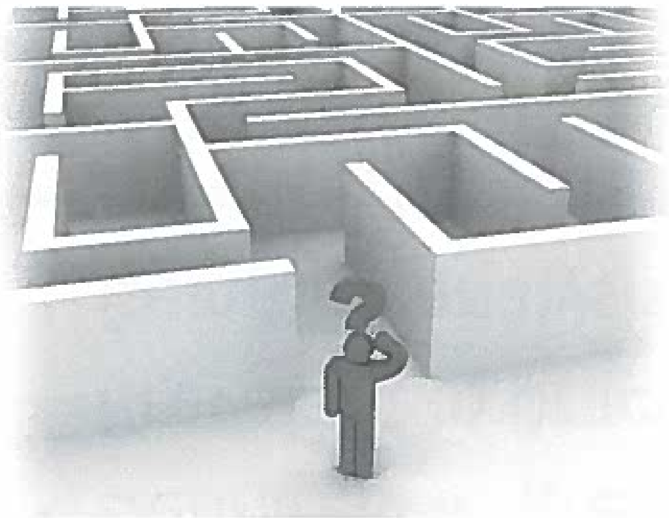
$\frac{5}{24}$

July 22, 18

Week 3



Design and Analysis
of Algorithms I

# QuickSort

# Overview

Overview - 1

We had MergeSort algo. Now quick sort

# QuickSort

- Definitely a "greatest hit" algorithm
- Prevalent in practice
- Beautiful analysis
- $O(n \log n)$ time "on average", works in place

  constants on big-oh notations are small.

  important — i.e., minimal extra memory needed
- See course site for optional lecture notes

Tim Roughgarden

# The Sorting Problem

Input : array of n numbers, unsorted

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 3 | 8 | 2 | 5 | 1 | 4 | 7 | 6 |

Output : Same numbers, sorted in increasing order

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |

Assume : all array entries distinct. (No duplicates).

**Exercise : extend QuickSort to handle duplicate entries**

Tim Roughgarden

overview 2

# Partitioning Around a Pivot

Key Idea : partition array around a pivot element. how to choose this element? later.

-Pick element of array

| 3 | 8 | 2 | 5 | 1 | 4 | 7 | 6 |
|---|---|---|---|---|---|---|---|

pivot

main idea: the meaning of partitioning

-Rearrange array so that

-Left of pivot => less than pivot

-Right of pivot => greater than pivot

final position of sorted array

| 2 | 1 | 3 | 6 | 7 | 4 | 5 | 8 |
|---|---|---|---|---|---|---|---|

< pivot

> pivot

Note : puts pivot in its "rightful position".

we don't put the elements before or after the pivot in the right position

Tim Roughgarden

# Two Cool Facts About Partition

and

1. Linear O(n) time, no extra memory advantage on mergeSort
   [see next video]

2. Reduces problem size

Tim Roughgarden

# QuickSort: High-Level Description

[ Hoare circa 1961 ]

QuickSort (array A, length n)

-If n=1   return

-p = ChoosePivot(A,n) (later)

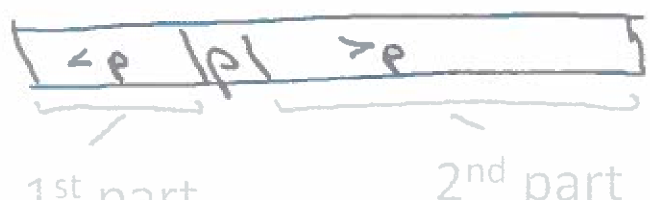
naive way is to choose the first element

-Partition A around p

-Recursively sort 1st part

-Recursively sort 2nd part (no merge step)

[ currently unimplemented ]




2 recursive calls: (the difference with MergeSort: we first split the array into pieces, recurse and then combine – here the recursives come last

Tim Roughgarden

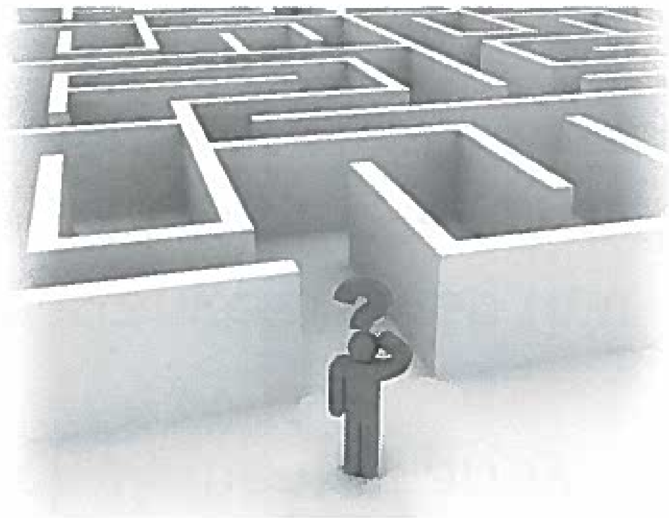
# Outline of QuickSort Videos

- The Partition subroutine
- Correctness proof [optional]
- Choosing a good pivot
- Randomized QuickSort
- Analysis
  - A Decomposition Principle
  - The Key Insight
  - Final Calculations



overview -4

July 27, 2018 25



Design and Analysis
of Algorithms I

# QuickSort

## The Partition Subroutine

Partition-1

# Partitioning Around a Pivot

Key Idea : partition array around a pivot element.

-Pick element of array

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 3 | 8 | 2 | 5 | 1 | 4 | 7 | 6 |

pivot

-Rearrange array so that

-Left of pivot => less than pivot

-Right of pivot => greater than pivot

$$\underbrace{2 \mid 1}_{<\ \text{pivot}} \mid 3 \mid \underbrace{6 \mid 7 \mid 4 \mid 5 \mid 8}_{>\ \text{pivot}}$$

Note : puts pivot in its "rightful position".

(In-place)

# Two Cool Facts About Partition

1. Linear O(n) time, no extra memory
   [see next video]

2. Reduces problem size

The question is how to implement the partitioning.

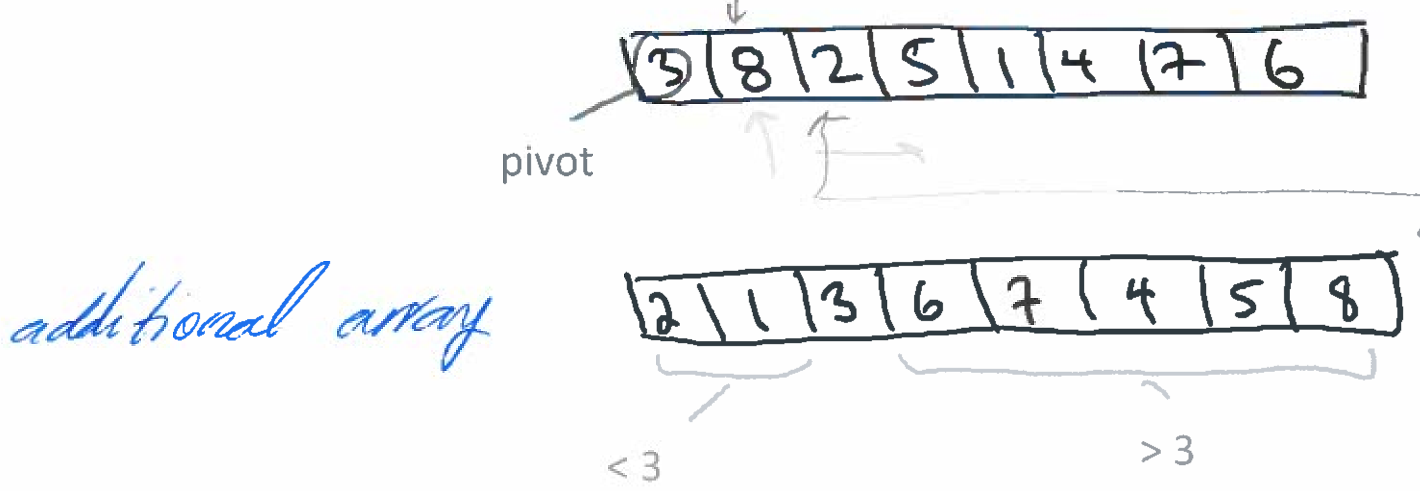
Partition_2

Let's leave out the ~~[illegible]~~ fact the we want the in-place requirement

# The Easy Way Out

Partitioning subroutine in linear time.

