MA 252: Data Structures and Algorithms

Lecture 6

http://www.iitg.ernet.in/psm/indexing_ma252/y12/index.html

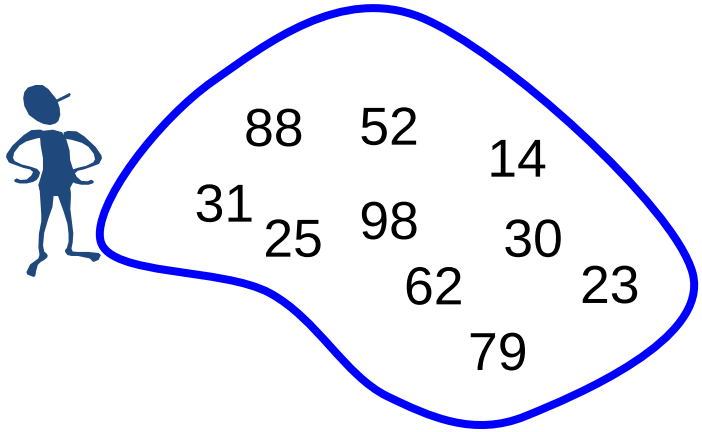
Partha Sarathi Mandal

Dept. of Mathematics, IIT Guwahati

Quicksort

- Proposed by C.A.R. Hoare in 1962.
- Divide-and-conquer algorithm.
- Sorts “in place”(like insertion sort, but not like merge sort).
- Very practical.

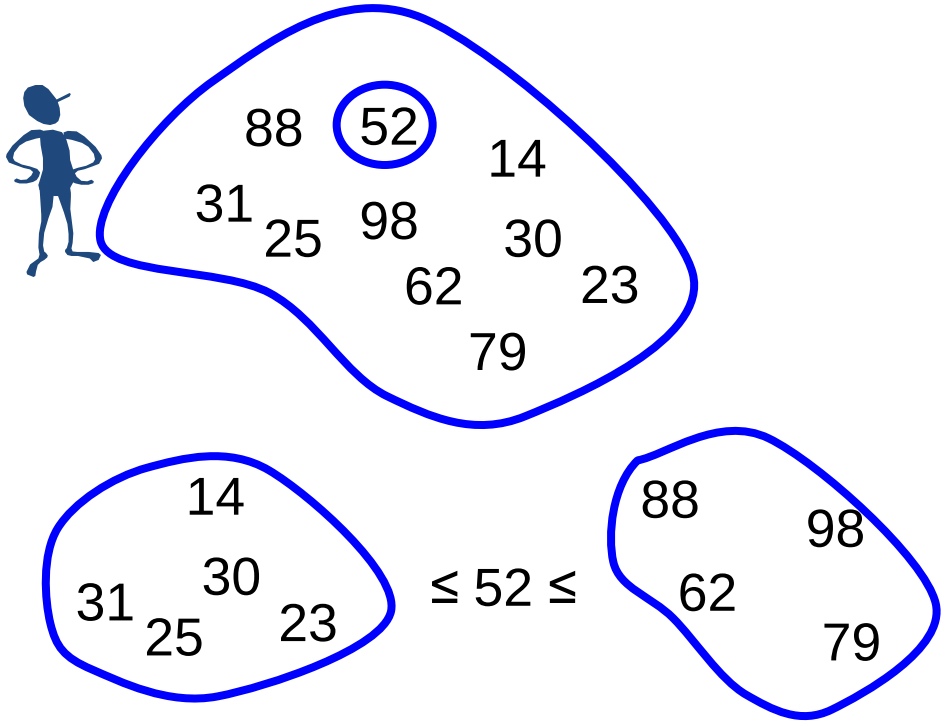
Quicksort



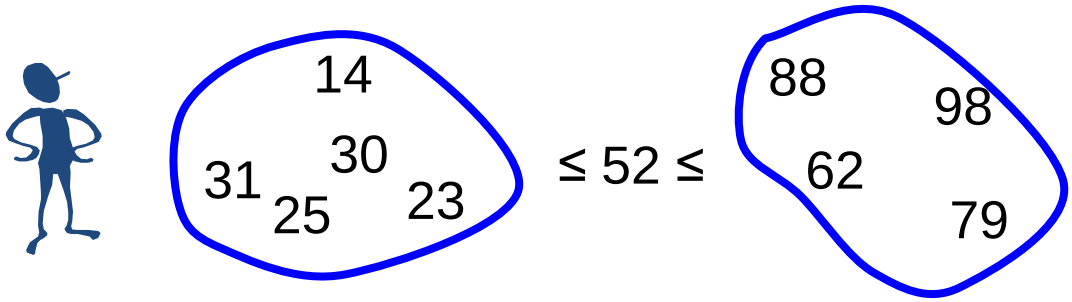
Divide and Conquer

Quicksort

Partition set into two
using randomly chosen pivot



Quicksort



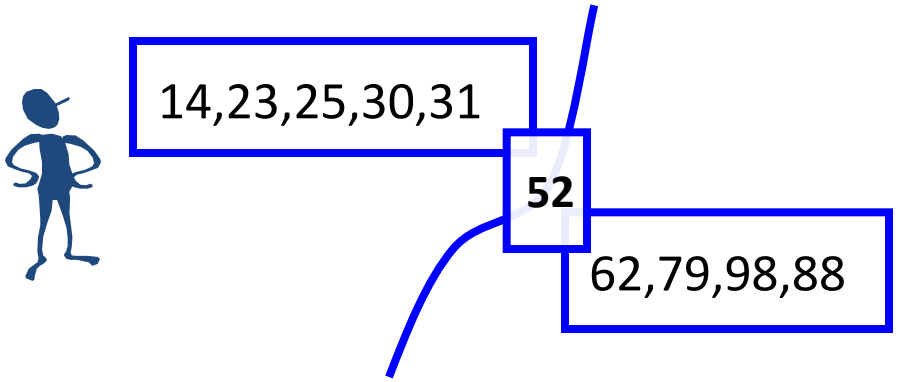
Get one friend to sort the first half.

14,23,25,30,31

Get one friend to sort the second half.

