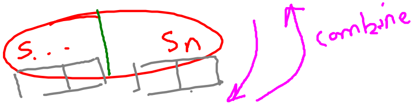
Data Structures and Algorithms

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

Department of Computer Science and Engineering
IIT Bombay

Session: Merge Sort

Divide and Conquer Approach to Sorting

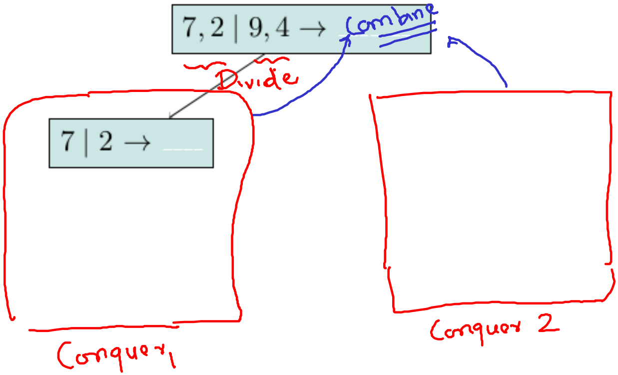


- **Divide** the problem into a number of sub-problems.
- **Conquer** the sub-problems by solving them recursively. For small sizes, recursive call could be replaced by other straightforward alternatives.
- **Combine** the solutions to the sub-problems into the solution for the original problem.

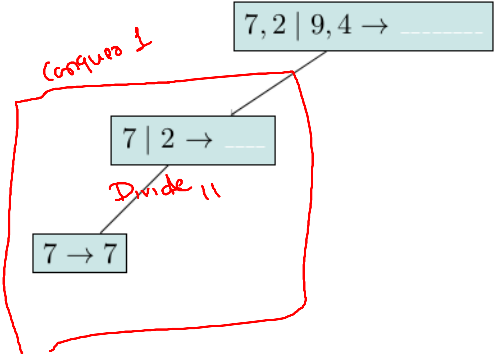
Divide and Conquer Approach to Sorting [Merge Sort]

- **Divide** the problem into a number of sub-problems.
  - Divide the n-element sequence to be sorted into two sub-sequences of $\frac{n}{2}$ elements each
- **Conquer** the sub-problems by solving them recursively. For small sizes, recursive call could be replaced by other straightforward alternatives.
  - Sort the two subsequences recursively using merge sort.
- **Combine** the solutions to the sub-problems into the solution for the original problem.
  - Merge the two sorted subsequences to produce the sorted answer.

