CS146: Data Structures and Algorithms
Lecture 7

QUICKSORT

INSTRUCTOR: KATERINA POTIKA
CS SJSU

Quicksort (Ch 7)

- Sorts in place
- Sorts $O(n \lg n)$ in the average case
- Sorts $O(n^2)$ in the worst case
  - But in practice, it's quick
- And the worst case doesn't happen often (but more on this later…)

Quicksort

- Another *divide-and-conquer* algorithm
  - The array A[p..r] is *partitioned* into two non-empty subarrays A[p..q] and A[q+1..r]
    - **Invariant**: All elements in A[p..q] are less than all elements in A[q+1..r]
  - The subarrays are recursively sorted by calls to quicksort
  - Unlike merge sort, **no** combining step: two subarrays form an already-sorted array

Quicksort Code


```
Quicksort(A, p, r)
{
    if (p < r)
    {
        q = Partition(A, p, r)
        Quicksort(A, p, q-1)
        Quicksort(A, q+1, r)
    }
}
```

Partition

- All action takes place in the **`partition()`** function
- Rearranges the subarray *in place*
- End result:
  - Two subarrays
  - All values in first subarray ≤ all values in second
- Returns the index of the "pivot" element separating the two subarrays (i.e. q)

How do we implement this?

Partition In Words

Partition(A, p, r):

Select an element to act as the "pivot" (which?)

Grow two regions, A[p..i] and A[j..r]

All elements in A[p..i] <= pivot

All elements in A[i+1..j] >= pivot

Increment j

if A[j] <= pivot

Increment i

Swap A[i] and A[j]

Repeat until j <=r-1

Swap A[i+1] and A[r]

Return i+1

Partition Code

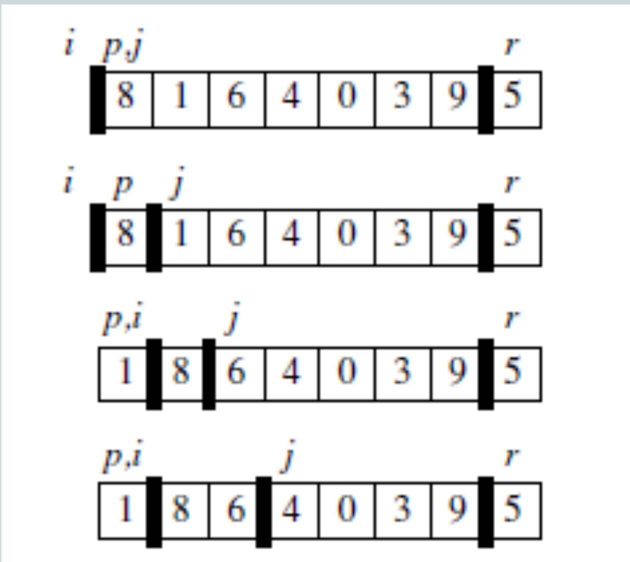

```
Partition(A, p, r)
    x = A[r]
    i = p - 1
    for j=p to r-1
        if A[j]<=x
            i++;
            Swap(A, i, j);
    Swap(A, i+1, r);
return i+1;
```

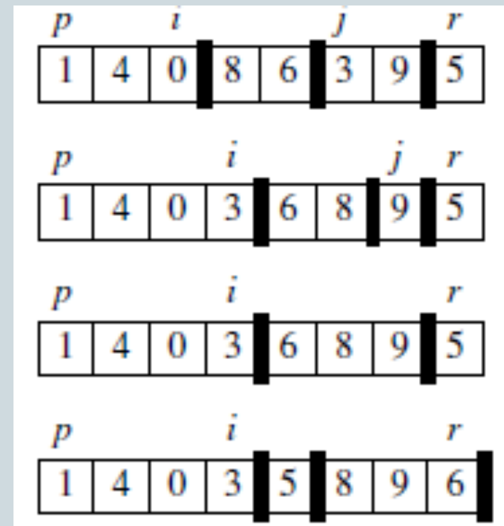
What is the running time of ***`partition()`****?*