Note : Using O(n) extra memory, easy to partition around pivot in O(n) time.




The idea is this:

- we start with 8, 8 > pivot (3), thus we put it at the end of the array.
- The 2 < pivot (3), thus goes to the beginning of the array and so on.

Tim Roughgarden

This high level algo. is only valid for the case that the pivot is (but we can use it for the general case too.

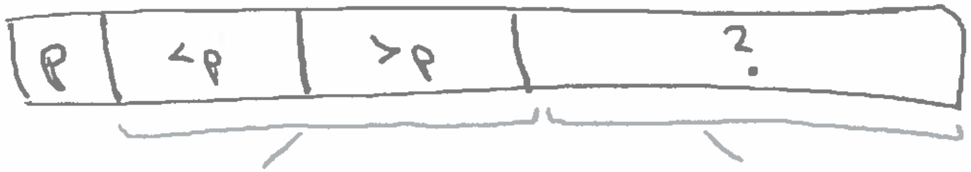
# In-Place Implementation

<u>Assume</u> : pivot = $1^{st}$ element of array

[ if not, swap pivot <--> $1^{st}$ element as preprocessing step ]

This means select the pivot in the middle, and move it to the first place.

<u>High – Level Idea</u> :




(Already partitioned we've seen) (unpartitioned we have not seen.)

(linear scan)

-Single scan through array $O(n)$.

- invariant : everything looked at so far is partitioned

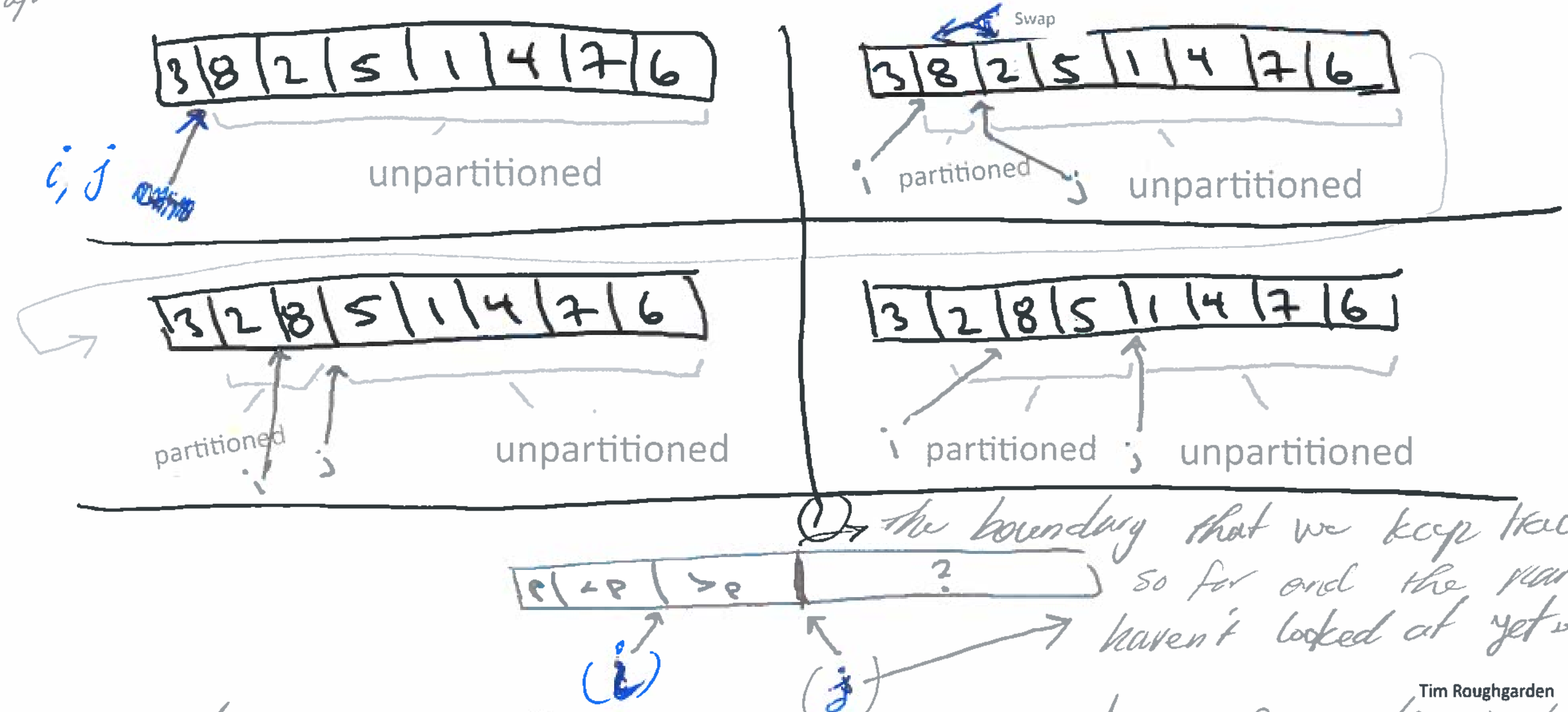
Tim Roughgarden

There are two boundaries that we need to keep track of.

In each step we advance j

# Partition Example

step 0: step 1:

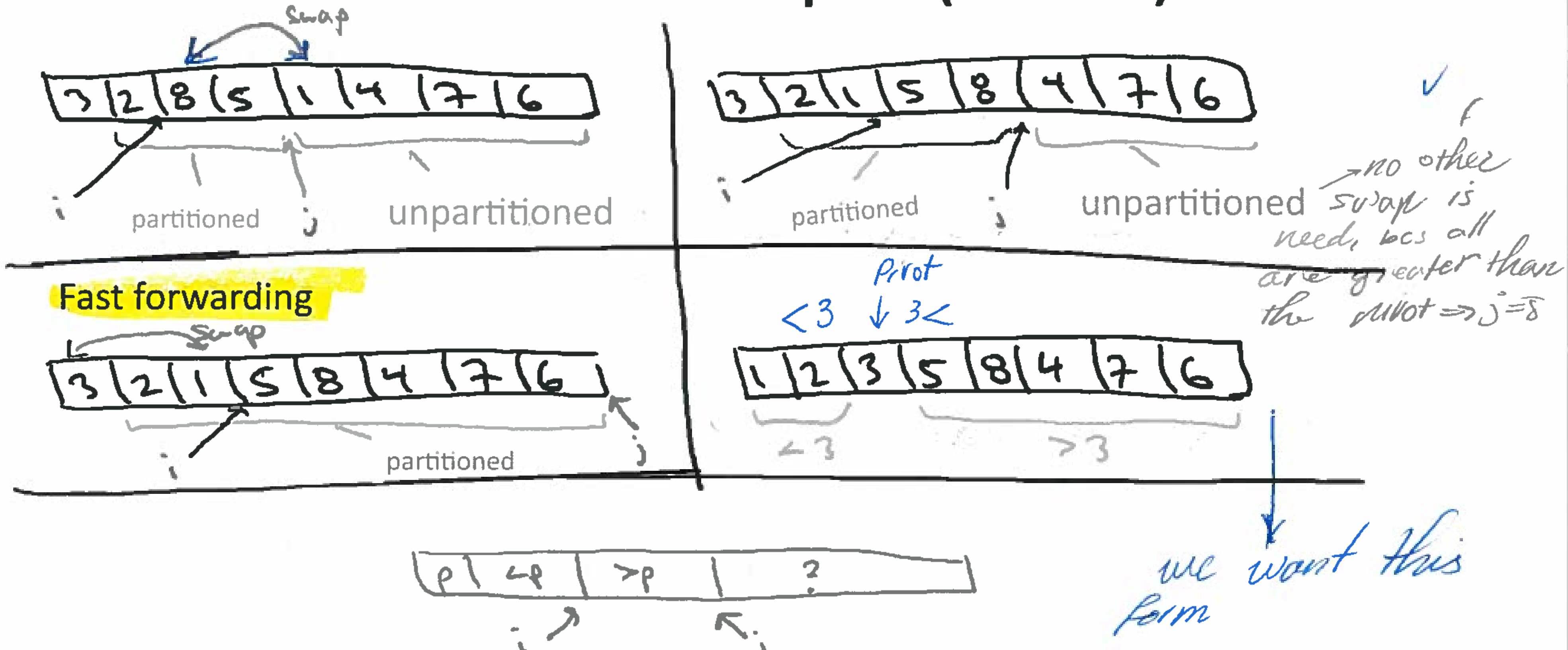



① the boundary that we keep tracked of so far and the part we haven't looked at yet.

Tim Roughgarden

② boundary amongst the no. we have seen, where the split is b/w the pivot < 5 >. (i)

# Partition Example (con'd)




Tim Roughgarden

Partition ...

The partition is going to be called recursively, from a quick sort algo. At any point in the quick sort, we would be recursing off some subset of the original array.

Two array indices: leftmost index, right most index

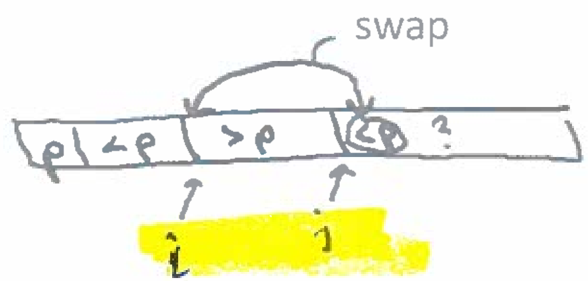
If A(1,2,...n), then, we want to partition A[l] to A[r].

# Pseudocode for Partition

→ passed to subroutine

Partition (A,l,r) [ input corresponds to A[l...r]]

pivot:
- p:= A[l] (first entry in the array, left most index)
- i:= l+1 (just right of the pivot)
- for j=l+1 to r
  - if A[j] < p [if A[j] > p, do nothing ]
    - swap A[j] and A[i]
    - i:= i+1
- swap A[l] and A[i-1]




Tim Roughgarden

what if the first entry after pivot is smaller than the pivot. (extra swap).

not in the book

time 20:00 check

# Running Time

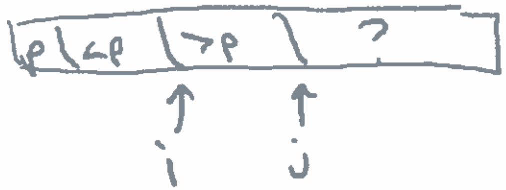
Running time = $O(n)$, where $n = r - l + 1$ is the length of the input (sub) array.

<u>Reason</u> : $O(1)$ work per array entry.

<u>Also</u> : clearly works in place (repeated swaps)

# Correctness

<u>Claim</u> : the for loop maintains the invariants :



1. A[l+1],..,A[i-1] are all less than the pivot
2. A[i],…,A[j-1] are all greater than pivot.

[ Exercise : check this, by induction. ]

<u>Consequence</u> : at end of for loop, have:

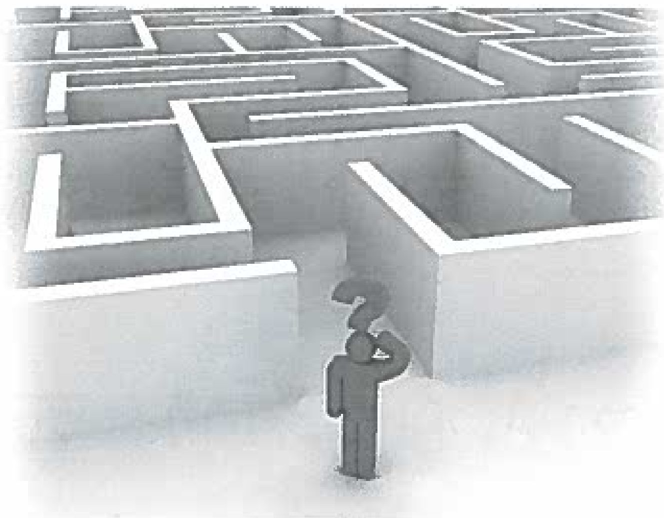
=> after final swap, array partitioned around pivot.

Q.E.D

Tim Roughgarden

July 23, 2018 26

Appendix A of the book



Design and Analysis
of Algorithms I

# QuickSort

# Proof of Correctness

Proof by induction for divide and conquer, specifically for quicksort.

The ~~[illegible]~~ format of proof by induction.

# Induction Review

something that you say or write that you strongly believe

Let P(n) = assertion parameterized by positive integers n. → a little abstrac.

For us : P(n) is "Quick Sort correctly sorts every input array of length n"

How to prove P(n) for all n >= 1 by induction :

1. [base case] directly prove that P(1) holds.
2. [inductive step] for every n>=2, prove that:
   If P(k) holds for all k<n, then P(n) holds as well.

INDUCTIVE HYPOTHESIS

imp

You apply the inductive step (n-1) times and you got.

Tim Roughgarden

not in the book.

# Correctness of QuickSort

P(n) = " QuickSort correctly sorts every input array of length n "

<u>Claim :</u> P(n) holds for every n >= 1 [no matter how pivot is chosen]

<u>Proof by induction :</u>

1. [base case] every input array of length 1 is already sorted.
   Quick Sort returns the input array which is correct (so P(1) holds)
2. [inductive step] Fix n>=2. Fix some input array of length n.

   <u>Need to show</u> : if P(k) holds for all k < n, then P(n) holds as well.

   INDUCTIVE STEP

Tim Roughgarden

correct_2

# Correctness of QuickSort (con'd)

Recall : QuickSort first partitions A around some pivot p.