62,79,98,88

Quicksort

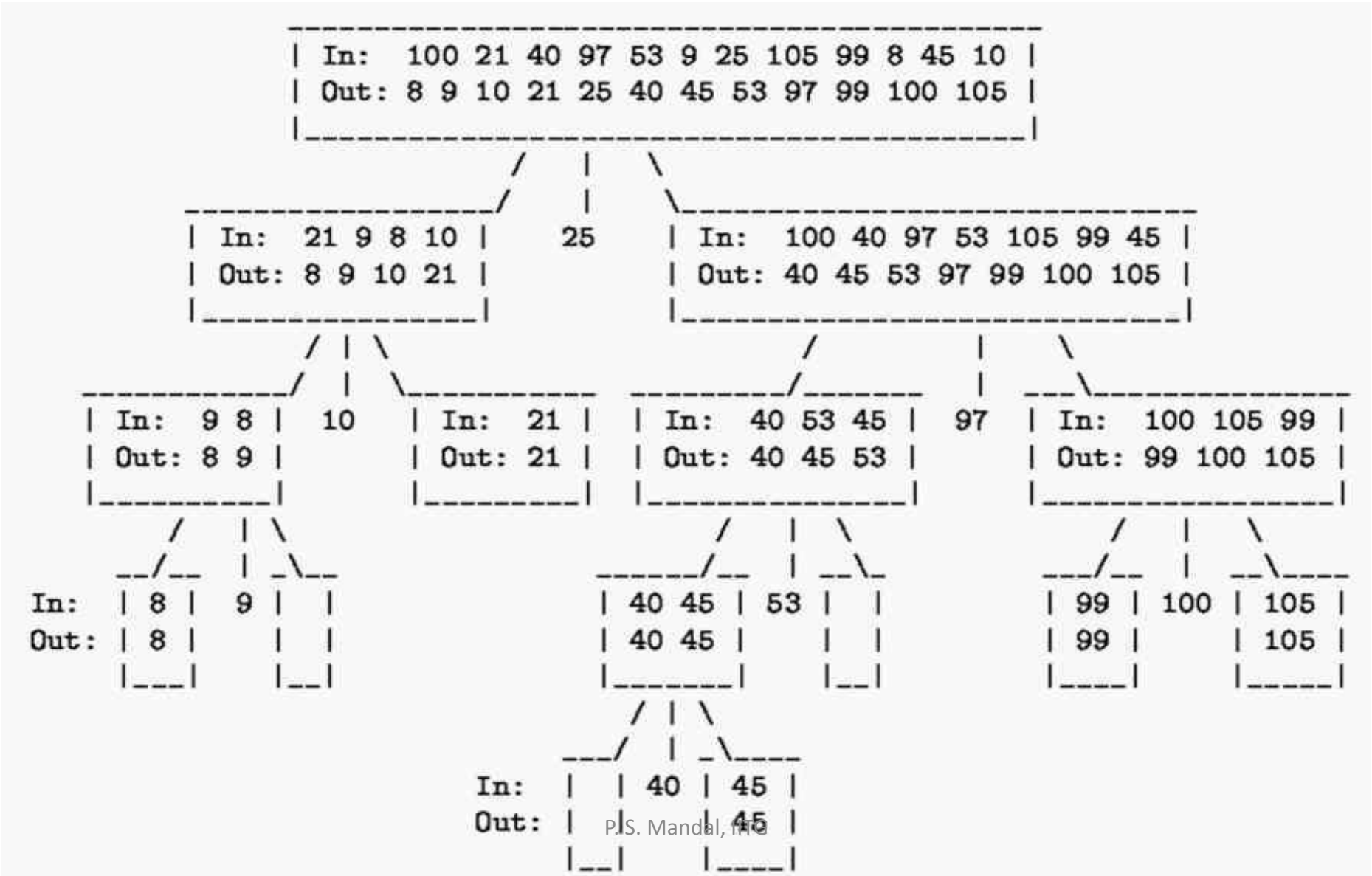


Glue pieces together

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

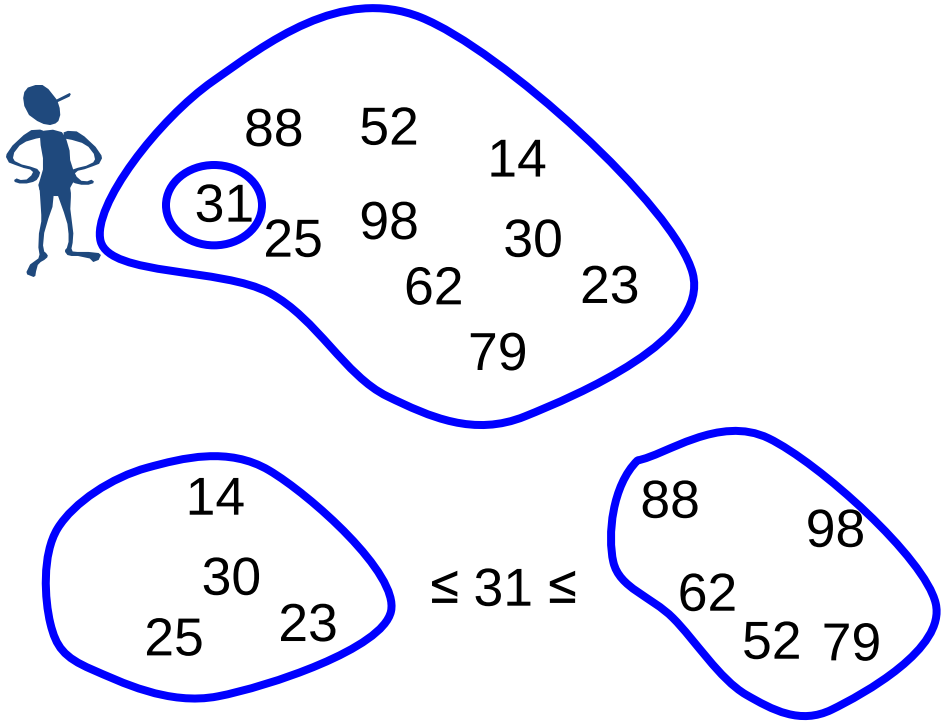
No real work…

Quicksort


In: 100 21 40 97 53 9 25 105 99 8 45 10
Out: 8 9 10 21 25 40 45 53 97 99 100 105
In: 21 9 8 10
Out: 8 9 10 21
25
In: 100 40 97 53 105 99 45
Out: 40 45 53 97 99 100 105
In: 9 8
Out: 8 9
10
In: 21
Out: 21
In: 40 53 45
Out: 40 45 53
97
In: 100 105 99
Out: 99 100 105
In: 8 9
Out: 8 9
In: 40 45 53
Out: 40 45 40 45
In: 99 100 105
Out: 99 99 105
In: 40 45
Out: 40 45

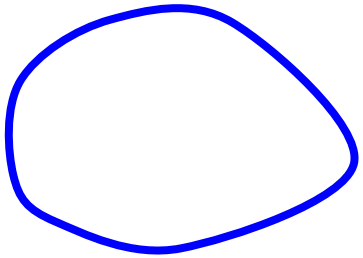
Quicksort

Let pivot be the first element in the list…



Quicksort

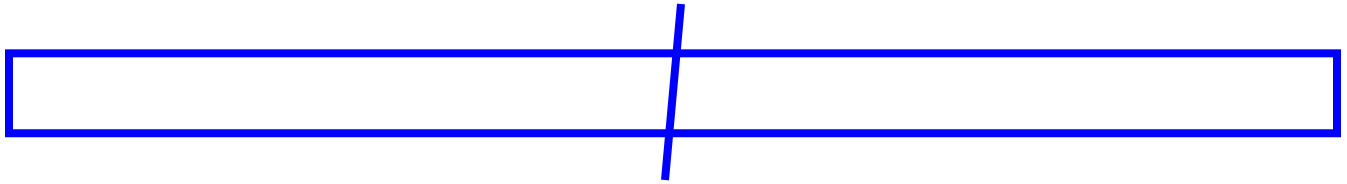
14,23,25,30,31,52,62,79,88,98



≤ 14 ≤ 23,25,30,31,52,62,79,88,98

If the list is already sorted,
then the slit is worst case unbalanced.