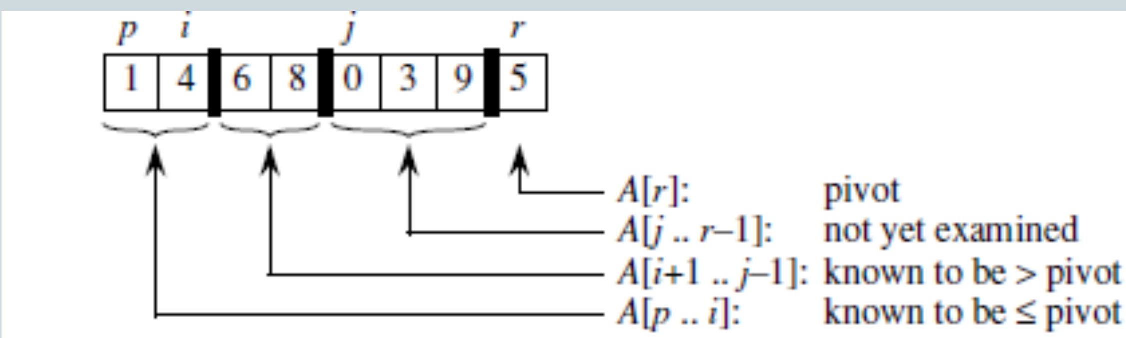
O(n)

Why?

Example for Partition







Loop Invariant

PARTITION always selects the last element $A[r]$ in the subarray $A[p \mathinner{..} r]$ as the ***pivot***—the element around which to partition.

As the procedure executes, the array is partitioned into four regions, some of which may be empty:

Loop invariant:

1. All entries in $A[p \mathinner{..} i]$ are ≤ pivot.
2. All entries in $A[i + 1 \mathinner{..} j - 1]$ are > pivot.
3. $A[r]$ = pivot.

It's not needed as part of the loop invariant, but the fourth region is $A[j \mathinner{..} r-1]$, whose entries have not yet been examined, and so we don't know how they compare to the pivot.