$k_1$ and $k_2$ are at most $n-1$


Note : $k_1, k_2 < n$

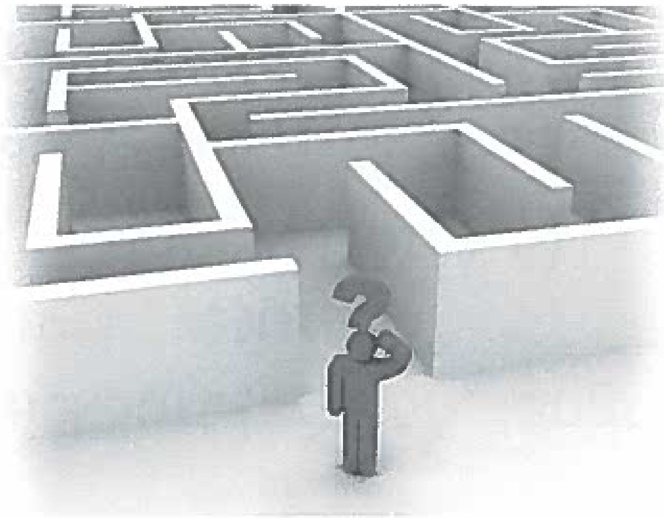
Note : pivot winds up in the correct position.

Let $k_1, k_2$ = lengths of 1st, 2nd parts of partitioned array.

Using $P(k_1)$, $P(k_2)$

By inductive hypothesis : 1st, 2nd parts get sorted correctly by recursive calls. So after recursive calls, entire array correctly sorted.

July 24, 2018 27



Design and Analysis
of Algorithms I

# QuickSort

# Choosing a Good Pivot

Pivot -1

# QuickSort: High-Level Description

[ Hoare circa 1961 ]

QuickSort (array A, length n)

-If n=1   return

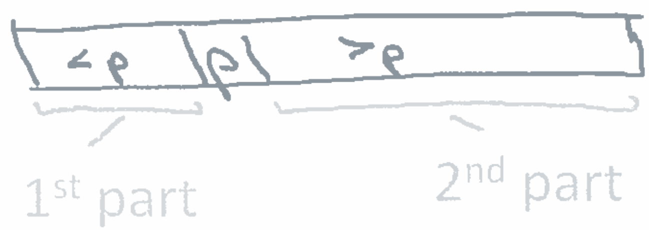
-p = ChoosePivot(A,n)

-Partition A around p

-Recursively sort 1st part

-Recursively sort 2nd part

[ currently unimplemented ]



MergeSort is O(nlogn) is it any better.

# The Importance of the Pivot

Q : running time of QuickSort ?

A : depends on the <u>quality of the pivot.</u> thus, we are not in the right position to discuss the running time, bcs we don't have enough info. It depends on the pivot.

.Imp: what is a good pivot? a pivot that splits the domain into two equal size subproblem is great.

Tim Roughgarden

low quality: splits unequal size subproblems.

Pivot -2

For the sorted array, essentially it does nothing - returns the same array.

This is the worst case in terms of performance.

Ter
ver

Suppose we implement QuickSort so that ChoosePivot always selects the first element of the array. What is the running time of this algorithm on an input array that is already sorted?

- Not enough information to answer question
- $\theta(n)$
- $\theta(n \log n)$
- $\theta(n^2)$

Reason :

Recurse on these

empty (*)

Length n-1 (still sorted)

1st n/2 terms are all at least n/2

Runtime : $\geq n + (n-1) + (n-2) + \ldots + 1$

$= \theta(n^2)$

(*) thus one of the recursive calls is vacuous.

For each subarray of length $k$, the recursive call is going to do $k$ operations.

This is the best case (running time scenario) why that matters?

we want to know how good we may do using this method

This will draw a line in the sand → the average running time cannot be better than the best case.

→ what is the highest quality pivot → (median element) give us two equal subproblems (n/2)

Te
ve

Suppose we run QuickSort on some input, and, magically, every recursive call chooses the median element of its subarray as its pivot. What's the running time in this case?

- Not enough information to answer question
- $\theta(n)$
- $\theta(n\log n)$
- $\theta(n^2)$

Similar to MergeSort

Reason : Let T(n) = running time on arrays of size n.

only b's median

Because pivot = median

choosePivot

partition

Then : $T(n) \le 2T(n/2) + \theta(n)$

$\Rightarrow T(n) = \theta(n \log n)$ [like MergeSort]

master method

a=2 = 2

Point -3

# Random Pivots

Key Question : how to choose pivots ?        BIG IDEA : RANDOM PIVOTS!

That is : in every recursive call, choose the pivot randomly.
(each element equally likely)

Hope : a random pivot is "pretty good"   "often enough".

Intuition : 1.) if always get a 25-75 split, good enough for O(nlog(n)) running time. [this is a non-trivial exercise : prove via recursion tree ]
2.) half of elements give a 25-75 split or better

Q : does this really work ? (later)

Tim Roughgarden

Assume that we have an array of 100 elements, and (1 ~ 100 itself as elements).
· Which numbers give us a 25-75 split? 26 ~ 75 anything in this range.
Thus 50% of elements are good enough. ≡ Flip a coin.

next 3 videos are on the proof of quick sort.

# Average Running Time of QuickSort

QuickSort Theorem : for every input array of length n, the average running time of QuickSort (with random pivots) is O(nlog(n)). Proof: next videos.

Note : holds for every input. [no assumptions on the data ]

- recall our guiding principles !
- "average" is over random choices made by the algorithm (i.e., the pivot choices )

means you run it many times, you get a different no. run time

$O(n \log n) < \dots < O(n^2)$. but on average the running time is the best case. $O(n \log n)$.

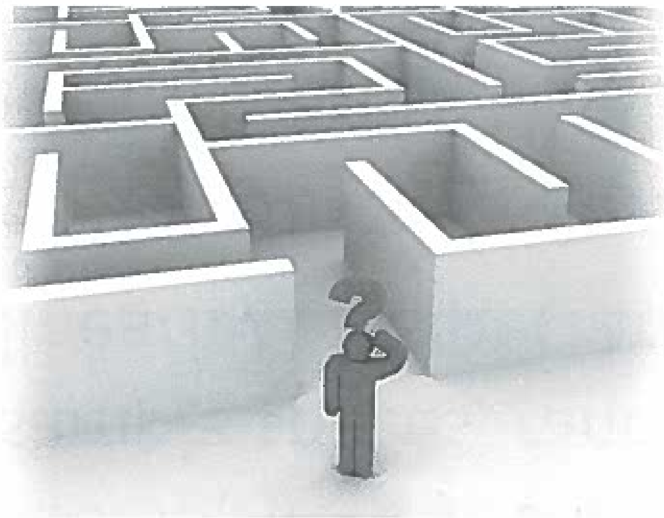
Tim Roughgarden

P.1nt - 21

6

28

July ~~27~~ 2018



# QuickSort

## Analysis I: A Decomposition Principle

Design and Analysis of Algorithms I

In this lecture we want to show that the "average" running time of QuickSort is $O(n \log n)$.

AnaI - 1

# Necessary Background

Assumption: you know and remember (finite) sample spaces, random variables, expectation, linearity of expectation. For review:

- Probability Review I (video)
- Lehman-Leighton notes (free PDF)
- Wikibook on Discrete Probability

remember the worst case is $O(n^2)$, best case $O(n \log n)$

The running time is closer to the best case not to the worst case

# Average Running Time of QuickSort

we make no assumption on the data

<u>QuickSort Theorem</u> : for every input array of length n, the average running time of QuickSort (with random pivots) is $O(n\log(n))$.