Quicksort



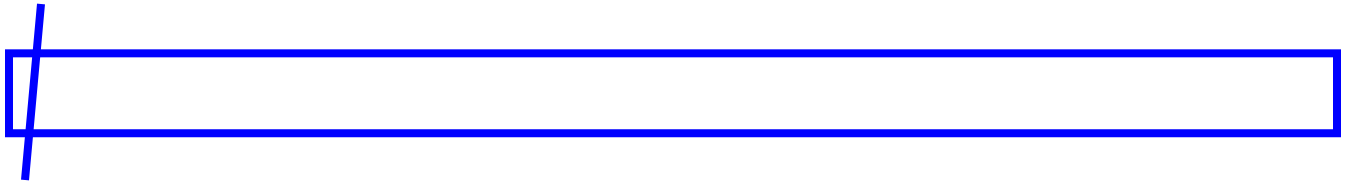
Best Time: $T(n) = 2T(n/2) + \Theta(n)$

$= \Theta(n \log(n))$

Worst Time:

Expected Time:

Quicksort



Best Time: $T(n) = 2T(n/2) + \Theta(n)$

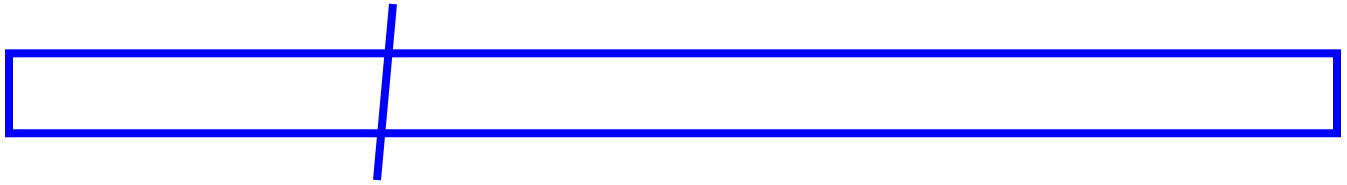
$= \Theta(n \log(n))$

Worst Time: $T(n) = T(0) + T(n-1) + \Theta(n)$

$= \Theta(n^2)$

Expected Time:

Quicksort



Best Time: $T(n) = 2T(n/2) + \Theta(n)$

$= \Theta(n \log(n))$

Worst Time: $T(n) = T(0) + T(n-1) + \Theta(n)$

$= \Theta(n^2)$

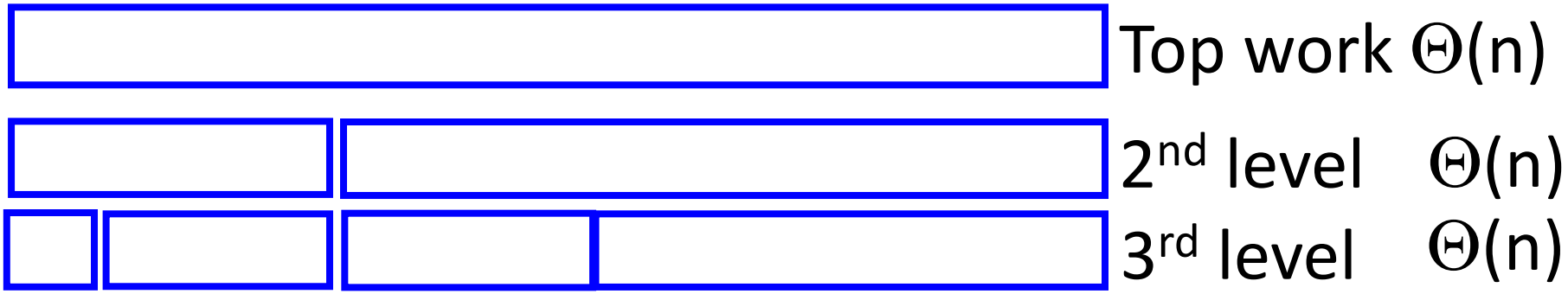
Expected Time: $T(n) = T(^1/_3 n) + T(^2/_3 n) + \Theta(n)$

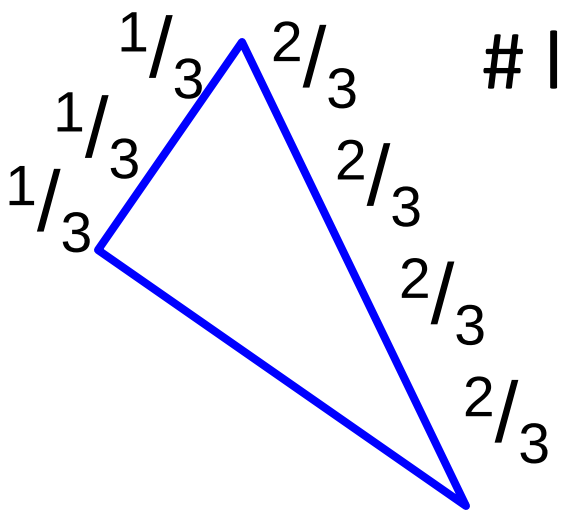
$= \Theta(n \log(n))$

Quicksort

Expected Time: $T(n) = T(1/3 n) + T(2/3 n) + \Theta(n)$

$= \Theta(n \log(n))$





levels = $\log_{3/2}(n)$

$= \Theta(\log(n))$

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

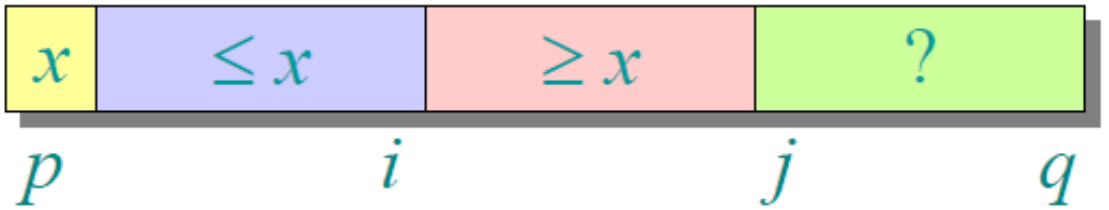
```
PARTITION(A, p, q)  ▷ A[p . . q]
    x ← A[p]        ▷ pivot = A[p]
    i ← p
    for j ← p + 1 to q
        do if A[j] ≤ x
                then  i ← i + 1
                      exchange A[i] ↔ A[j]
    exchange A[p] ↔ A[i]
    return i
```

Partitioning subroutine

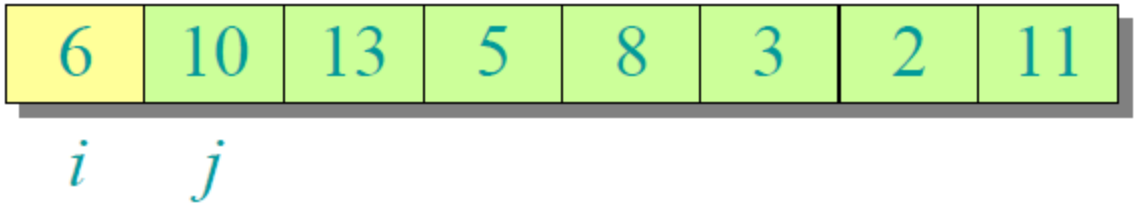
```
PARTITION(A, p, q)  ▷ A[p . . q]
    x ← A[p]        ▷ pivot = A[p]
    i ← p
    for j ← p + 1 to q
        do if A[j] ≤ x
                then  i ← i + 1
                      exchange A[i] ↔ A[j]
    exchange A[p] ↔ A[i]
    return i
```