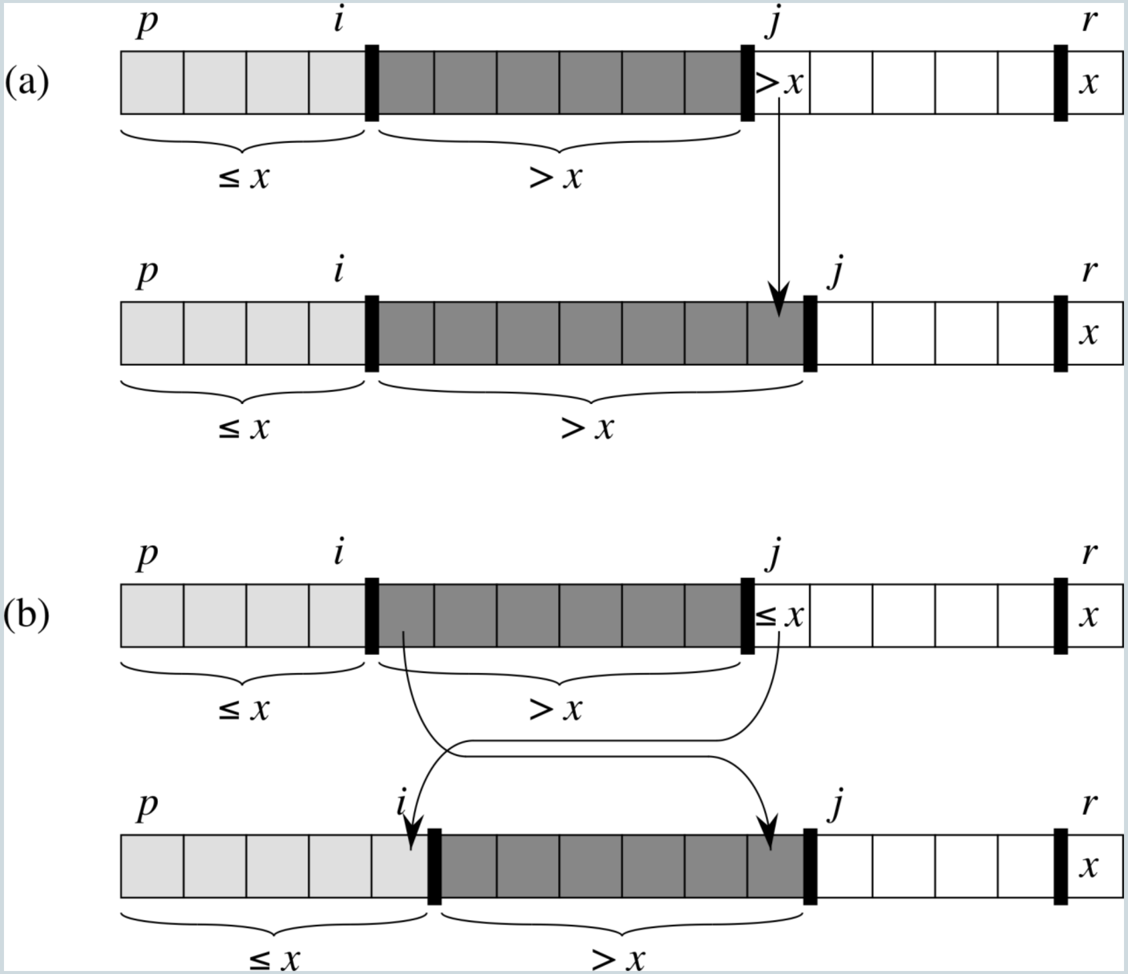
Correctness of partition

- **Initialization**: Before the loop starts, all the conditions of the loop invariant are satisfied, because r is the pivot and the subarrays
  $A[p \, . \, . \, i]$ and $A[i+1 \, . \, . \, j-1]$ are empty.

- **Maintenance**: While the loop is running, if $A[j] \le$ pivot, then
  $A[j]$ and $A[i+1]$ are swapped and then i and j are incremented. If $A[j]$ > pivot, then increment only j .

- **Termination**: When the loop terminates, $j = r$, so all elements in A are partitioned into one of the three cases:
  $A[p \, . \, . \, i] \le$ pivot,
  $A[i+1 \, . \, . \, r-1] >$ pivot, and
  $A[r] =$ pivot

Correctness of Partition (maintenance)



Analyzing Quicksort

What will be the worst case for the algorithm?

Partition is always unbalanced

What will be the best case for the algorithm?

Partition is perfectly balanced

Which is more likely?

The latter, by far, except...

Will any particular input evoke the worst case?

Yes: Already-sorted input

Analyzing Quicksort

In the worst case:

$T(1) = \Theta(1)$

$T(n) = T(n - 1) + \Theta(n)$

Works out to

$T(n) = \Theta(n^2)$ Why?

Analyzing Quicksort

In the best case:

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

What does this work out to be?

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

Improving Quicksort

The real liability of quicksort is that it runs in $O(n^2)$ on already-sorted input

Book discusses two solutions:

- Randomize the input array, OR
- *Pick a random pivot element*

How will these solve the problem?

By insuring that no particular input can be chosen to make quicksort run in $O(n^2)$ time

Randomized Algorithms

- Worst case occurs only if we get "unlucky" numbers from the random number generator
- Worst case becomes less likely
  - Randomization can NOT eliminate the worst-case but it can make it less likely!