<u>Note</u> : holds for every input. [no assumptions on the data ]

- recall our guiding principles !
- "average" is over random choices made by the algorithm (i.e., the pivot choices )

# Preliminaries

Fix input array A of length n

Sample Space $\Omega$ = all possible outcomes of random choices in QuickSort (i.e., pivot sequences)

Key Random Variable : for $\sigma \in \Omega$

governs the average running time of the QS

$C(\sigma)$ = # of comparisons between two input elements made by QuickSort (given random choices $\sigma$) $\equiv$ in the algorithm $A[j] < P$

Lemma: running time of QuickSort dominated by comparisons.

this says that the overal running time of the QuickSort algorithm is only different by a constant from the comparisons

There exist constant c s.t. for all $\sigma \in \Omega$ , $RT(\sigma) \leq c \cdot C(\sigma)$ (see notes)

in the book (page 137)

Running Time of QuickSort

Remaining goal : E[C] = O(nlog(n))

any pivot in the array.

Tim Roughgarden

For two reasons ① random no. of recursions ② unequal subproblems.

# Building Blocks

Note can't apply Master Method [random, unbalanced subproblems]

but we do something similar

A: (unsorted) [6 | 8 | 10 | 2]

[A = final input array] length $n$;

$z_2$ $z_3$ $z_4$ $z_1$

2nd smallest 1st smallest

A decomposition theory

Notation : $z_i$ = $i^{th}$ smallest element of A

$z_i$ is not the element in the "ith" position of the input unsorted array.

For $\sigma \in \Omega$, indices $i < j$ $i,j \in (1, \dots, n)$

given

$X_{ij}(\sigma)$ = # of times $z_i, z_j$ get compared in QuickSort with pivot sequence $\sigma$

Tim Roughgarden

Ana I - 3

Ter
ver

Fix two elements of the input array. How many times can these two elements get compared with each other during the execution of QuickSort?

- ○ 1
- ○ 0 or 1
- ○ 0, 1, or 2
- ○ Any integer between 0 and $n-1$

Reason : two elements compared only when one is the pivot, which is excluded from future recursive calls.

Thus : each $X_{ij}$ is an "indicator" (i.e., 0-1) random variable

both cannot be pivots

Fig A.27
A.29
A.21

# A Decomposition Approach

<u>So</u> : $C(\sigma)$ = # of comparisons between input elements

$X_{ij}(\sigma)$ = # of comparisons between $z_i$ and $z_j$

<u>Thus</u> : $\forall \sigma, C(\sigma) = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} X_{ij}(\sigma)$

<u>By Linearity of Expectation :</u> $E[\underbrace{C}_{\text{complicated}}] = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} E[\underbrace{X_{ij}}_{\text{simple}}]$

Since $E[X_{ij}] = 0 \cdot Pr[X_{ij} = 0] + 1 \cdot Pr[X_{ij} = 1] = Pr[X_{ij} = 1]$ ✓

Thus : $E[C] = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \underbrace{Pr[z_i, z_j \textit{ get compared}]}_{\text{Next video}} \quad (*)$

Tim Roughgarden

Ano I-4

# A General Decomposition Principle

1. Identify random variable Y that you really care about

2. Express Y as sum of indicator random variables :

$$Y = \sum_{l=1}^{m} X_e$$

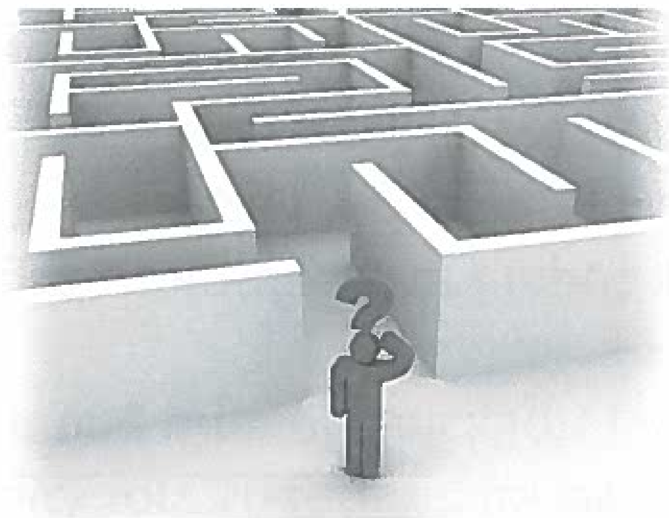
3. Apply Linearity of expectation :

$$E[Y] = \sum_{l=1}^{m} Pr[X_e = 1]$$

"just" need to understand these!

6

July 27/2018 29



# QuickSort

## Analysis II: The Key Insight

Design and Analysis of Algorithms I

Ana I – 1

# Average Running Time of QuickSort

<u>QuickSort Theorem</u> : for every input array of length n, the average running time of QuickSort (with random pivots) is $O(n\log(n))$.

<u>Note</u> : holds for every input. [no assumptions on the data ]

- recall our guiding principles !
- "average" is over random choices made by the algorithm (i.e., the pivot choices )

# The Story So Far

$C(\sigma)$ = # of comparisons between input elements

$X_{ij}(\sigma)$ = # of comparisons between $\underbrace{z_i \text{ and } z_j}_{i^{th},\ j^{th} \text{ smallest entries in array}}$

**Recall** : $E[C] = \sum_{i=1}^{n-1} \sum_{k=i+1}^{n} Pr[X_{ij} = 1]$

$= Pr[z_i\ z_j \ get\ compared]$

Key Claim : for all $i < j$, $Pr[z_i, z_j$ get compared $] = 2/(j-i+1)$

for example the chance that $z_3$ and $z_7$ get compared is $\frac{2}{7-3+1} = \frac{2}{5} = 40\%$

Tim Roughgarden

Anat I - 2

choosing two (bad) no. from $j-i+1$ numbers $\Rightarrow Pr = \frac{2}{j-i+1}$.

# Proof of Key Claim

$Pr[z_i, z_j$ get compared $] = 2/(j-i+1)$

Fix $z_i$, $z_j$ with $i < j$ total elements $j-i+1$

Consider the set $\underbrace{z_i, z_{i+1}, \ldots, z_{j-1}, z_j}$

If pivot is bigger they go to left side and vice versa

**Inductively** : as long as none of these are chosen as a pivot, all are passed to the same recursive call. until one of them is chosen as a pivot.

Consider the first among $z_i, z_{i+1}, \ldots, z_{j-1}, z_j$ that gets chosen as a pivot.

1. If $z_i$ or $z_j$ gets chosen first, then $z_i$ and $z_j$ get compared
2. If one of $z_{i+1}, \ldots, z_{j-1}$ gets chosen first then $z_i$ and $z_j$ are never compared [split into different recursive calls]

KEY INSIGHT

Tim Roughgarden

they only compared if one of them compared as pivot.

