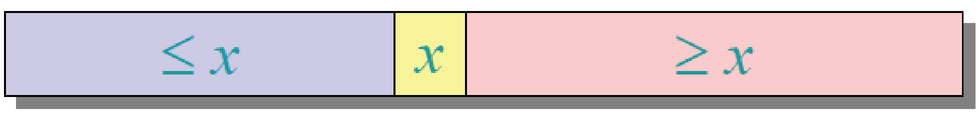
Divide and conquer

Quicksort an n-element array:

1. **Divide:** Partition the array into two subarrays around a pivot x such that elements in lower subarray $\leq x \leq$ elements in upper subarray.



2. **Conquer:** Recursively sort the two subarrays.
3. **Combine:** Trivial.

Key: Linear-time partitioning subroutine.

Partitioning subroutine

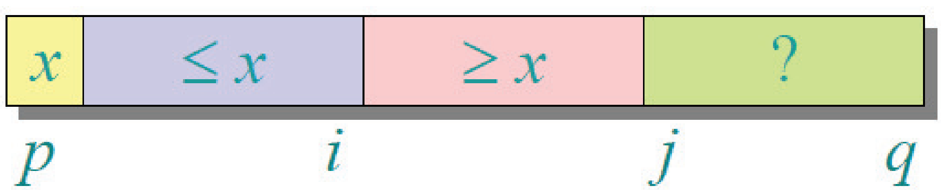
```
PARTITION(A, p, q)  ▷ A[p . . q]
    x ← A[p]          ▷ pivot = A[p]
    i ← p
    for j ← p + 1 to q
        do if A[j] ≤ x
              then  i ← i + 1
                    exchange A[i] ↔ A[j]
    exchange A[p] ↔ A[i]
    return i
```

Partitioning subroutine

```
PARTITION(A, p, q)    ▷ A[p . . q]
    x ← A[p]          ▷ pivot = A[p]
    i ← p
    for j ← p + 1 to q
        do if A[j] ≤ x
              then  i ← i + 1
                    exchange A[i] ↔ A[j]
    exchange A[p] ↔ A[i]
    return i
```

Running time $= O(n)$ for n elements.

Invariant:

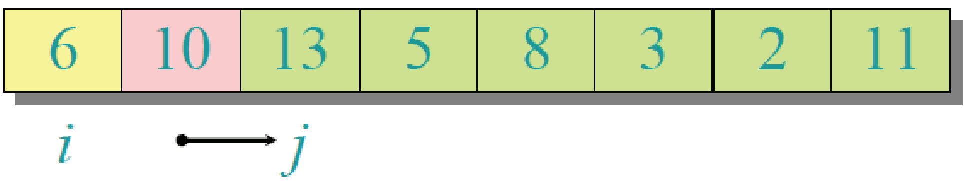


Example of partitioning

6	10	13	5	8	3	2	11
i	j						

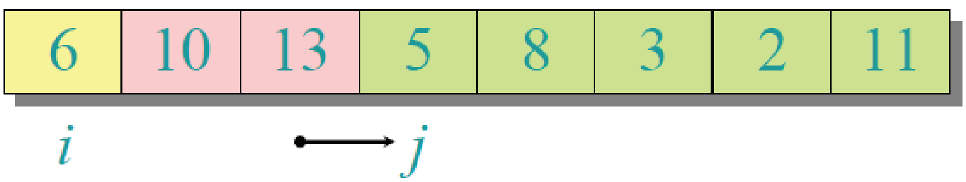
PARTITION(A, p, q) ▷ $A[p \,.\,.\, q]$
  $x \leftarrow A[p]$ ▷ pivot $= A[p]$
  $i \leftarrow p$
  for $j \leftarrow p + 1$ **to** q
    do if $A[j] \leq x$
      then $i \leftarrow i + 1$
        exchange $A[i] \leftrightarrow A[j]$
  exchange $A[p] \leftrightarrow A[i]$
  return i