Randomized Quicksort

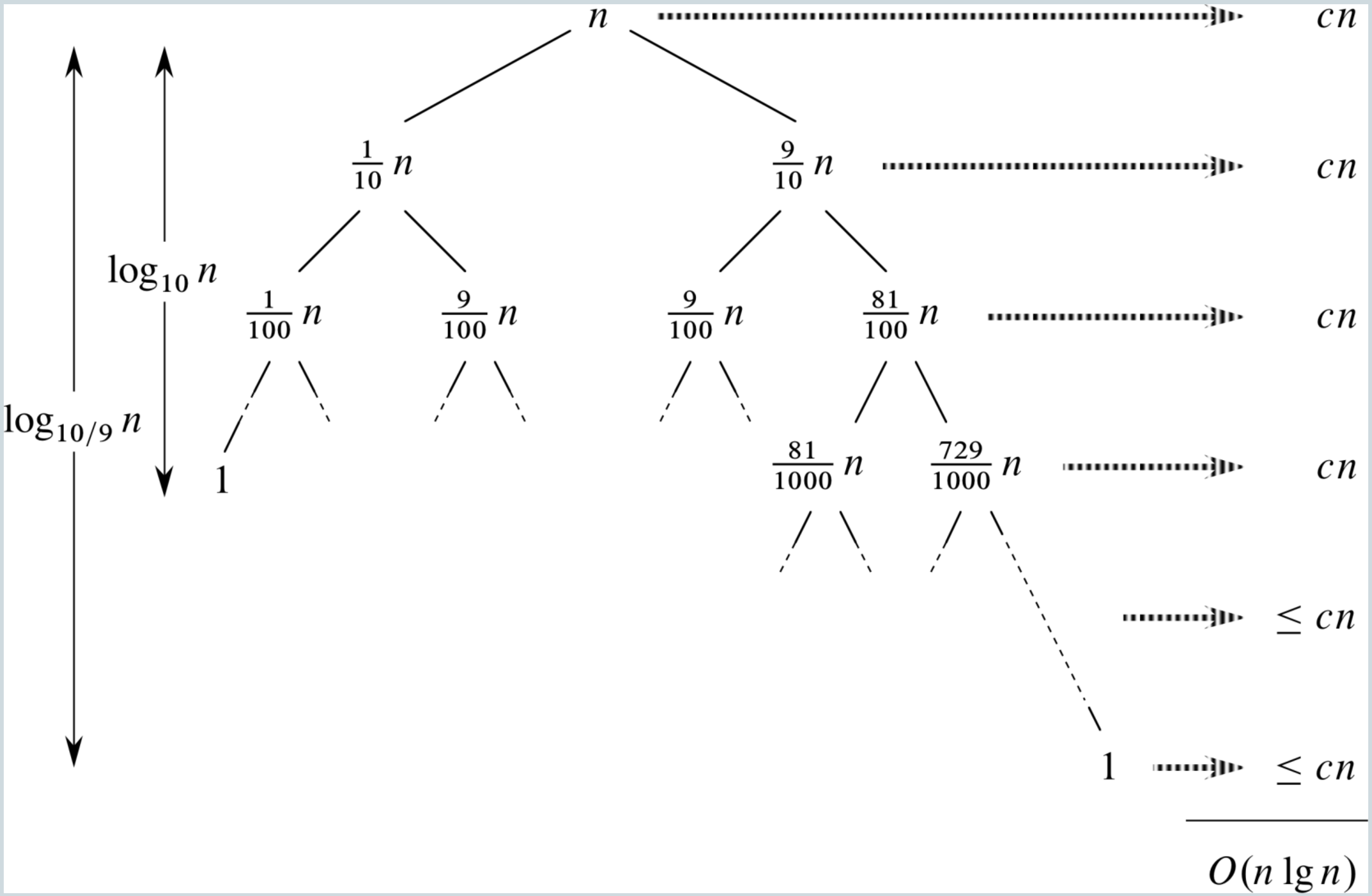

```
RANDOMIZED-QUICKSORT(A, p, r)
  if p < r
        q = RANDOMIZED-PARTITION(A, p, r);
        RANDOMIZED-QUICKSORT(A, p, q -1);
        RANDOMIZED-QUICKSORT(A, q+1,r);

RANDOMIZED-PARTITION(A, p, r)
   i = RANDOM(p, r);
   swap(A[r], A[i]);
  return PARTITION(A, p, r);
```

Analyzing Quicksort: Average Case

- Assuming random input, average-case running time is much closer to O(n lg n) than O($n\uparrow 2$)
- First, a more intuitive explanation/example:
  - Suppose that partition always produces a 9-to-1 split. This looks quite unbalanced!
  - The recurrence is thus:

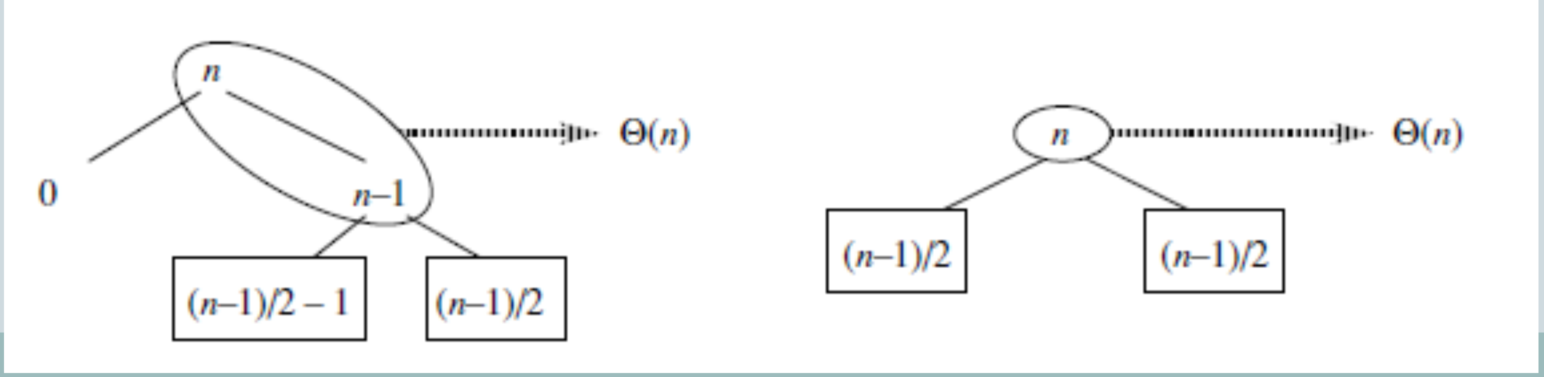
    T(n) = T(9n/10) + T(n/10) + n
  - How deep will the recursion go?