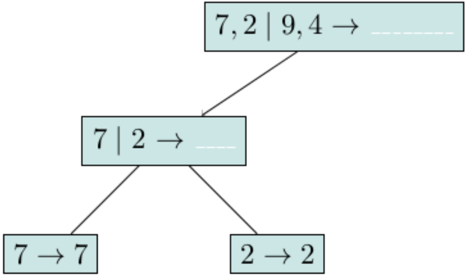
Merge-Sort Execution Tree



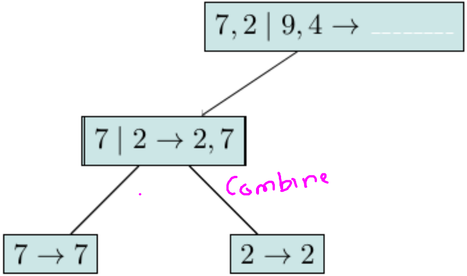
Merge-Sort Execution Tree



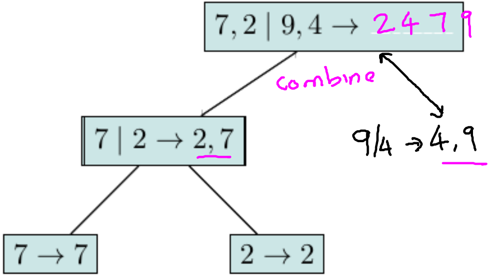
Merge-Sort Execution Tree



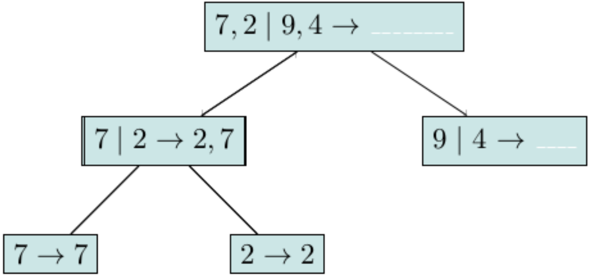
Merge-Sort Execution Tree



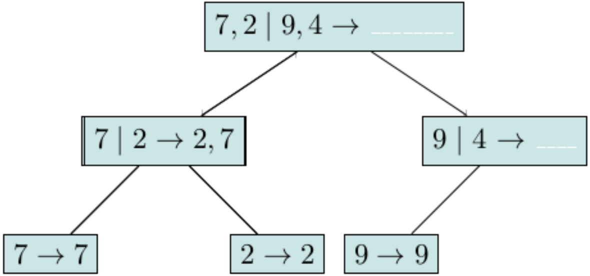
Merge-Sort Execution Tree



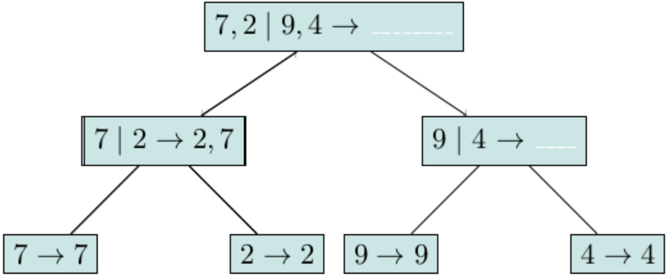
Merge-Sort Execution Tree



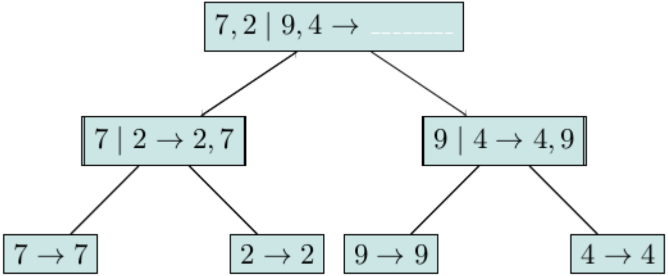
Merge-Sort Execution Tree



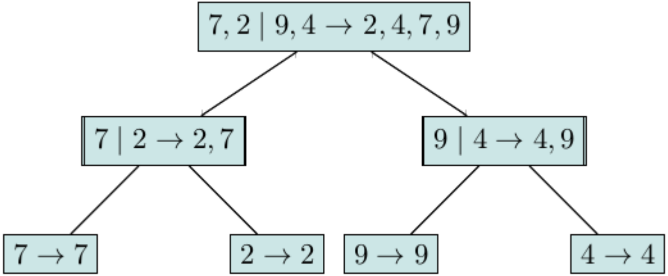
Merge-Sort Execution Tree



Merge-Sort Execution Tree



Merge-Sort Execution Tree



Merge Sort Algorithm

Merge Sort using additional space: O(n)
→ O(nlog n)

Algorithm MergeSort(S)
Input: Sequence S
Output: Sequence S sorted in increasing order
if $S.length > 1$ **then**
1. $(S_1, S_2) \leftarrow$ partition$(S, \frac{n}{2})$ → Divide
2. MergeSort(S_1)
3. MergeSort(S_2)

} Conquer

4. $S \leftarrow$ Merge(S_1,S_2) → Merge / Combine

end if

Figure: In-place Merge Sort

In-place merge sort : Sort part of S & use rest as working area for merging
O(n²)

Merge Subroutine

Algorithm Merge(S)
Input: Sorted Sequence S_1 and S_2 with $\frac{n}{2}$ elements each
Output: Sorted Sequence S union of S_1 and S_2
$S \leftarrow$ empty sequence
while $not(S_1.isEmpty()) \wedge not(S_2.isEmpty())$ **do**
  if $S_1.firstElement() < S_2.firstElement()$ **then**
    $S.append(S_1.removeFirstElement())$
  else
    $S.append(S_2.removeFirstElement())$
  end if
end while

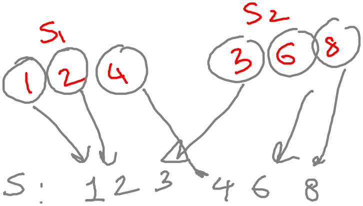


Figure: In-place Merge Sort

Merge Subroutine

Algorithm Merge(S) ...contd
while $not(S_1.isEmpty())$ **do**
  $S.append(S_1.removeFirstElement())$
end while
while $not(S_2.isEmpty())$ **do**
  $S.append(S_2.removeFirstElement())$
end while

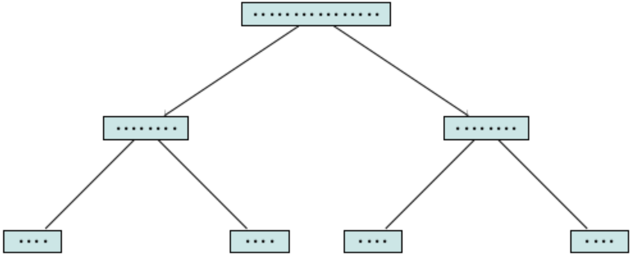
S_1 S_2

$\underline{1 \quad 2 \quad 4}$ $\quad \underline{3} \quad \underbrace{6 \quad 8}$