Example of partitioning



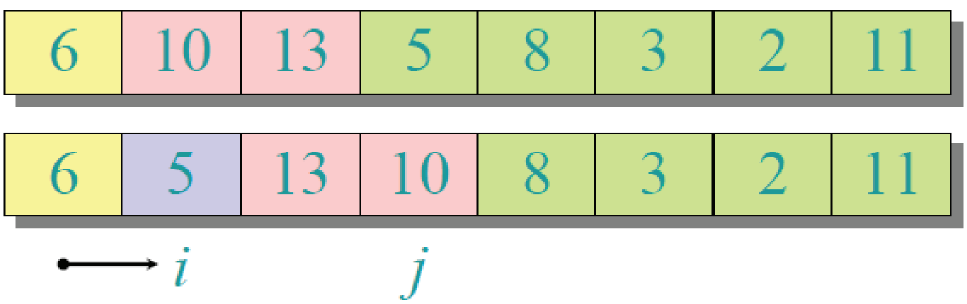

```
PARTITION(A, p, q)  ▷ A[p . . q]
    x ← A[p]          ▷ pivot = A[p]
    i ← p
    for j ← p + 1 to q
        do if A[j] ≤ x
               then  i ← i + 1
                     exchange A[i] ↔ A[j]
    exchange A[p] ↔ A[i]
    return i
```

Example of partitioning



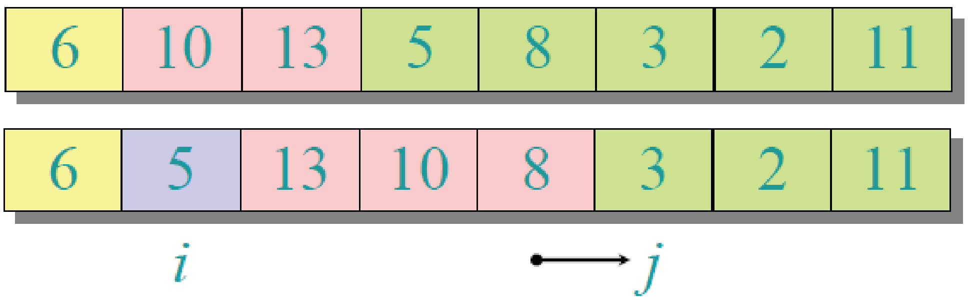

```
PARTITION(A, p, q)  ▷ A[p . . q]
    x ← A[p]          ▷ pivot = A[p]
    i ← p
    for j ← p + 1 to q
        do if A[j] ≤ x
              then  i ← i + 1
                    exchange A[i] ↔ A[j]
    exchange A[p] ↔ A[i]
    return i
```

Example of partitioning



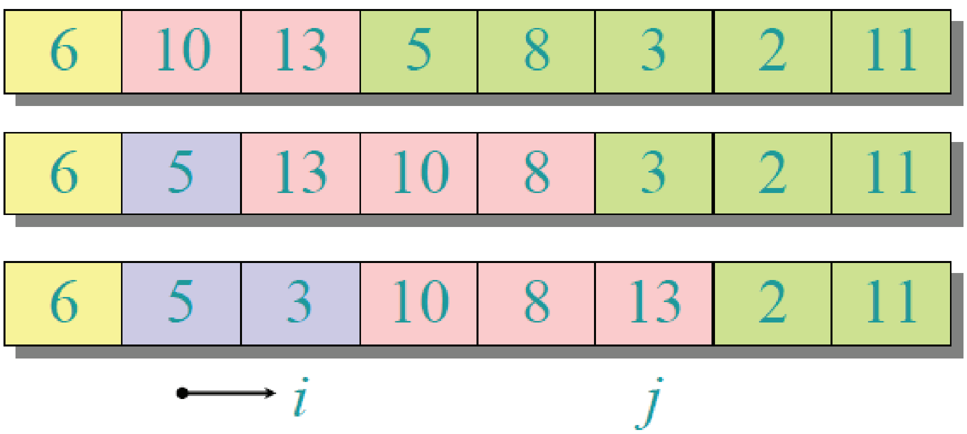

```
PARTITION(A, p, q)  ▷ A[p . . q]
    x ← A[p]        ▷ pivot = A[p]
    i ← p
    for j ← p + 1 to q
        do if A[j] ≤ x
              then  i ← i + 1
                    exchange A[i] ↔ A[j]
    exchange A[p] ↔ A[i]
    return i
```