Recursion tree for Quicksort with 9-to-1 split



Analyzing Quicksort: Average Case

- Intuitively, a real-life run of quicksort will produce a mix of "bad" and "good" splits
  - Randomly distributed among the recursion tree
  - Pretend for intuition that they alternate between best-case (n/2 : n/2) and worst-case (n-1 : 1)
  - What happens if we bad-split root node, then good-split the resulting size (n-1) node?



Analyzing Quicksort: Average Case

- a real-life run of quicksort will produce a mix of "bad" and "good" splits
  - Randomly distributed among the recursion tree
  - Pretend for intuition that they alternate between best-case (n/2 : n/2) and worst-case (n-1 : 1)
  - 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 -1 = 2n -1 = O(n)
    - No worse than if we had good-split the root node!

Analyzing Quicksort: Average Case

- Intuitively, the O(n) cost of a bad split (or 2 or 3 bad splits) can be absorbed into the O(n) cost of each good split
- Thus running time of alternating bad and good splits is still O(n lg n), with slightly higher constants
- How can we be more strict?

Analyzing Quicksort: Average Case

- For simplicity, assume:
  - All inputs distinct (no repeats)
  - Slightly different partition() procedure
    - partition around a random element, which is not included in subarrays
    - all splits (0:n-1, 1:n-2, 2:n-3, … , n-1:0) equally likely
- What is the probability of a particular split happening?
- Answer: 1/n

Analyzing Quicksort: Average Case

- So partition generates splits
  (0:n-1, 1:n-2, 2:n-3, … , n-2:1, n-1:0)
  each with probability 1/n
- If T(n) is the expected running time,

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

- What is each term under the summation for?
- What is the Θ(n) term for?

Analyzing Quicksort: Average Case

- So…

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

$$= \frac{2}{n}\sum_{k=0}^{n-1} T(k) + \Theta(n)$$

  - Note: this is just like the book's recurrence [see also problem 7-3],

Analyzing Quicksort: Average Case

- We can solve this recurrence using the substitution method
  - Guess the answer
  - Assume that the inductive hypothesis holds
  - Substitute it in for some value $< n$
  - Prove that it follows for n

Analyzing Quicksort: Average Case

- We can solve this recurrence using the dreaded substitution method
  - Guess the answer
    - $T(n) = O(n \lg n)$
  - Assume that the inductive hypothesis holds
  - Substitute it in for some value < n
  - Prove that it follows for n

Analyzing Quicksort: Average Case

- We can solve this recurrence using the dreaded substitution method
  - Guess the answer
    - $T(n) = O(n \lg n)$
  - Assume that the inductive hypothesis holds
    - $T(n) \le an \lg n$ for some constants a
  - Substitute it in for some value < n
  - Prove that it follows for n

Analyzing Quicksort: Average Case

- We can solve this recurrence using the dreaded substitution method
  - Guess the answer
    - T(n) = O(n lg n)
  - Assume that the inductive hypothesis holds
    - T(n) <= an lg n   for some constants a
  - Substitute it in for some value < n
    - The value k in the recurrence
  - Prove that it follows for n

Analyzing Quicksort: Average Case

The recurrence to be solved

$$T(n) = \frac{2}{n}\sum_{k=0}^{n-1} T(k) + \Theta(n)$$

Plug in inductive hypothesis

$$\leq \frac{2}{n}\sum_{k=0}^{n-1} (ak\lg k) + \Theta(n)$$

Expand out the k=0 case

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

Note: leaving the same recurrence as the book

Analyzing Quicksort: Average Case

$$T(n) = \frac{2}{n}\sum_{k=1}^{n-1}\left(ak\lg k\right) + \Theta(n)$$

The recurrence to be solved

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

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

This summation at the end

Analyzing Quicksort: Average Case

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

The recurrence to be solved

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

We'll prove this later

$$= an \lg n - \frac{a}{4} n + \Theta(n)$$

Distribute the (2a/n) term

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

Remember, our goal is to get $T(n) \le an \lg n$

$$\le an \lg n$$

Pick a large enough that $an/4$ dominates $\Theta(n) <= cn$

Analyzing Quicksort: Average Case

- So T(n) =an lg n   for certain a
  - Thus the induction holds
  - Thus T(n) = O(n lg n)
  - Thus quicksort runs in O(n lg n) time on average
- Forgot something, the summation…see (or not) Appendix

Average-Case Analysis of Quicksort using Probability (Appendix C.2 -3, Ch 5)

