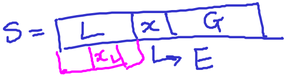
Data Structures and Algorithms

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. } Conquer
- Merge the sorted subsequences L, E and G to produce the sorted answer. } Combine

Quick-Sort Execution Tree

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

Quick-Sort Execution Tree

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

Divide

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

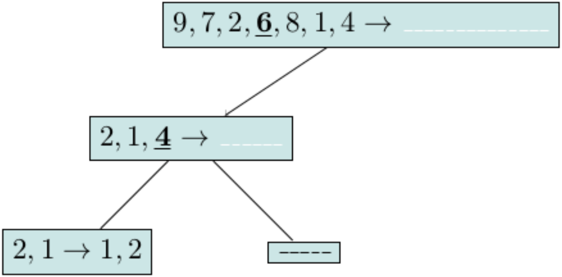
Quick-Sort Execution Tree

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

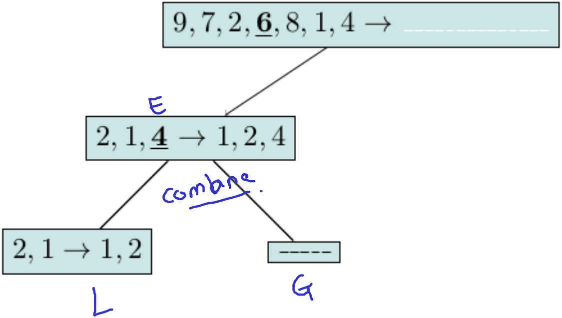
$$2, 1, \underline{\mathbf{4}} \rightarrow$$

