

Analysis of Algorithms

CSC 402

2022-23



Subject Incharge

Bidisha Roy

Associate Professor

Room No. 401

email: bidisharoy@sfit.ac.in



Module 2 – Divide and Conquer



DIVIDE AND CONQUER APPROACH



General Method

- Works on the approach of dividing a given problem into smaller sub problems (ideally of same size). ← ***Divide***
- Sub problems are solved independently using recursion. ← ***Conquer***
- Solutions for sub problems then combined to get solution for the original problem. ← ***Combine***



Examples to be considered

- Merge Sort
- Quick Sort
- Binary search
- Finding Minimum and Maximum



Merge Sort

- **Divide:** If A has at least two elements (nothing needs to be done if A has zero or one elements), remove all the elements from A and put them into two sequences, L and R , each containing about half of the elements of A. (i.e. L contains the first $n/2$ elements and R contains the remaining $n/2$ elements).
- **Conquer:** Sort sequences L and R using Merge Sort.
- **Combine:** Put back the elements into A by merging the sorted sequences L and R into one sorted sequence



Merge Sort ... contd

- Merge_Sort (A, low, high)
 - if $low < high$ then
 - $mid \leftarrow (low+high)/2$
 - Merge_Sort(A, low, mid)
 - Merge_Sort(A, mid+1, high)
 - Merge (A, low, mid, high)



Merge(A, low, mid, high)

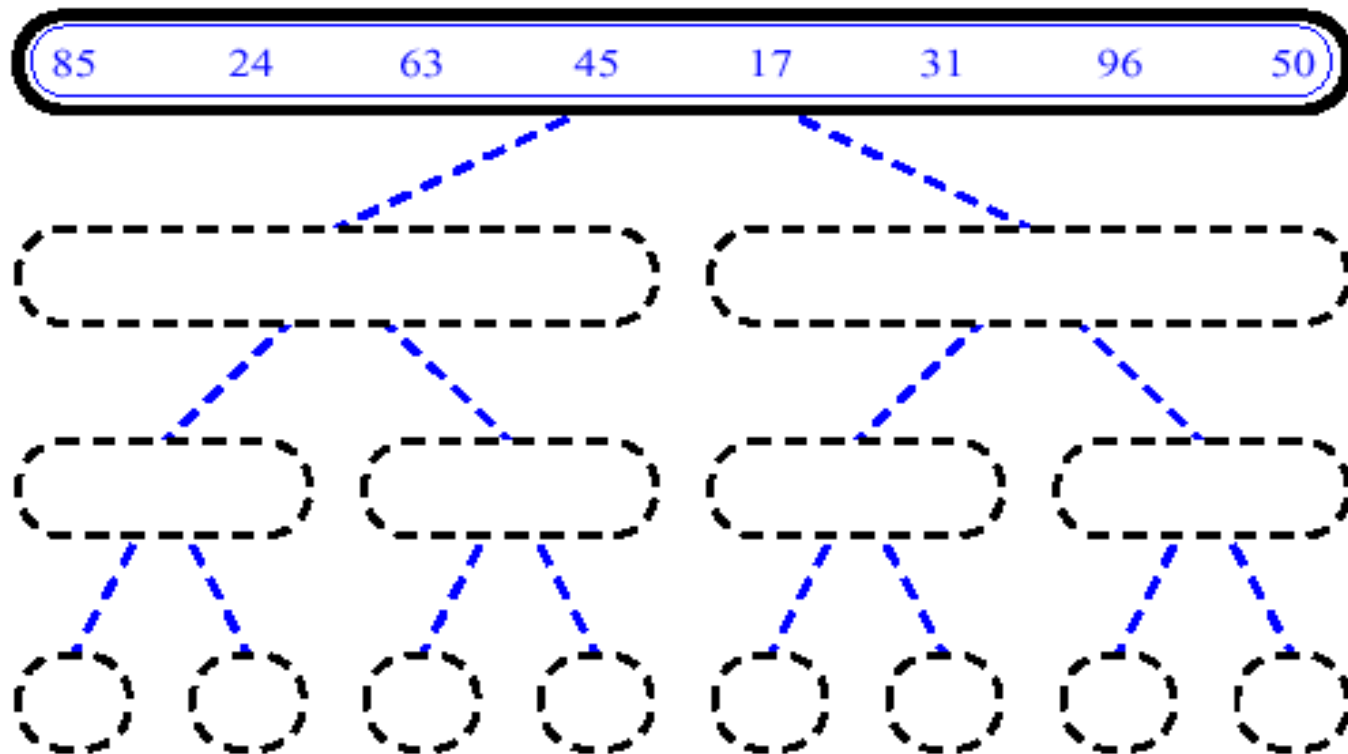
- $h \leftarrow \text{low}, i \leftarrow \text{low}, j \leftarrow \text{mid} + 1$
- Create auxiliary array B[]
- while ($h \leq \text{mid}$ and $j \leq \text{high}$) do
 - if ($A[h] \leq A[j]$) then
 - $B[i] \leftarrow A[h]$
 - $h \leftarrow h+1$
 - Else
 - $B[i] \leftarrow A[j]$
 - $j \leftarrow j+1$
 - $i \leftarrow i+1$
- if ($h > \text{mid}$) then
 - for $k \leftarrow j$ to high
 - $B[i] \leftarrow A[k]$
 - $i \leftarrow i+1$
- else
 - for $k \leftarrow h$ to mid
 - $B[i] \leftarrow A[k]$
 - $i \leftarrow i+1$
- for $k \leftarrow \text{low}$ to high
 - $A[k] \leftarrow B[k]$

Merge(A, low, mid , high)

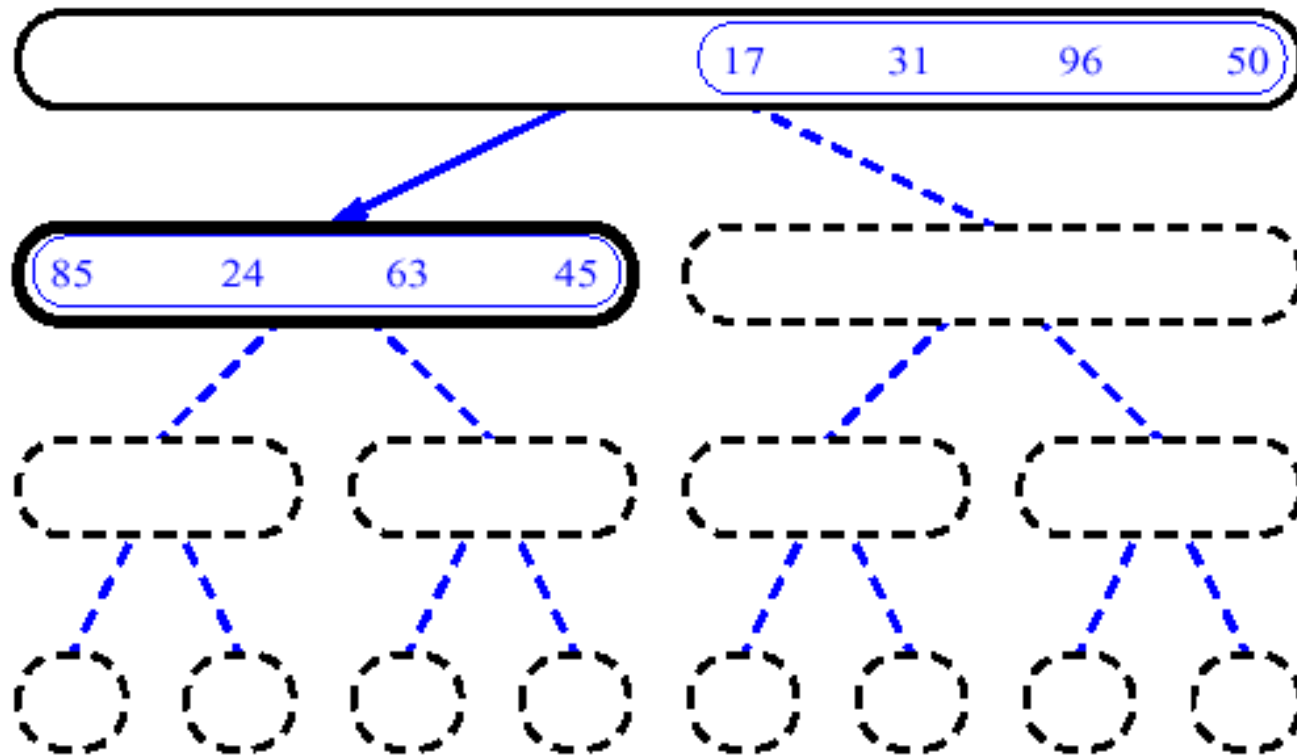
Take the smallest of the two topmost elements of sequences $A[\text{low}..\text{mid}]$ and $A[\text{mid}+1..\text{high}]$ and put into the resulting sequence. Repeat this, until both sequences are empty. Copy the resulting sequence into $A[\text{low}..\text{high}]$.



