# *Analysis of Algorithms*
# *CMPSC* 565

## LECTURES 38-39

**Randomized Algorithms II**

- Quickselect
- Quicksort
- Running time

**Adam Smith**

# Types of "randomized" analysis

- "Average-case analysis":
  - Assume data is distributed according to a given distribution
  - Very common in networking applications
  - PROS: understand "typical" behavior
  - CONS: no guarantees if data not as expected
- Randomized worst-case:
  - consider worst possible input
  - but look at expected running time over the choice of "coins"

# Divide and conquer

Quickselect: Input array and target k

***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.

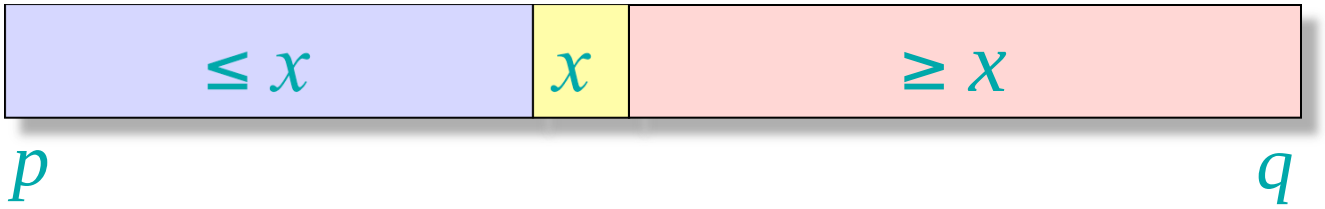

***2.Conquer:*** Recursively search in one of the two subarrays. Position of x tells you where to look.

***3.Combine:*** Nothing.

**Key:** *Linear-time partitioning subroutine*

# PARTITION($A,p,r$)

- Gets array $A$ and bounds $p$, $r$
- Basic idea
  - Use *first element* $A[p]$ of array as pivot
  - Arrange elements in array so that:



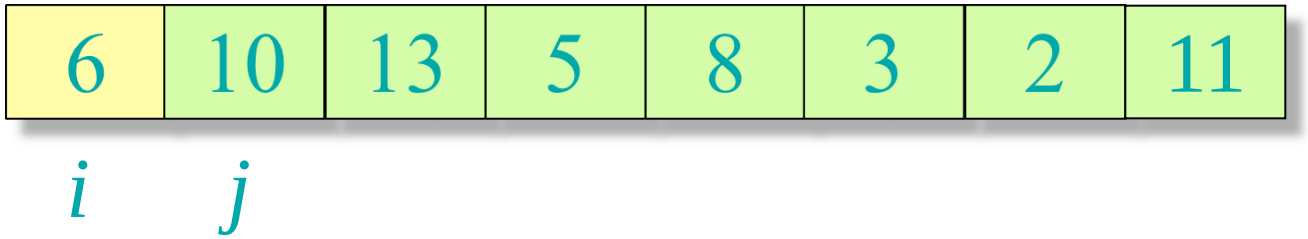

- After partition: Sort left and right pieces recursively

# Partitioning subroutine

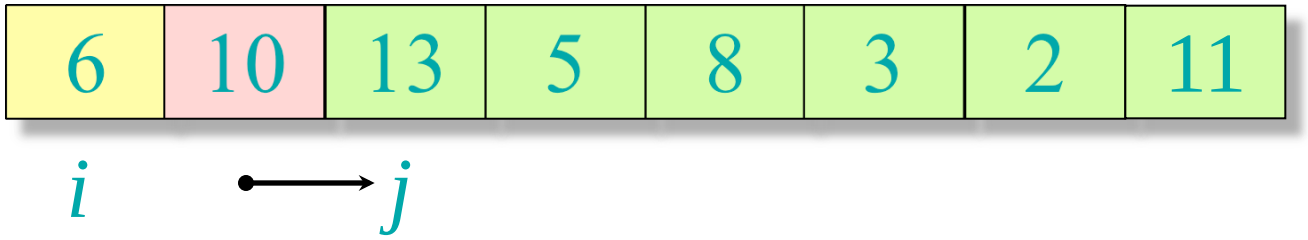
```
PARTITION(A, p, q)  ▷ A[p . . q]
    x ← A[p]        ▷ pivot = A[p]
    i ← p
    for j ← p + 1 to q
        do if A[j] ≤ x
                then  i ← i + 1
                      exchange A[i] ↔ A[j]
    exchange A[p] ↔ A[i]
    return i
```