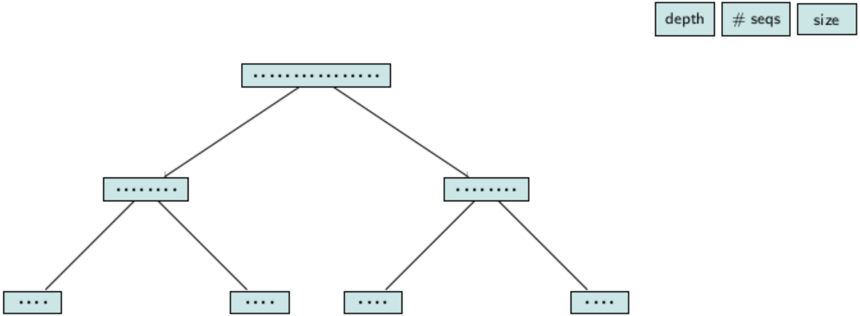
Inserting remaining portion of $S_1 | S_2$

Figure: In-place Merge Sort

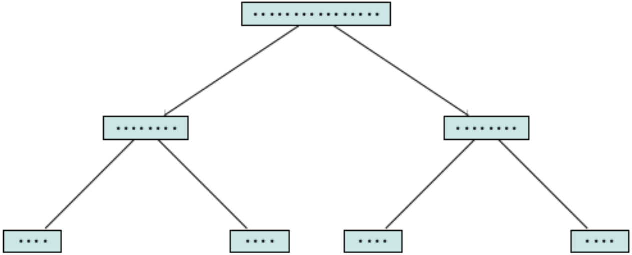
Analysis of Merge-Sort



Analysis of Merge-Sort

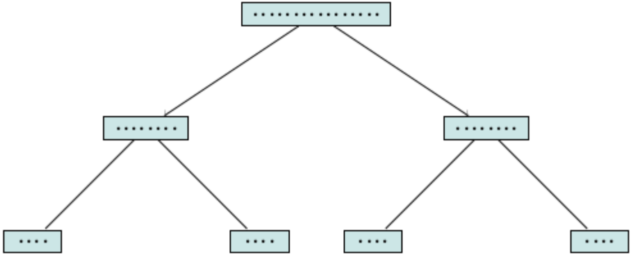


Analysis of Merge-Sort



depth	# seqs	size
0	1	n

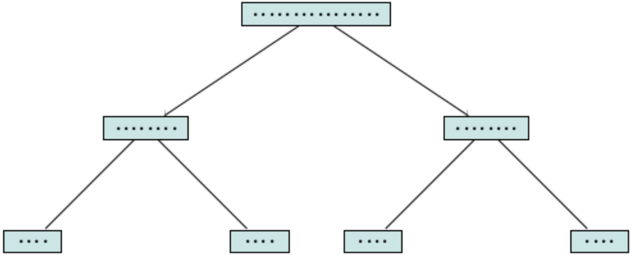
Analysis of Merge-Sort



depth	# seqs	size
0	1	n
1	2	$n/2$

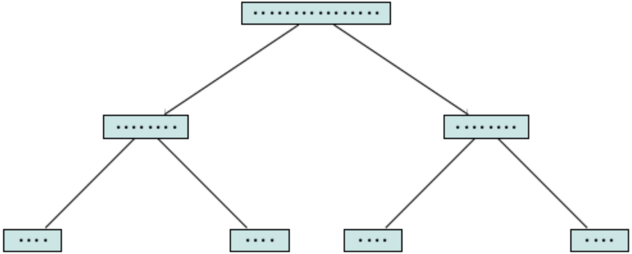
seq × size = n

Analysis of Merge-Sort



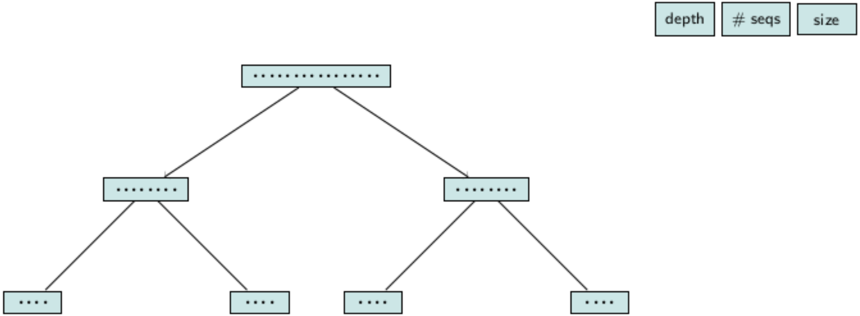
depth	# seqs	size
0	1	n
1	2	$n/2$
i	2^i	$n/(2^i)$

