CS3230 Lecture 4 (revised)



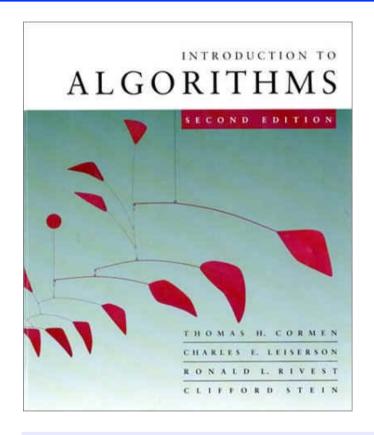
"A Review of Sorting, Quicksort Analysis, and Augmenting Data Structures"

- ☐ Lecture Topics and Readings
 - (Quick Review) of Sorting Methods [CLRS]-C?
 - Quicksort and Randomized QS [CLRS]-C7
 - **❖** Augmenting Data Structures [CLRS]-C14

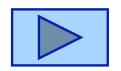
Creative View of Sorting Methods
Quicksort (only 40% sub-optimal)
Be more aware of augmenting Data Structure

(CS3230 Outline) Page 1

[CLRS]...



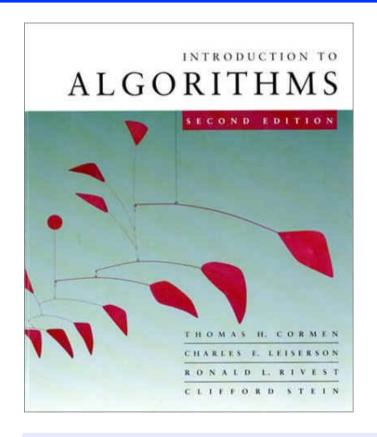




Sabbatical leave at NUS Computer Science Dept 1995/96

[CLRS] & Charles Leiserson.

[CLRS] @500K

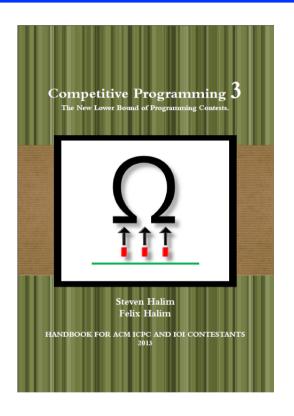




[CLRS]-90, [CLRS]-01, [CLRS]-09 Celebrating 500,000 copies sold

(CS3230 Outline) Page 3

[HH2013]... 3rd edition





Steven Halim



Felix Halim

[HH13] *Competitive Programming*, (3rd edition) by Steven Halim and Felix Halim, 2013.

(CS3230 Outline) Page 4

Antony Hoare (1934 –)



Invented Quicksort (at age 26)

Developed Hoare's Logic (for program correctness)

Developed CSP (including dining philisophers' problem)

Quote: (about difficulties of creating software systems)

"There are two ways of constructing a software design: One way is to make it so simple that there are obviously no deficiencies, and the other way is to make it so complicated that there are no obvious deficiencies. The first method is far more difficult."



Tony Hoare, Singapore 2008 Computing in the 21st Century

- Turing Award, 1980
- Knighted, 2000

Thank you.

QLA



CS3230 Lecture 4



"A Review of Sorting, Lower Bounds, and Sorting in Linear Time"

☐ Lecture Topics and Readings

(Quick Review)	of Sorting Methods	[CLRS]-C?
----------------	--------------------	-----------

Quicksort and Randomized QS [CLRS]-C7

❖ Lower Bound for Sorting [CLRS]-C8

❖ Sorting in Linear Time [CLRS]-C8

Creative Review of Sorting,
Lower Bound and Optimal Sorting,
Busting the Lower Bound

(CS3230 Review of Sorting) Page 1



Sorting Animation: by Steven Halim & students

http://www.comp.nus.edu.sg/~stevenha/visualization/sorting.html



The problem of sorting

Input: sequence $\langle a_1, a_2, ..., a_n \rangle$ of numbers.

Output: permutation $\langle a'_1, a'_2, ..., a'_n \rangle$ such that $a'_1 \le a'_2 \le \cdots \le a'_n$.

Example: *Input:* 8 2 4 9 3 6

Output: 2 3 4 6 8 9

Sorting Animation: by Steven Halim & students

http://www.comp.nus.edu.sg/~stevenha/visualization/sorting.html

Sorting: Problem and Algorithms

Problem: Sorting

Given a list of *n* numbers, sort them

Algorithms:

❖ Selection Sort $\Theta(n^2)$

* Insertion Sort $\Theta(n^2)$

❖ Bubble Sort $\Theta(n^2)$

❖ Merge Sort $\Theta(n \lg n)$

❖ Quicksort $\Theta(n \lg n)^*$

* average case

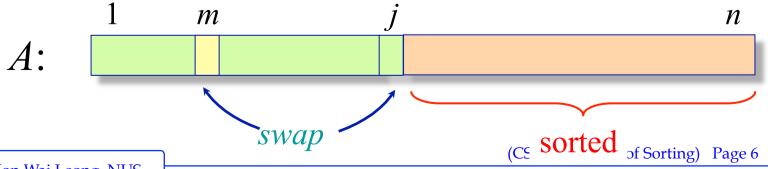


Selection Sort Algorithm «

Recall from Lecture 2

```
SELECTION-SORT (A, n) \triangleright A[1 ... n]
j \leftarrow n
while j > 1
do m \leftarrow Find-Max (A, j)
Swap (A[m], A[j])
j \leftarrow j - 1
Let's make this recursive.
```