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

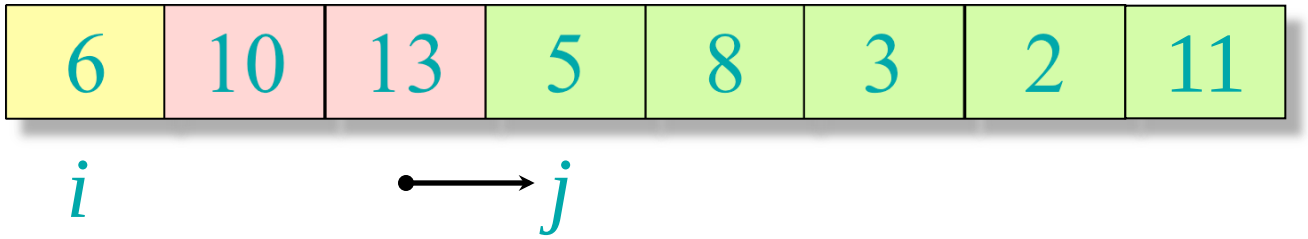
# Example of partitioning



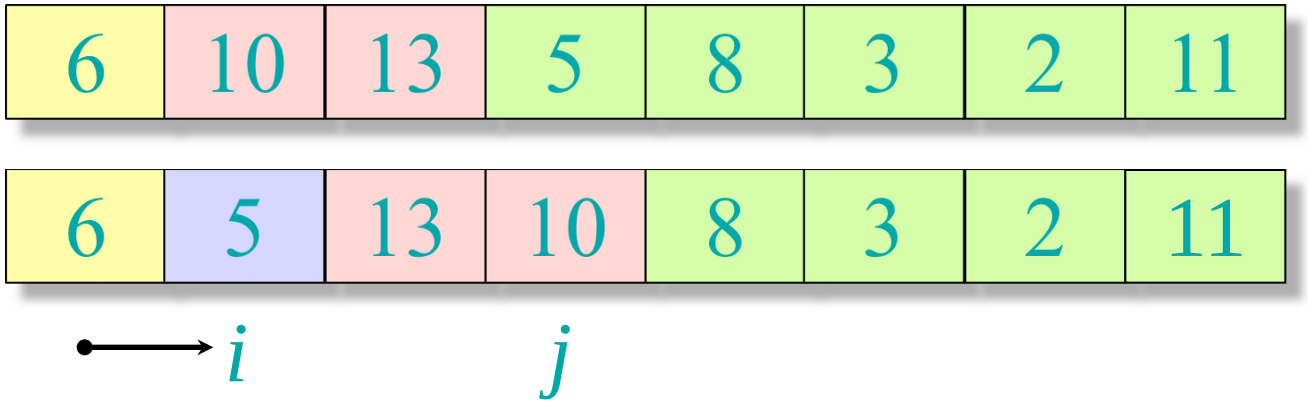
# Example of partitioning



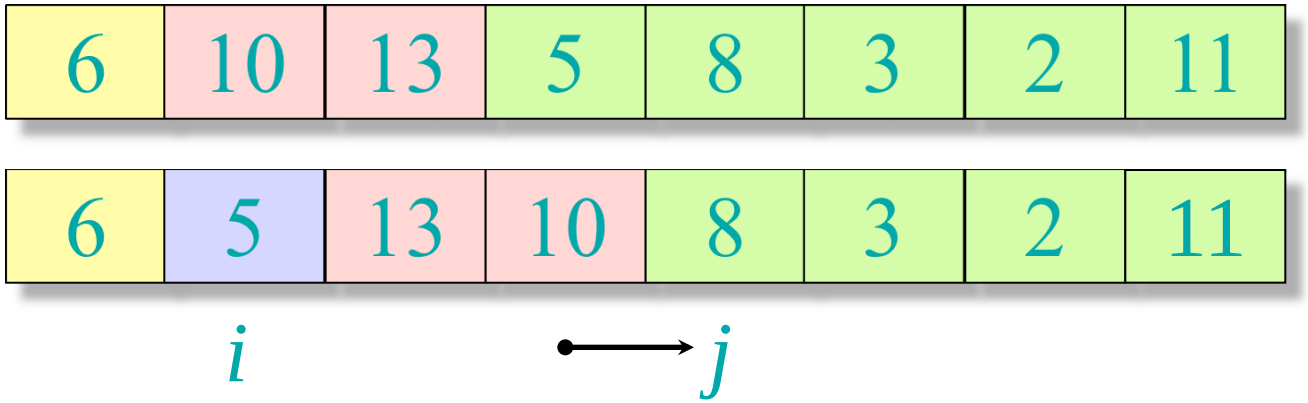
# Example of partitioning



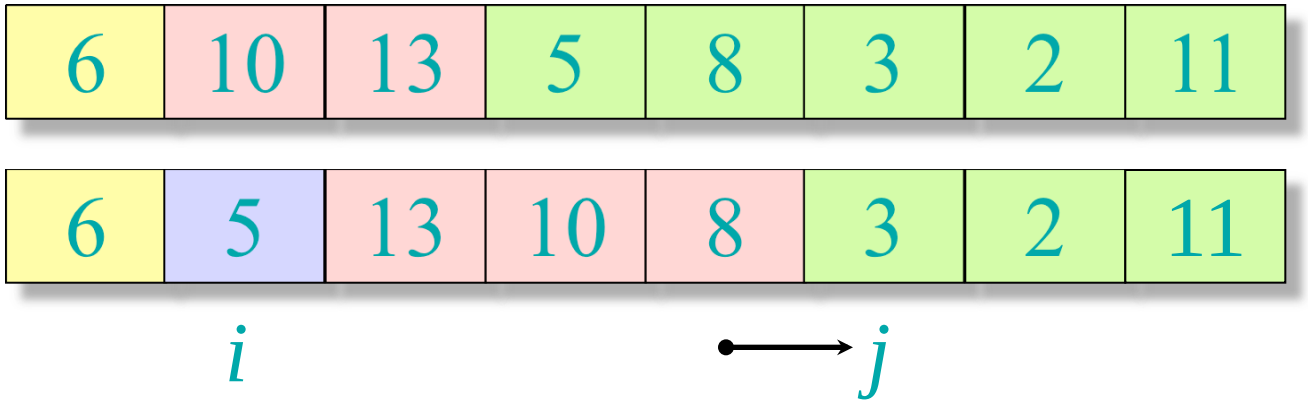
# Example of partitioning



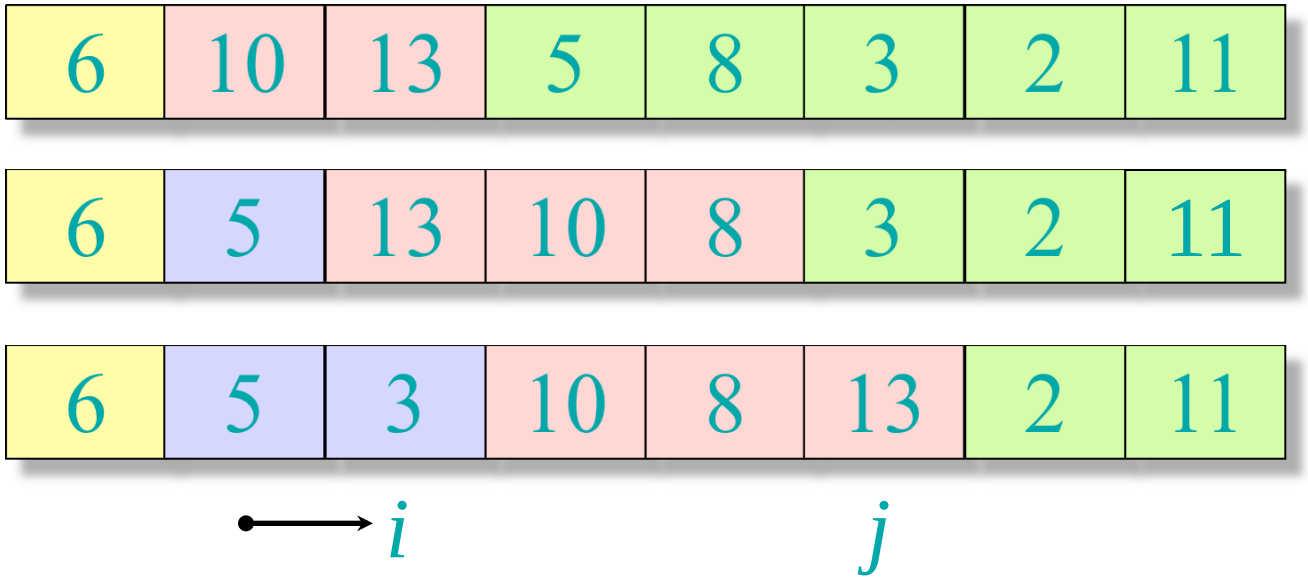
# Example of partitioning



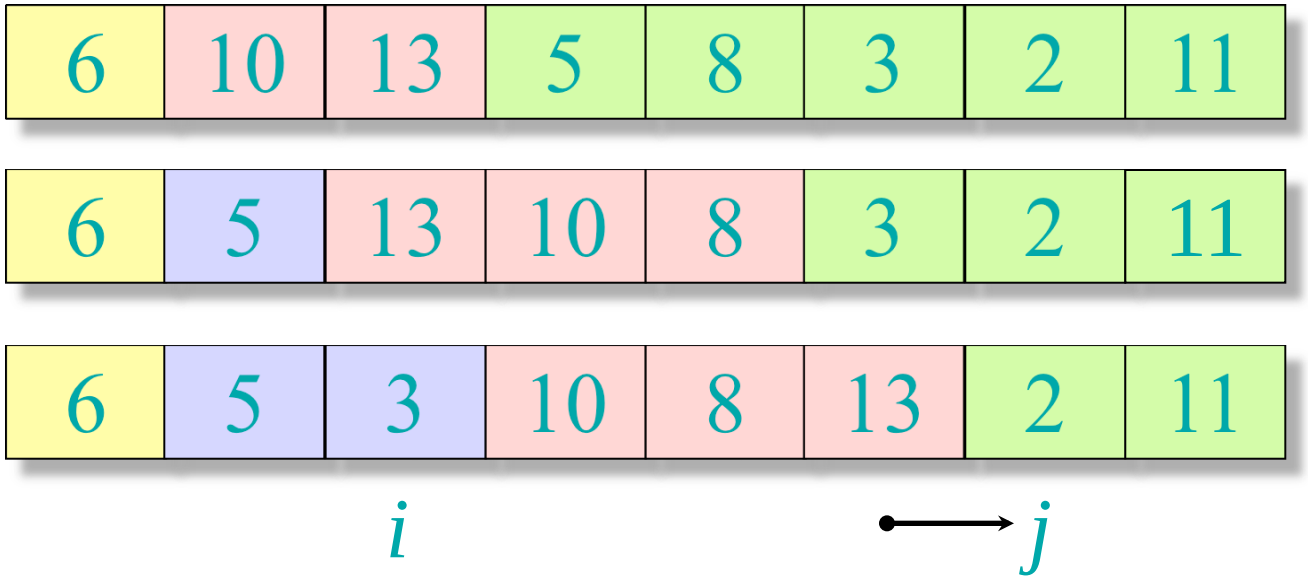
# Example of partitioning



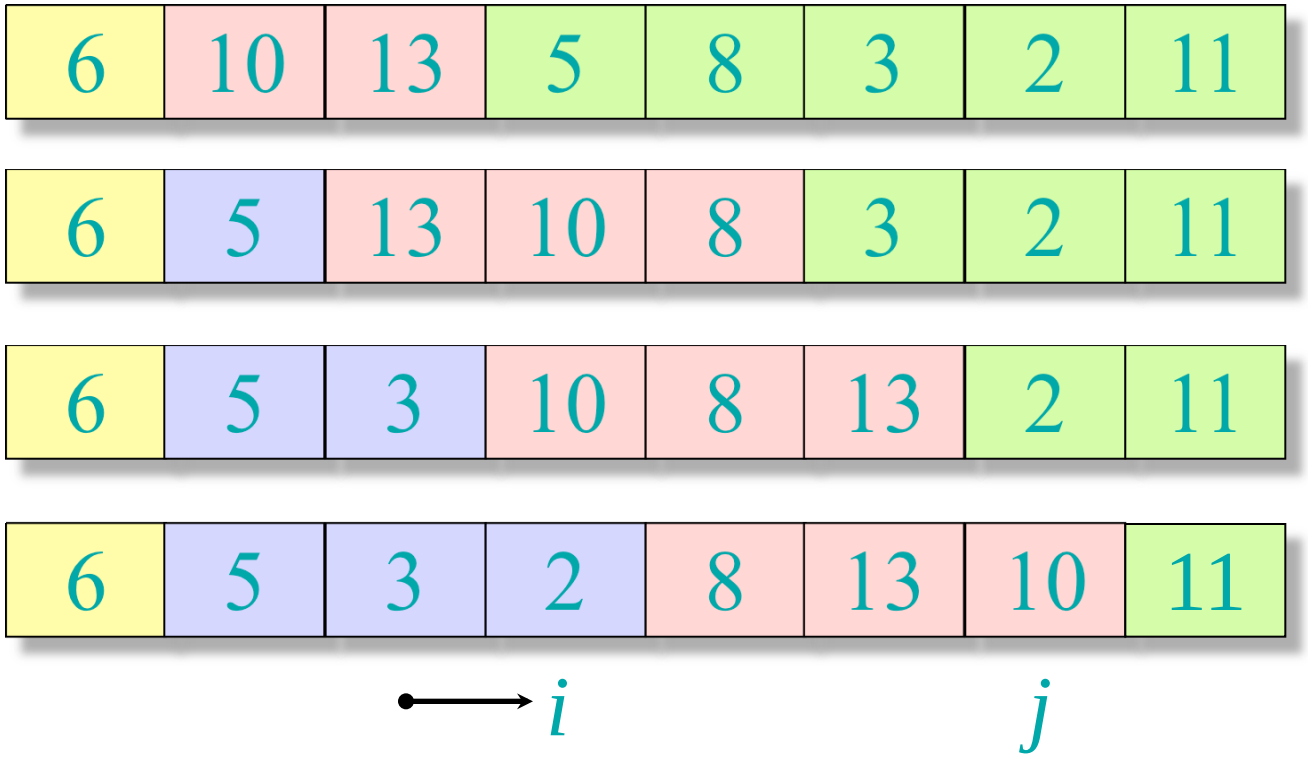
# Example of partitioning



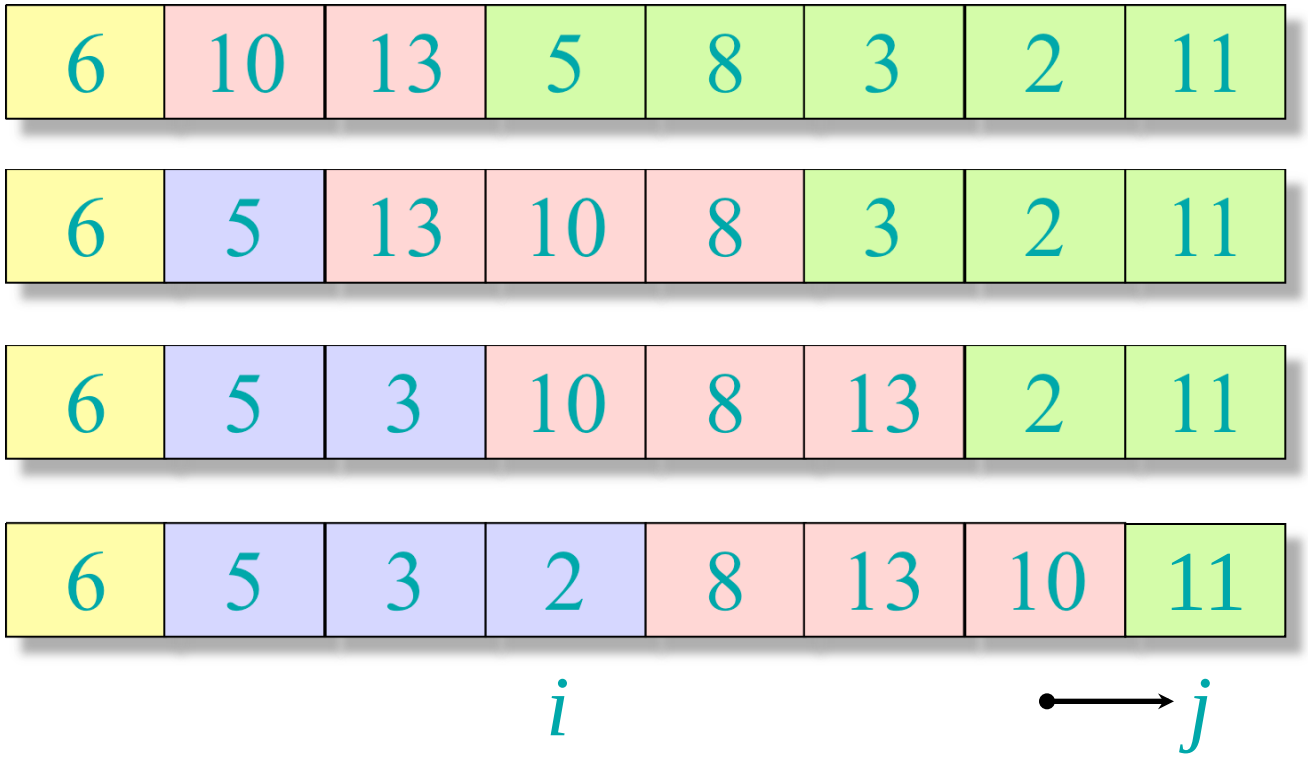
# Example of partitioning



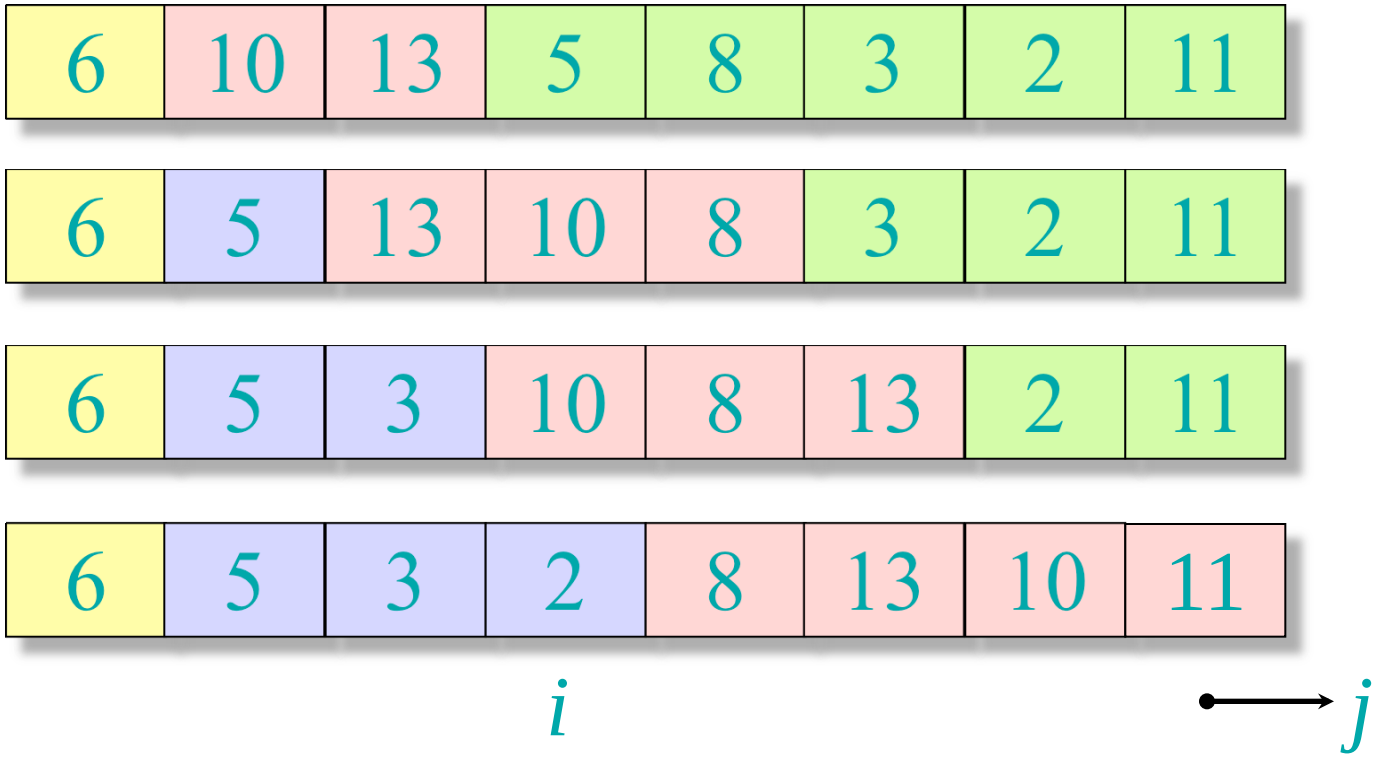
# Example of partitioning



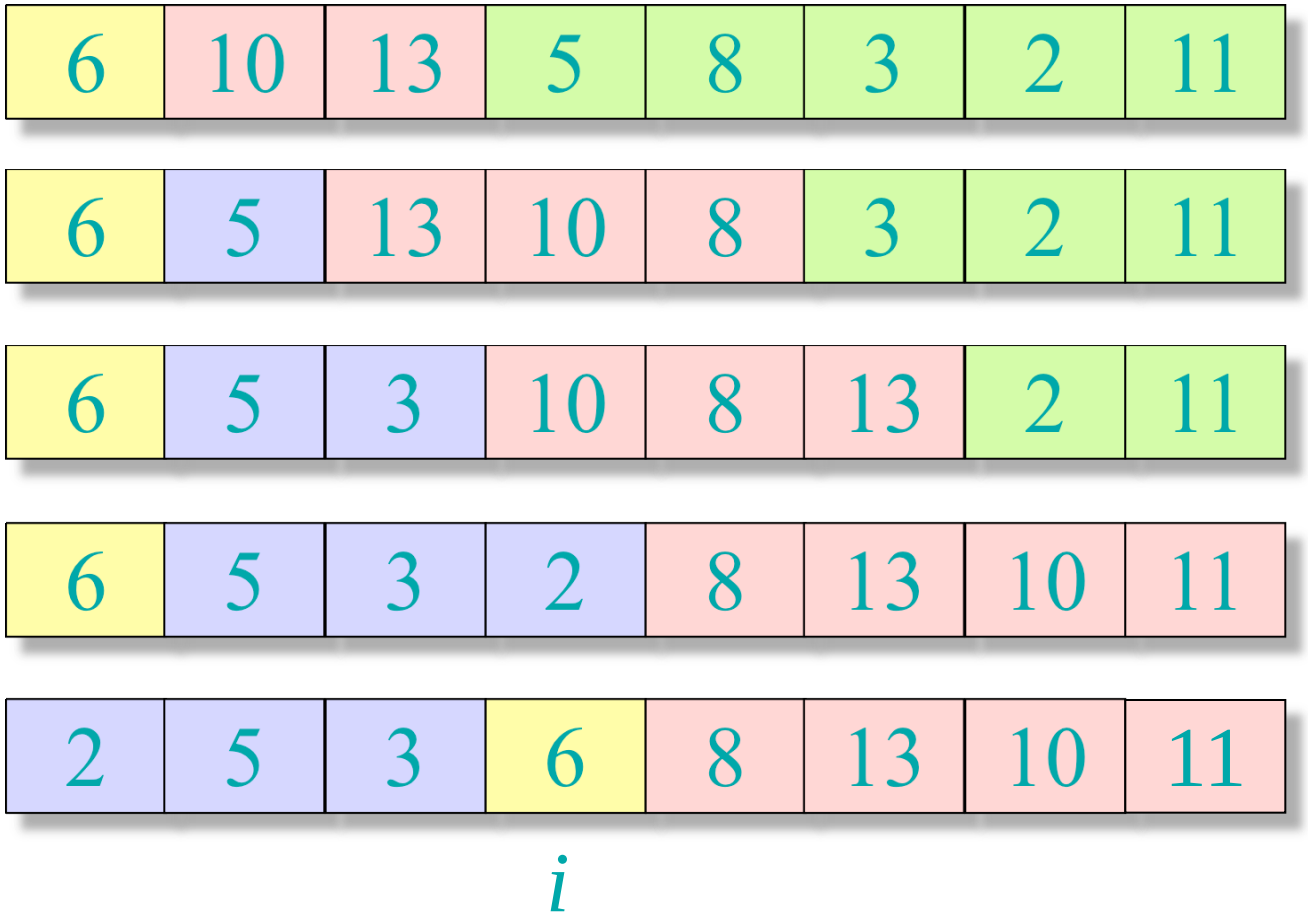
