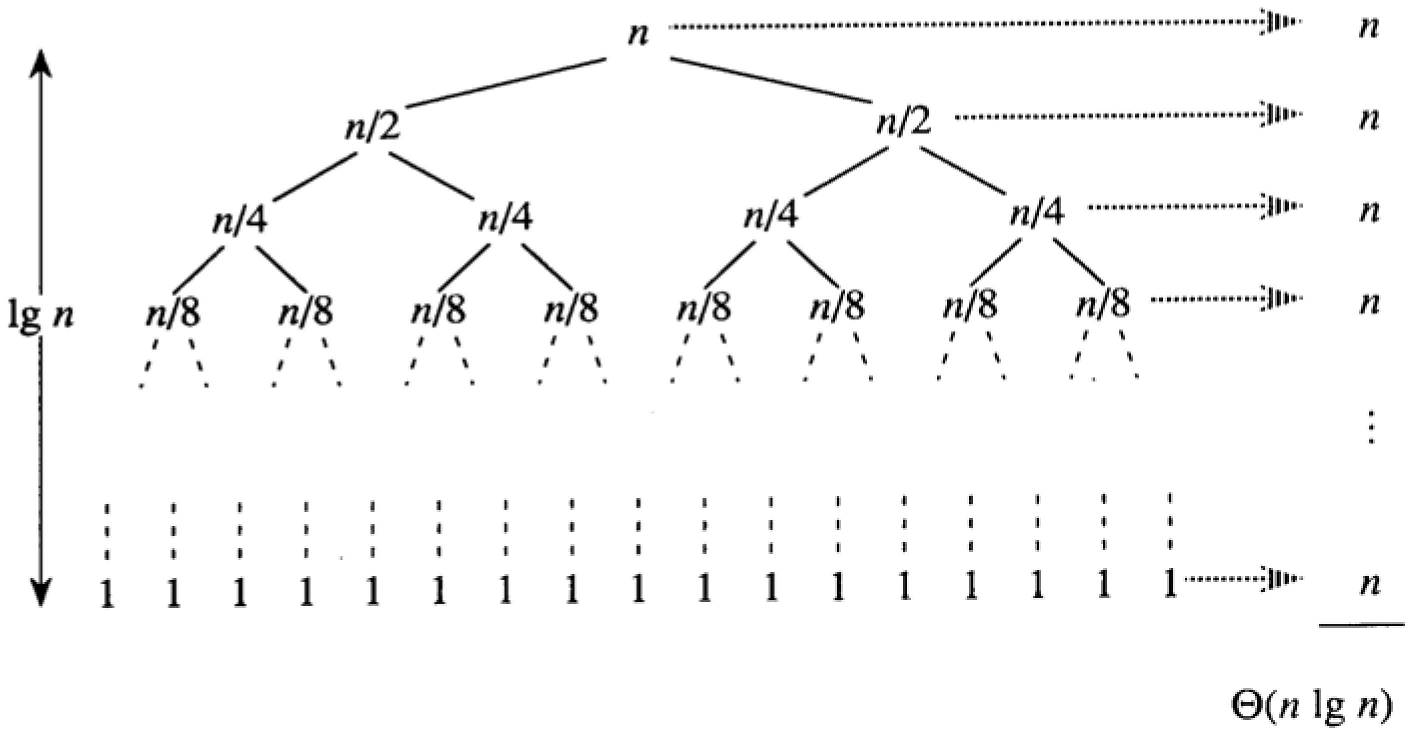
Iterative Method

- We iteratively apply the recurrence equation to itself and see if we can find a pattern:

$$
\begin{aligned}
T(n) &= 2T(n/2) + n \\
&= 2(2T(n/2^2)) + (n/2)) + n \\
&= 2^2 T(n/2^2) + 2n \\
&= 2^3 T(n/2^3) + 3n \\
&= 2^4 T(n/2^4) + 4n \\
&= \ldots \\
&= 2^i T(n/2^i) + in
\end{aligned}
$$

- Note that base, T(n)=c, case occurs when 2^i=n. That is, i = log n.
- So, $T(n) = cn + n\log n$
- Thus, T(n) is O(n log n).