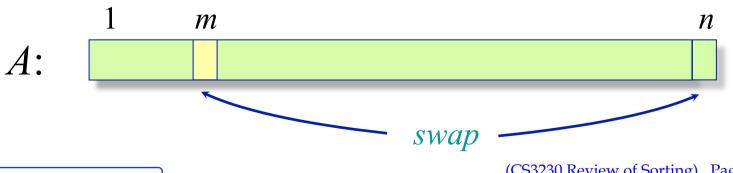
A[m] is largest among A[1..j]



Recursive Selection Sort

```
SELECTION-SORT-R (A, n) \triangleright A[1 ... n]
  if n = 1 then return
  m \leftarrow \text{Find-Max}(A, n)
  Swap (A[m], A[n])
  SELECTION-SORT-R (A, n-1)
```

A[m] is largest among A[1..n]



Hon Wai Leong, NUS

Recursive Selection Sort

```
Selection-Sort-R (A, n) \triangleright A[1 ... n]

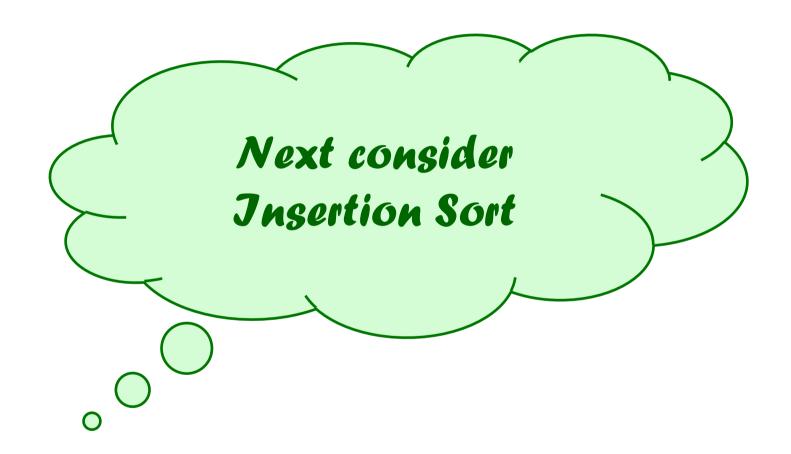
if n = 1 then return

m \leftarrow \text{Find-Max } (A, n)

Swap (A[m], A[n])

Selection-Sort-R (A, n-1)
```

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1; \\ T(n-1) + \Theta(n) & \text{if } n > 1. \end{cases}$$





Insertion sort

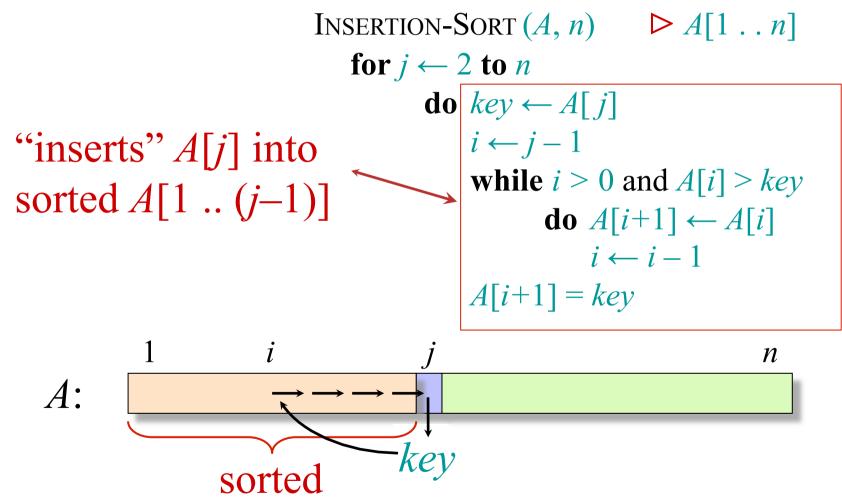


"pseudocode"

INSERTION-SORT (A, n) \triangleright A[1 ... n]for $j \leftarrow 2$ to ndo $key \leftarrow A[j]$ $i \leftarrow j - 1$ while i > 0 and A[i] > keydo $A[i+1] \leftarrow A[i]$ $i \leftarrow i - 1$ A[i+1] = key



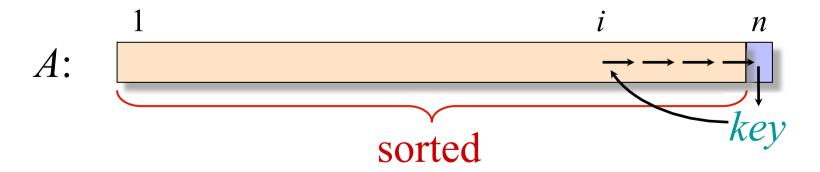
Insertion sort

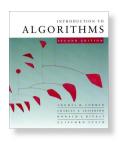




Recursive Insertion sort

```
Insertion-Sort-R (A, n) \triangleright A[1 ... n]
if n = 1 then return
Insertion-Sort-R (A, n-1)
insert A[n] into sorted A[1 ... n-1]
```

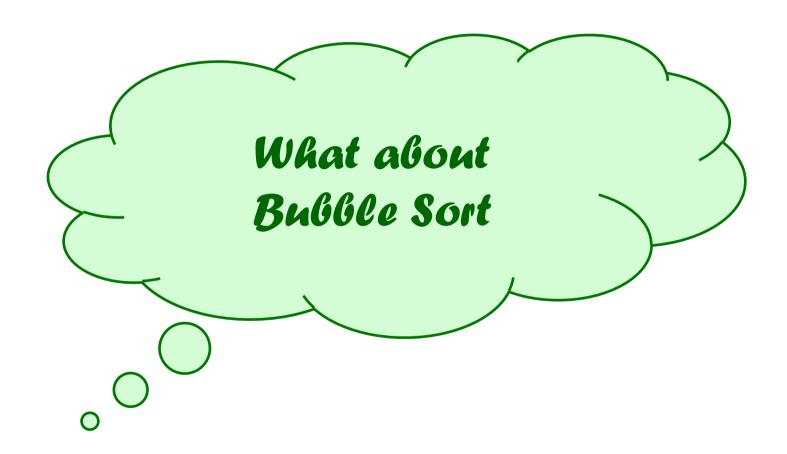




Recursive Insertion sort

```
INSERTION-SORT-R (A, n) \triangleright A[1 ... n]
if n = 1 then return
INSERTION-SORT-R (A, n-1)
insert A[n] into sorted A[1 ... n-1]
```

