Data Structures and Algorithms

Prof. Ganesh Ramakrishnan,
Prof. Ajit Diwan,
Prof. D.B. Phatak

Department of Computer Science and Engineering
IIT Bombay

Session: Quick-Sort

QuickSort: Sorting by Randomized Divide and Conquer

- Pick a random (pivot) element x and **Divide** the n-element sequence to be sorted into two sub-sequences L and G
  - L has elements less than x
  - G has elements greater than x
  - E has elements equal to x
- Sort the two subsequences L and G recursively using merge sort.
- Merge the sorted subsequences L, E and G to produce the sorted answer.

Quick-Sort Execution Tree

$9, 7, 2, \underline{\mathbf{6}}, 8, 1, 4 \rightarrow$ ___________

Quick-Sort Execution Tree

$9, 7, 2, \underline{\mathbf{6}}, 8, 1, 4 \to$

$2, 1, \underline{\mathbf{4}} \to$

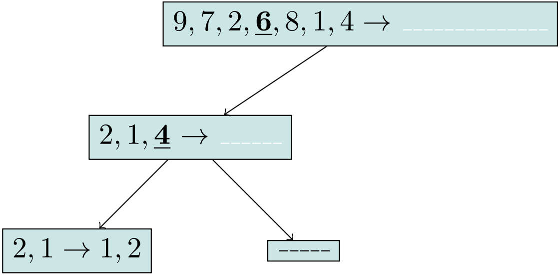
Quick-Sort Execution Tree

$9, 7, 2, \underline{\mathbf{6}}, 8, 1, 4 \rightarrow$ ____________

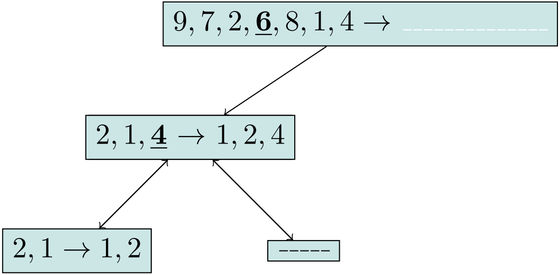
$2, 1, \underline{\mathbf{4}} \rightarrow$ ______