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

Invariant:

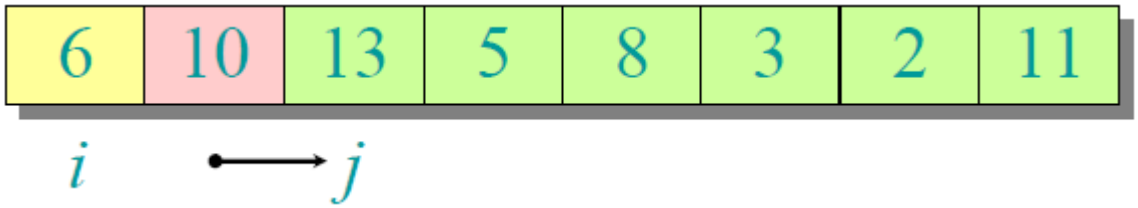


Example of partitioning



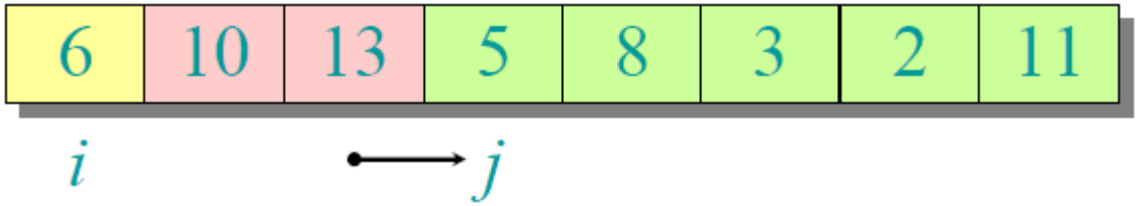

```
PARTITION(A, p, q)  ▷ A[p . . q]
    x ← A[p]        ▷ pivot = A[p]
    i ← p
    for j ← p + 1 to q
        do if A[j] ≤ x
                then  i ← i + 1
                      exchange A[i] ↔ A[j]
    exchange A[p] ↔ A[i]
    return i
```

Example of partitioning



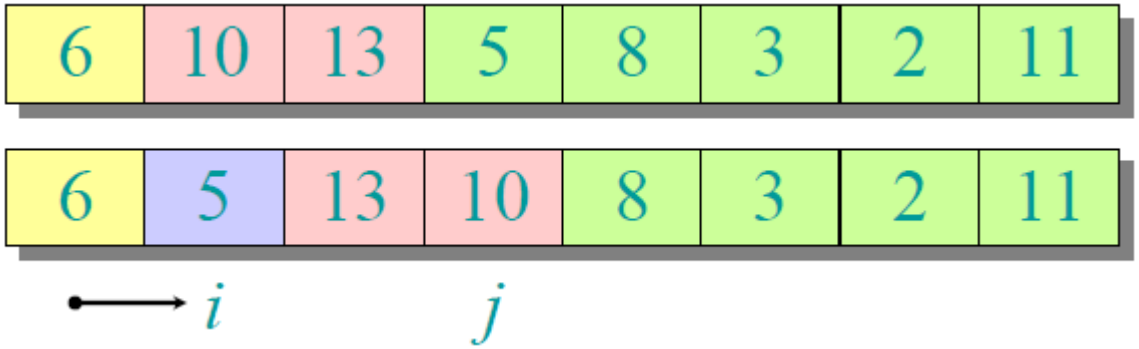

```
PARTITION(A, p, q)  ▷ A[p . . q]
    x ← A[p]        ▷ pivot = A[p]
    i ← p
    for j ← p + 1 to q
        do if A[j] ≤ x
                then  i ← i + 1
                      exchange A[i] ↔ A[j]
    exchange A[p] ↔ A[i]
    return i
```

Example of partitioning



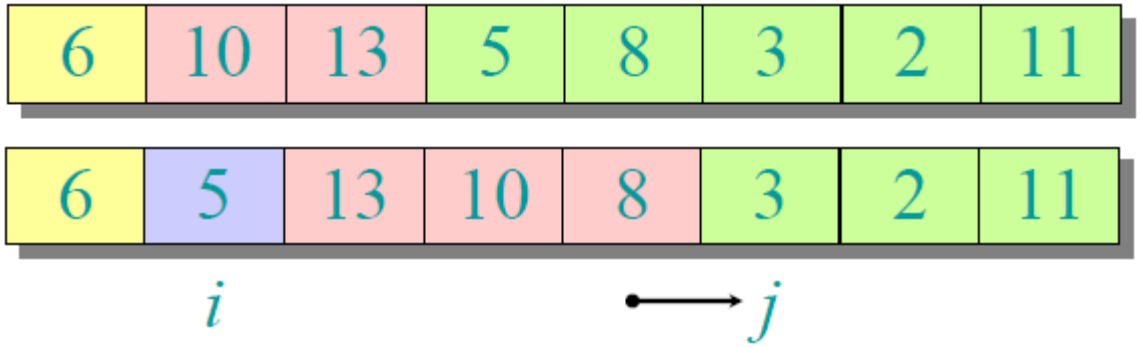

```
PARTITION(A, p, q)  ▷ A[p . . q]
    x ← A[p]        ▷ pivot = A[p]
    i ← p
    for j ← p + 1 to q
        do if A[j] ≤ x
                then  i ← i + 1
                      exchange A[i] ↔ A[j]
    exchange A[p] ↔ A[i]
    return i
```

Example of partitioning



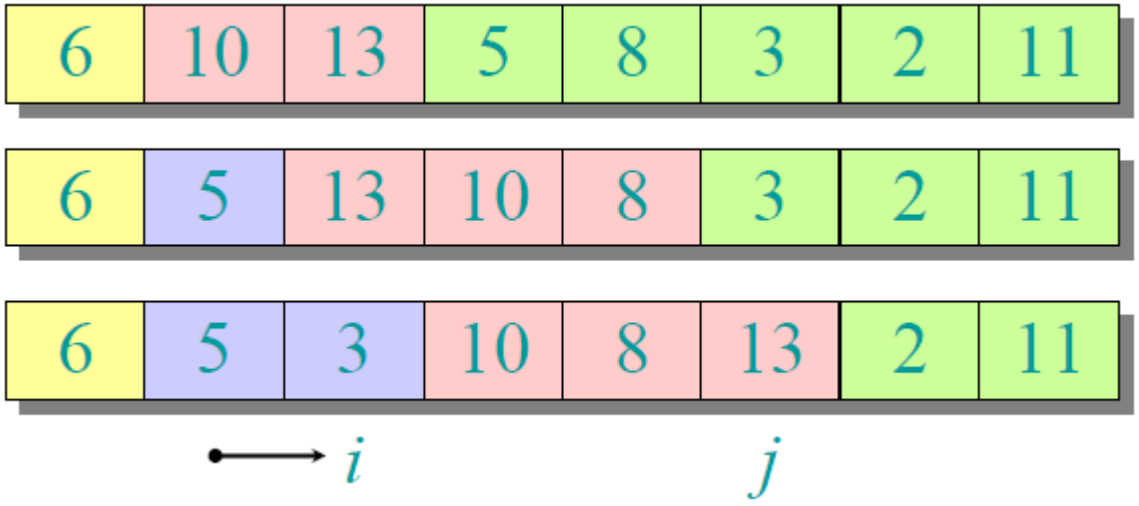

```
PARTITION(A, p, q)  ▷ A[p . . q]
    x ← A[p]        ▷ pivot = A[p]
    i ← p
    for j ← p + 1 to q
        do if A[j] ≤ x
                then  i ← i + 1
                      exchange A[i] ↔ A[j]
    exchange A[p] ↔ A[i]
    return i
```