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1; \\ T(n-1) + \Theta(n) & \text{if } n > 1. \end{cases}$$



Recursive Bubble Sort

```
Bubble-Sort-R (A, n) \triangleright A[1 ... n]

if n = 1 then return

One bubble-phase on A[1 ... n]

Bubble-Sort-R (A, n-1)
```

Recursive Bubble Sort

Bubble-Sort-R
$$(A, n) \triangleright A[1 ... n]$$

if $n = 1$ then return
One bubble-phase on $A[1 ... n]$
Bubble-Sort-R $(A, n-1)$

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1; \\ T(n-1) + \Theta(n) & \text{if } n > 1. \end{cases}$$

All have the *same* recurrence

Selection Sort, Insertion Sort, Bubble Sort

Selection Sort, insertion Sort, Bubb
$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1; \\ T(n-1) + \Theta(n) & \text{if } n > 1. \\ (n-1), 0 \end{cases}$$
Extreme imbalance

All have the same recurrence

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1; \\ T(n-1) + \Theta(n) & \text{if } n > 1. \end{cases}$$

How to "solve" this recurrence

Answer: Use TELESCOPING

How to "solve" this recurrence

Answer: Use TELESCOPING

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

$$= cn + c(n-1) + T(n-2)$$

$$= cn + c(n-1) + c(n-2) + T(n-3)$$

$$= cn + c(n-1) + c(n-2) + \dots + c^{2} + T^{2}$$

$$= cn + c(n-1) + c(n-2) + \dots + c^{2} + c^{2}$$

$$= c + c(n-1) + c(n-2) + \dots + c^{2} + c^{2}$$

$$= c + c(n-1) + (n-2) + \dots + c^{2} + c^{2}$$

Observation:

Selection Sort, Insertion Sort, Bubble Sort

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1; \\ T(n-1) + \Theta(n) & \text{if } n > 1. \end{cases}$$

They all have running time: $T(n) = \Theta(n^2)$

Imbalance in Divide & Conquer algorithms produces inefficient algorithms



Merge sort (Perfect balance)

MERGE-SORT $A[1 \dots n]$

- 1. If n = 1, done.
- 2. Recursively sort A[1..[n/2]] and A[[n/2]+1..n].
- 3. "Merge" the 2 sorted lists.

$$T(n) = \begin{cases} \Theta(1) \text{ if } n = 1; \\ 2T(n/2) + \Theta(n) \text{ if } n > 1. \end{cases}$$

M-Thm:
$$a = 2$$
, $b = 2 \implies n^{\log_b a} = n^{\log_2 2} = n$
 \Rightarrow Case 2 $(k = 0) \Rightarrow T(n) = \Theta(n \lg n)$.

(CS3230 Review of Sorting) Page 22

What about Heapsort, Quicksort

```
Heapsort \Theta(n \lg n) builds a data structure – Heap \Theta(n) sort efficiently using the Heap \Theta(n \lg n)
```

Quicksort

Partitions array about a pivot $\Theta(n)$

Recursively sort each partition O(??)

How balanced is QuickSort? (We'll see in next section)

view or sorting) Page 23

Thank you.

Q&A



CS3230 Lecture 4



"A Review of Sorting, Lower Bounds, and Sorting in Linear Time"

- ☐ Lecture Topics and Readings
 - * (Quick Review) of Sorting Methods [CLRS]-C?
 - **❖ Quicksort and Randomized QS** [CLRS]-C7
 - ***** Lower Bound for Sorting [CLRS]-C8
 - **❖ Sorting in Linear Time** [CLRS]-C8

Creative Review of Sorting,
Lower Bound and Optimal Sorting,
Busting the Lower Bound

(CS3230 Quicksort Analysis) Page 1



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

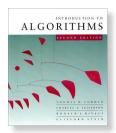
Quicksort an *n*-element array:

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

```
\leq x x \geq x
```

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

Key: Linear-time partitioning subroutine.



Partitioning subroutine

```
Partition(A, p, q) \triangleright A[p \dots q]
    x \leftarrow A[p] \qquad \triangleright \text{pivot} = A[p]
                                                      Running time
    i \leftarrow p
                                                       = O(n) for n
    for j \leftarrow p + 1 to q
                                                       elements.
         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
Invariant:
                                              \geq \chi
                             \leq \chi
```