Example of partitioning



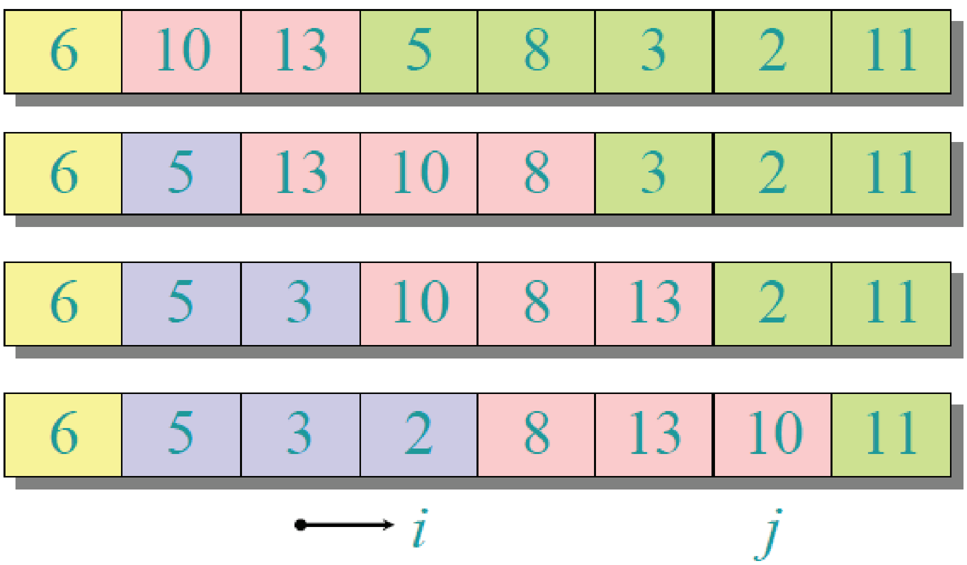

```
PARTITION(A, p, q)  ▷ A[p . . q]
    x ← A[p]            ▷ pivot = A[p]
    i ← p
    for j ← p + 1 to q
        do if A[j] ≤ x
              then  i ← i + 1
                    exchange A[i] ↔ A[j]
    exchange A[p] ↔ A[i]
    return i
```

Example of partitioning



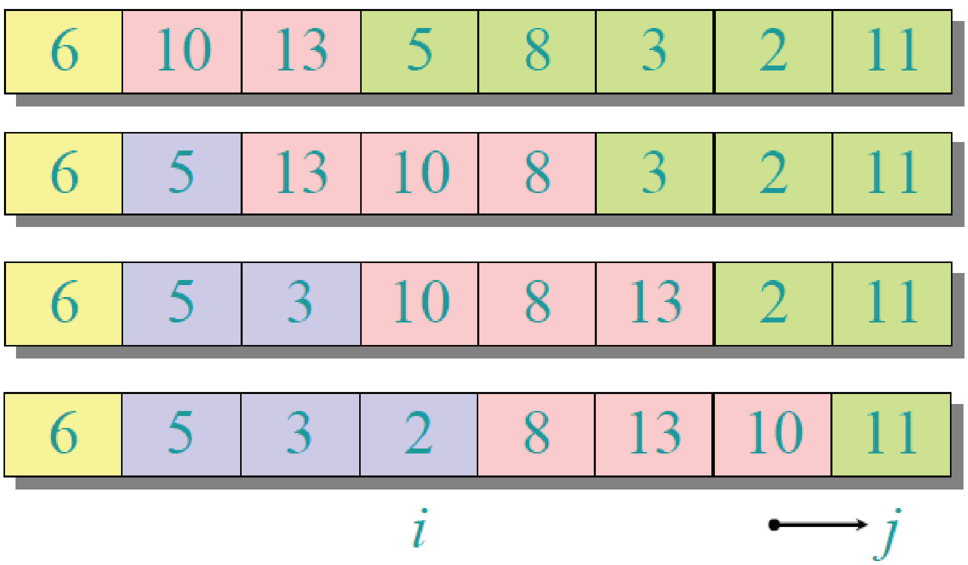

```
PARTITION(A, p, q)  ▷ A[p . . q]
    x ← A[p]        ▷ pivot = A[p]
    i ← p
    for j ← p + 1 to q
        do if A[j] ≤ x
              then  i ← i + 1
                    exchange A[i] ↔ A[j]
    exchange A[p] ↔ A[i]
    return i
```