Example of partitioning



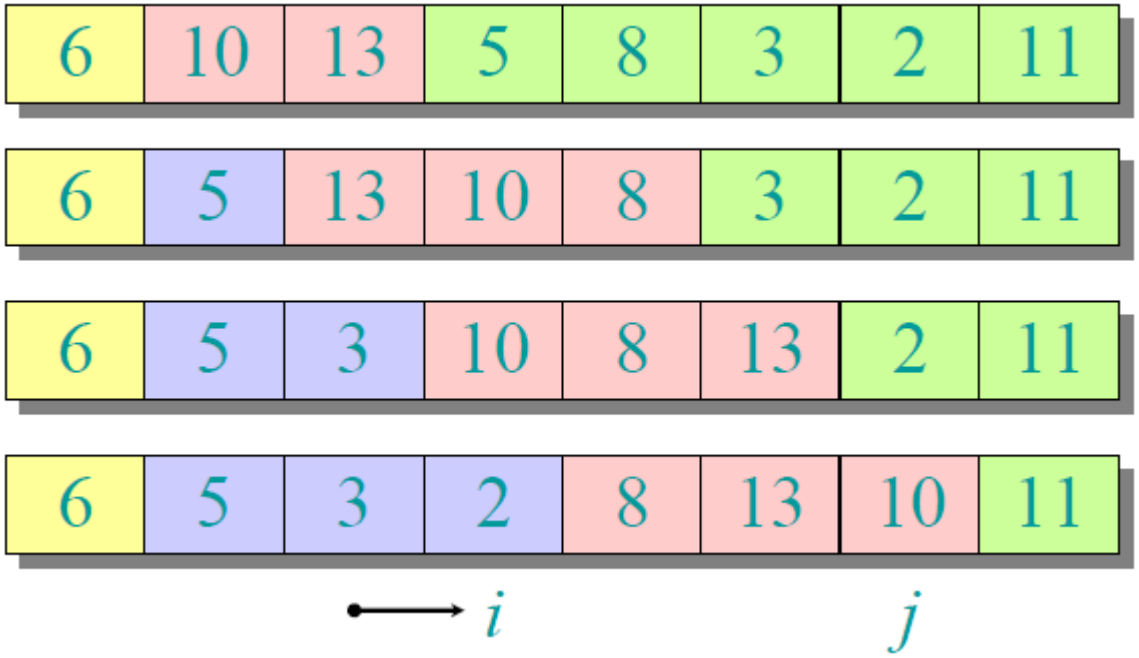

```
PARTITION(A, p, q)  ▷ A[p . . q]
    x ← A[p]        ▷ pivot = A[p]
    i ← p
    for j ← p + 1 to q
        do if A[j] ≤ x
                then  i ← i + 1
                      exchange A[i] ↔ A[j]
    exchange A[p] ↔ A[i]
    return i
```

Example of partitioning



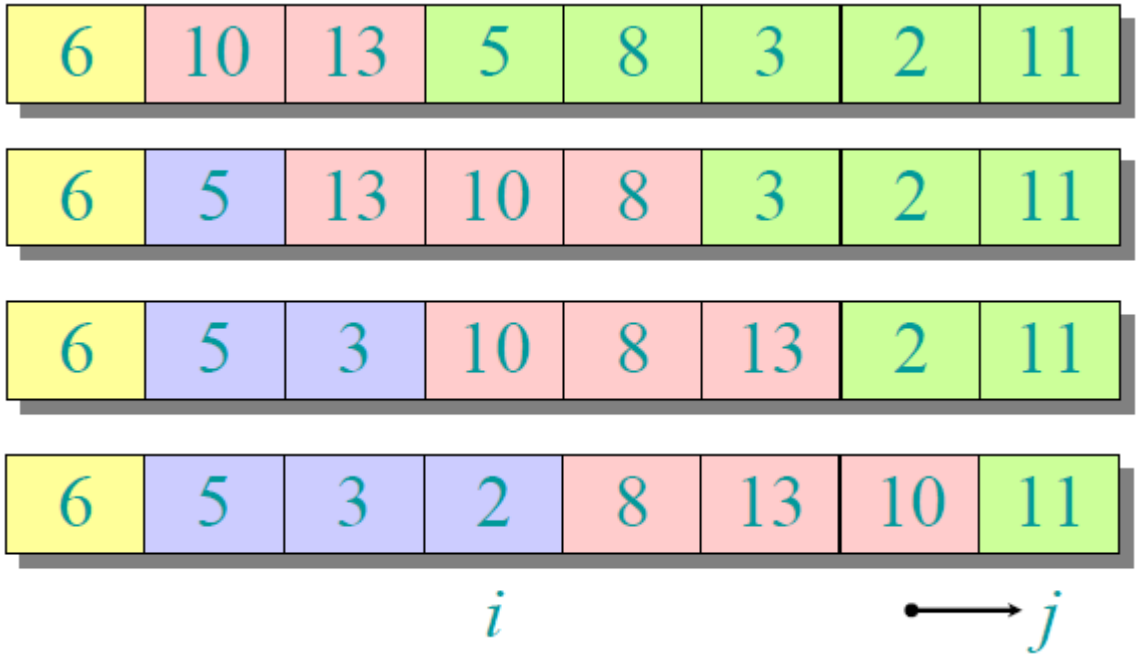

```
PARTITION(A, p, q)  ▷ A[p . . q]
    x ← A[p]        ▷ pivot = A[p]
    i ← p
    for j ← p + 1 to q
        do if A[j] ≤ x
                then  i ← i + 1
                      exchange A[i] ↔ A[j]
    exchange A[p] ↔ A[i]
    return i
```