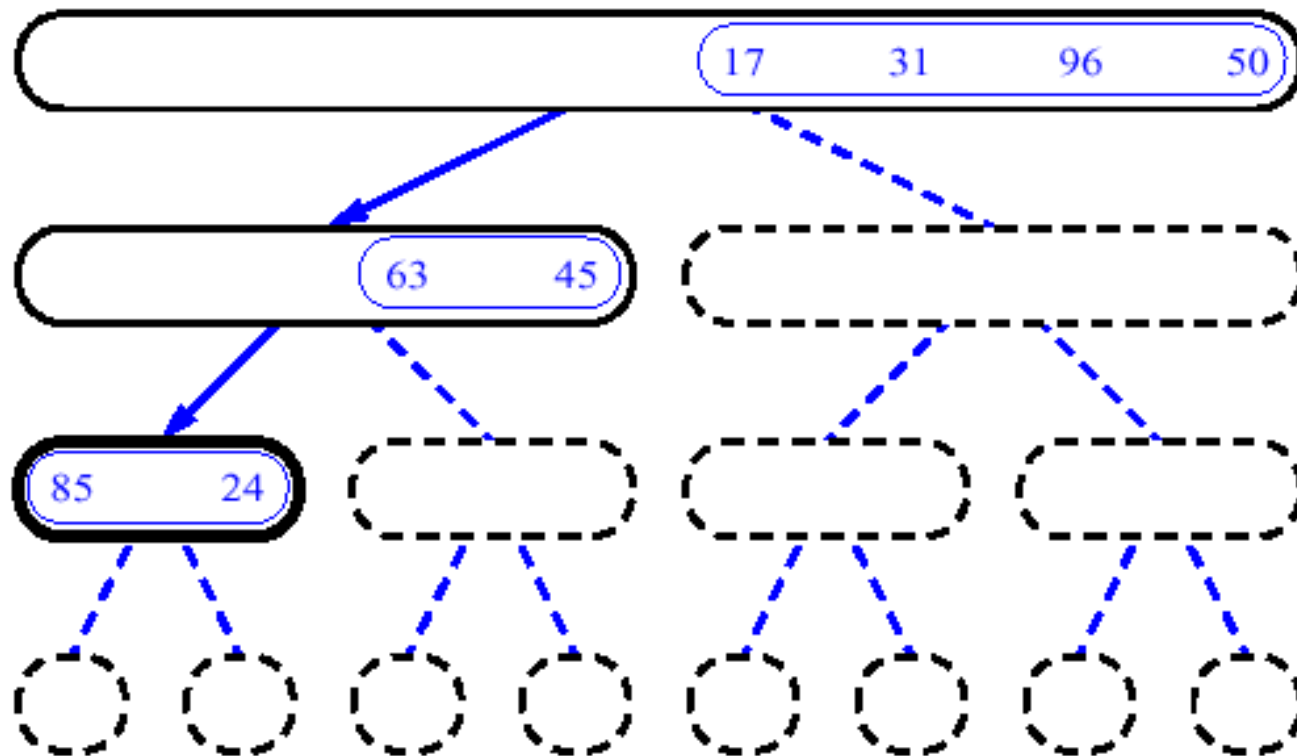
Merge Sort (Example) - 1



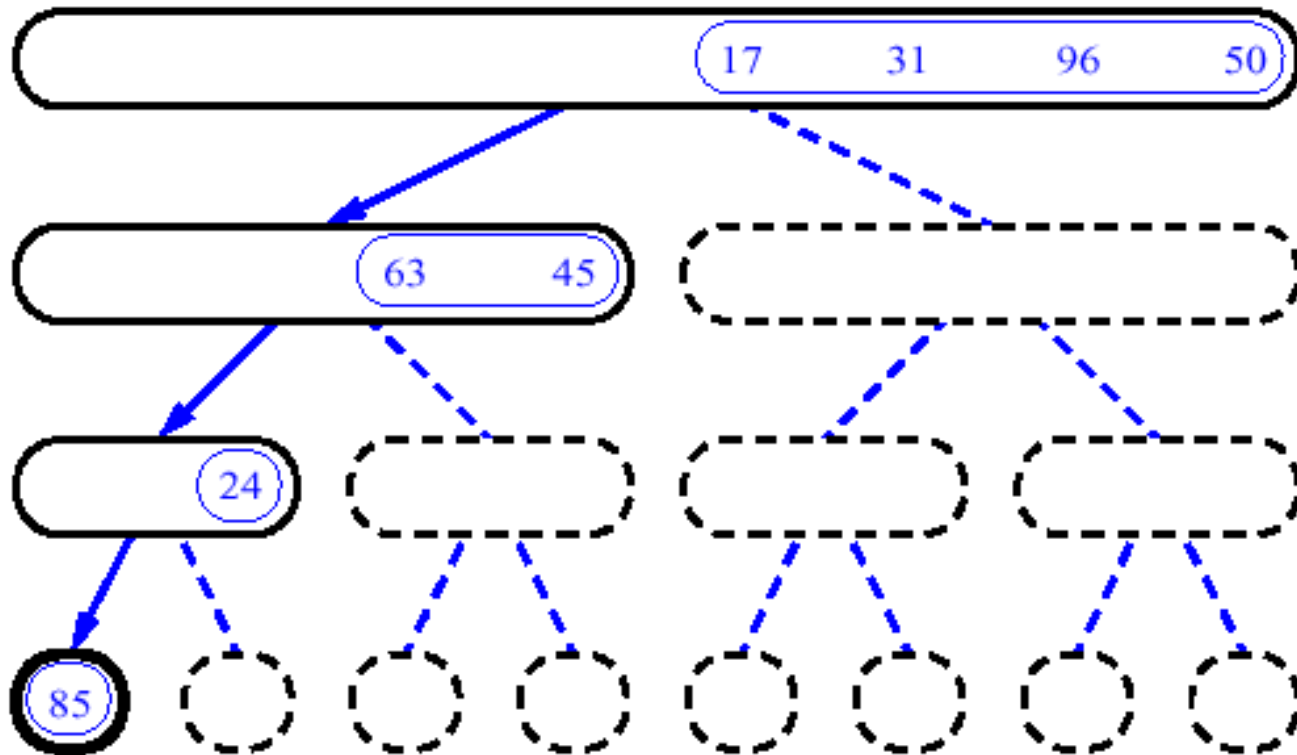
Merge Sort (Example) - 2



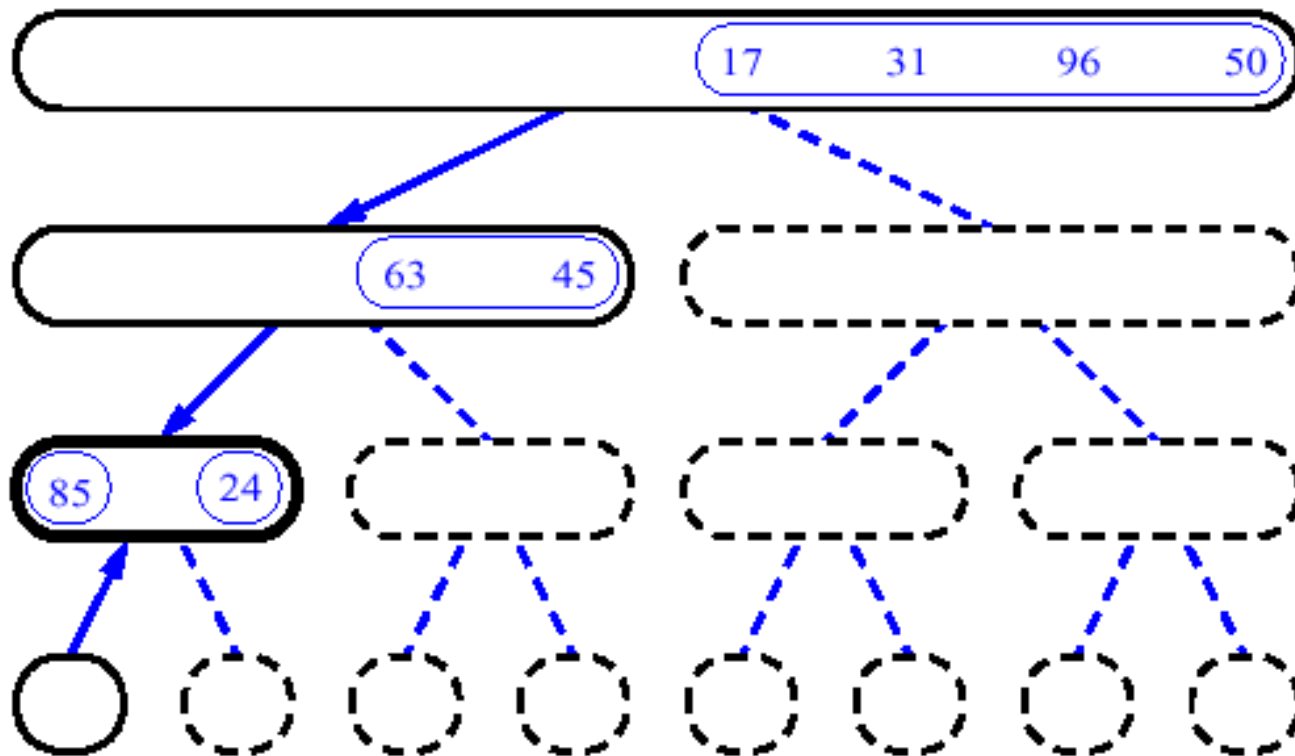
Merge Sort (Example) - 3



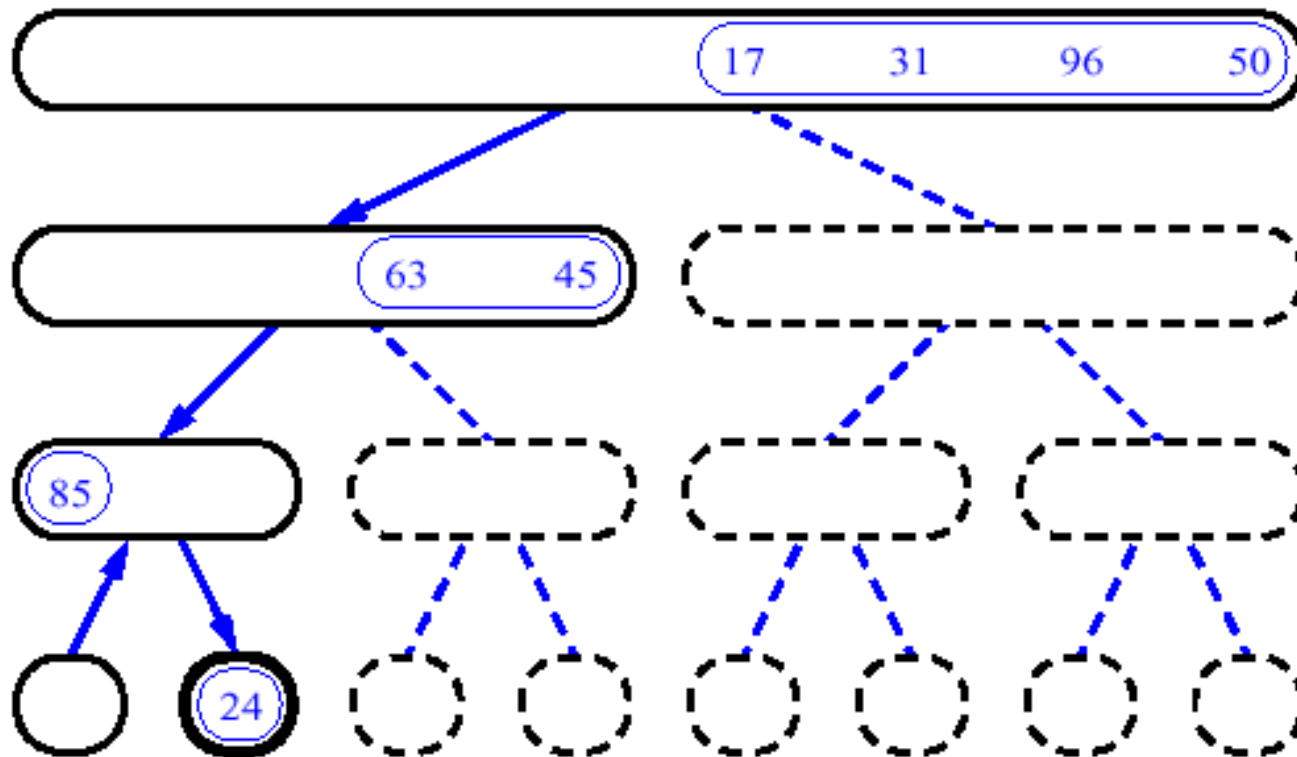
Merge Sort (Example) - 4



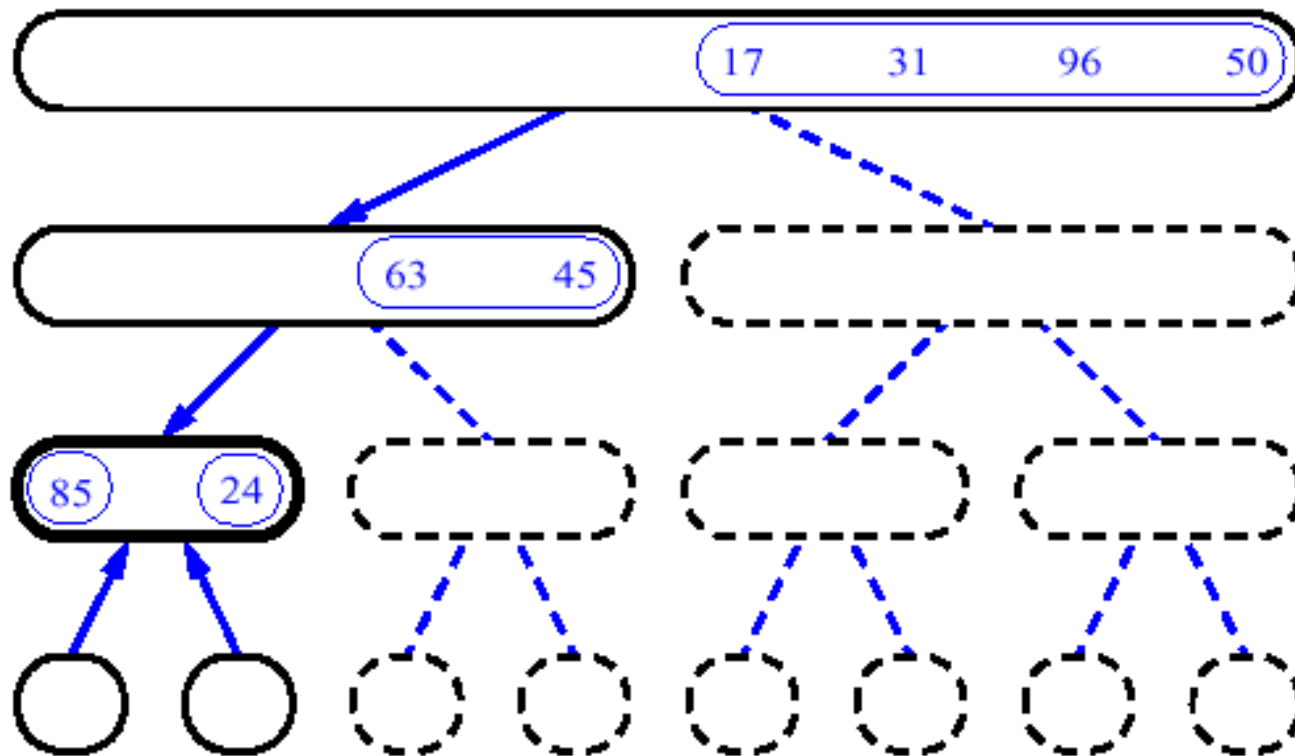
Merge Sort (Example) - 5



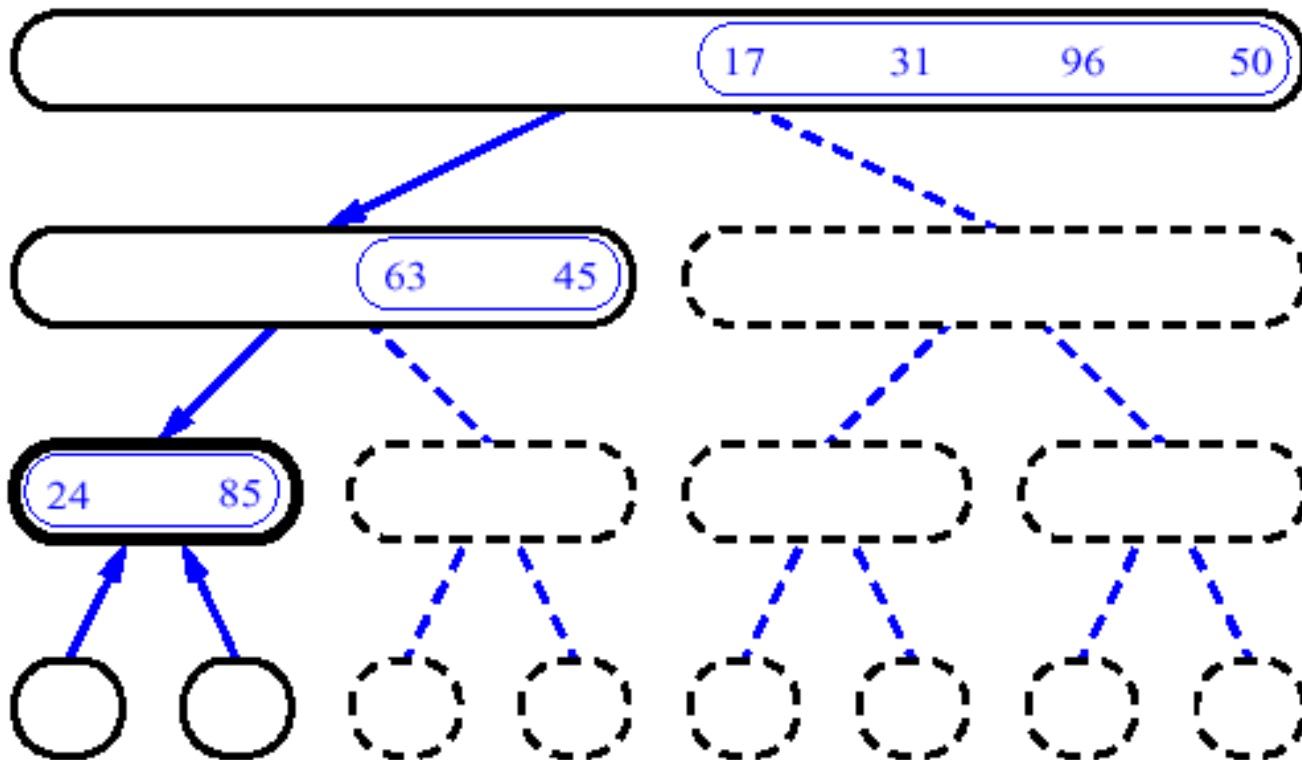
Merge Sort (Example) - 6



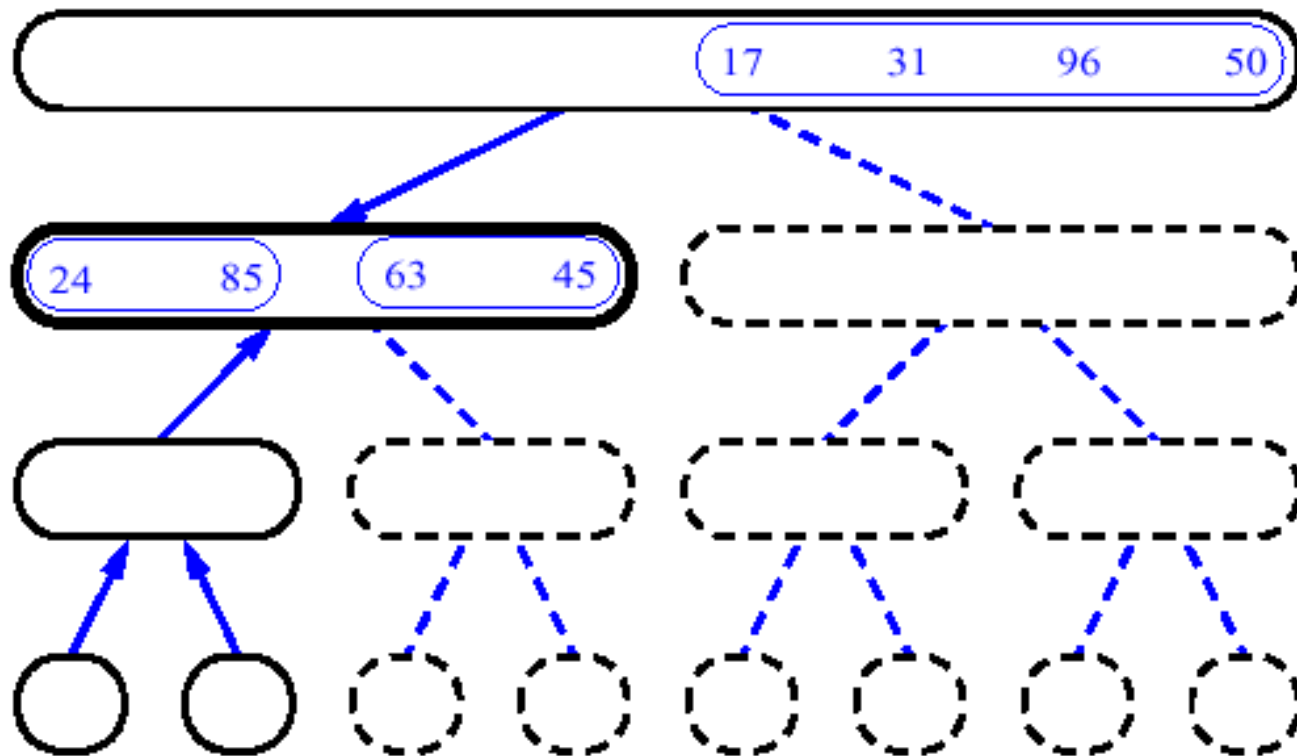