Example of partitioning



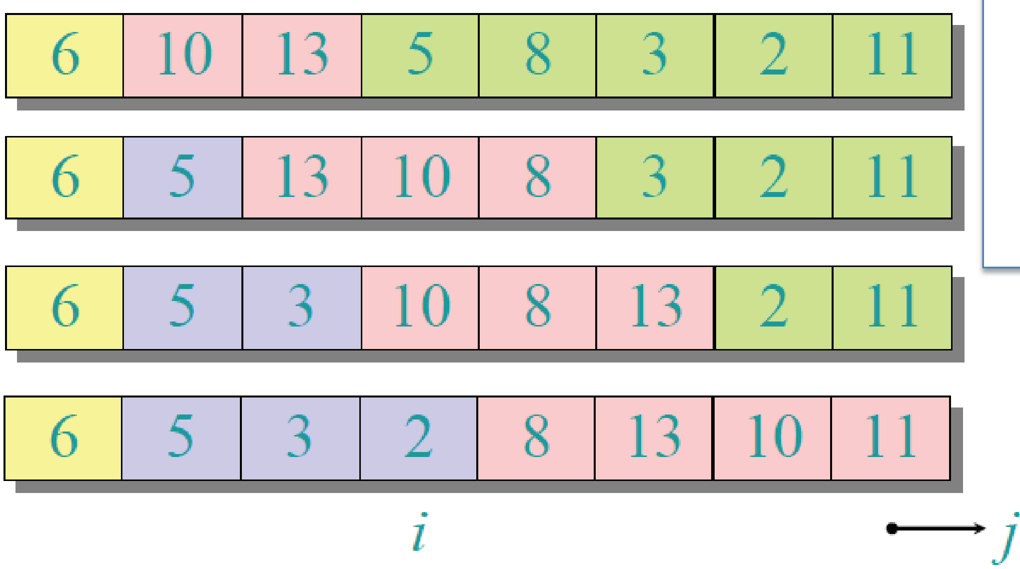

```
PARTITION(A, p, q)  ▷ A[p . . q]
    x ← A[p]            ▷ pivot = A[p]
    i ← p
    for j ← p + 1 to q
        do if A[j] ≤ x
              then  i ← i + 1
                    exchange A[i] ↔ A[j]
    exchange A[p] ↔ A[i]
    return i
```

Example of partitioning



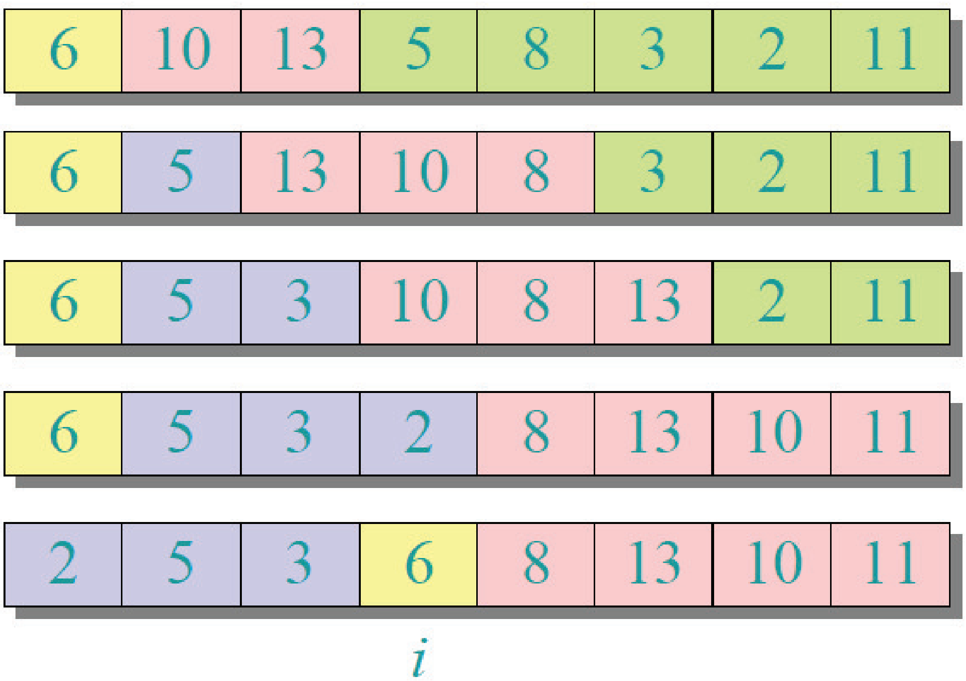
PARTITION(A, p, q) ▷ $A[p \,..\, q]$
  $x \leftarrow A[p]$ ▷ pivot $= A[p]$
  $i \leftarrow p$
  for $j \leftarrow p + 1$ **to** q
    do if $A[j] \leq x$
      then $i \leftarrow i + 1$
        exchange $A[i] \leftrightarrow A[j]$
  exchange $A[p] \leftrightarrow A[i]$
  return i