Example of partitioning



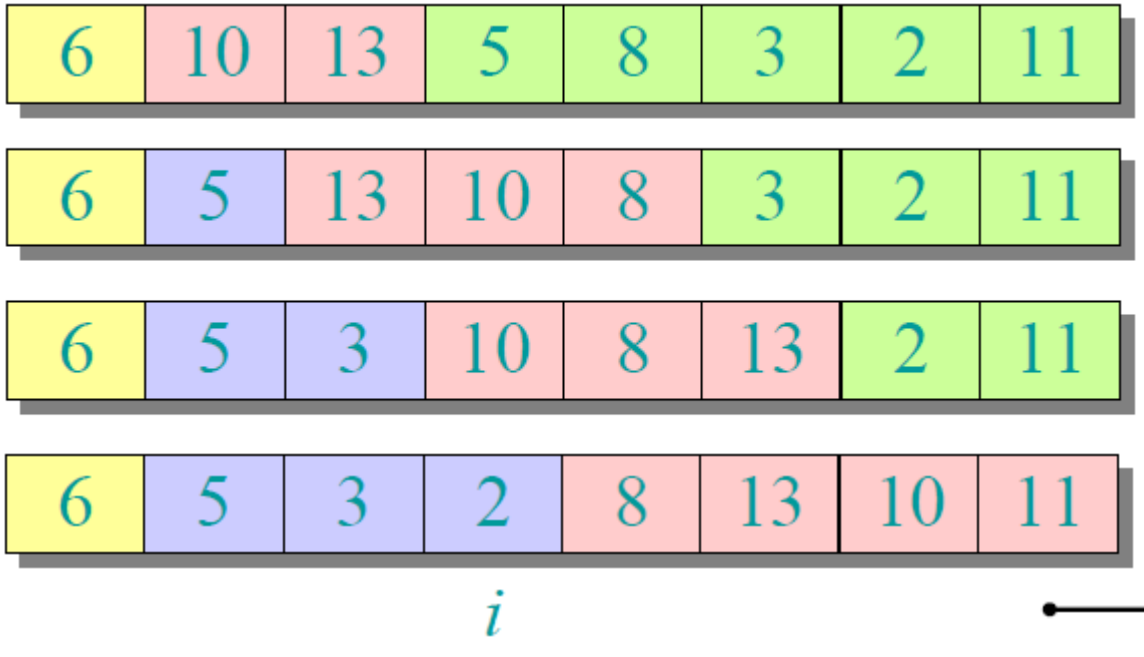

```
PARTITION(A, p, q)  ▷ A[p . . q]
    x ← A[p]        ▷ pivot = A[p]
    i ← p
    for j ← p + 1 to q
        do if A[j] ≤ x
                then  i ← i + 1
                      exchange A[i] ↔ A[j]
    exchange A[p] ↔ A[i]
    return i
```

Example of partitioning



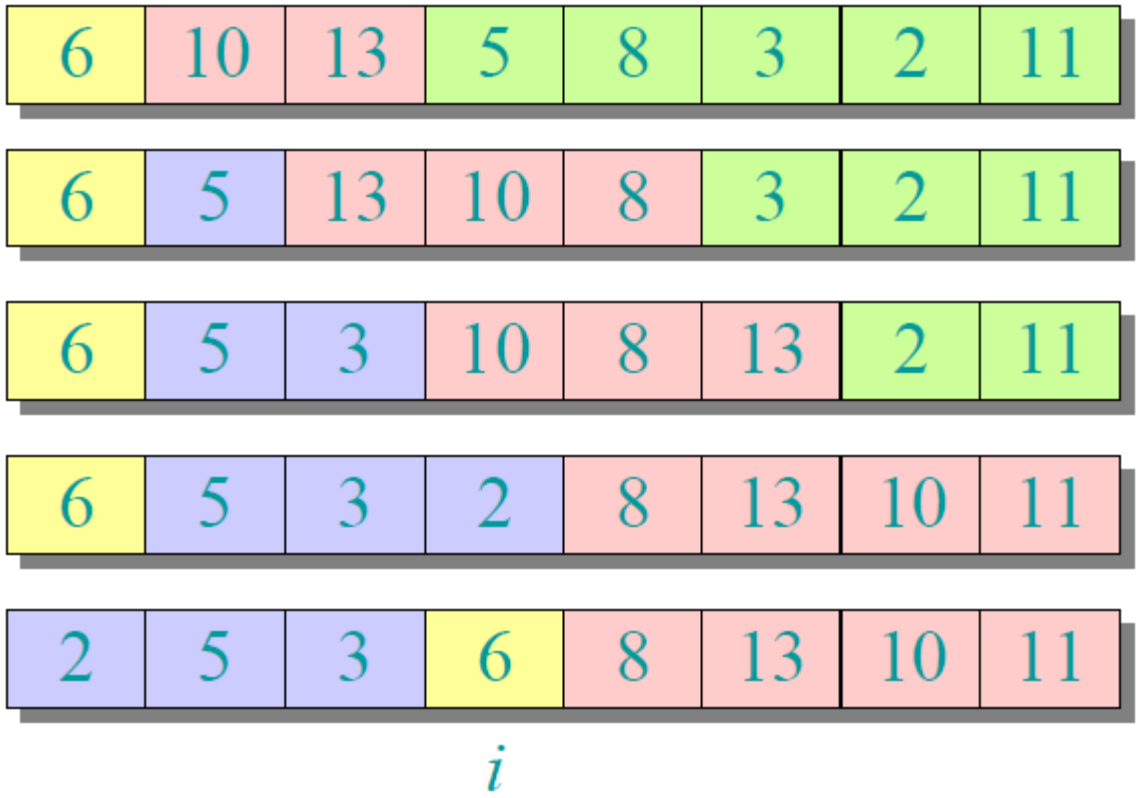

```
PARTITION(A, p, q)  ▷ A[p . . q]
    x ← A[p]        ▷ pivot = A[p]
    i ← p
    for j ← p + 1 to q
        do if A[j] ≤ x
                then  i ← i + 1
                      exchange A[i] ↔ A[j]
    exchange A[p] ↔ A[i]
    return i
```

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
```

Pseudocode for quicksort

```
QUICKSORT(A, p, r)
    if p < r
        then q ← 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.

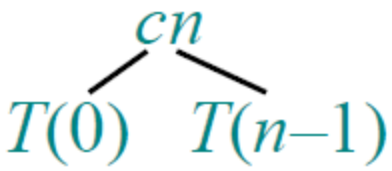
$$
\begin{aligned}
T(n) &= T(0) + T(n-1) + \Theta(n) \\
&= \Theta(1) + T(n-1) + \Theta(n) \\
&= T(n-1) + \Theta(n) \\
&= \Theta(n^2)
\end{aligned}
$$