Merge Sort (Example) - 7



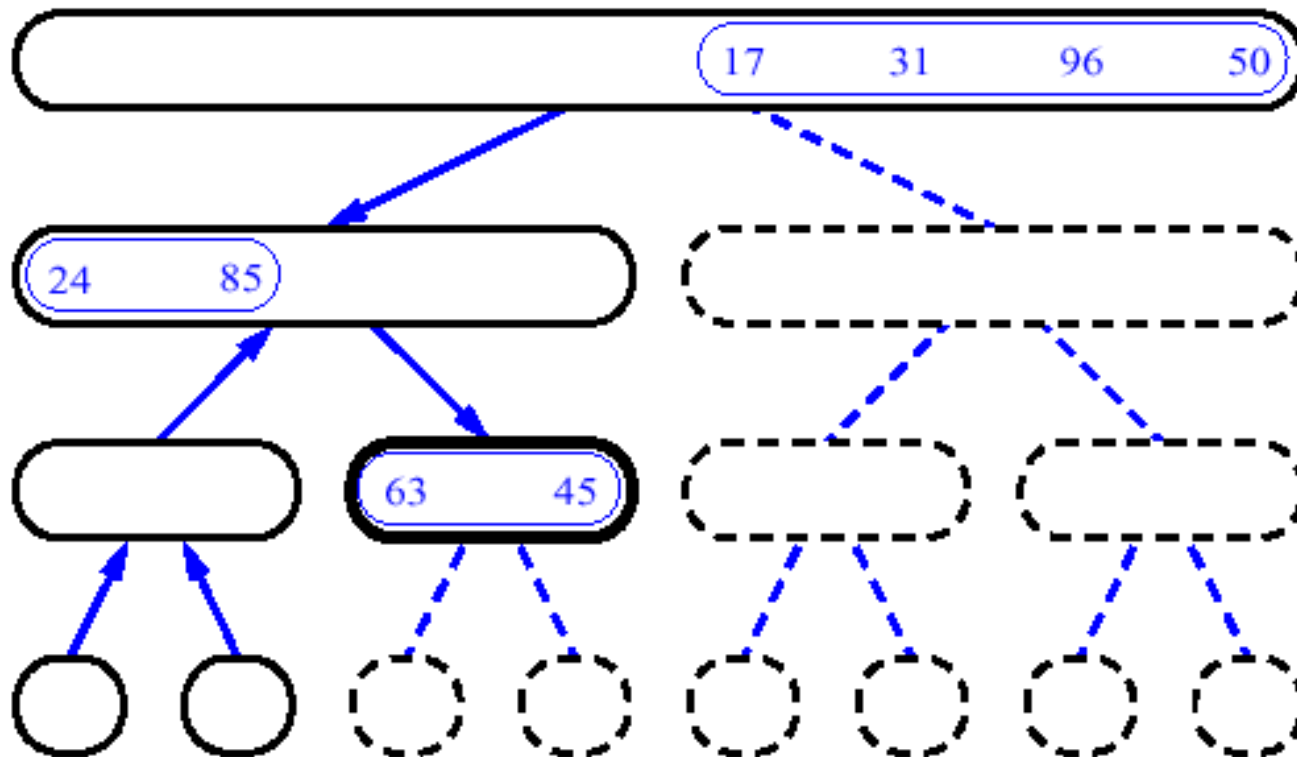
Merge Sort (Example) - 8



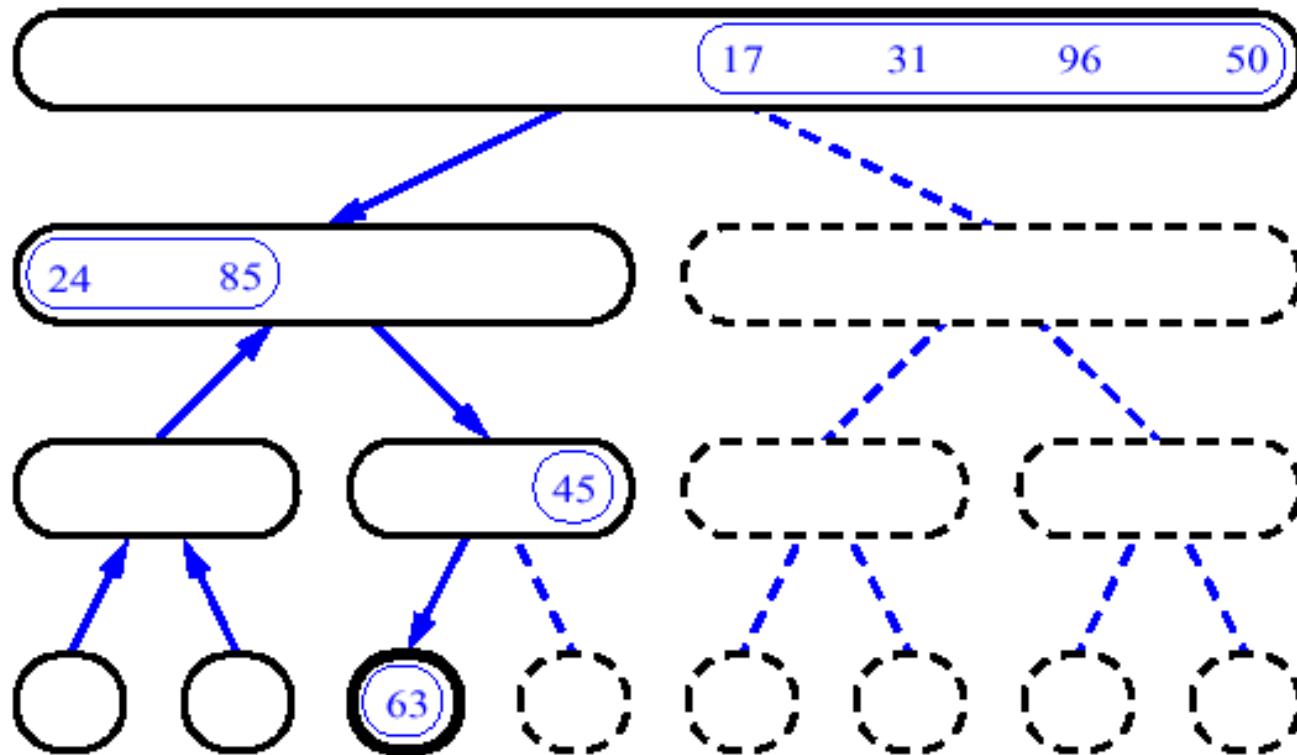
Merge Sort (Example) - 9



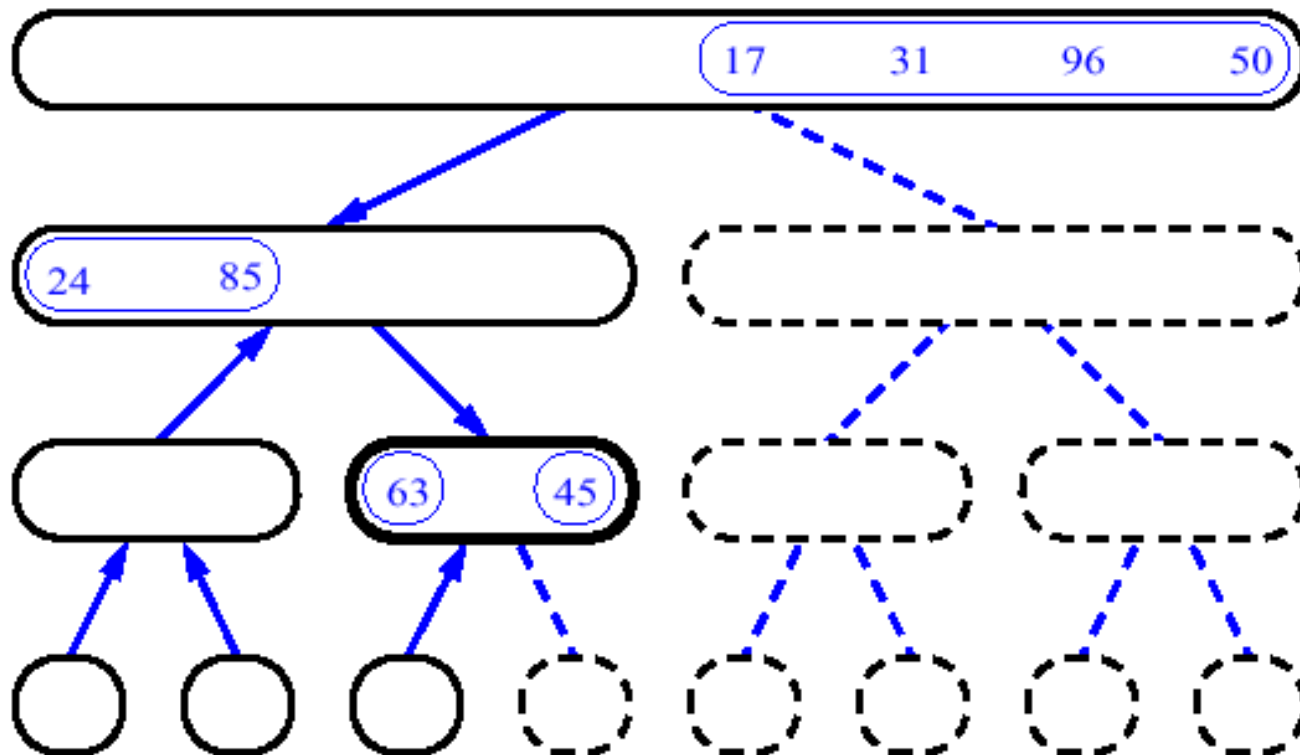
Merge Sort (Example) - 10



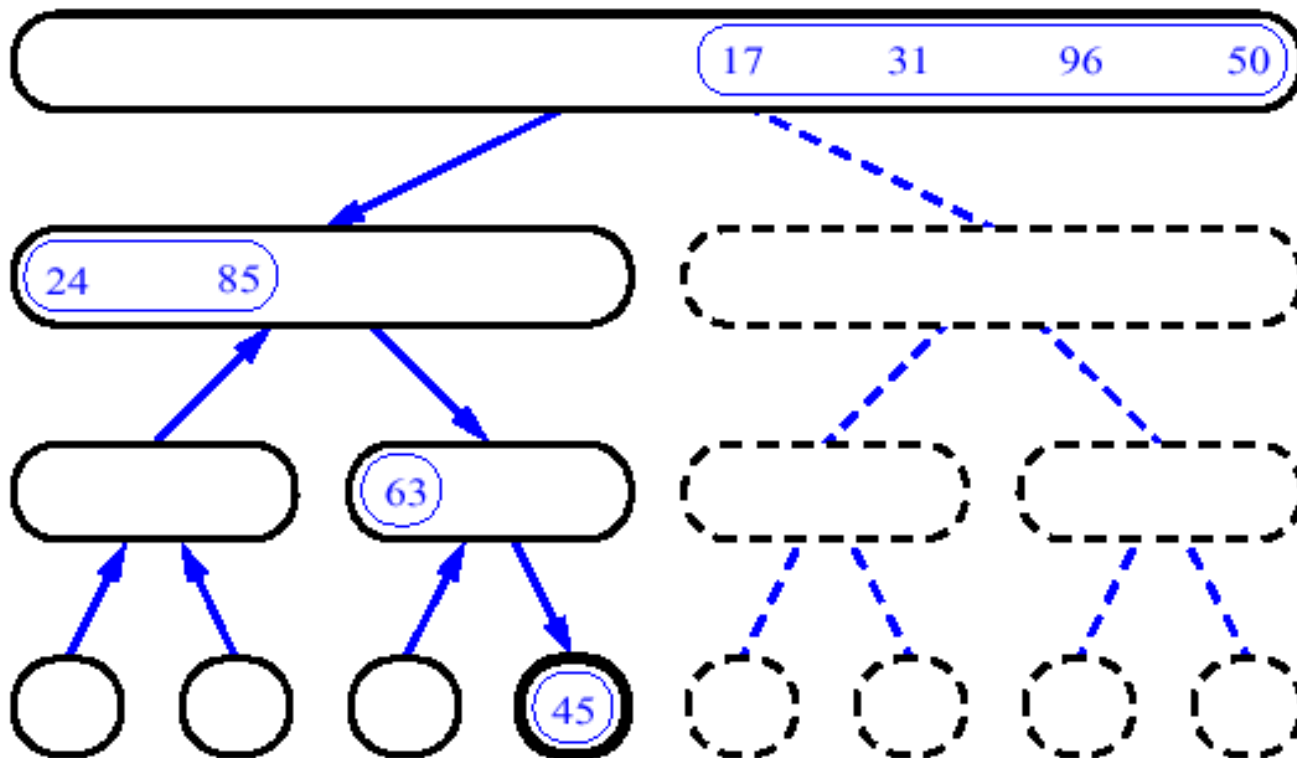
Merge Sort (Example) - 11



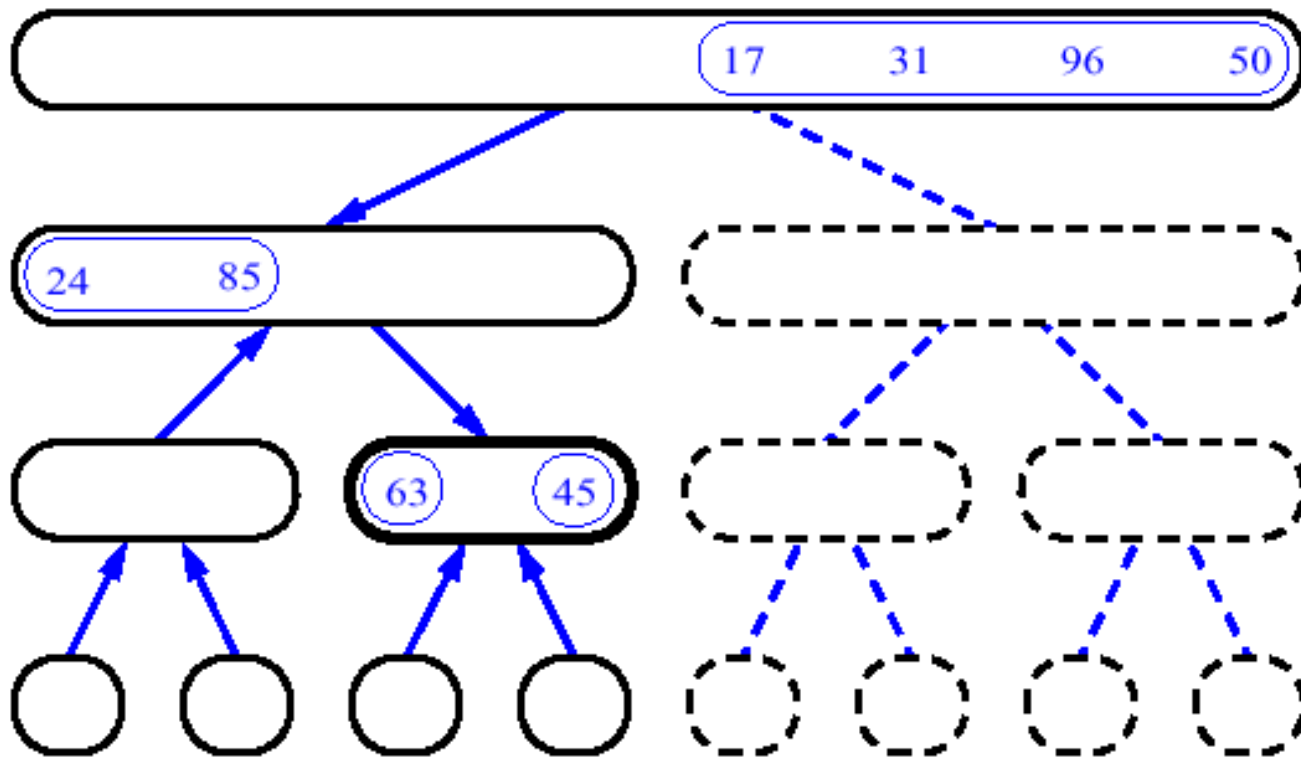
Merge Sort (Example) - 12



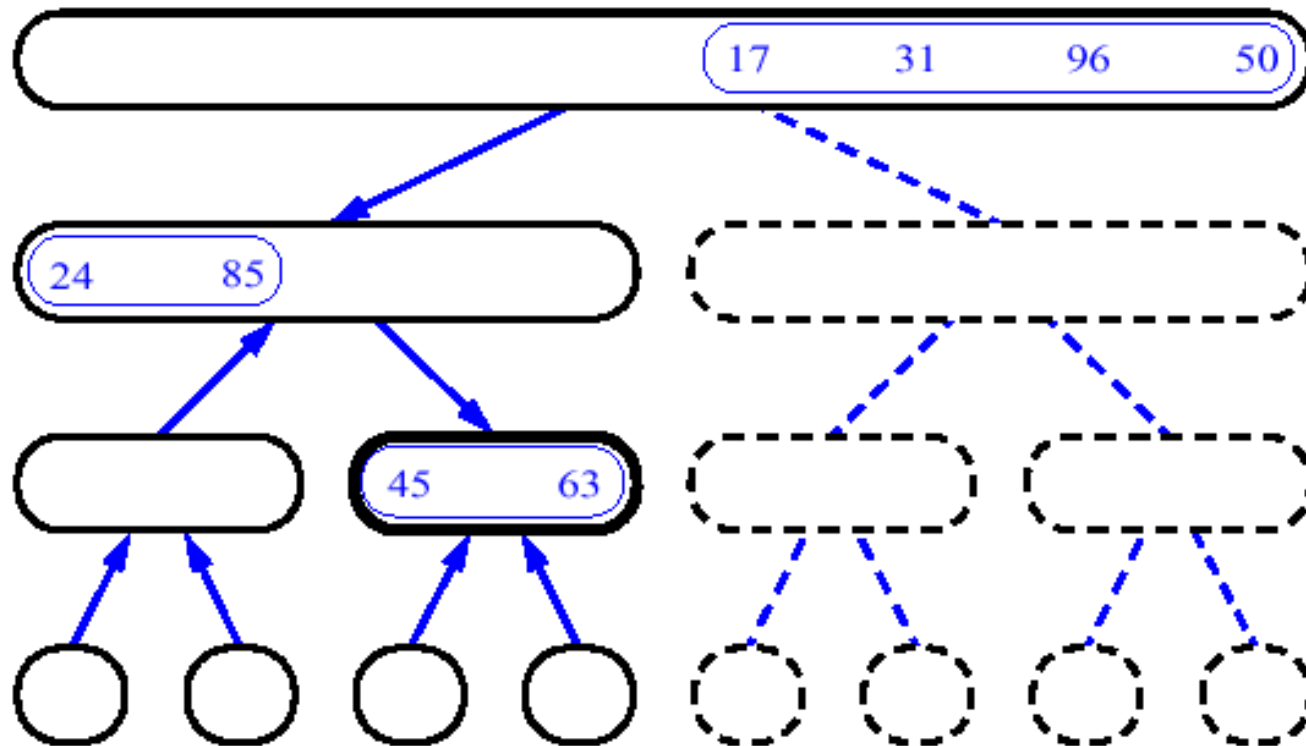
Merge Sort (Example) - 13