$$2, 1 \rightarrow 1, 2$$

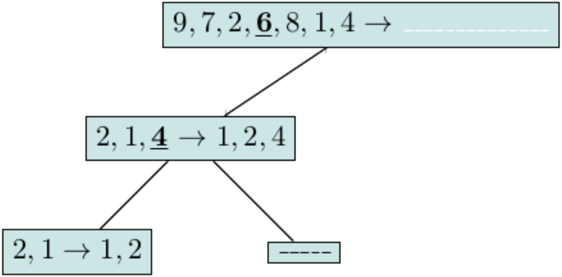
Quick-Sort Execution Tree



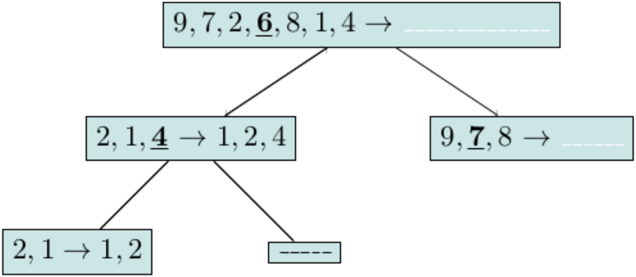
Quick-Sort Execution Tree



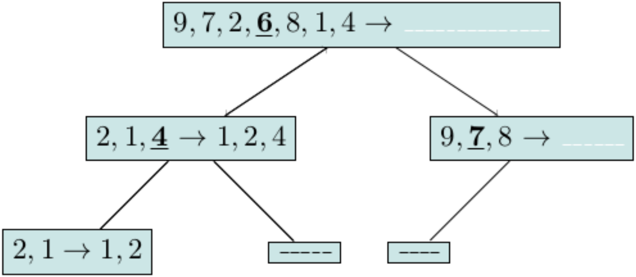
Quick-Sort Execution Tree



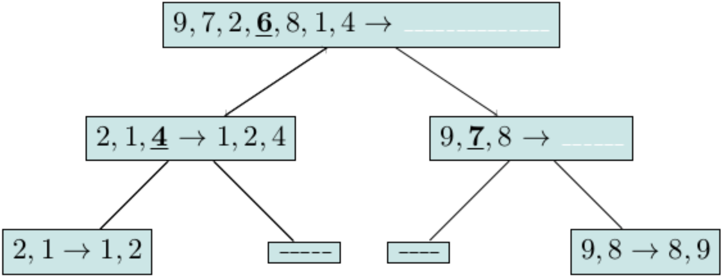
Quick-Sort Execution Tree



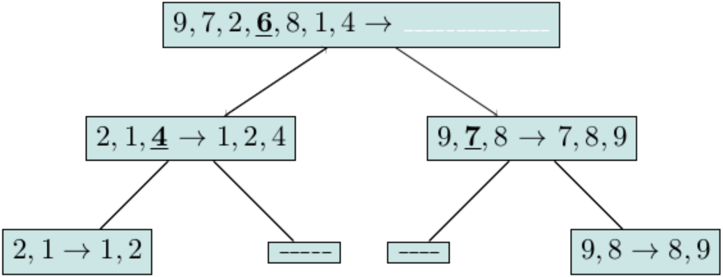
Quick-Sort Execution Tree



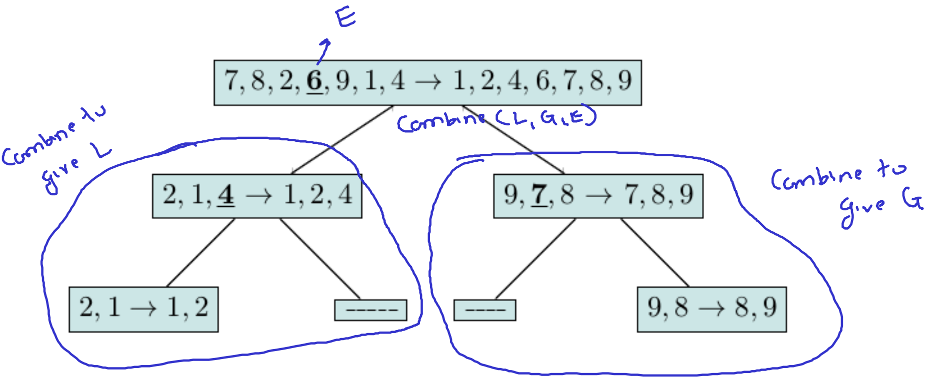
Quick-Sort Execution Tree



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 : Consider simpler case when p = S.length()
3. QuickSort(S_1)
4. QuickSort(S_2) } conquer
5. S' = Merge(S_1,p) //Merge same as before
6. S = Merge(S',S_2) → Exercise 3-way merge Merge(S_1, P, S_2)

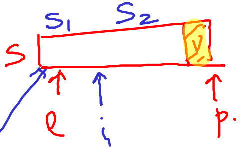
end if

indices into S

Figure: In-place Quick-Sort

begins at l & ends at p

partition $(S, p) = S_1, S_2$



$v \leftarrow S[p]$

$i = l - 1$

for $j = l$ to $p-1$ {

if $S[j] \leq v$ {

$i \leftarrow i + 1$

$S[i] \longleftrightarrow S[j]$

}

}

$S[i+1] \longleftrightarrow S[p]$

j = iterator over elements of $S \ldots [l, p-1]$

<u>goal:</u>

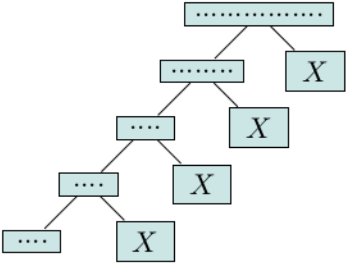
$\forall\ k \leq i \quad S[k] \leq v$

$k > i \quad S[k] > v$

H/W: What abt p that is a random index?

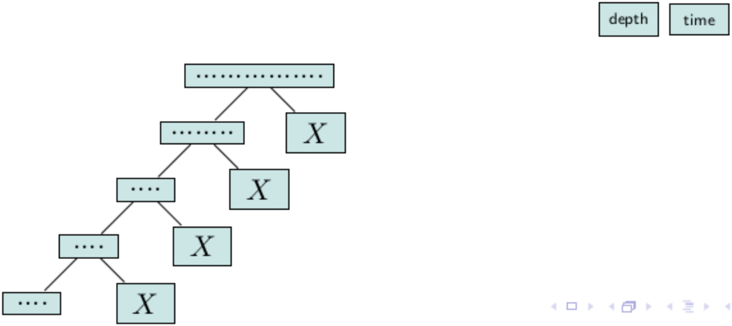
Worst case Running time of QuickSort

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



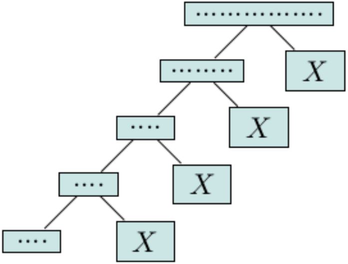
Worst case Running time of QuickSort

- 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
-
-

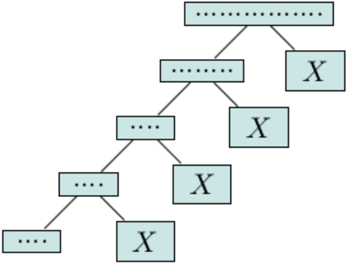


depth	time
0	n

Worst case Running time of QuickSort

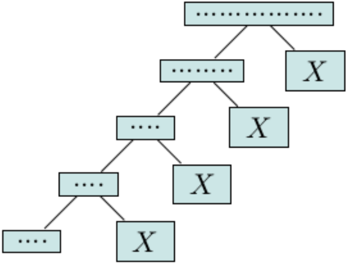
- When $S[p]$ is consistently the unique minimum or maximum element of S
  - Case for unique minimum is illustrated below
- One of S_1 or S_2 has size $length(S) - 1$ and other has 0
-
-

depth	time
0	n
1	$n-1$



Worst case Running time of QuickSort

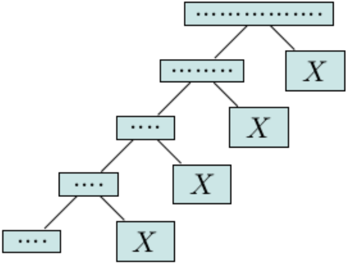
- When $S[p]$ is consistently the unique minimum or maximum element of S
  - Case for unique minimum is illustrated below
- One of S_1 or S_2 has size $length(S) - 1$ and other has 0
-
-