Analysis of Merge-Sort [Assuming auxilliary storage space]



- Each recursive call means division into two tasks of half original size $\Rightarrow$ Height h of tree $= O(\log n)$
- Amount of work done at any height $i = 2^i$ (number of sequences) $\times \frac{n}{2^i}$ (size of each sequence) $= O(n)$
- $\Longrightarrow$ total runtime $= O(n \log n)$

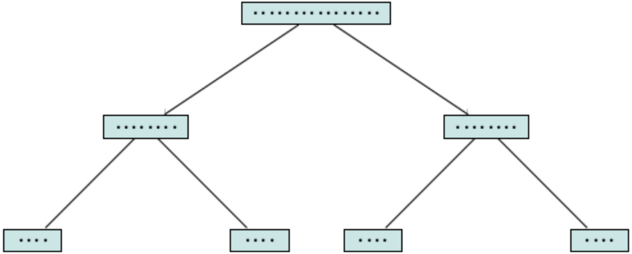
Analysis of Merge-Sort



- Each recursive call means division into two tasks of half original size $\Rightarrow$ Height h of tree $= O(\log n)$
- Amount of work done at any height $i = 2^i$ (number of sequences) $\times \frac{n}{2^i}$ (size of each sequence) $= O(n)$
- $\Longrightarrow$ total runtime $= O(n \log n)$

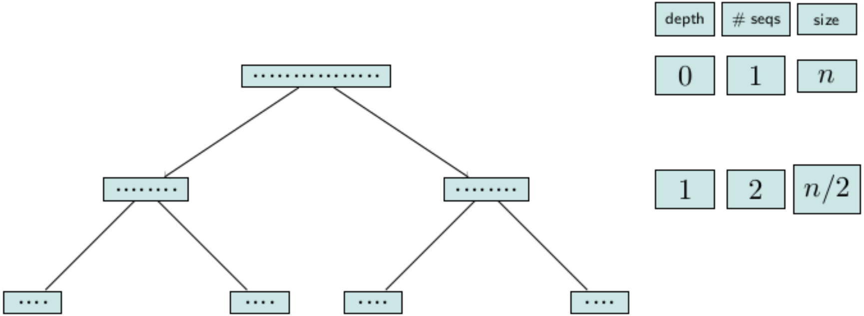
Analysis of Merge-Sort

depth	# seqs	size
0	1	n



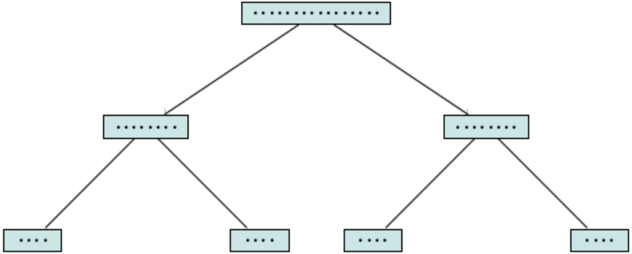
- Each recursive call means division into two tasks of half original size $\Rightarrow$ Height h of tree $= O(\log n)$
- Amount of work done at any height $i = 2^i$ (number of sequences) $\times \frac{n}{2^i}$ (size of each sequence) $= O(n)$
- $\Longrightarrow$ total runtime $= O(n \log n)$

Analysis of Merge-Sort



- Each recursive call means division into two tasks of half original size $\Rightarrow$ Height h of tree $= O(\log n)$
- Amount of work done at any height $i = 2^i$ (number of sequences) $\times \frac{n}{2^i}$ (size of each sequence) $= O(n)$
- $\Longrightarrow$ total runtime $= O(n \log n)$

Analysis of Merge-Sort



depth	# seqs	size
0	1	n
1	2	$n/2$
i	2^i	$n/(2^i)$

- Each recursive call means division into two tasks of half original size $\Rightarrow$ Height h of tree $= O(\log n)$
- Amount of work done at any height $i = 2^i$ (number of sequences) $\times \frac{n}{2^i}$ (size of each sequence) $= O(n)$

- Next Session: Quick Sort

Advantage: Natural in-place implementation

Disadvantage: Complexity depends on nature of input

Worst case = $O(n^2)$

Thank you