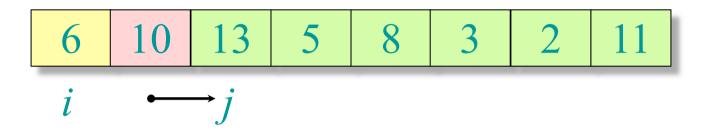


Example of partitioning

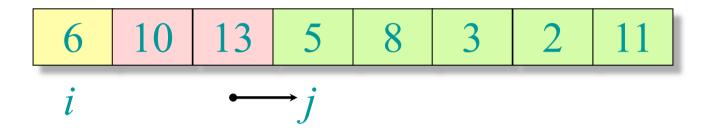




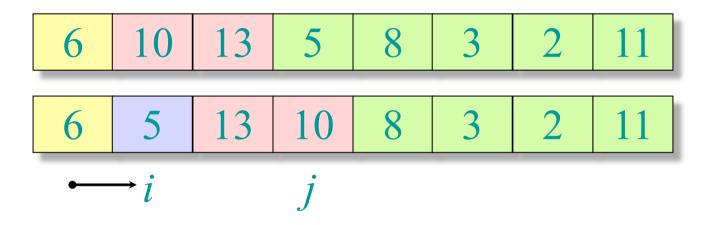
Example of partitioning



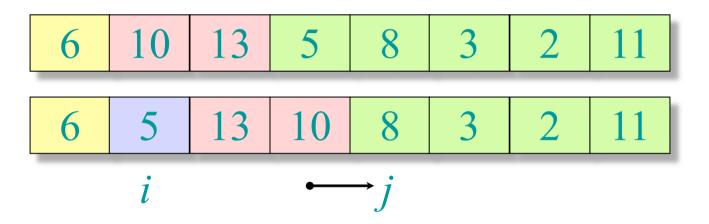




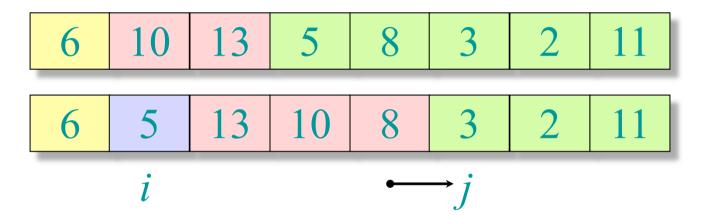




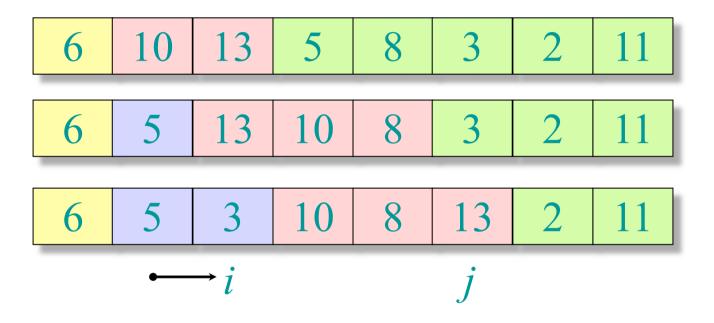




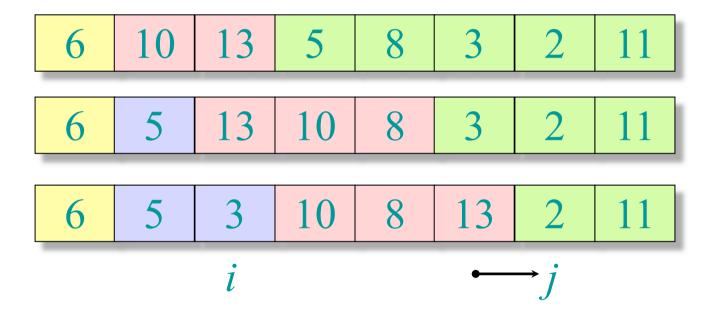




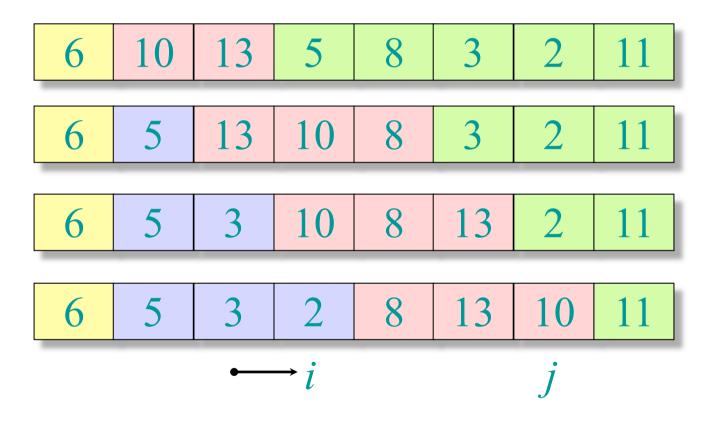




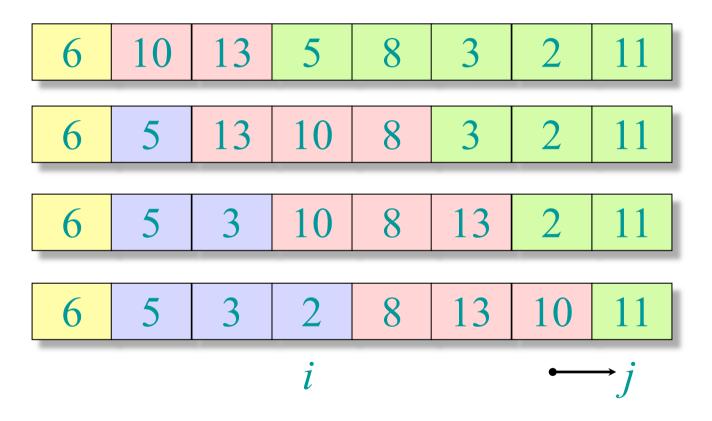




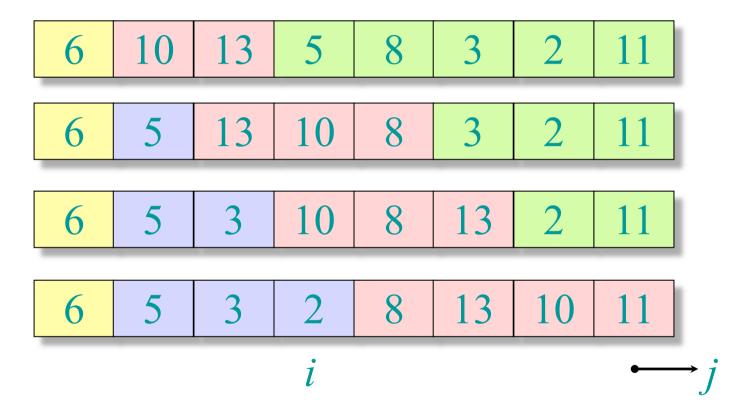


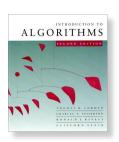


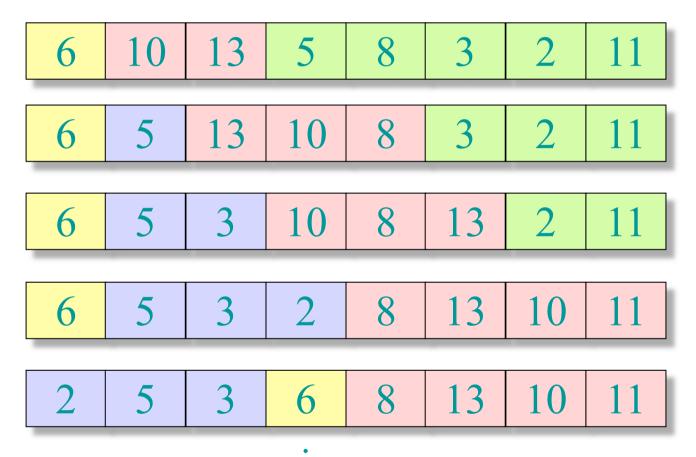














Pseudocode for quicksort

```
Quicksort(A, p, r)

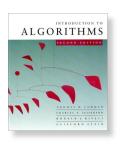
if p < r

then q \leftarrow \text{Partition}(A, p, r)

Quicksort(A, p, q-1)

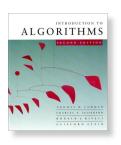
Quicksort(A, p, q+1, r)
```

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.

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

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