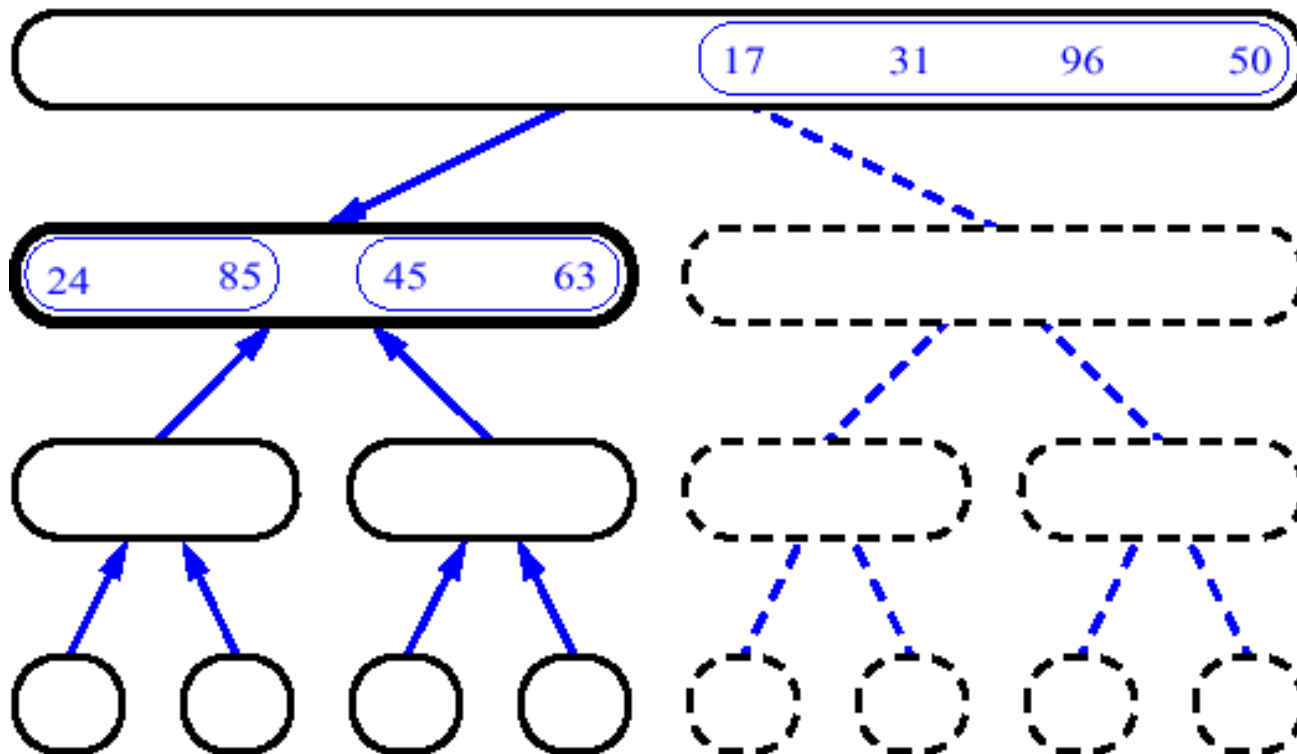
Merge Sort (Example) - 14



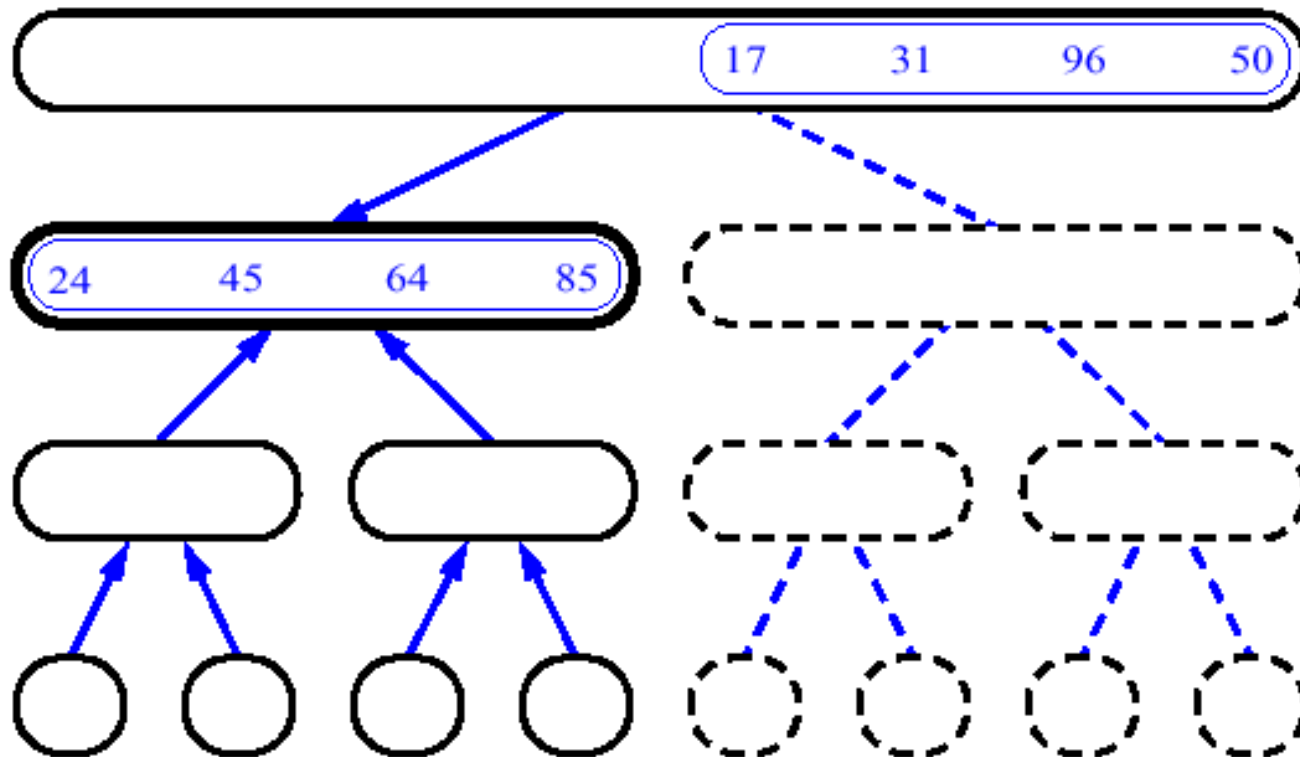
Merge Sort (Example) - 15



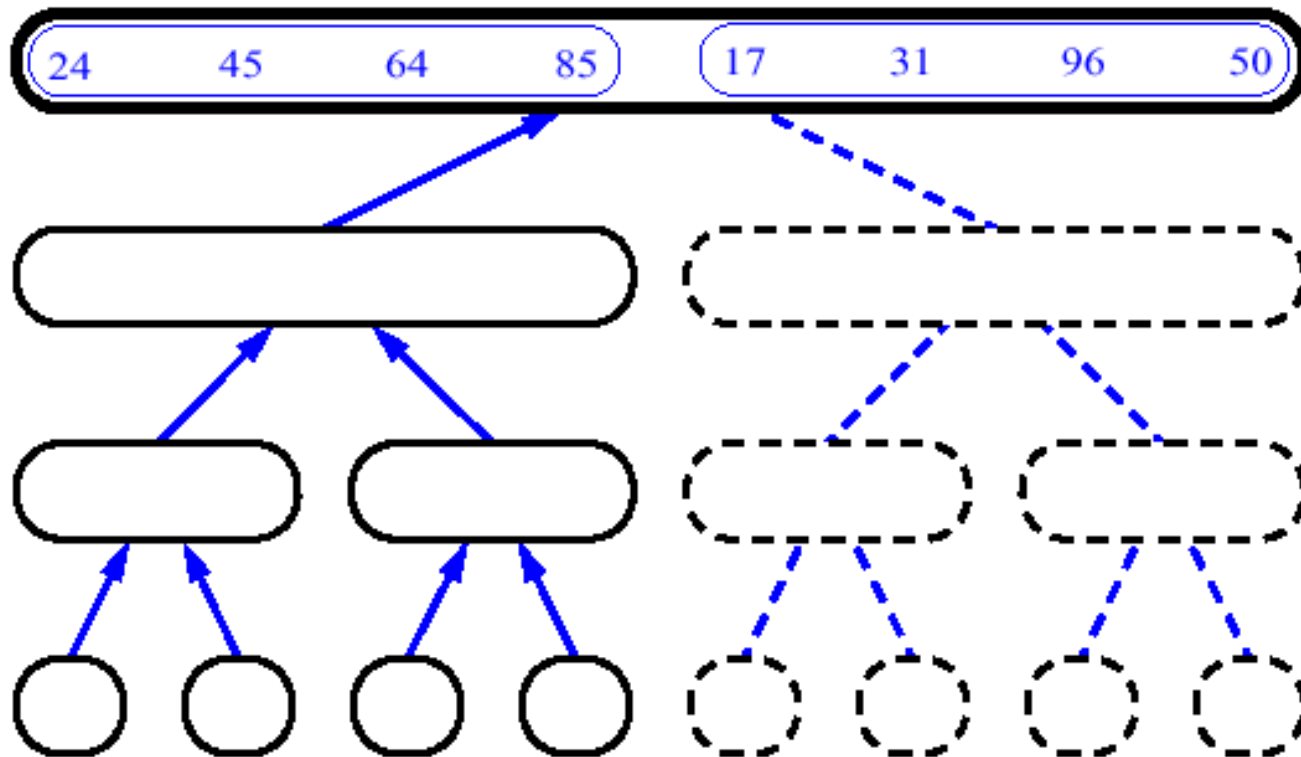
Merge Sort (Example) - 16



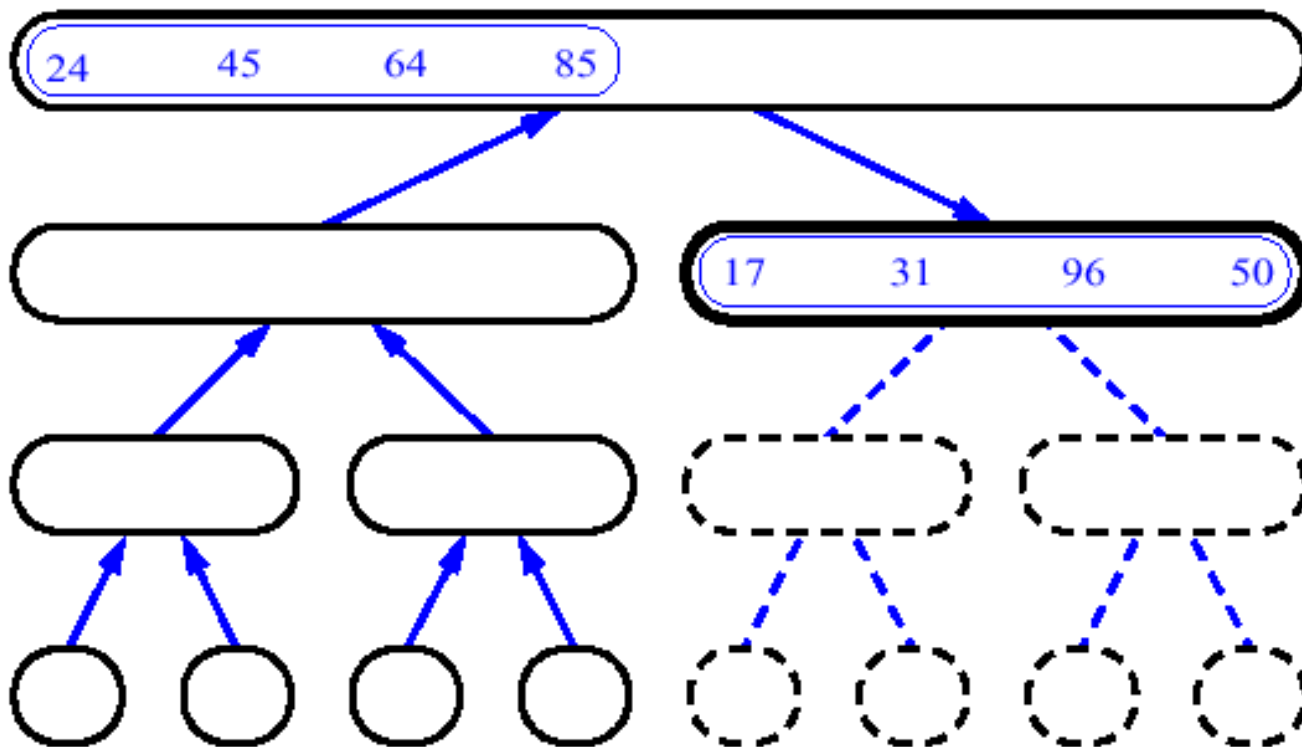
Merge Sort (Example) - 17



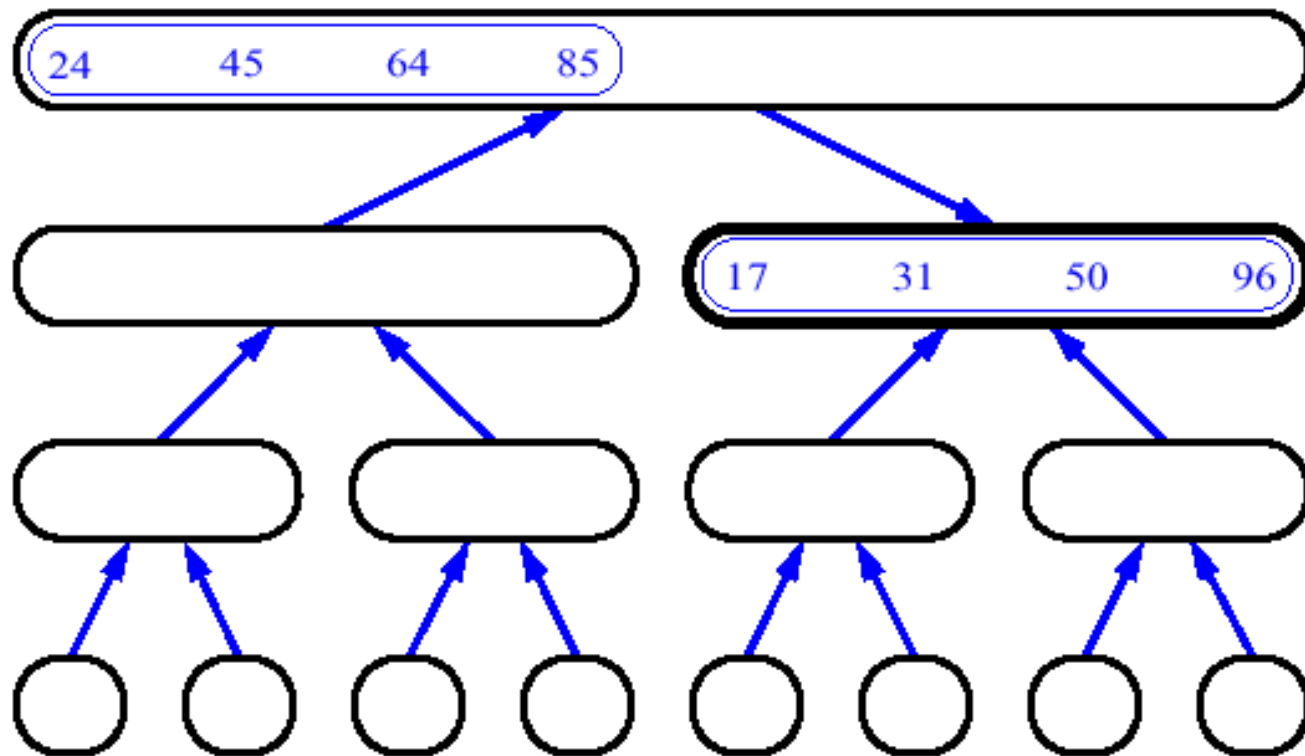
Merge Sort (Example) - 18



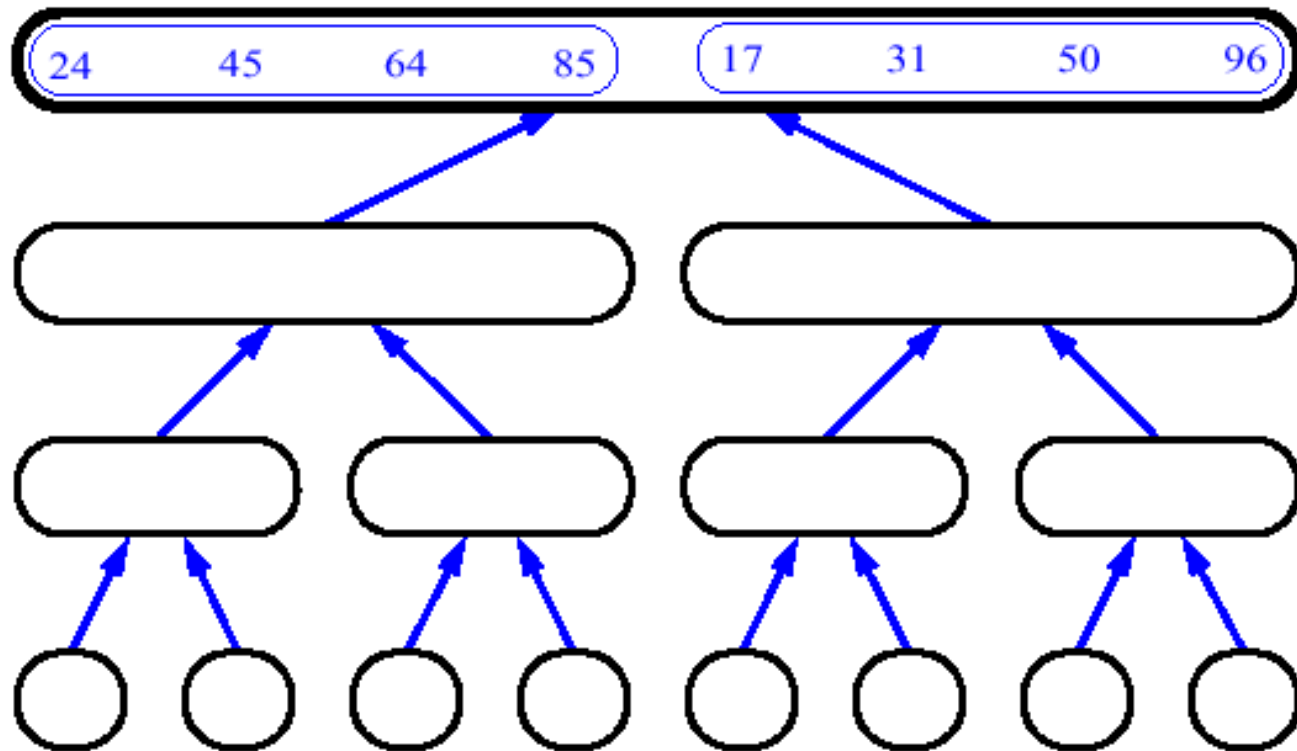
Merge Sort (Example) - 19



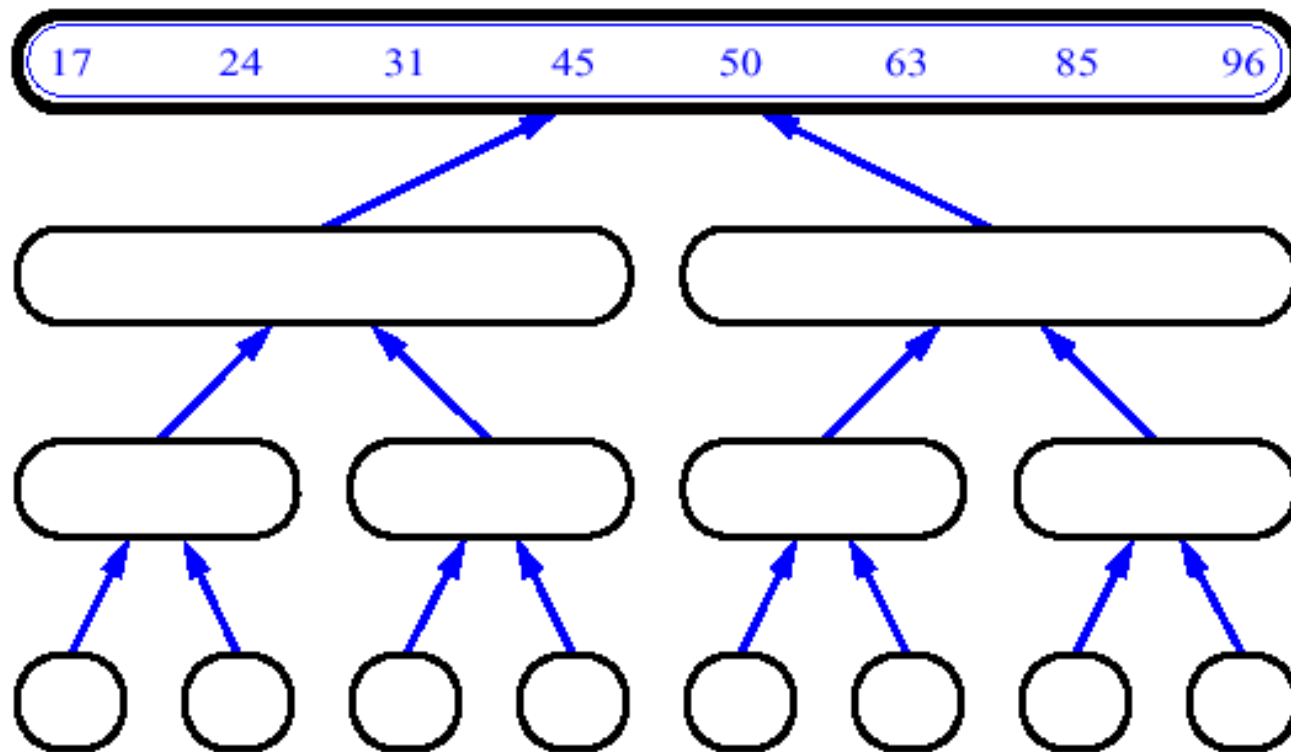