# Proof of Key Claim (con'd)

1. $z_i$ or $z_j$ gets chosen first => they get compared
2. one of $z_{i+1},\ldots,z_{j-1}$ gets chosen first => $z_i$ , $z_j$ never compared

Note : Since pivots always chosen uniformly at random, each of $z_i,z_{i+1},\ldots,z_{j-1},z_j$ is equally likely to be the first

$\Rightarrow$Pr[$z_i$,$z_j$ get compared ] = 2/(j-i+1)

Choices that lead to case (1)

Total # of choices

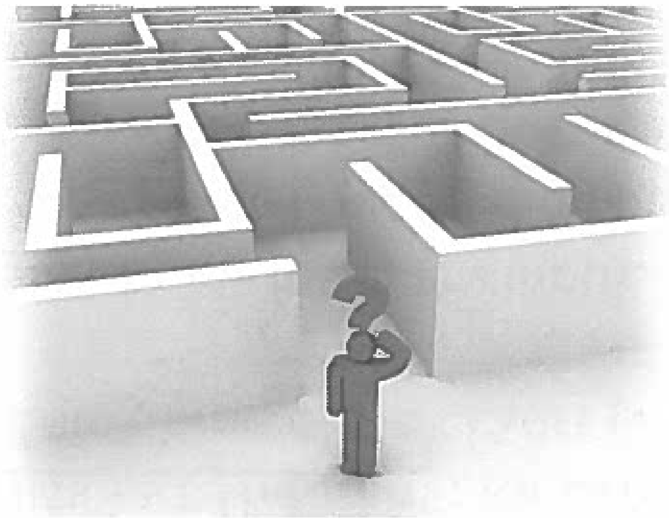
So : $E[C] = \sum_{i=1}^{n-1} \sum_{j=1}^{n} \frac{2}{j-i+1}$

[Still need to show this is O(nlog(n))

Tim Roughgarden

Ana I -3

7 (36)



Design and Analysis of Algorithms I

# QuickSort

## Analysis III: Final Calculations

Ana III - 1

# Average Running Time of QuickSort

<u>QuickSort Theorem</u> : for every input array of length n, the average running time of QuickSort (with random pivots) is $O(n\log(n))$

<u>Note</u> : holds for every input. [no assumptions on the data ]

- recall our guiding principles !
- "average" is over random choices made by the algorithm (i.e., pivot choices )

Tim Roughgarden

what is the upper bound? the best case is that the denominator is small, so that we have the largest value. what is the smallest denominator? "2" bcs $j \geq i+1 \Rightarrow 2n \cdot n \left(\frac{1}{2}\right) = O(n^2)$.

# The Story So Far

$$E[C] = 2\sum_{i=1}^{n-1}\underbrace{\sum_{j=1}^{n}\overbrace{\frac{1}{j-i+1}}^{\text{How big can this be ?}}}_{\theta(n^2)\ terms} \quad (*)$$

<= n choices for i

Note : for each fixed i, the inner sum is

$$\sum_{j=i+1}^{n}\frac{1}{j-i+1} = \overbrace{1/2 + 1/3 + \ldots + \frac{1}{n-i+1}}^{(n-i)\ \text{terms}}$$

$$So \quad E[C] \leq 2 \cdot n \cdot \sum_{k=2}^{n}\frac{1}{k}$$

Claim : this is <= ln(n)

Tim Roughgarden

$$\leq 2\sum_{i=1}^{n-1}\sum_{j=i+1}^{n}\frac{1}{j-i+1}$$

inner sum assume $i=1$

$$= \sum_{i=1}^{n-1}\sum_{j=2}^{n}\frac{2}{j} = 2(n-1)\sum_{j=2}^{n}\frac{1}{j}$$

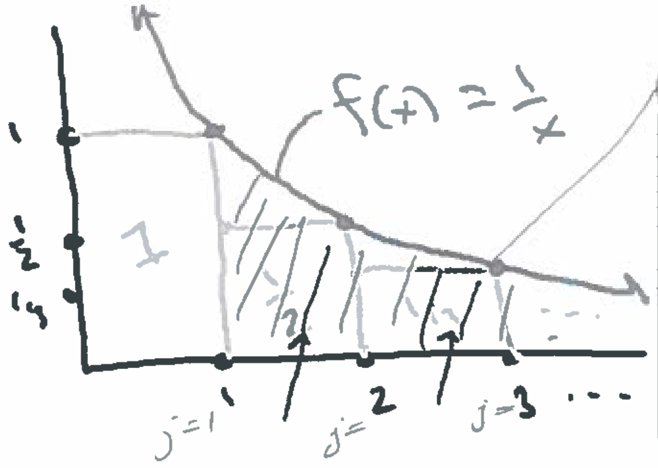
Ana III_2

# Completing the Proof

$$E[C] \leq 2 \cdot n \cdot \sum_{k=2}^{n} \frac{1}{k}$$

$$\boxed{Claim \quad \sum_{k=2}^{n} \frac{1}{k} \leq \ln n}$$

Proof of Claim




$$So \quad \sum_{k=2}^{n} \frac{1}{n} \leq \int_{1}^{n} \frac{1}{x} dx$$

$$= \ln x \,\Big|_1^n$$

$$= \ln n - \ln 1$$

$$= \ln n$$

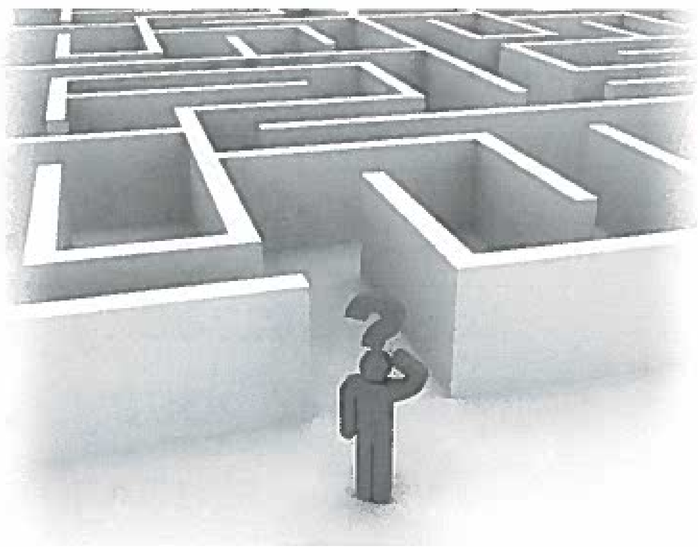
Q.E.D. (CLAIM)

So :
E[C] <=
2n ln n

Q.E.D.

Tim Roughgarden

July 25, 2018 30



Design and Analysis
of Algorithms I

# Probability Review

# Part I

Probability I - 1

Topics in Discrete probability.

# Topics Covered

Discussing minimum cut in graph

- Sample spaces
- Events
- Random variables
- Expectation
- Linearity of Expectation

See also:

- Lehman-Leighton notes (free PDF)
- Wikibook on Discrete Probability

# Concept #1 – Sample Spaces

Sample Space $\Omega$ : "all possible outcomes"
[ in algorithms, $\Omega$ is usually finite ]

what is the probability of one outcome?

Also : each outcome $i \in \Omega$ has a probability p(i) >= 0

Constraint : $\boxed{\sum_{i \in \Omega} p(i) = 1}$

equally likely : $p(i) = \frac{1}{36}$ 36 different outcome.

Example #1 : Rolling 2 dice. $\Omega$ = {(1,1), (2,1), (3,1),…,(5,6),(6,6) }
Example #2 : Choosing a random pivot in outer QuickSort call.
$\Omega$ = {1,2,3,…,n} (index of pivot) and p(i) = 1/n for all $i \in \Omega$

In quicksort we randomly choose one pivot.