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

$$= \Theta(n^2) \qquad (arithmetic series)$$



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



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

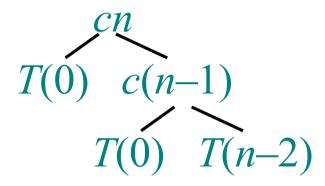


$$T(n) = T(0) + T(n-1) + cn$$

$$T(0)$$
 $T(n-1)$

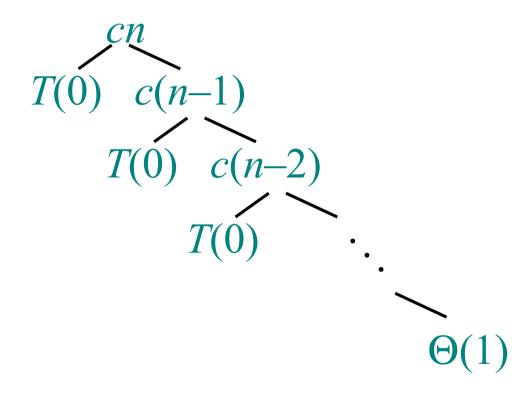


$$T(n) = T(0) + T(n-1) + cn$$



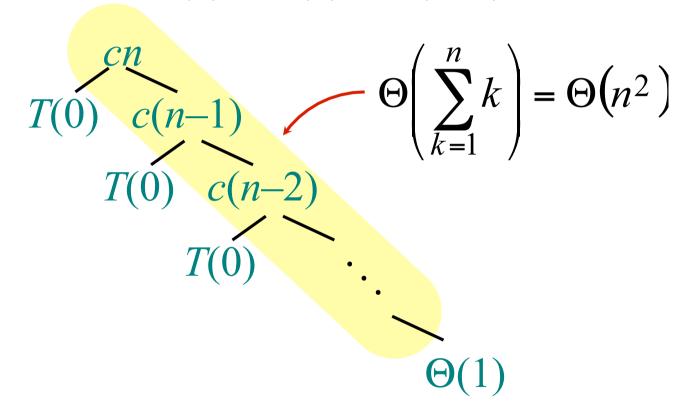


$$T(n) = T(0) + T(n-1) + cn$$



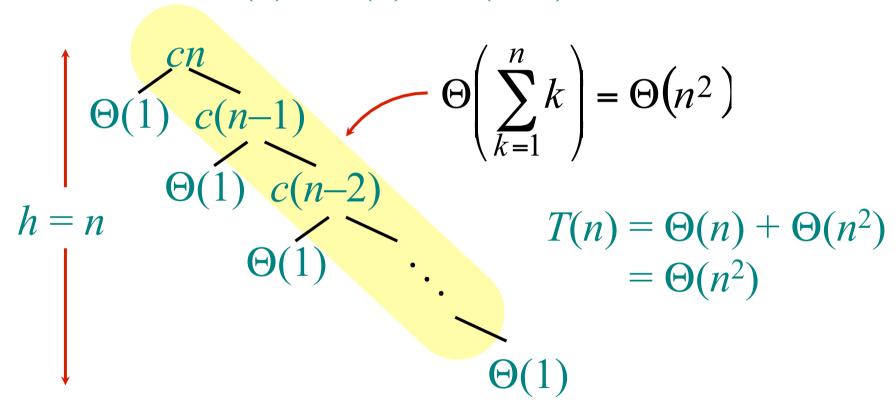


$$T(n) = T(0) + T(n-1) + cn$$





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





Best-case analysis

(For intuition only!)

If we're lucky, Partition splits the array evenly:

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

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

What if the split is always $\frac{1}{10}$: $\frac{9}{10}$?

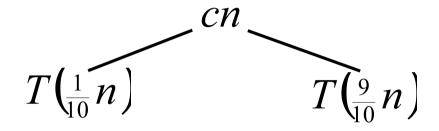
$$T(n) = T\left(\frac{1}{10}n\right) + T\left(\frac{9}{10}n\right) + \Theta(n)$$

What is the solution to this recurrence?