$2, 1 \rightarrow 1, 2$

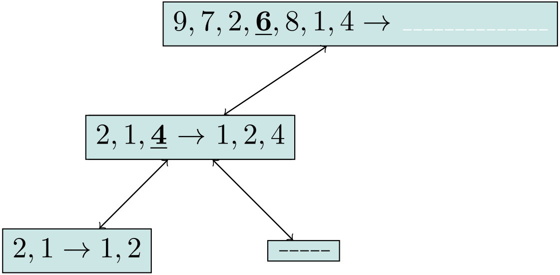
Quick-Sort Execution Tree



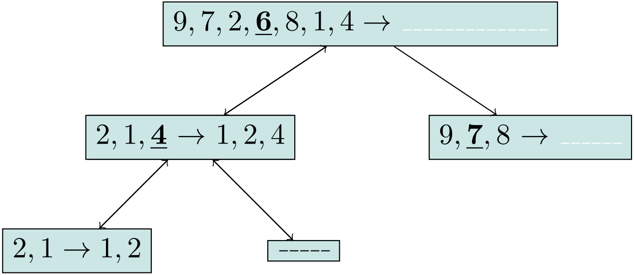
Quick-Sort Execution Tree



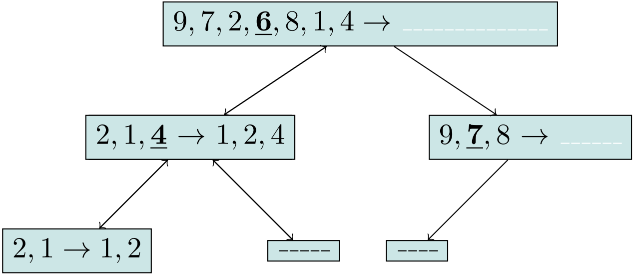
Quick-Sort Execution Tree



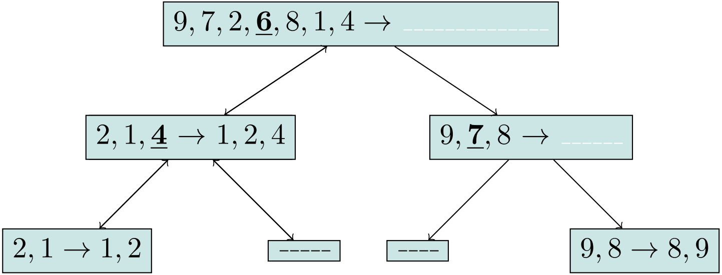
Quick-Sort Execution Tree



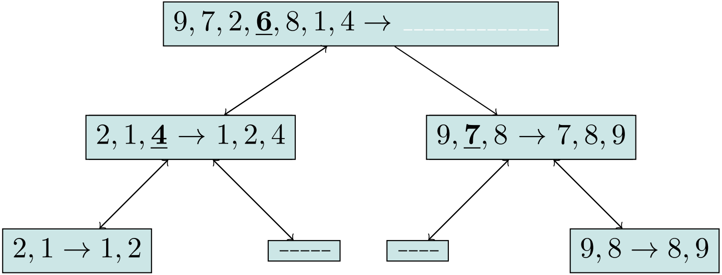
Quick-Sort Execution Tree



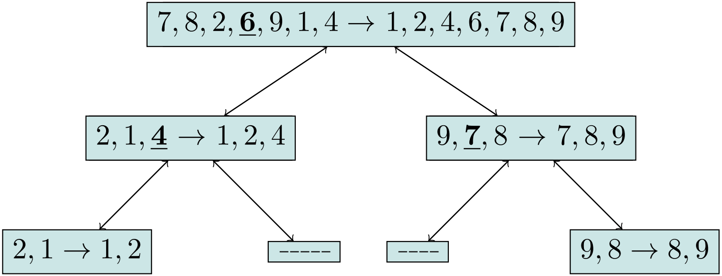
Quick-Sort Execution Tree



Quick-Sort Execution Tree



Quick-Sort Execution Tree



Quick Sort Algorithm

Algorithm QuickSort(S)
Input: Sequence S
Output: Sequence S sorted in increasing order
if $length(S) > 1$ **then**

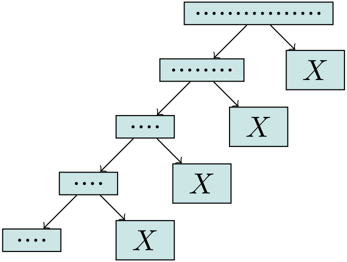
1. Let p be the random position of pivot in S
2. (S_1, S_2) = partition(S,p) //S_1 contains (positions of) elements with value less than $S[p]$ and S_2 contains those with value greater than p
3. QuickSort(S_1)
4. QuickSort(S_2)
5. S' = Merge(S_1,p) //Merge same as before
6. S = Merge(S',S_2)

end if

Figure: In-place Quick-Sort

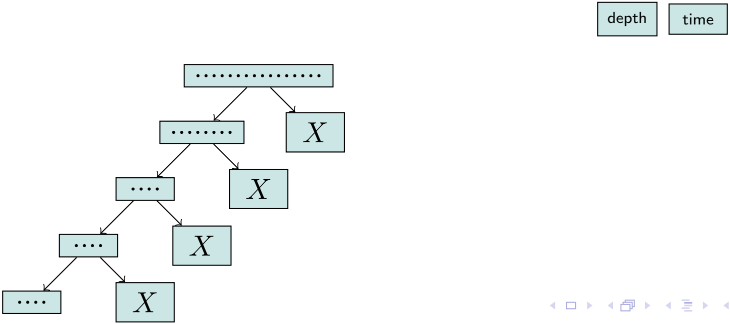
Worst case Running time of QuickSort

- When $S[p]$ is consistently the unique minimum or maximum element of S
  - Case for unique minimum is illustrated below
- One of S_1 or S_2 has size $length(S) - 1$ and other has 0



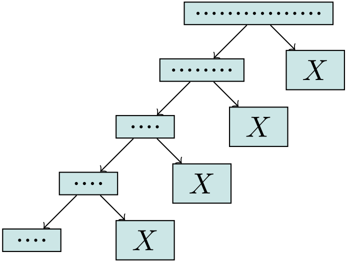
Worst case Running time of QuickSort

- When $S[p]$ is consistently the unique minimum or maximum element of S
  - Case for unique minimum is illustrated below
- One of S_1 or S_2 has size $length(S) - 1$ and other has 0



Worst case Running time of QuickSort

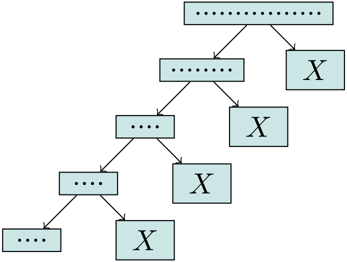
- When $S[p]$ is consistently the unique minimum or maximum element of S
  - Case for unique minimum is illustrated below
- One of S_1 or S_2 has size $length(S) - 1$ and other has 0



depth	time
0	n

Worst case Running time of QuickSort

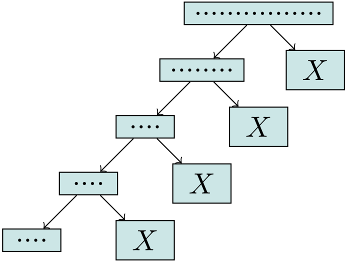
- When $S[p]$ is consistently the unique minimum or maximum element of S
  - Case for unique minimum is illustrated below
- One of S_1 or S_2 has size $length(S) - 1$ and other has 0