Example of partitioning




```
PARTITION(A, p, q)  ▷ A[p . . q]
    x ← A[p]         ▷ pivot = A[p]
    i ← p
    for j ← p + 1 to q
        do if A[j] ≤ x
                then  i ← i + 1
                      exchange A[i] ↔ A[j]
    exchange A[p] ↔ A[i]
    return i
```

Example of partitioning




```
PARTITION(A, p, q)  ▷ A[p . . q]
    x ← A[p]        ▷ pivot = A[p]
    i ← p
    for j ← p + 1 to q
        do if A[j] ≤ x
               then  i ← i + 1
                     exchange A[i] ↔ A[j]
    exchange A[p] ↔ A[i]
    return i
```

Pseudocode for quicksort

QUICKSORT(A, p, r)
  if $p < r$
    then $q \leftarrow$ PARTITION(A, p, r)
      QUICKSORT$(A, p, q-1)$
      QUICKSORT$(A, q+1, r)$

Initial call: QUICKSORT$(A, 1, n)$

Analysis of quicksort

- Assume all input elements are distinct.
- In practice, there are better partitioning algorithms for when duplicate input elements may exist.
- Let $T(n)$ = worst-case running time on an array of n elements.

Worst-case of quicksort

- Input sorted or reverse sorted.
- Partition around min or max element.
- One side of partition always has no elements.

$$
\begin{aligned}
T(n) &= T(0) + T(n-1) + \Theta(n) \\
&= \Theta(1) + T(n-1) + \Theta(n) \\
&= T(n-1) + \Theta(n) \\
&= \Theta(n^2)
\end{aligned}
$$