# Example of partitioning



# Example of partitioning

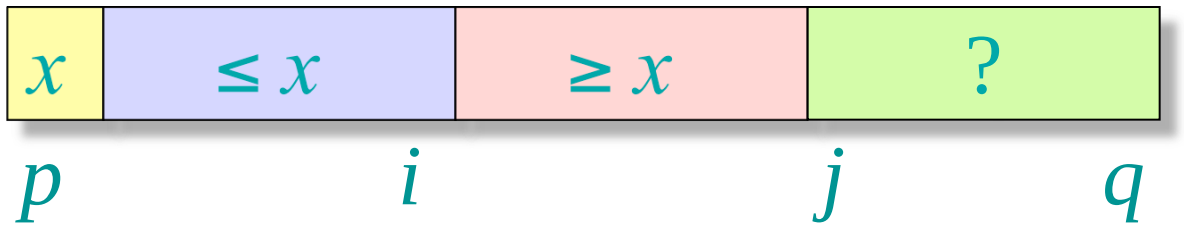


# Example of partitioning



# Invariant for the for loop

- Reminder: an *invariant* is a statement about a particular point in the code. The invariant should always be true when that point is reached (see CLRS Chapter 2).
- Goals of invariants: help *understand* and *explain* what the algorithm is doing, and help *prove* correctness.
- Example: In the PARTITION algorithm, before each execution of the **for** loop: $A[p] = x$,
  $A[p+1], \ldots, A[i] \leq x$, and
  $A[i+1], \ldots, A[j-1] \geq x$



# Quickselect

- Quickselect(A,i,j,k)
  - Pick pivot p at random in A[i,j]
  - Partition around p
  - r = location of p after partition
  - rank = r-p+1
  - If rank = k