Tim Roughgarden

Prob I – 7

very good

# Concept #2 – Events

An event is a subset $S \subseteq \Omega$

The probability of an event S is $\sum_{i \in S} p(i)$

Consider the event (i.e., the subset of outcomes for which) “the sum of the two dice is 7”. What is the probability of this event?

$$S = \{(1,6),(2,5),(3,4),(4,3),(5,2),(6,1)\}$$

$$\Pr[S] = 6/36 = 1/6$$

- ○ 1/36
- ○ 1/12
- ○ 1/6
- ○ 1/2

ProbI-3

Consider the event (i.e., the subset of outcomes for which) "the chosen pivot gives a 25-75 split of better". What is the probability of this event?

Event

- ○ $1/n$
- ○ $1/4$
- ○ $1/2$
- ○ $3/4$

$S = \{(n/4+1)^{th}$ smallest element,.., $(3n/4)^{th}$ smallest element

$Pr[S] = (n/2)/n = 1/2$

# Concept #2 – Events

An event is a subset

The probability of an event S is

<u>Ex#1</u> : sum of dice = 7. S = {(1,1),(2,1),(3,1),…,(5,6),(6,6)}
Pr[S] = 6/36 = 1/6

<u>Ex#2</u> : pivot gives 25-75 split or better.
S = {$(n/4+1)^{th}$ smallest element,…,$(3n/4)^{th}$ smallest element]
Pr[S] = (n/2)/n = 1/2

Tim Roughgarden

Prob I – 4

# Concept #3 - Random Variables

A Random Variable X is a real-valued function

$$X : \Omega \rightarrow \Re$$

$\mathbb{R}$

↓ real-valued function

Ex#1 : Sum of the two dice

Ex#2 : Size of subarray passed to 1st recursive call.

Tim Roughgarden

# Concept #4 - Expectation

Let $X : \Omega \rightarrow \Re$ be a random variable.

The expectation E[X] of X = average value of X

$$= \sum_{i \in \Omega} X(i) \cdot p(i)$$

weighted by the probability of that outcome

Tim Roughgarden

Prob I_5

What is the expectation of the sum of two dice?

$$E[X] = \sum_{i \in \Omega} X(i) \cdot \underbrace{p(i)}_{= \frac{1}{36}} = \frac{1}{36} \sum_{i \in \Omega} X(i) = \frac{1}{36}\big((1+1)+(1+2)+(1+3)+(1+4)+(1+5)+(1+6)$$
$$+(2+1)+(2+2) \cdots +(2+6)\big)$$

$$= 7$$

- ○ 6.5
- ○ 7
- ○ 7.5
- ○ 8

Which of the following is closest to the expectation of the size of the subarray passed to the first recursive call in QuickSort?

- $n/4$
- $n/3$
- $n/2$
- $3n/4$

Let X = subarray size

Then E[X] = (1/n)*0 + (1/n)*2 + ... + (1/n)*(n-1)
= (n-1)/2

RobI_6

# Concept #4 - Expectation

Let $X : \Omega \rightarrow \Re$ be a random variable.

The expectation E[X] of X = average value of X

$$= \sum_{i \in \Omega} X(i) \cdot p(i)$$

Ex#1 : Sum of the two dice, E[X] = 7

Ex#2 : Size of subarray passed to $1^{st}$ recursive call.

E[X] = (n-1)/2

Tim Roughgarden

very important.

# Concept #5 – Linearity of Expectation

Claim [LIN EXP] : Let $X_1,\dots,X_n$ be random variables defined on $\Omega$ . Then :

$$E[\sum_{j=1}^{n} X_j] = \sum_{j=1}^{n} E[X_j]$$

CRUCIALLY:
HOLDS EVEN WHEN $X_j$'s ARE NOT INDEPENDENT!
[WOULD FAIL IF REPLACE SUMS WITH PRODUCTS]

Ex#1 : if $X_1,X_2$ = the two dice, then
$E[X_j] = (1/6)(1+2+3+4+5+6) = 3.5$ ✓

By LIN EXP : $E[X_1+X_2] = E[X_1] + E[X_2] = 3.5 + 3.5 = 7$

Tim Roughgarden

Prob I-7

# Linearity of Expectation (Proof)

$$\sum_{j=1}^{n} E[X_j] = \sum_{j=1}^{n} \sum_{i \in \Omega} X_j(i)p(i)$$

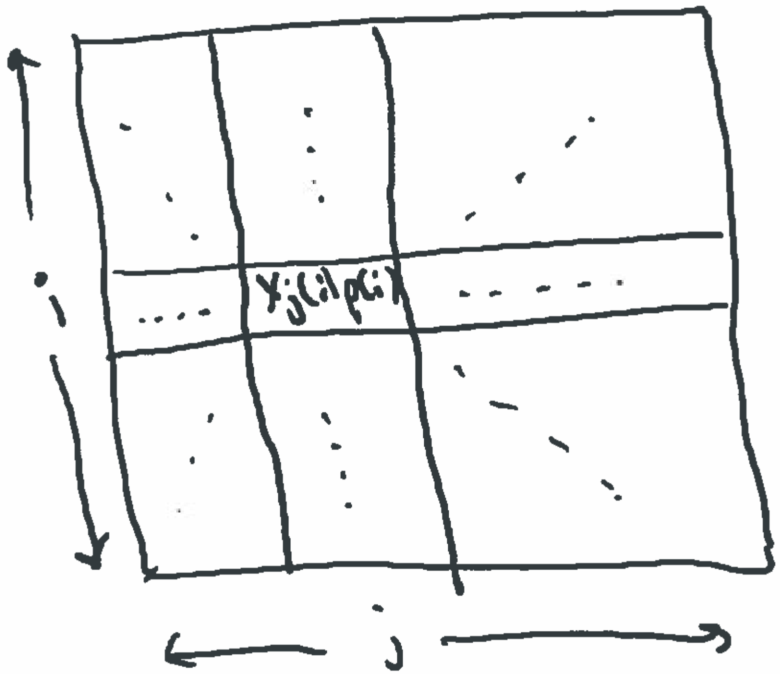
why we can change the summation

$$= \sum_{i \in \Omega} \sum_{j=1}^{n} X_j(i)p(i)$$

$$= \sum_{i \in \Omega} p(i) \sum_{j=1}^{n} X_j(i)$$

$$= E[\sum_{j=1}^{n} X_j]$$

Q.E.D.



# Example: Load Balancing

<u>Problem</u> : need to assign n processes to n servers.

<u>Proposed Solution</u> : assign each process to a random server

<u>Question</u> : what is the expected number of processes assigned to a server ?

Tim Roughgarden

# Load Balancing Solution

Sample Space $\Omega$ = all $n^n$ assignments of processes to servers, each equally likely.

Let Y = total number of processes assigned to the first server.