- Let X = total number of comparisons performed in all calls to PARTITION:
- (k calls of Partition)

$$X = \sum_k X_k$$

- The total work done over the entire execution of Quicksort is

$$O(nc+X)=O(n+X)$$

- (at most n calls to partition)
- Need to estimate E(X)

Review of Probabilities

- **Definitions**

  - random experiment: an experiment whose result is not certain in advance (e.g., throwing a die)

  - outcome: the result of a random experiment

  - sample space: the set of all possible outcomes (e.g., {1,2,3,4,5,6})

  - event: a subset of the sample space (e.g., obtain an odd number in the experiment of throwing a die = {1,3,5})

Review of Probabilities

- **Probability of an event**

  - The likelihood that an event will occur if the underlying random experiment is performed

$$P(event) = \frac{number\ of\ \ favorable\ outcomes}{total\ number\ of\ \ possible\ outcomes}$$

Example: $P(obtain\ an\ odd\ number) = 3/6 = 1/2$

Random Variables

- Def.: (Discrete) random variable X: a function from a sample space S to the real numbers.
  - It associates a real number with each possible outcome of an experiment.

X(j)

Computing Probabilities Using Random Variables

- Example: consider the experiment of throwing a pair of dice

Define the r.v. X="sum of dice"

$X = x$ corresponds to the event $A_x=\{s \in S / X(s) = x\}$

(e.g., $X = 5$ corresponds to $A_5=\{(1,4),(4,1),(2,3),(3,2)\}$

$$P(X = x) = P(A_x) = \sum_{s:X(s)=x} P(s)$$

$$(P(X = 5) = P((1,4)) + P((4,1)) + P((2,3)) + P((2,3)) = 4/36 = 1/9)$$

Expectation

- Expected value (expectation, mean) of a discrete random variable X is:
- $$E[X] = \sum_{x} x * Pr\{X = x\}$$
  - "Average" over all possible values of random variable X

Example of finding expectation

- Consider a game in which you flip two fair coins. You earn \$3 for each head but lose \$2 for each tail. The expected value of the random variable X representing your earnings is

$$
\begin{aligned}
\mathrm{E}[X] &= 6 \cdot \Pr\{2 \text{ H's}\} + 1 \cdot \Pr\{1 \text{ H}, 1 \text{ T}\} - 4 \cdot \Pr\{2 \text{ T's}\} \\
&= 6(1/4) + 1(1/2) - 4(1/4) \\
&= 1 .
\end{aligned}
$$

Indicator Random Variables

- Given a sample space S and an event A, we define the indicator random variable I{A} associated with A:
  - $I\{A\} = \begin{cases} 1 & \text{if A occurs} \\ 0 & \text{if A does not occur} \end{cases}$
- The expected value of an indicator random variable X_A =I{A} is:
- $E[X_A] = \Pr\{A\}$
- Proof:
- $E[X_A] = E[I\{A\}] = 1 * \Pr\{A\} + 0 * \Pr\{\bar{A}\} = \Pr\{A\}$

Notation

z_2	z_9	z_8	z_3	z_5	z_4	z_1	z_6	z_{10}	z_7
2	9	8	3	5	4	1	6	10	7

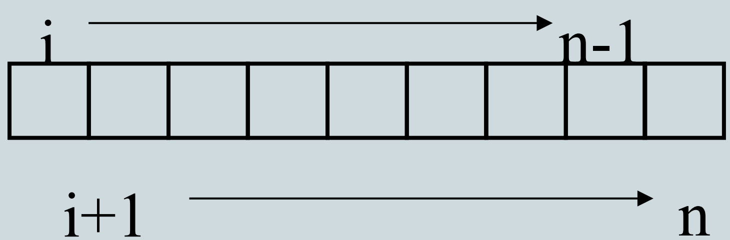
- Rename the elements of A as $z_1, z_2, \ldots, z_n$, with z_i being the <u>*i*-th smallest</u> element
- Define the set $Z_{ij} = \{z_i, z_{i+1}, \ldots, z_j\}$ the set of elements between z_i and z_j, inclusive

Total Number of Comparisons in PARTITION

Define X_{ij} = I {z_i is compared to z_j }

- Total number of comparisons X performed by the algorithm:

$$X = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} X_{ij}$$



Expected Number of Total Comparisons in PARTITION

- Compute the expected value of X:

$$E\left[\sum_{i=1}^{n-1}\sum_{j=i+1}^{n} X_{ij}\right] = \sum_{i=1}^{n-1}\sum_{j=i+1}^{n} E\left[X_{ij}\right] =$$

by linearity of expectation

indicator random variable

$E[X] =$

$$= \sum_{i=1}^{n-1}\sum_{j=i+1}^{n} \Pr\{z_i \text{ is compared to } z_j\}$$

the expectation of X_{ij} is equal to the probability of the event "z_i is compared to z_j"