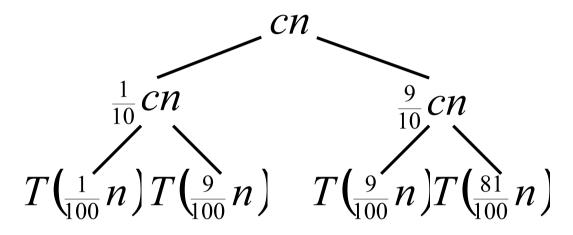


T(n)

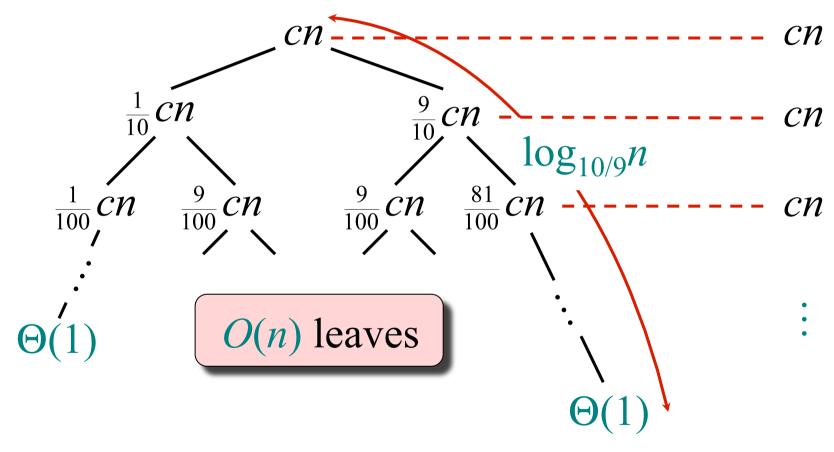




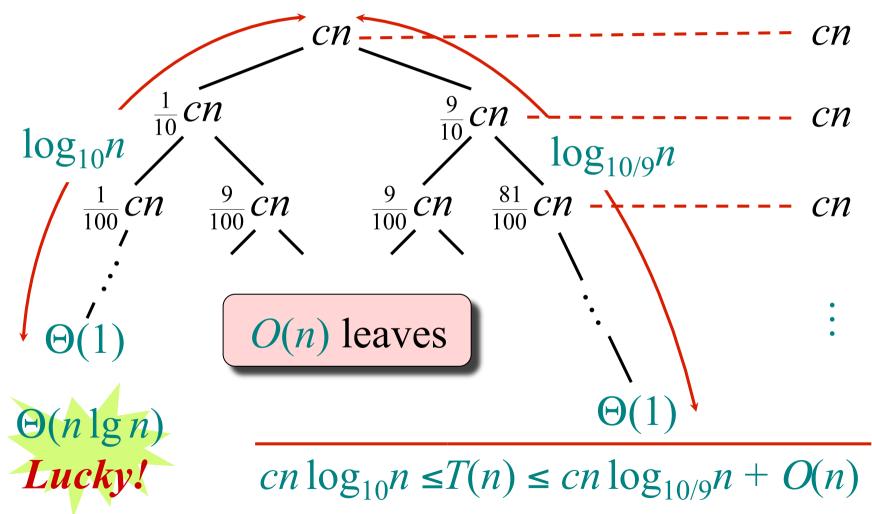














More intuition

Suppose we alternate lucky, unlucky, lucky, unlucky, lucky,

$$L(n) = 2U(n/2) + \Theta(n)$$
 lucky
 $U(n) = L(n-1) + \Theta(n)$ unlucky

Solving:

$$L(n) = 2(L(n/2 - 1) + \Theta(n/2)) + \Theta(n)$$

$$= 2L(n/2 - 1) + \Theta(n)$$

$$= \Theta(n \lg n) \quad Lucky!$$

How can we make sure we are usually lucky?



Randomized quicksort

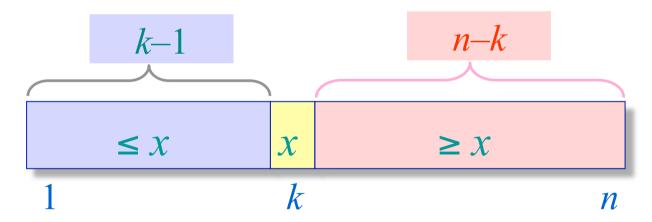
IDEA: Partition around a *random* element.

- 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.

Let T(n) = the *average* time taken to sort an array of size n using Quicksort

If pivot x ends up in position k,

then
$$T(n) = T(k-1) + T(n-k) + (n+1)$$



Prob(pivot is at pos k) = 1/n for all k

(CS3230 Quicksort Analysis) Page 35

Then, we have

$$T(n) = \begin{cases} T(0) + T(n-1) + (n+1) & \text{if } 0: n-1 \text{ split} \\ T(1) + T(n-2) + (n+1) & \text{if } 1: n-2 \text{ split} \\ T(2) + T(n-3) + (n+1) & \text{if } 2: n-3 \text{ split} \\ \vdots & \vdots & \vdots \\ T(n-1) + T(0) + (n+1) & \text{if } n-1: 0 \text{ split} \end{cases}$$

Prob(pivot is at pos k) = 1/n for all k

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

(CS3230 Quicksort Analysis) Page 36

Then, we have the following recurrence:

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

\rightarrow Expand the summations

Then, we have the following recurrence:

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

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

$$nT(n) = 2(T(0) + T(1) + ... + T(n-1)) + n(n+1)$$

→ Get rid of dependence on "full history"

Then, we get rid of "full history":

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

$$nT(n) = 2[T(0) + T(1) + \dots + T(n-2) + T(n-1)] + n(n+1)$$

$$(n-1)T(n-1) = 2[T(0) + T(1) + \dots + T(n-2)] + (n-1)n$$

$$nT(n) = (n+1)T(n-1) + 2n$$

 \rightarrow Divide by n(n+1)... (make it telescopic)

Divide by n(n+1)... (make it telescopic)

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

$$nT(n) = (n+1)T(n-1) + 2n$$

$$\frac{T(n)}{n+1} = \frac{T(n-1)}{n} + \frac{2}{n+1}$$

→ Now "telescope"...