depth	time
0	n
1	$n-1$

Worst case Running time of QuickSort

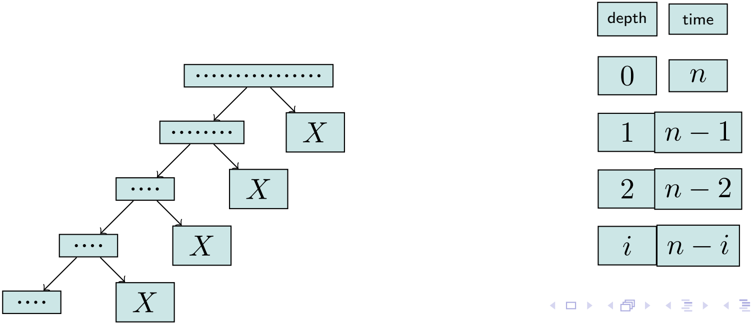
- When $S[p]$ is consistently the unique minimum or maximum element of S
  - Case for unique minimum is illustrated below
- One of S_1 or S_2 has size $length(S) - 1$ and other has 0
-
-



depth	time
0	n
1	$n-1$
2	$n-2$

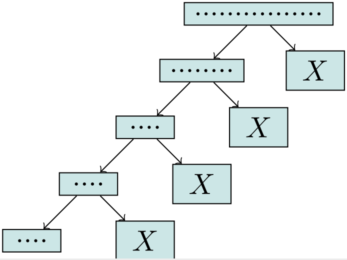
Worst case Running time of QuickSort

- When $S[p]$ is consistently the unique minimum or maximum element of S
  - Case for unique minimum is illustrated below
- One of S_1 or S_2 has size $length(S) - 1$ and other has 0



Worst case Running time of QuickSort

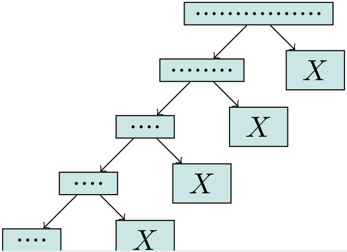
- When $S[p]$ is consistently the unique minimum or maximum element of S
  - Case for unique minimum is illustrated below
- One of S_1 or S_2 has size $length(S) - 1$ and other has 0
-
-



depth	time
0	n
1	$n-1$
2	$n-2$
i	$n-i$
$n-1$	1

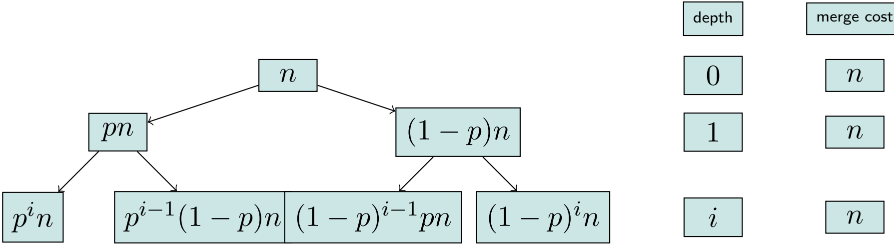
Worst case Running time of QuickSort

- When $S[p]$ is consistently the unique minimum or maximum element of S
  - Case for unique minimum is illustrated below
- One of S_1 or S_2 has size $S.length - 1$ and other has 0
- $\implies$ Runtime is proportional to $n + n - 1 + n - 2 \ldots 2 + 1$
- $\implies$ Runtime $= \Theta(n^2)$



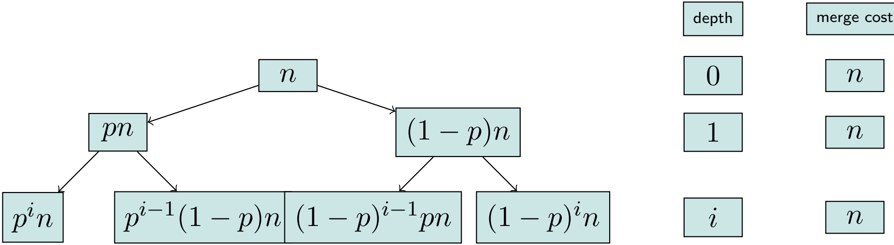
depth	time
0	n
1	$n-1$
2	$n-2$
i	$n-i$
$n-1$	1

Fixed Proportion splitting Analysis of Quick-Sort



- If sequence is split based on 'fixed' proportion p at each step:
  - Recursion will terminate at d such that, $p^{d-1}n = 1$ (if $p < 0.5$) or $(1-p)^d n = 1$ (if $1-p < 0.5$)
  - Since $p < 1$ (and also $1-p < 1$), $d = \Theta(\log n)$
- Amount of work done at each level $i = \Theta(n)$
- $\implies$ total runtime $= \Theta(n \log n)$
- Is solution to recurrence $T(n) \leq T(pn) + T((1-p)n) + cn$
- Also holds if the split prop. is upper bounded by p (or $1-p$)

Average Case Analysis of Quick-Sort



- Proof Sketch: show that the average number of comparisons made across all calls to the "partition" subroutine $= O(\log n)$
  - Also prove that this is the most frequently invoked of all steps
- Ref: Section 7.4.2 of the Second Edition of CLR

Thank you