Merge Sort (Example) - 20



Merge Sort (Example) - 21



Merge Sort (Example) - 22



<https://liveexample.pearsoncmg.com/dsanimation13ejava/MergeSorteBook.html>



Running time of Merge Sort

- To sort n numbers
 - if $n=1$ done
 - recursively sort 2 lists of numbers $(n/2)$ and $(n/2)$ elements
 - Merge 2 sorted lists in $O(n)$ time
- Again the running time can be expressed as a recurrence:

$$T(n) = \begin{cases} \text{solving_trivial_problem} & \text{if } n = 1 \\ \text{num_pieces } T(n / \text{subproblem_size_factor}) + \text{dividing} + \text{combining} & \text{if } n > 1 \end{cases}$$



Running time of Merge Sort

- Worst case and best case complexity same as average case complexity.
- Applications
 - Sorting linked list



Quicksort

- Also called ***partition exchange sort***.
- Fastest known sorting algorithm in practice.
- The array $A[p..r]$ is partitioned into two non-empty subarrays $A[p..q]$ and $A[q+1..r]$
- Unlike merge sort, no combining step



Quicksort ... contd

- Quicksort(A, lb, ub)
 - if ($lb < ub$) then
 - $pivot \leftarrow \text{Partition}(A, lb, ub)$
 - Quicksort(A, lb, pivot-1)
 - Quicksort(A, pivot+1, ub)
- To sort an entire array A, initially lb is 0/1 and ub is n-1/n.

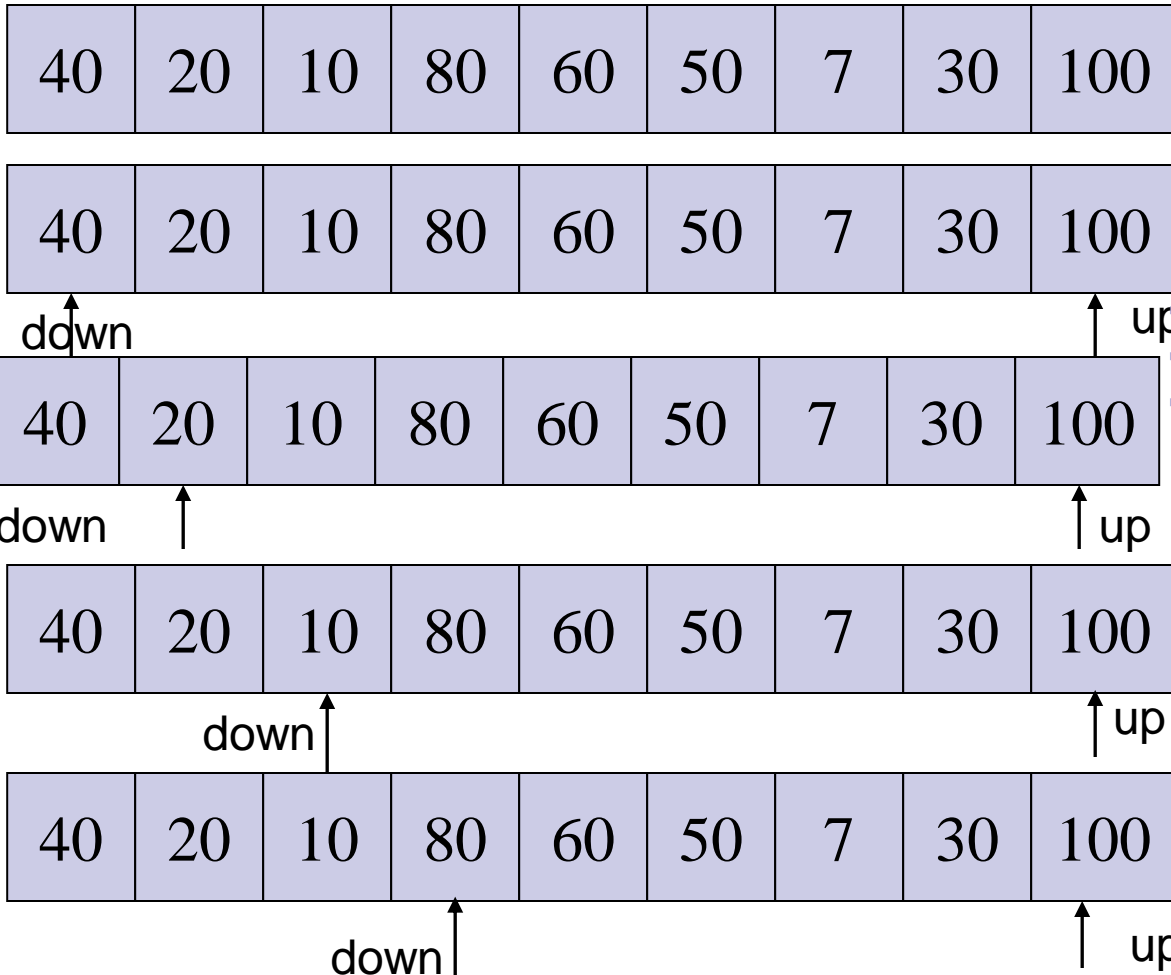


Quicksort ... contd

- Partition(A, lb, ub)
 - $x \leftarrow A[lb]$