Now, telescope...

$$\frac{T(n)}{(n+1)} = \frac{2}{(n+1)} + \frac{T(n-1)}{(n)}$$

$$= \frac{2}{(n+1)} + \frac{2}{(n)} + \frac{T(n-2)}{(n-1)}$$

$$= \frac{2}{(n+1)} + \frac{2}{(n)} + \frac{2}{(n-1)} + \frac{T(n-3)}{(n-2)}$$

$$= \frac{2}{(n+1)} + \frac{2}{(n)} + \frac{2}{(n-1)} + \dots + \frac{2}{(3)} + \frac{T(1)}{(2)}$$

$$\frac{T(n)}{(n+1)} = \frac{T(1)}{(2)} + 2\left[\frac{1}{(n+1)} + \frac{1}{(n)} + \frac{1}{(n-1)} + \dots + \frac{1}{(3)}\right]$$

(CS3230 Quicksort Analysis) Page 41

Then, we have the following recurrence:

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

$$\frac{T(n)}{(n+1)} = \frac{T(1)}{(2)} + 2\left[\frac{1}{(n+1)} + \frac{1}{(n)} + \frac{1}{(n-1)} + \dots + \frac{1}{(3)}\right]$$

$$T(n) = 2(n+1)H(n+1) + O(n)$$

$$H(n) = \sum_{k=1}^{n} \frac{1}{k}$$
 is the Harmonic series

Analysis of Randomized Quicksort

Avg running time of Randomized Quicksort:

$$T(n) = 2(n+1)H(n+1) + O(n)$$

$$H(n) = \ln n + O(1)$$
 [CLRS] - App.A

$$T(n) = 2(n+1)\ln n + O(n)$$

$$T(n) = 1.386n\lg n + O(n)$$

Randomized Quicksort is only 38.6% from optimal.

Optimal sorting is $T^*(n) = (n \lg n)$ [See L.B. for Sorting]

Recap...

Beautiful analysis of Randomized Quicksort to get...

$$T(n) = 1.386n\lg n + O(n)$$

Not that difficult, right?

Where are the key steps?

- Get rid of full history
- * Telescope

Recap: The Key Steps

This recurrence depends on full history

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

Step 1: Get rid of full history... to get

$$n \cdot T(n) = (n+1)T(n-1) + 2n$$

Step 2: Get to a form that can telescope...

$$\frac{T(n)}{(n+1)} = \frac{T(n-1)}{(n)} + \frac{2}{(n+1)}$$

Using the result...

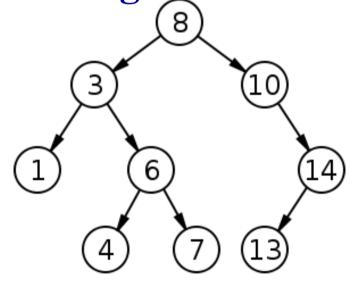
Using a similar analysis, we can show...

□ For a randomly built *n*-node BST (binary search tree), the expected height is

❖ 1.386 lg *n*

□ Try it out yourself...

Or read [CLRS]-C12.4



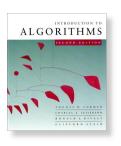


Randomized quicksort analysis [by CLRS]

[CLRS] uses a slightly different 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, ..., n-1, define the *indicator* random variable



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.

Thank you.

QLA



CS3230 Lecture 4 (revised)



"A Review of Sorting, Quicksort Analysis, and Augmenting Data Structures"

- ☐ Lecture Topics and Readings
 - * (Quick Review) of Sorting Methods [CLRS]-C?
 - **❖ Quicksort and Randomized QS** [CLRS]-C7
 - **❖** Augmenting Data Structures [CLRS]-C14

Creative View of Sorting Methods
Quicksort (only 40% sub-optimal)
Be more aware of augmenting Data Structure

Augmenting Data Structures

- **□** Why augment a data structure?
 - * When "standard" data structures are not adequate
 - **Need to support** *more* operations *efficiently*
- □ Readings: [CLRS]-C14

Note: For CS3230 Spring 2014, we use AVL tree (instead of the Red-Black tree) as our balanced BST.

So, when reading the notes and textbook, replace all references to "Red-Black tree" with "AVL tree".

Dynamic order statistics

OS-Select(i, S): returns the ith smallest element

in the dynamic set S.

OS-RANK(x, S): returns the rank of $x \in S$ in the

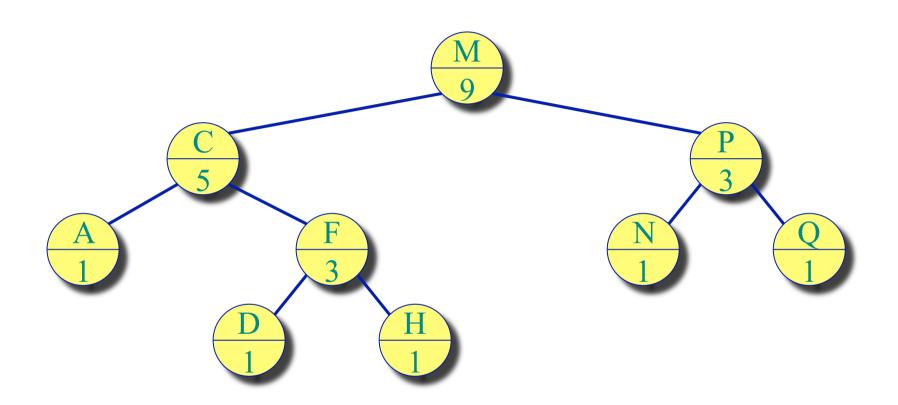
sorted order of S's elements.