Worst-case recursion tree

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

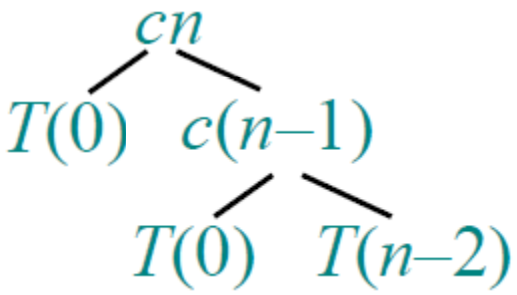
Worst-case recursion tree

$$T(n) = T(0) + T(n–1) + cn$$



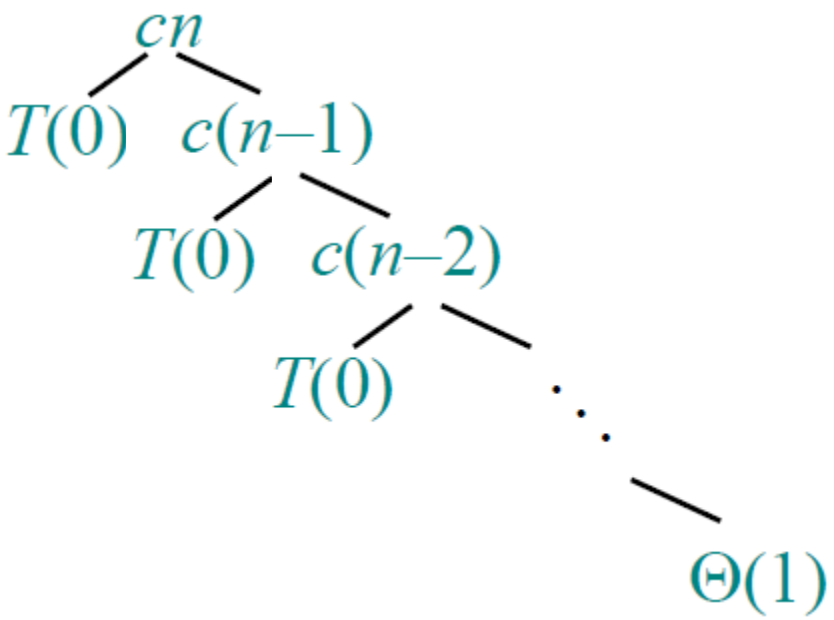
Worst-case recursion tree

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



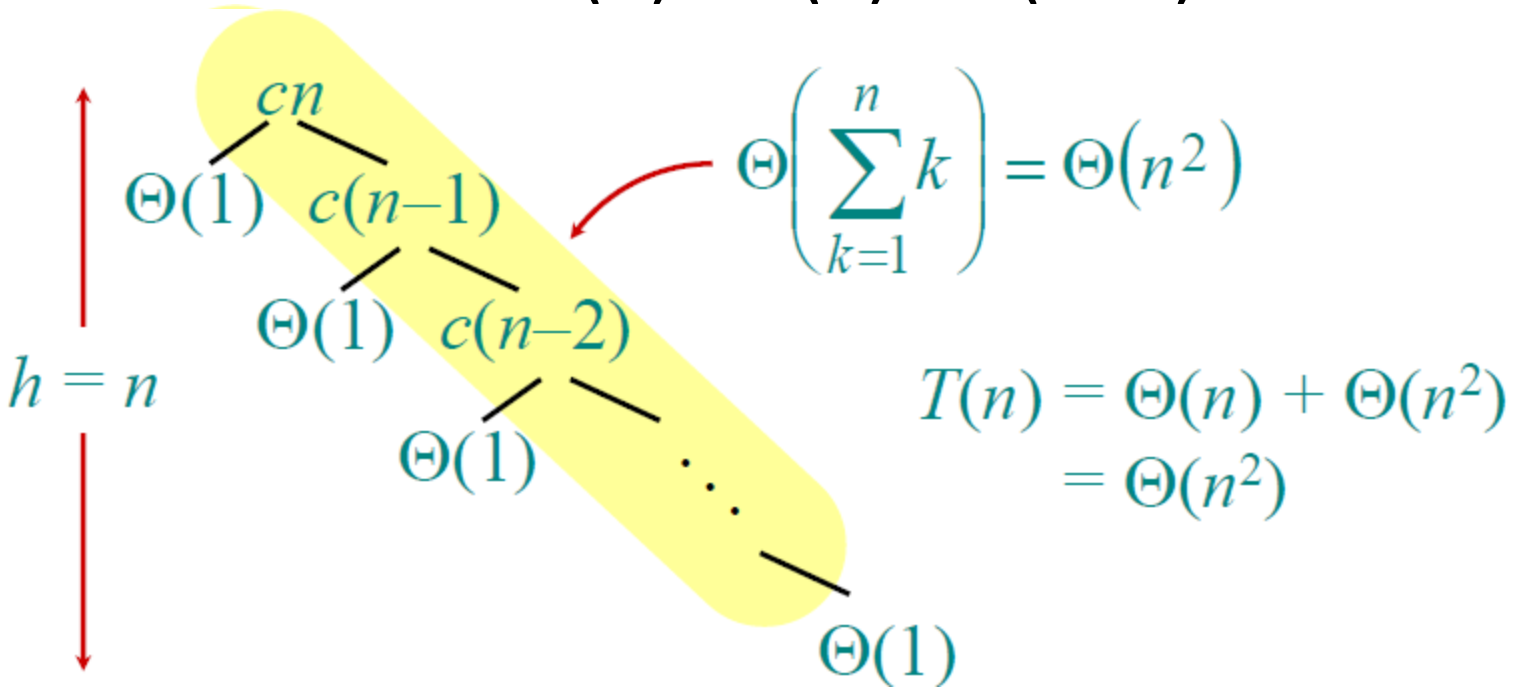
Worst-case recursion tree

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



Worst-case recursion tree

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



Best-case analysis

(For intuition only!)

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

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

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

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

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

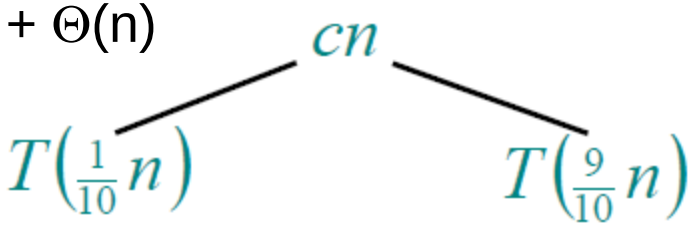
- What is the solution to this recurrence ?