Comparisons in PARTITION : Observation 1

- Each pair of elements is compared at most once during the entire execution of the algorithm
  - Elements are compared only to the pivot point!
  - Pivot point is excluded from future calls to PARTITION

Comparisons in PARTITION: Observation 2

- Only the pivot is compared with elements in both partitions!

z_2	z_9	z_8	z_3	z_5	z_4	z_1	z_6	z_{10}	z_7
2	9	8	3	5	4	1	6	10	7

$Z_{1,6} = \{1, 2, 3, 4, 5, 6\}$ $\{7\}$ $Z_{8,9} = \{8, 9, 10\}$

pivot

Elements between different partitions are <u>never</u> compared!

Comparisons in PARTITION

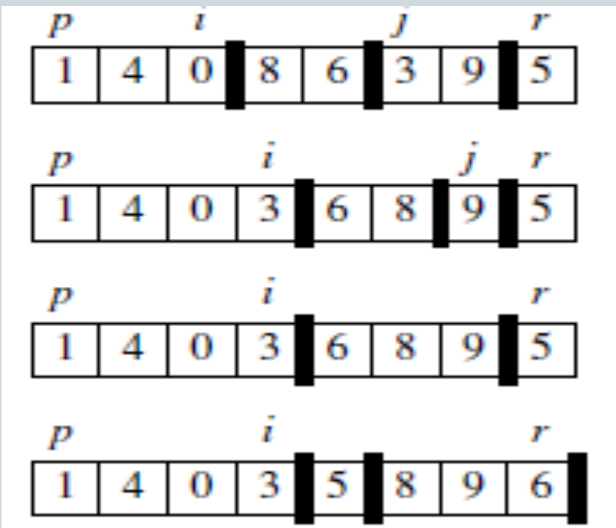
z_2	z_9	z_8	z_3	z_5	z_4	z_1	z_6	z_{10}	z_7
2	9	8	3	5	4	1	6	10	7

$$Z_{1,6} = \{1, 2, 3, 4, 5, 6\} \qquad \{7\} \qquad Z_{8,9} = \{8, 9, 10\}$$

$$\Pr\{z_i \text{ is compared to } z_j\}?$$

- Case 1: pivot chosen such as: $z_i < x < z_j$
  - zi and zj will never be compared
- Case 2: zi or zj is the pivot
  - zi and zj will be compared
  - only if one of them is chosen as pivot before any other element in range zi to zj

This is why



z_2 z_4 z_1 z_3 z_5 z_7 z_8 z_6

z2 will never be compared with z6 since z5 (which belongs to [z_2, z_6]) was chosen as a pivot first !

Probability of comparing zi with zj

$\Pr\{ z_i \text{ is compared to } z_j \} =$

$\Pr\{ z_i \text{ is the first pivot chosen from } Z_{ij} \}$ $+$

$\Pr\{ z_j \text{ is the first pivot chosen from } Z_{ij} \}$

$= 1/(j - i + 1) + 1/(j - i + 1) = 2/(j - i + 1)$

- There are $j – i + 1$ elements between z_i and z_j

– Pivot is chosen randomly and independently

– The probability that any particular element is the first one chosen is $1/(j - i + 1)$

Number of Comparisons in PARTITION

xpected number of comparisons in PARTITION:

$$E[X] = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \Pr\{z_i \text{ is compared to } z_j\}$$

$$E[X] = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{2}{j-i+1} = \sum_{i=1}^{n-1} \sum_{k=1}^{n-i} \frac{2}{k+1} < \sum_{i=1}^{n-1} \sum_{k=1}^{n} \frac{2}{k} = \sum_{i=1}^{n-1} O(\lg n)$$

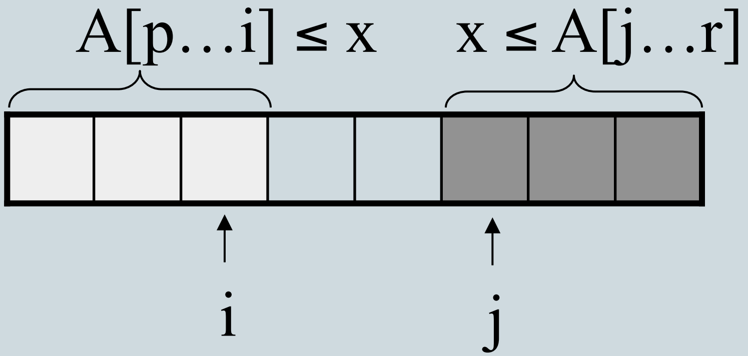
(set k=j-i) (harmonic series)

$$= O(n \lg n)$$

$\Rightarrow$ Expected running time of Quicksort using RANDOMIZED-PARTITION is O(nlgn)