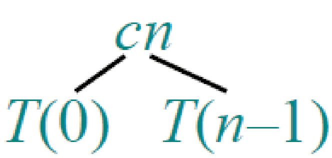
Worst-case recursion tree

$$T(n) = T(0) + T(n–1) + cn$$

$T(n)$

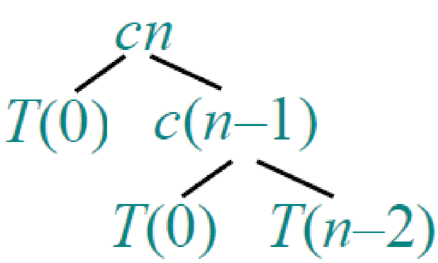
Worst-case recursion tree

$$T(n) = T(0) + T(n-1) + cn$$



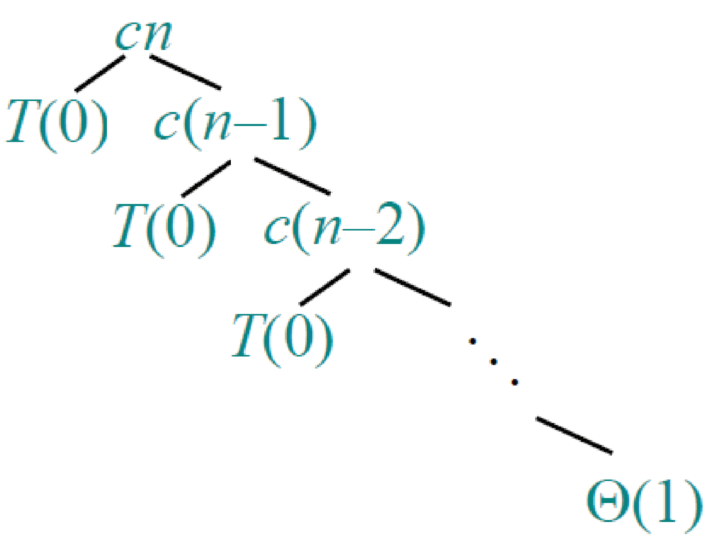
Worst-case recursion tree

$$T(n) = T(0) + T(n–1) + cn$$



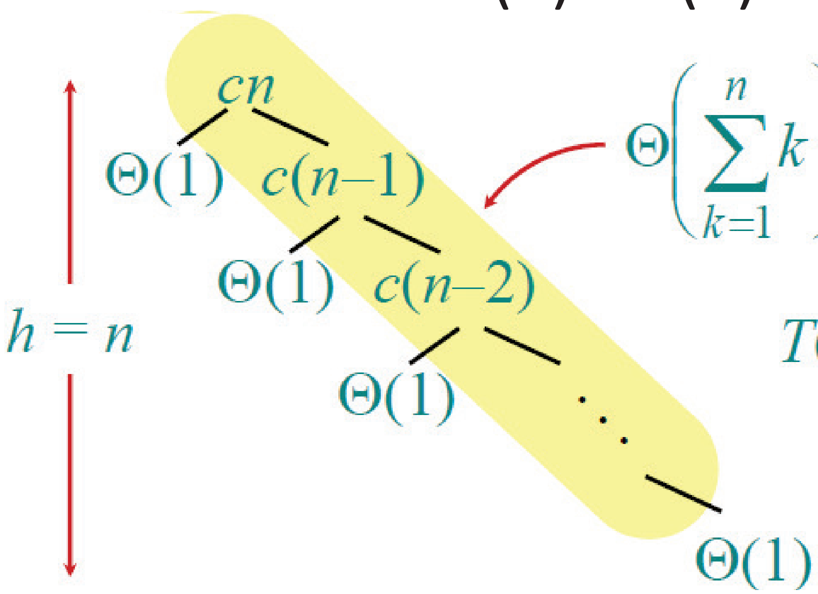
Worst-case recursion tree

$$T(n) = T(0) + T(n–1) + cn$$



Worst-case recursion tree

T(n) = T(0) + T(n–1) + cn



$$\Theta\left(\sum_{k=1}^{n} k\right) = \Theta\left(n^2\right)$$

$$\begin{aligned} T(n) &= \Theta(n) + \Theta(n^2) \\ &= \Theta(n^2) \end{aligned}$$

Best-case analysis

(For intuition only!)

- If we're lucky, PARTITION splits the array evenly:

  $T(n) = 2T(n/2) + \Theta(n)$

  $= \Theta(n\lg n)$ (same as merge sort)

- What if the split is always 1/10 : 9/10 ?

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

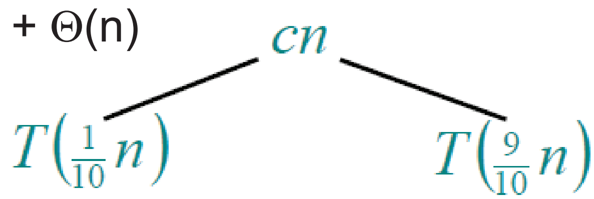
- What is the solution to this recurrence ?