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

Worst case Running time of QuickSort

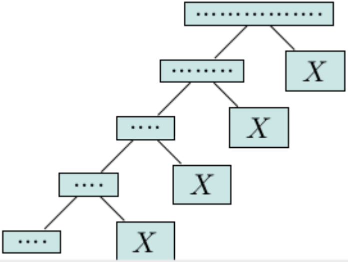
- When $S[p]$ is consistently the unique minimum or maximum element of S
  - Case for unique minimum is illustrated below
- One of S_1 or S_2 has size $length(S) - 1$ and other has 0



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

Worst case Running time of QuickSort

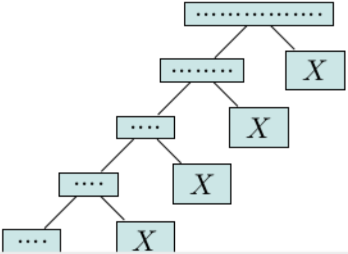
- When $S[p]$ is consistently the unique minimum or maximum element of S
  - Case for unique minimum is illustrated below
- One of S_1 or S_2 has size $length(S) - 1$ and other has 0
- $\sum_{d=0}^{n-1} (n-d)$
-



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 $= \Theta(n^2)$

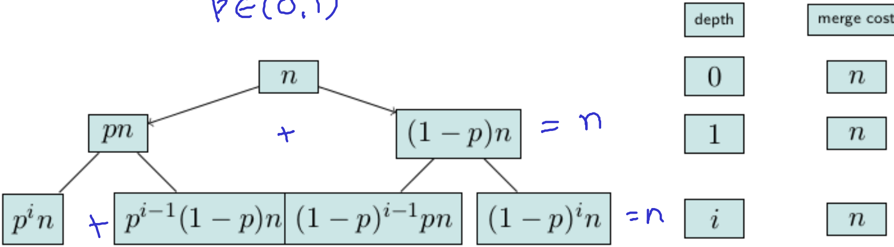


depth	time
0	n
1	$n-1$
2	$n-2$
i	$n-i$
$n-1$	1

(Best case $p = 0.5$)

Fixed Proportion splitting Analysis of Quick-Sort

$p \in (0, 1)$



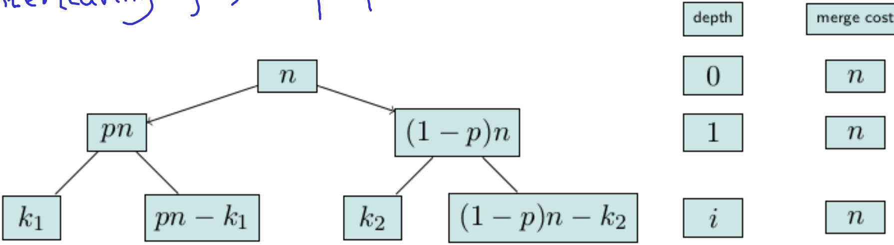
- If sequence is split based on 'fixed' proportion p at each step:
  - Recursion will terminate at d such that, $p^{d-1}n = 1$ (if $p < 0.5$) or $(1-p)^d n = 1$ (if $1 - p < 0.5$)
  - Since $p < 1$ (and also $1 - p < 1$), $d = \Theta(\log n)$
- Amount of work done at each level $i = \Theta(n)$
- $\implies$ 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 $1 - p$)

$(\min\{p, 1-p\})^{d-1} n = 1$

$(d-1)\log(p) + \log(n) = 0$

Average Case Analysis of Quick-Sort

(interleaving of fixed proportion & worst case)



$k_1 = 1$ $k_2 = 1$

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

$X_{ij} = 1$ if $S[i]$ got compared with $S[j]$

$S_{ij} = [i \quad j]$

$$\left.\begin{aligned} \Pr(X_{ij}=1) = {} & \Pr(i \text{ is chosen as pivot from } S_{ij}) \\ & + \Pr(j \text{ is chosen as pivot from } S_{ij}) \end{aligned}\right\} \text{Mutually exclusive}$$

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

$$E[X] = \sum_i \sum_j E[X_{ij}] = \sum_i \sum_j \frac{2}{\underbrace{j-i+1}_{k}} = \sum_{i=1}^{n-1} \sum_{k=1}^{n} \frac{2}{k+1}$$

$$\leq \sum_i \underbrace{\sum_k \frac{2}{k}}_{O(\lg n)} = \sum_i O(\lg n) = O(n \lg n)$$

Thank you