Analysis of “almost-best” case

T(n) = T(n/10) + T(9n/10) + Θ(n) $T(n)$

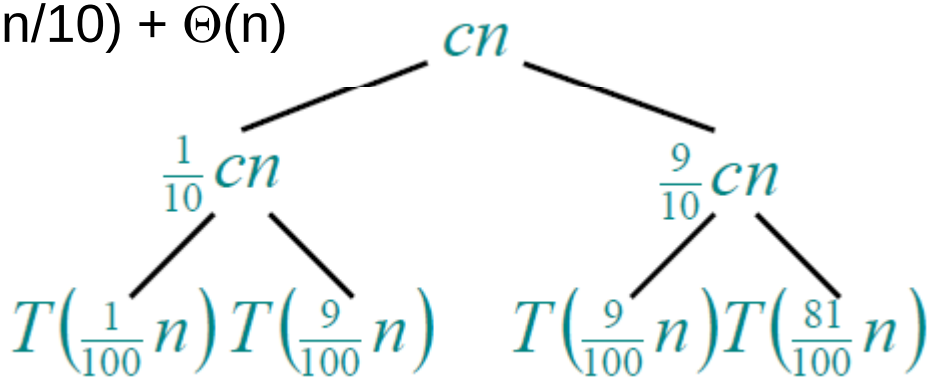
Analysis of “almost-best” case

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



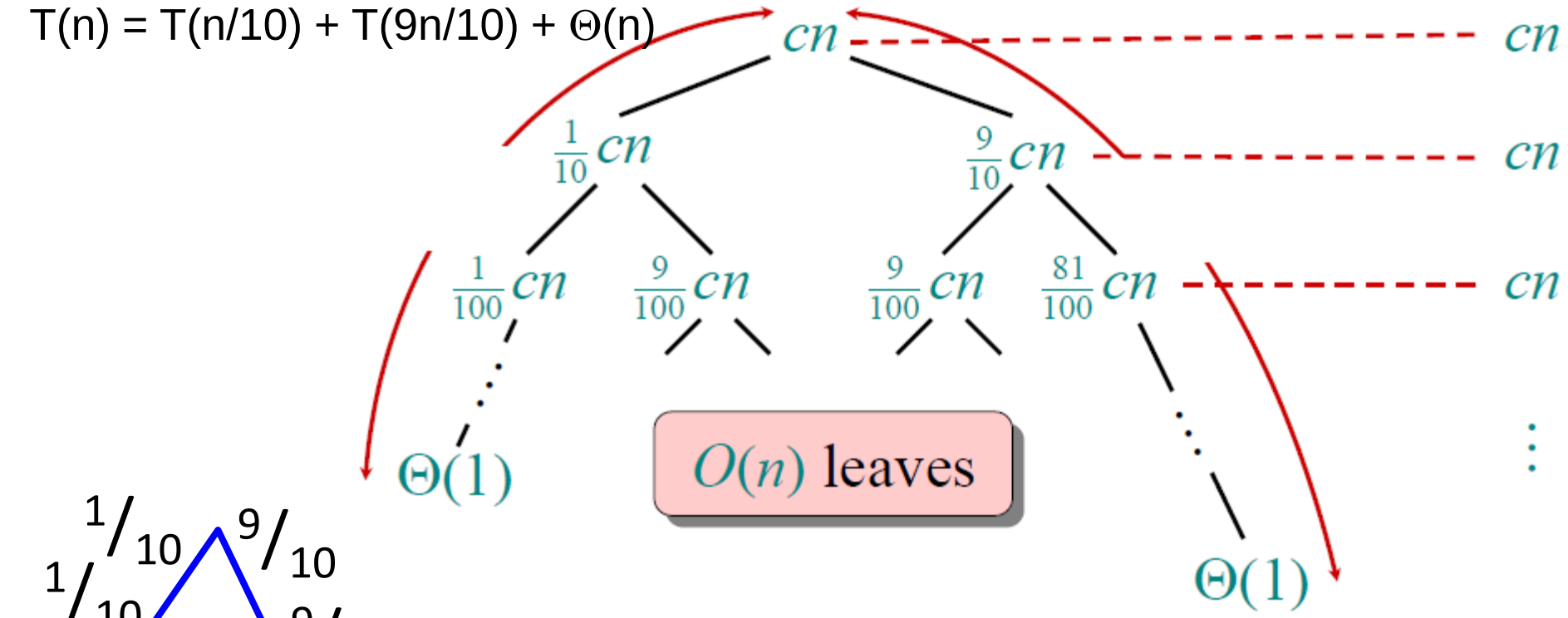
Analysis of “almost-best” case

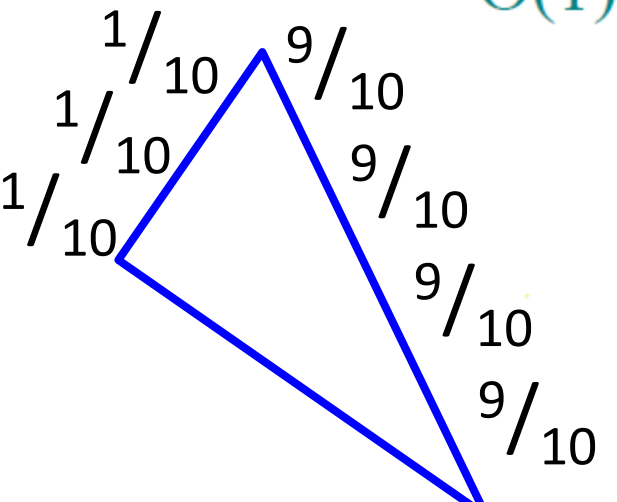
$T(n) = T(n/10) + T(9n/10) + \Theta(n)$

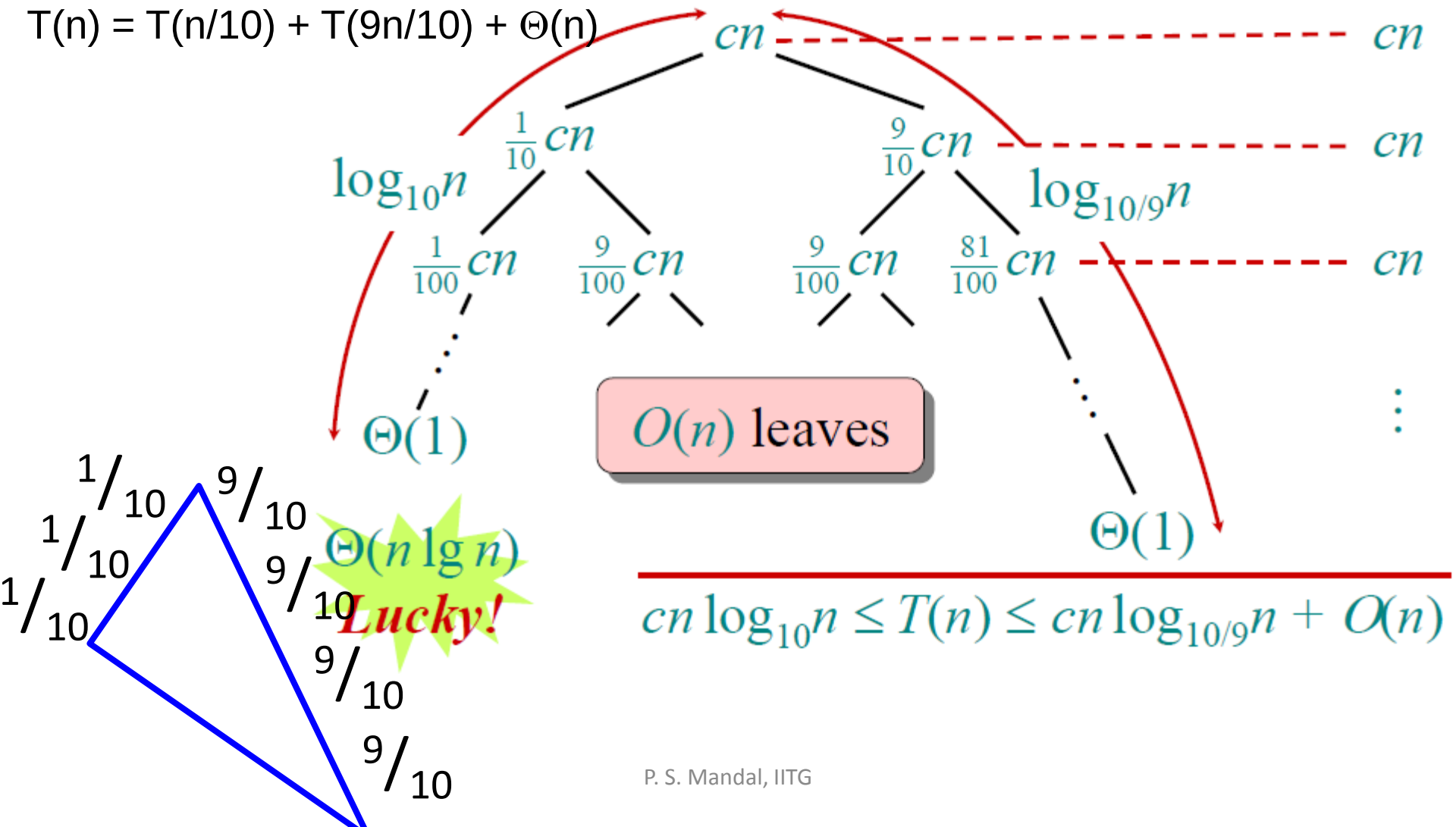


Analysis of “almost-best” case

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





Analysis of “almost-best” case

T(n) = T(n/10) + T(9n/10) + Θ(n)