Analysis of “almost-best” case

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

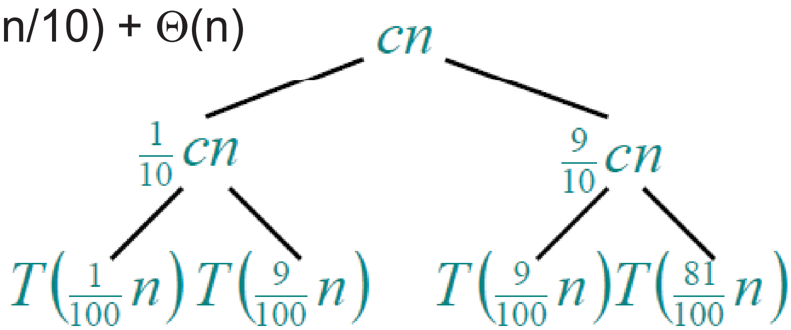
Analysis of “almost-best” case

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



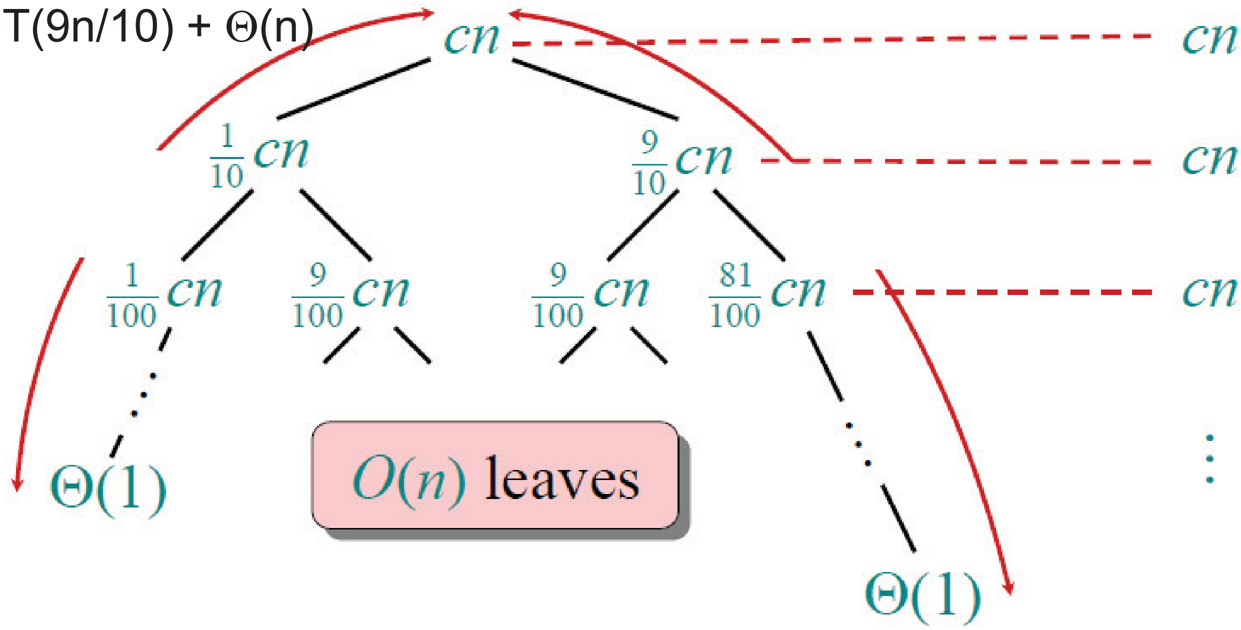
Analysis of “almost-best” case

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



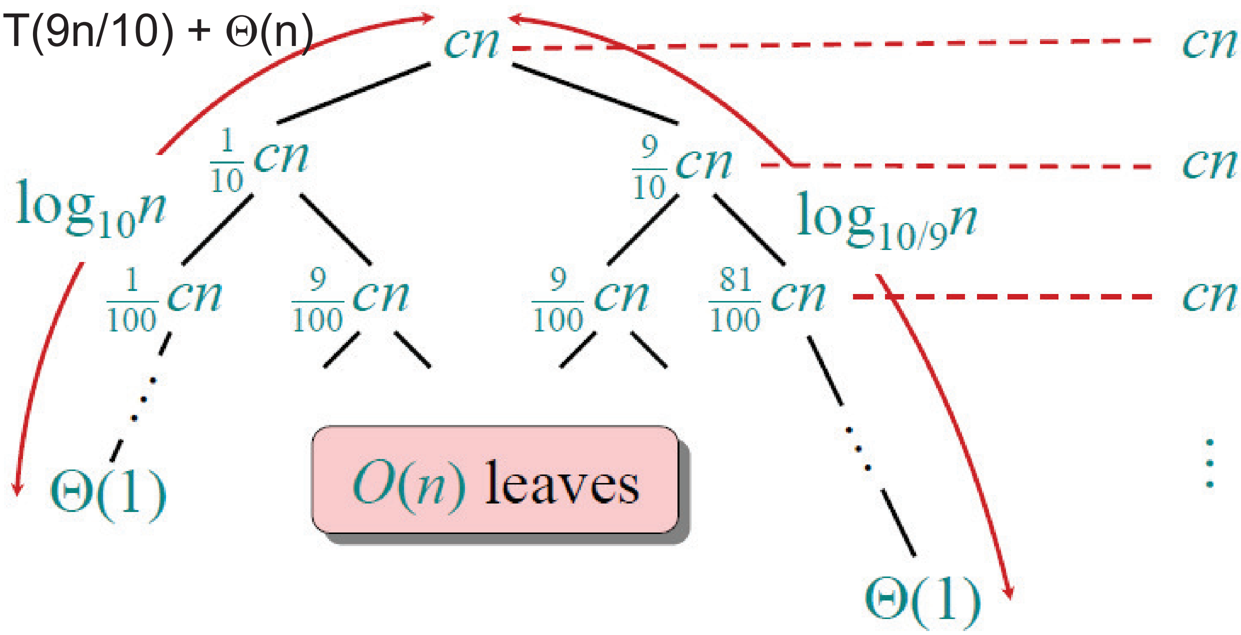
Analysis of “almost-best” case

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



Analysis of “almost-best” case

T(n) = T(n/10) + T(9n/10) + Θ(n)



$cn\log_{10}n \le T(n) \le cn\log_{10/9}n + O(n)$