    - return p
  - Else if k < rank
    - return Quickselect(A, i, r-1,k)
  - Else
    - return Quickselect(A, r+1, j, k-rank)

# Running time?

- Worst case:
  - Every time, pivot has rank 1
  - Size of subarray decreases by 1 each time
  - $T(n) = T(n-1) + cn$
  - Quadratic time
- Average?
  - If array shrinks by ½ at each stage, $T(n) = T(n/2) + cn$ so $T(n) = O(n)$
- Expectation: Pr(pivot has rank i) = 1/n
  - $T'(n) = E(T(n)) = cn + (T'(1)+\ldots+T'(n))/n$
  - Solves to $T(n) = O(n)$ (by substitution)

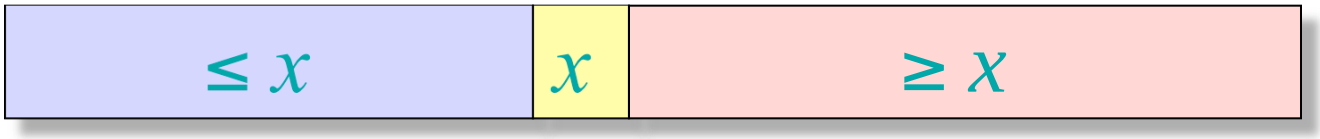
# Quicksort

- 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 (with tuning).

# 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:*** Nothing.

**Key:** *Linear-time partitioning subroutine.*

# Pseudocode for quicksort

```
QUICKSORT(A, p, r)
                ▷ Sort A[p . . q] in place.
  if p < r
    then q ← PARTITION(A, p, r)
         QUICKSORT(A, p, q–1)
         QUICKSORT(A, q+1, r)
```

If $p \geq r$ do nothing

$\Theta(n)$ time

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

# Review Question

- How long does Quicksort run on a sorted array, e.g. [1,2,…,*n*] ?
  - Write down a recurrence
  - Guess a solution.

(Answers in the next few slides)

# 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.

$$
\begin{aligned}
T(n) &= T(0) + T(n-1) + \Theta(n) \\
&= \Theta(1) + T(n-1) + \Theta(n) \\
&= T(n-1) + \Theta(n) \\
&= \Theta(n^2) \qquad \textit{(arithmetic series, as with insertion sort)}
\end{aligned}
$$