IDEA: Use an AVL tree for the set *S*, but keep subtree sizes in the nodes.

Notation for each node: & balance (not shown)



Example of an OS-tree



$$size[x] = size[left[x]] + size[right[x]] + 1$$

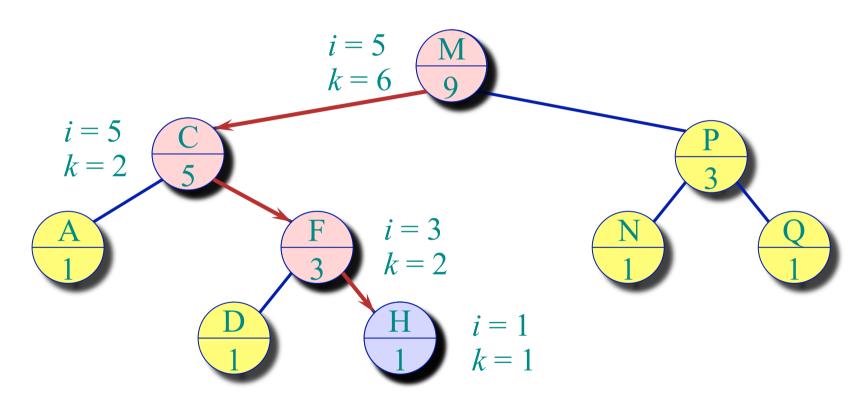
Selection

```
Implementation trick: Use a sentinel
(dummy record) for NIL such that size[NIL] = 0.
OS-Select(x, i) > ith smallest element in the
                     subtree rooted at x
  k \leftarrow size[left[x]] + 1   \triangleright k = rank(x)
   if i = k then return x
   if i < k
      then return OS-SELECT(left[x], i)
      else return OS-SELECT(right[x], i - k)
```

(OS-RANK is in the textbook.)

Example

OS-SELECT(*root*, 5)



Running time = $O(h) = O(\lg n)$ for AVL trees.

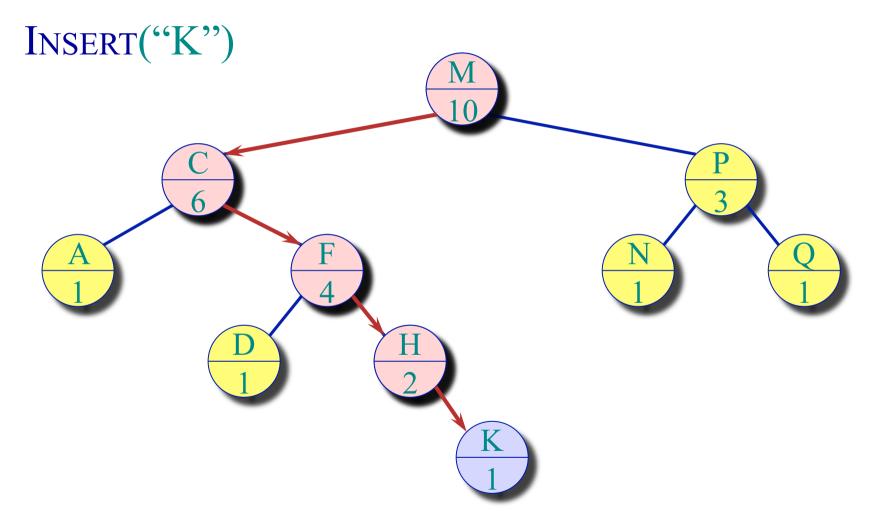
Data structure maintenance

- Q. Why not keep the ranks themselves in the nodes instead of subtree sizes?
- A. They are hard to maintain when the AVL tree is modified.

Modifying operations: Insert and Delete.

Strategy: Update subtree sizes when inserting or deleting.

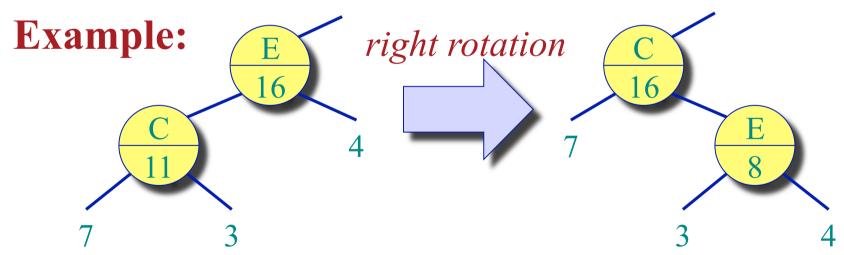
Example of insertion



Handling rebalancing (updated)

Don't forget that AVL-INSERT and AVL-DELETE may also need to modify the AVL tree in order to maintain AVL tree size & balance.

• *Rotations*: fix up subtree sizes in O(1) time & update balance of nodes affected



 \therefore AVL-Insert & AVL-Delete still run in $O(\lg n)$ time.

Data-structure augmentation

Methodology: (e.g., order-statistics trees)

- 1. Choose an underlying data structure (AVL trees).
- 2. Determine additional information to be stored in the data structure (*subtree sizes*).
- 3. Verify that this information can be maintained for modifying operations (*AVL-INSERT, AVL-DELETE don't forget rotations*).
- 4. Develop new dynamic-set operations that use the information (*OS-SELECT and OS-RANK*).
- These steps are guidelines, not rigid rules.

Augmenting Data Structures

- **□** Optional Readings:
 - * Read up [CLRS] C14.3 Interval Trees
- **□** Homework:
 - **Try R-problem:** [CLRS] Ex 14.1-1, 14.1-2

Thank you.

Q & A