Recursion Tree



A recursion tree for QUICKSORT in which PARTITION always balances the two sides of the partition equally (the best case). The resulting running time is $\Theta(n \lg n)$.

Master Method

- Recurrence: Relation for Quick Sort

$$T(n) = \begin{cases} c & \text{if } n < 2 \\ 2T(n/2) + n & \text{if } n \geq 2 \end{cases}$$

a= 2 b=2   f(n)= n

$n^{\log_b a} = n^{\log_2 2} = n^1 = n$

Case 2: if $f(n) = \Theta(n^{\log_b a})$, then: $T(n) = \Theta(n^{\log_b a} \lg n)$

$T(n) = \Theta(n^{\log_2 2} \lg n) = \Theta(n^1 \lg n) = \Theta(n \lg n)$

Average Case

- Assuming random input, average-case running time is much closer to $\Theta(n \lg n)$ than $\Theta(n^2)$

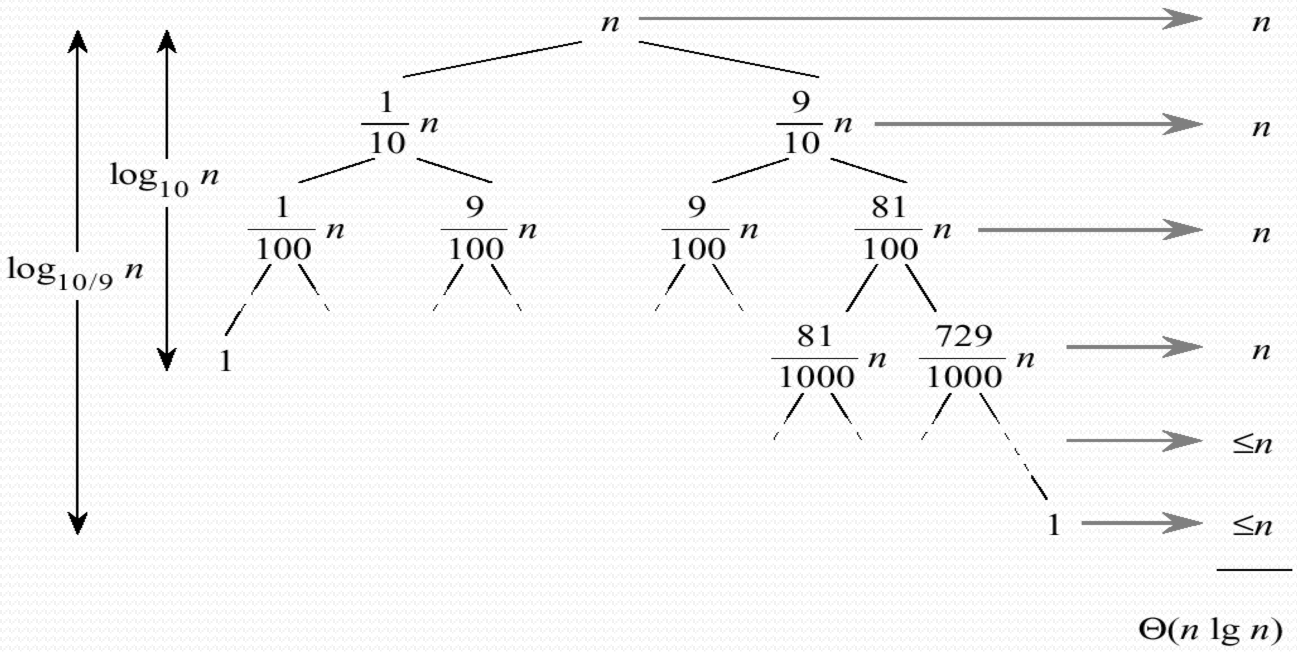
- First, a more intuitive explanation/example:
  - Suppose that **partition()** always produces a 9-to-1 **proportional** split. This looks quite unbalanced!
  - The recurrence is thus:

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

  [Using recursion tree method to solve]

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

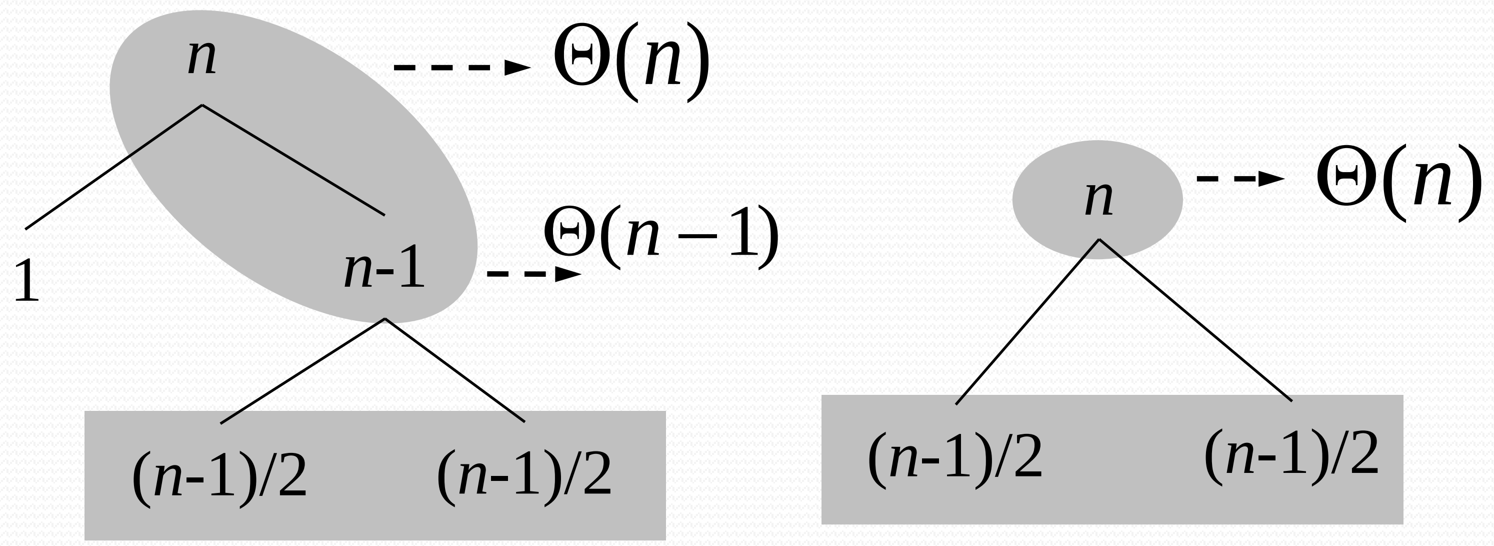
- Every level of the tree has cost cn, until a boundary condition is reached at depth $\log_{10} n = \Theta(\lg n)$, and then the levels have cost at most cn.

- 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 = $n + n\text{-}1 = 2n - 1 = \Theta(n)$



Intuition for the Average Case

- Suppose, we alternate lucky and unlucky cases to get an average behavior

$$L(n) = 2U(n/2) + \Theta(n) \quad \text{lucky}$$

$$U(n) = L(n-1) + \Theta(n) \quad \text{unlucky}$$

we consequently get

$$\begin{aligned} L(n) &= 2(L(n/2-1) + \Theta(n/2)) + \Theta(n) \\ &= 2L(n/2-1) + \Theta(n) \\ &= \Theta(n \log n) \end{aligned}$$

The combination of good and bad splits would result in

***T*(*n*) = O (*n* lg *n*)**, but with slightly **larger constant** hidden by the O-notation.