# Worst-case recursion tree

$$T(n) = T(0) + T(n–1) + cn$$

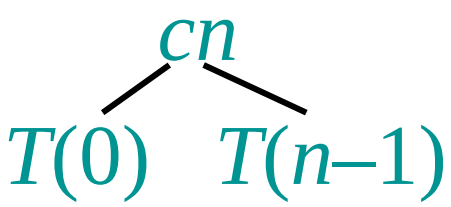
# Worst-case recursion tree

$$T(n) = T(0) + T(n–1) + cn$$

$T(n)$

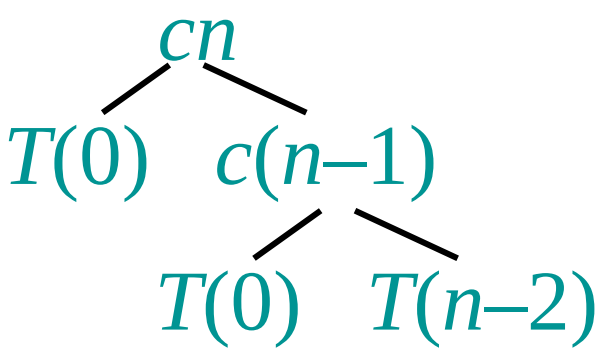
# Worst-case recursion tree

$$T(n) = T(0) + T(n–1) + cn$$



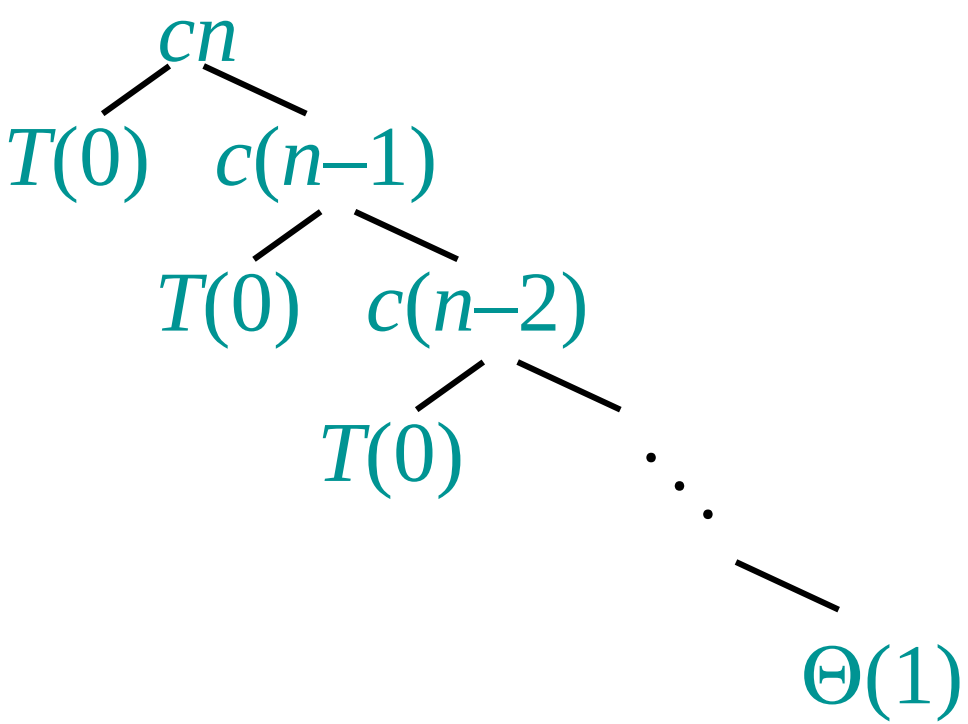
# Worst-case recursion tree

$$T(n) = T(0) + T(n–1) + cn$$



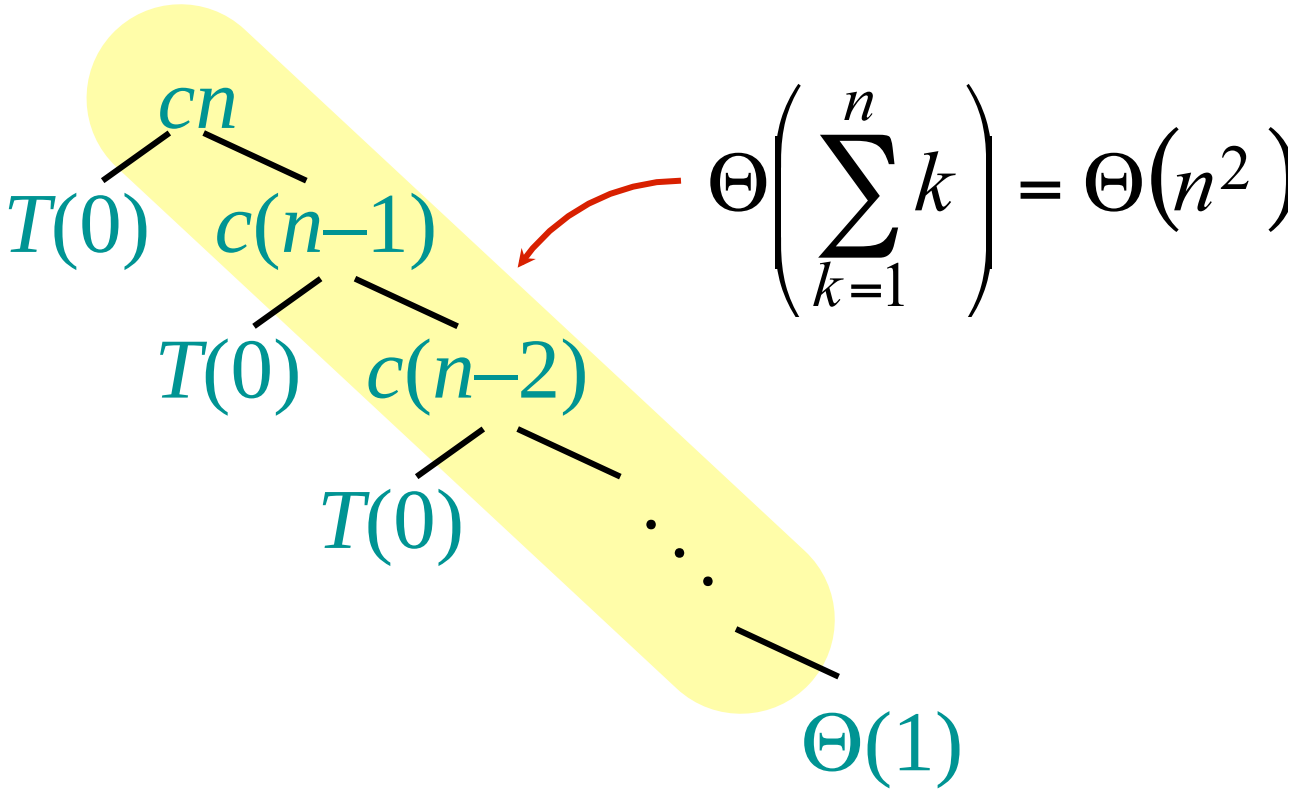
# Worst-case recursion tree

$$T(n) = T(0) + T(n–1) + cn$$



# Worst-case recursion tree

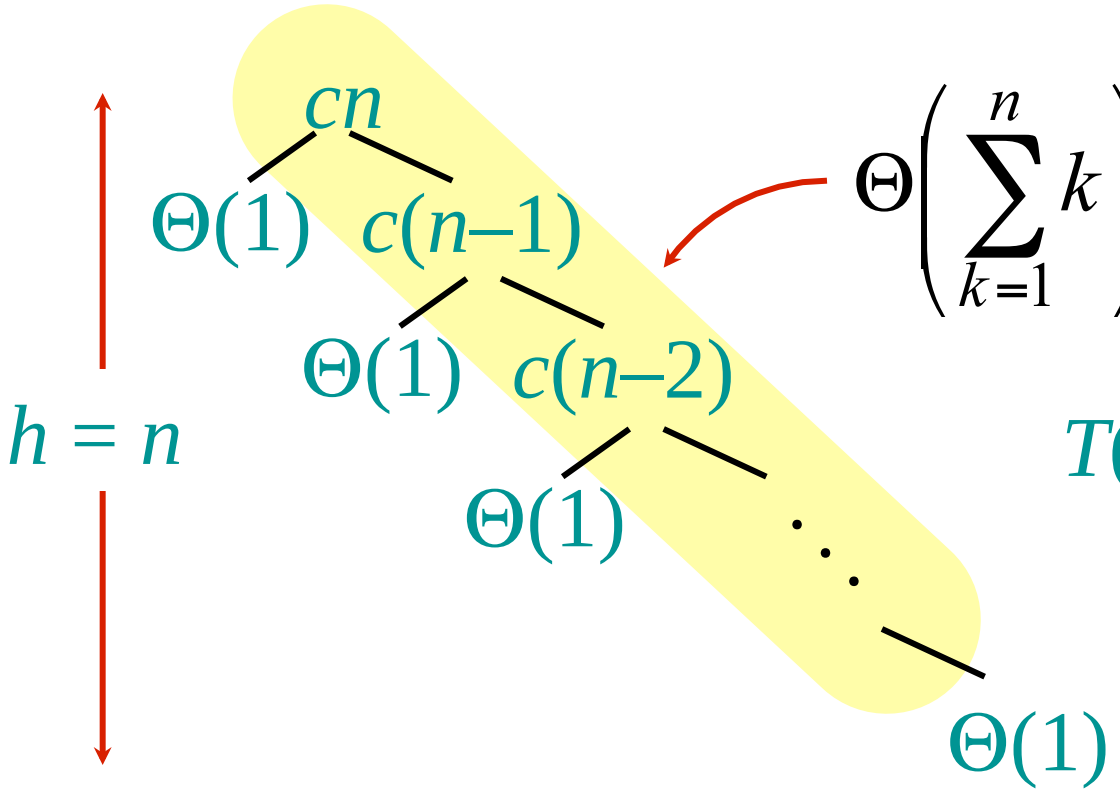
$$T(n) = T(0) + T(n–1) + cn$$




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

# Worst-case recursion tree

$$T(n) = T(0) + T(n–1) + cn$$




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

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

# 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)

# An "OK" case analysis

A split (or pivot) is "OK" if the smaller piece has at least $\lfloor n/4 \rfloor$ elements

What if the split is always OK, i.e. $\frac{1}{4} : \frac{3}{4}$ ?

$$T(n) = T\left(\tfrac{1}{4} n\right) + T\left(\tfrac{3}{4} n\right) + \Theta(n)$$

What is the solution to this recurrence?

$T(n) = \Theta(n \lg n)$ (homework)

# Average Split?

- If input is sorted, every split is bad
- What if input is in **random** order?
  - Quality of split depends on **rank** of first element

$$\text{rank}(x) = \#\{\, i : A[i] < x \}$$

  - $E[$ size of **smaller** subarray $]$

$$= E[\text{ random number between } 0 \text{ and } n/2\ ]$$
$$= n / 4$$

- Problem: inputs are not typically random

# Randomized Quicksort

**BIG IDEA**: Partition around a ***random*** element.

- Running time is independent of the input order.
- No assumptions need to be made about the input distribution.
- No specific input elicits the worst-case behavior.
- The worst case is determined only by the output of a random-number generator.

# Pseudocode for quicksort

```
RQUICKSORT(A, p, r)
                ▷ Sort A[p . . r] in place.
  if p < r
     then t ← RANDOM(p,r)
          Exchange A[p] ↔ A[t]
          q ← PARTITION(A, p, r)
          RQUICKSORT(A, p, q–1)
          RQUICKSORT(A, q+1, r)
```

Swap random element to be pivot

**Initial call:** RQUICKSORT(*A*, 1, *n*)

# Random Binary Search Trees

- Suppose I want to insert n keys into a binary search tree
  - Suppose keys are 1, ..., n
  - Insert in ascending order ➔ tree of depth n
  - Insert in random order ➔ ?
- Two questions:
  - # comparisons for quicksort with randomized pivot?
  - # comparisons to build random binary search tree?
- Same quantity!
  - Same distribution, expectation, variance, etc

# Quicksort, Analysis 1

See, e.g. Erickson Notes

# Expected Running Time?

- Indicator variable: random variable that is either 1 or 0
  - Note: Every element is eventually a pivot
  - Let $X_{ij}$ = 1 if element of rank j gets compared to i when i is the pivot
    0 otherwise
  - $E(X_{ij}) = Pr(X_{ij}=1) = ?$