Goal : compute E[Y]

"indicator random variable"

Let $X_j = \begin{cases} 1 & \text{if jth process assigned to first server} \\ 0 & \text{otherwise} \end{cases}$

$$Note \quad Y = \sum_{j=1}^{n} X_j$$

# Load Balancing Solution (con'd)

We have

$$E[Y] = E[\sum_{j=1}^{n} X_j]$$

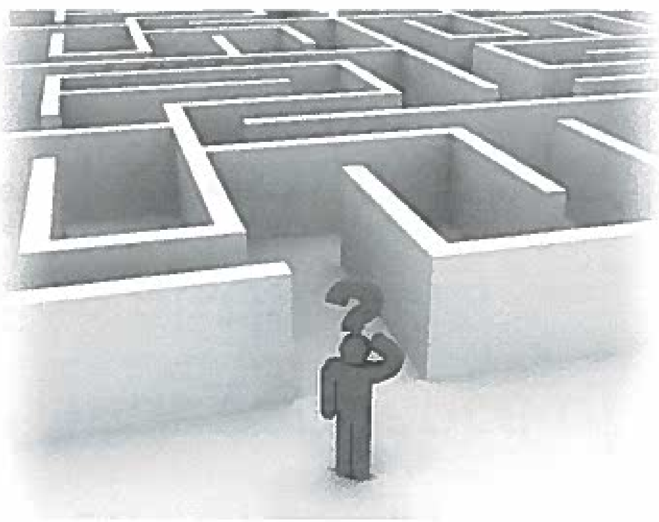
$$= \sum_{j=1}^{n} E[X_j]$$

$$= \sum_{j=1}^{n} (\cancelto{0}{Pr[X_j = 0]} \cdot 0 + \underbrace{Pr[X_j = 1]}_{= 1/n \text{ (servers chosen uniformly at random)}} \cdot 1)$$

$$= \sum_{j=1}^{n} \frac{1}{n} \quad = 1$$

Tim Roughgarden

July 26, 2018 30 ''



Design and Analysis
of Algorithms I

# Probability Review

# Part II

Prob II_1

# Topics Covered

- Conditional probability
- Independence of events and random variables

See also:

- Lehman-Leighton notes (free PDF)
- Wikibook on Discrete Probability

# Concept #1 – Sample Spaces

Sample Space $\Omega$ : "all possible outcomes"
[ in algorithms, $\Omega$ is usually finite ]

Also : each outcome $i \in \Omega$ has a probability p(i) >= 0

Constraint : $\sum_{i \in \Omega} p(i) = 1$

An event is a subset $S \subseteq \Omega$

The probability of an event S is $\sum_{i \in S} p(i)$

Tim Roughgarden

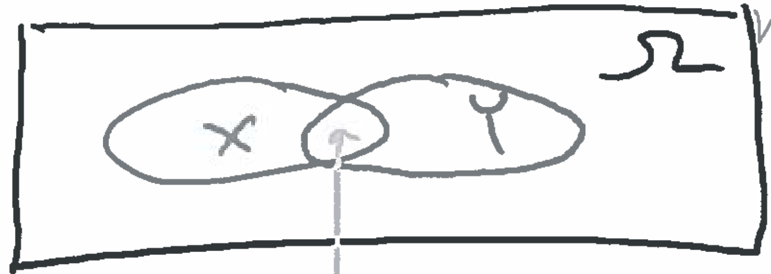
Prob II_ 2

conditional probability of an event given the second event.

# Concept #6 – Conditional Probability

*Let* $X, Y \subseteq \Omega$ *be events.*

Venn diagram




$X \cap Y$

*Then* $Pr[X|Y] = \dfrac{Pr[X \cap Y]}{Pr[Y]}$

("X given Y")

Intersection of X and Y.

$X \cup Y$ Union of two sets.

$X = \{1,1 \quad 1,2 \quad 1,3 \quad 1,4 \quad 1,5, \quad 1,6 \quad 2,1 \quad 3,1 \quad 4,1 \quad 5,1 \quad 6,1\}$ $\quad Y = \{1,6 \quad 2,5 \quad 3,4 \quad 4,3 \quad 5,2 \quad 6,1\}$

$P[X] = \frac{12}{36} = \frac{1}{3}$ $\qquad P[Y] = \frac{6}{36} = \frac{1}{6}$

$X \cap Y = \{(1,6), (6,1)\} \Rightarrow P[X \cap Y] = 2/36 = \frac{1}{18}$

$P[X|Y] = \frac{1/18}{1/6} = \frac{1}{3}$ $\qquad$ We don't care about $Pr[X]$.

Te
ve

Suppose you roll two fair dice. What is the probability that at least one die is a 1, given that the sum of the two dice is 7?

dice

x $\qquad$ y

- ○ 1/36
- ○ 1/6
- ○ 1/3
- ○ 1/2

X = at least one die is a 1
Y = sum of two dice = 7
= {(1,6),(2,5),(3,4),(4,3),(5,2),(6,1)}

$$=> X \cap Y = \{(1,6),(6,1)\}$$

$$Pr[X|Y] = \frac{Pr[X \cap Y]}{Pr[Y]} = \frac{(2/36)}{(6/36)} = \frac{1}{3}$$

Prob II 3

# Concept #7 – Independence (of Events)

Definition : Events $X, Y \subseteq \Omega$ are independent if (and only if) $Pr[X \cap Y] = Pr[X] \cdot Pr[Y]$

You check : this holds if and only if Pr[X | Y ] = Pr[X]
<==> Pr[Y|X] = Pr{Y]

WARNING : can be a very subtle concept.
(intuition is often incorrect!)

Tim Roughgarden

# Independence (of Random Variables)

Definition : random variables A, B (both defined on $\Omega$ )
are independent if and only if the events Pr[A=a], Pr[B=b] are
independent for all a,b. [<==> Pr[A = a and B = b] = Pr[A=z]*Pr[B=b] ]

Claim : if A,B are independent, then E[AB] = E[A]*E[B]

Proof : $E[AB] = \sum_{a,b} (a \cdot b) \cdot Pr[A = a \text{ and } B = b]$

$$= \sum_{a,b} (a \cdot b) \cdot Pr[A = a] \cdot Pr[B = b]$$ (Since A,B independent)

$$= \underbrace{\left(\sum_a a \cdot Pr[A = a]\right)}_{E[A]} \underbrace{\left(\sum_b b \cdot Pr[B = b]\right)}_{E[B]}$$

Q.E.D.

Tim Roughgarden

Prob II-4

| $X_1$ | $X_2$ | $X_1 \oplus X_2$ |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 1 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |

means $X_1$ or $X_2$ are either 0 or 1

# Example

XOR = Exclusive OR

Let $X_1, X_2 \in \{0,1\}$ be random, and $X_3 = X_1 \oplus X_2$

<u>formally</u> : $\Omega = \{000, 101, 011, 110\}$, each equally likely.

<u>Claim</u> : $X_1$ and $X_3$ are independent random variables (you check)

bcs in the four possible outcome in the sample space, we have all possible cases.

| | |
|---|---|
| 0 | 0 |
| 0 | 1 |
| 1 | 0 |
| 1 | 1 |

<u>Claim</u> : $X_1X_3$ and $X_2$ are not independent random variables.

<u>Proof</u> : suffices to show that

$$E[X_1X_2X_3] \neq E[X_1X_3]E[X_2]$$

$E[X_1X_2X_3] = 0$

$= \sum 0() + 0() + 0() + 0()$

$= 0$

$E[X_2] = 1/2 = \sum 0 \cdot \frac{1}{2} + 1 \cdot \frac{1}{2}$

$E[X_1X_3] = E[X1]E[X3] = 1/4$ Since $X_1$ and $X_3$ independent

$\frac{1}{8}$

⟹ great

Tim Roughgarden