Revisit Partitioning

- Hoare's partition
  - Select a pivot element x around which to partition
  - Grows two regions
- A[p…i] ≤ x
- x ≤ A[j…r]



Ex. Quicksort

- Sort the array below to ascending order using quick sort. Pivot: last element
- [45 16 8 32 10 6 33 29]

Appendix (skim): Tightly Bounding The Key Summation

$$\sum_{k=1}^{n-1} k \lg k = \sum_{k=1}^{\lceil n/2 \rceil - 1} k \lg k + \sum_{k=\lceil n/2 \rceil}^{n-1} k \lg k$$

Split the summation for a tighter bound

$$\le \sum_{k=1}^{\lceil n/2 \rceil - 1} k \lg k + \sum_{k=\lceil n/2 \rceil}^{n-1} k \lg n$$

The* lg *k* *in the second term is bounded by* lg *n

$$= \sum_{k=1}^{\lceil n/2 \rceil - 1} k \lg k + \lg n \sum_{k=\lceil n/2 \rceil}^{n-1} k$$

Move the* lg *n* *outside the summation

Tightly Bounding
The Key Summation

$$\sum_{k=1}^{n-1} k \lg k \le \sum_{k=1}^{\lceil n/2 \rceil - 1} k \lg k + \lg n \sum_{k=\lceil n/2 \rceil}^{n-1} k$$

The summation bound so far

$$\le \sum_{k=1}^{\lceil n/2 \rceil - 1} k \lg(n/2) + \lg n \sum_{k=\lceil n/2 \rceil}^{n-1} k$$

The* lg *k in the first term is bounded by* lg *n/2

$$= \sum_{k=1}^{\lceil n/2 \rceil - 1} k(\lg n - 1) + \lg n \sum_{k=\lceil n/2 \rceil}^{n-1} k$$

lg *n/2* = lg *n* - 1

$$= (\lg n - 1) \sum_{k=1}^{\lceil n/2 \rceil - 1} k + \lg n \sum_{k=\lceil n/2 \rceil}^{n-1} k$$

Move (*lg *n* - 1*) outside the summation

Tightly Bounding The Key Summation

$$\sum_{k=1}^{n-1} k \lg k \le (\lg n - 1) \sum_{k=1}^{\lceil n/2 \rceil - 1} k + \lg n \sum_{k=\lceil n/2 \rceil}^{n-1} k$$

The summation bound so far

$$= \lg n \sum_{k=1}^{\lceil n/2 \rceil - 1} k - \sum_{k=1}^{\lceil n/2 \rceil - 1} k + \lg n \sum_{k=\lceil n/2 \rceil}^{n-1} k$$

Distribute the (lg n - 1)

$$= \lg n \sum_{k=1}^{n-1} k - \sum_{k=1}^{\lceil n/2 \rceil - 1} k$$

The summations overlap in range; combine them

$$= \lg n \left(\frac{(n-1)(n)}{2} \right) - \sum_{k=1}^{\lceil n/2 \rceil - 1} k$$

The Guassian sum

Tightly Bounding
The Key Summation

$$\sum_{k=1}^{n-1} k \lg k \le \left(\frac{(n-1)(n)}{2}\right)\lg n - \sum_{k=1}^{\lceil n/2 \rceil - 1} k$$

The summation bound so far

$$\le \frac{1}{2}\left[n(n-1)\right]\lg n - \sum_{k=1}^{n/2-1} k$$

Rearrange first term, place upper bound on second

$$\le \frac{1}{2}\left[n(n-1)\right]\lg n - \frac{1}{2}\left(\frac{n}{2}\right)\left(\frac{n}{2}-1\right)$$

X Guassian sum

$$\le \frac{1}{2}\left(n^2 \lg n - n \lg n\right) - \frac{1}{8}n^2 + \frac{n}{4}$$

Multiply it all out

Tightly Bounding, The Key Summation

$$\sum_{k=1}^{n-1} k \lg k \le \frac{1}{2}\left(n^2 \lg n - n \lg n\right) - \frac{1}{8}n^2 + \frac{n}{4}$$

$$\le \frac{1}{2}n^2 \lg n - \frac{1}{8}n^2 \text{ when } n \ge 2$$

Done!!!