    - Comparison happens only if i is chosen as a pivot before every element in {i+1,…j} if i < j (or {j,…,i-1} if j<i)
    - so $Pr(X_{ij}=1) = 1/|j-i+1|$

# Expected Running Time, Cont'd

- So $T'(n) = E(\Sigma_{ij} X_{ij})$
- Let $H_n$ be the nth Harmonic number,
  $H_n = 1+1/2+1/3+\ldots+1/n$
- Consider a single value i:
  - $\Sigma_j X_{ij} = H_{i-1} + H_{n-i}$
- So $T'(n) = \Sigma_i (H_{i-1} + H_{n-i}) = \Theta(n \log n)$

# Quicksort, Analysis 2

Also bounds max. depth of recursion tree

# Randomized quicksort analysis

- Look at recursion tree for a given run of the algorithm
  - depends on random choices; some splits are good, some are not
  - no matter what, total work at each level is $\leq \boldsymbol{cn}$ (since the sum of lengths of subarrays at each level is at most n)
  - So it suffices to show that the height is small
- What is the height?
  - We get to a leaf when one element is alone in its subarray
  - How deep does this typically occur?
  - Sufficient to get $\log_{4/3}(n)$ "ok" splits
  - Each is "ok" w.p. ½ independently of previous splits
  - Expect to wait 2 splits to see a good one…

# Chernoff Bound

- **Theorem**: Let
  - $X_1,\ldots, X_k$ be independent random variables in [0,1]
  - $S = X_1 + \ldots + X_k$
  - ( Note that $E(S) = E(X_1) + \ldots + E(X_k)$ )

  Then we have: $\Pr(|S-E(S)| > t) < \exp(-2t^2/k)$

# Randomized quicksort analysis

- **Fact**: The probability that $10\log_{4/3}(n)$ fair coin flips will produce fewer than $\log_{4/3}(n)$ heads is $\leq 1/n^2$
  - Follows from Chernoff bound
- Let $E_i$ be event the leaf i is at depth $>10\log_{4/3}(n)$
- Probability that some leaf at depth more than $10\log_{4/3}(n)$ is:
  - $\Pr(\cup_i E_i) \leq \Pr(E_1) + \Pr(E_2) + \ldots \Pr(E_n)$
    $= (1/n^2) + \ldots (1/n^2) = n(1/n^2) = 1/n$
- So for some ***c*** (depending on partition subroutine), the probability that the running time exceeds $\boldsymbol{cn}\log(\boldsymbol{n})$ is at most $\boldsymbol{1/n}$

# Randomized Quicksort

- We have proved
  **Theorem:** There is a constant ***c*** such that for every input of length ***n***, with probability at least **1-1/*n*** over the random choices of the algorithm, Randomized Quicksort runs in time at most ***cn*log(*n*)**.
- A direct consequence ("corollary") is:
  **Corollary:** The expectation of the running time of Randomized Quicksort is **O(*n*log(*n*))**.

# Quicksort, Analysis 3

Attack recurrence directly

Based on CLRS

# Randomized quicksort analysis

Let $T(n)$ = the **random variable** for the running time of randomized quicksort on an input of size $n$, assuming random numbers are independent.

For $k = 0, 1, \ldots, n-1$, define the ***indicator random variable***

$$X_k = \begin{cases} 1 & \text{if PARTITION generates a } k : n-k-1 \text{ split,} \\ 0 & \text{otherwise.} \end{cases}$$

$E[X_k] = \Pr\{X_k = 1\} = 1/n$, since all splits are equally likely, assuming elements are distinct.

# Analysis (continued)

$$T(n) = \begin{cases} T(0) + T(n-1) + \Theta(n) & \text{if } 0 : n-1 \text{ split,} \\ T(1) + T(n-2) + \Theta(n) & \text{if } 1 : n-2 \text{ split,} \\ \vdots & \\ T(n-1) + T(0) + \Theta(n) & \text{if } n-1 : 0 \text{ split,} \end{cases}$$

$$= \sum_{k=0}^{n-1} X_k \left( T(k) + T(n-k-1) + \Theta(n) \right)$$

- Big Idea #2: Analyze the expected value of T(n)

# Calculating expectation

$$E[T(n)] = E\left[\sum_{k=0}^{n-1} X_k \left(T(k) + T(n-k-1) + \Theta(n)\right)\right]$$

Take expectations of both sides.

# Calculating expectation

$$E[T(n)] = E\left[\sum_{k=0}^{n-1} X_k \left(T(k) + T(n-k-1) + \Theta(n)\right)\right]$$

$$= \sum_{k=0}^{n-1} E\left[X_k \left(T(k) + T(n-k-1) + \Theta(n)\right)\right]$$

Linearity of expectation.

# Calculating expectation

$$
\begin{aligned}
E[T(n)] &= E\left[\sum_{k=0}^{n-1} X_k \left(T(k) + T(n-k-1) + \Theta(n)\right)\right] \\
&= \sum_{k=0}^{n-1} E\left[X_k \left(T(k) + T(n-k-1) + \Theta(n)\right)\right] \\
&= \sum_{k=0}^{n-1} E\left[X_k\right] \cdot E\left[T(k) + T(n-k-1) + \Theta(n)\right]
\end{aligned}
$$

Independence of $X_k$ from other random choices.

# Calculating expectation

$$
\begin{aligned}
E[T(n)] &= E\left[\sum_{k=0}^{n-1} X_k \left(T(k) + T(n-k-1) + \Theta(n)\right)\right] \\
&= \sum_{k=0}^{n-1} E\left[X_k \left(T(k) + T(n-k-1) + \Theta(n)\right)\right] \\
&= \sum_{k=0}^{n-1} E\left[X_k\right] \cdot E\left[T(k) + T(n-k-1) + \Theta(n)\right] \\
&= \frac{1}{n}\sum_{k=0}^{n-1} E\left[T(k)\right] + \frac{1}{n}\sum_{k=0}^{n-1} E\left[T(n-k-1)\right] + \frac{1}{n}\sum_{k=0}^{n-1} \Theta(n)
\end{aligned}
$$

Linearity of expectation; $E[X_k] = 1/n$ .

# Calculating expectation

$$
\begin{aligned}
E[T(n)] &= E\left[\sum_{k=0}^{n-1} X_k \left(T(k) + T(n-k-1) + \Theta(n)\right)\right] \\
&= \sum_{k=0}^{n-1} E\left[X_k \left(T(k) + T(n-k-1) + \Theta(n)\right)\right] \\
&= \sum_{k=0}^{n-1} E\left[X_k\right] \cdot E\left[T(k) + T(n-k-1) + \Theta(n)\right] \\
&= \frac{1}{n}\sum_{k=0}^{n-1} E\left[T(k)\right] + \frac{1}{n}\sum_{k=0}^{n-1} E\left[T(n-k-1)\right] + \frac{1}{n}\sum_{k=0}^{n-1} \Theta(n) \\
&= \frac{2}{n}\sum_{k=1}^{n-1} E\left[T(k)\right] + \Theta(n)
\end{aligned}
$$

Summations have identical terms.

# A recurrence on expected value

- We want to know the asymptotc behaviour of

$$F(n) = E[T(n)]$$

real number

random variable

- We know it satisfies

$$F(n) = \frac{2}{n}\sum_{k=2}^{n-1} F(k) + \Theta(n)$$

# Hairy recurrence

$$F(n) = \frac{2}{n}\sum_{k=2}^{n-1} F(k) + \Theta(n)$$

(The $k = 0, 1$ terms can be absorbed in the $\Theta(n)$.)

Prove: $F(n) \le a\,n \lg n$ for constant $a > 0$ .

- Choose $a$ large enough so that $a\,n \lg n$ dominates $E[T(n)]$ for sufficiently small $n \ge 2$.

Use fact: $\sum_{k=2}^{n-1} k \lg k \le \frac{1}{2} n^2 \lg n - \frac{1}{8} n^2$ (exercise).

# Substitution method

$$F(n) \leq \frac{2}{n} \sum_{k=2}^{n-1} ak \lg k + \Theta(n)$$

Substitute inductive hypothesis.

# Substitution method

$$F(n) \le \frac{2}{n}\sum_{k=2}^{n-1} ak \lg k + \Theta(n)$$

$$\le \frac{2a}{n}\left(\frac{1}{2}n^2 \lg n - \frac{1}{8}n^2\right) + \Theta(n)$$

Use fact.

# Substitution method

$$F(n) \le \frac{2}{n}\sum_{k=2}^{n-1} ak \lg k + \Theta(n)$$

$$\le \frac{2a}{n}\left(\frac{1}{2}n^2 \lg n - \frac{1}{8}n^2\right) + \Theta(n)$$

$$= an \lg n - \left(\frac{an}{4} - \Theta(n)\right)$$

Express as *desired – residual*.

# Substitution method

$$F(n) \le \frac{2}{n}\sum_{k=2}^{n-1} ak \lg k + \Theta(n)$$

$$= \frac{2a}{n}\left(\frac{1}{2}n^2 \lg n - \frac{1}{8}n^2\right) + \Theta(n)$$

$$= an \lg n - \left(\frac{an}{4} - \Theta(n)\right)$$

$$\le an \lg n$$

if $a$ is chosen large enough so that $an/4$ dominates the $\Theta(n)$.

# Quicksort, Analysis 4

See Kleinberg-Tardos
(no notes on this)

# Quicksort in practice

- Quicksort is a great general-purpose sorting algorithm.
- Quicksort is typically over twice as fast as merge sort.
- Quicksort can benefit substantially from *code tuning*.
- Quicksort behaves well even with caching and virtual memory.

# Background: Properties of the Expectation

Here are some useful properties of the expectation:

- For any random variables $A$, $B$, and constants $c$,$d$:

$$E[cA+dB] = c\,E[\mathrm{A}] + d\,E[B]$$

- For any two **independent** random variables

$$E[AB] = E[A]\,E[B]$$

- If $A$ only takes the values 0 and 1 then

$$E[A] = \mathrm{Prob}[\ A=1\ ]$$

# Background: Geometric Random Variables

- Suppose we flip a coin many times independently, and each time the probability that it comes up "heads" is $p$
  - If $X$ is the number of times we flip the coin before getting heads, then $X$ is a geometric random variable with parameter $p$
- *E.g.*
  - "How many times do we roll a die until we see a 1"? (Geometric with $p = 1/6$.)
  - "How many times do we flip a fair (50:50) coin before getting heads?" (Geometric with $p = ½$.)
  - "How many pivots do we choose before seeing an OK split?" (Geometric with $p = ½$.)
- For geometric r.v. with parameter $p$, $\boldsymbol{E(X) = 1/p}$