 - $up \leftarrow ub$
 - $down \leftarrow lb$
 - while ($down < up$) do
 - while ($A[down] \leq x$ and $down < ub$)
 - $down \leftarrow down + 1$
 - while ($A[up] > x$)
 - $up \leftarrow up - 1$
 - if ($down < up$)
 - Swap ($A[down]$, $A[up]$)
 - $A[lb] \leftarrow A[up]$
 - $A[up] \leftarrow x$
 - Return up



Quicksort ... contd



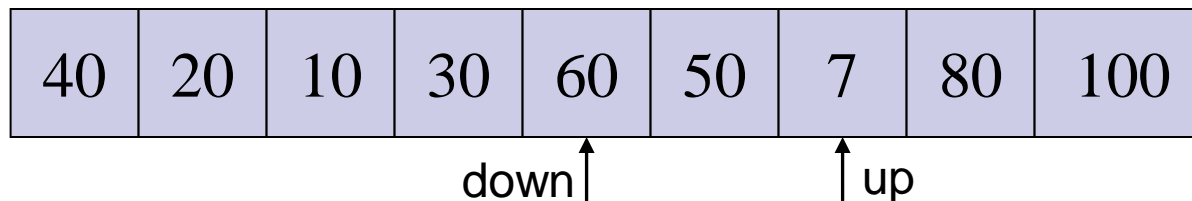
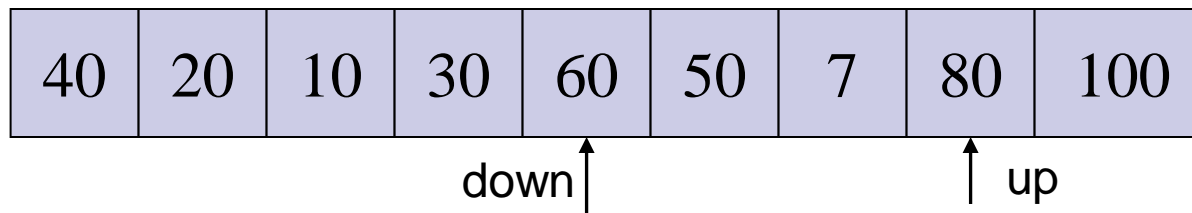
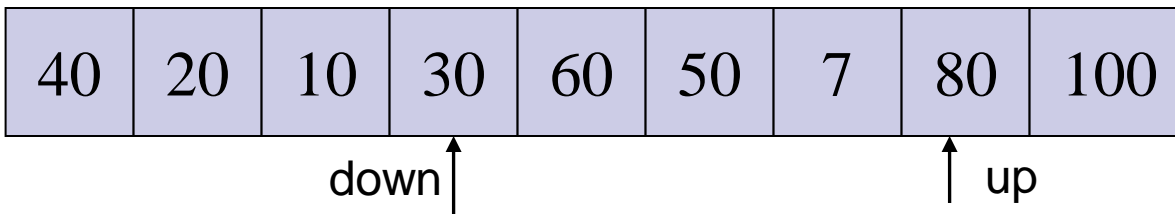
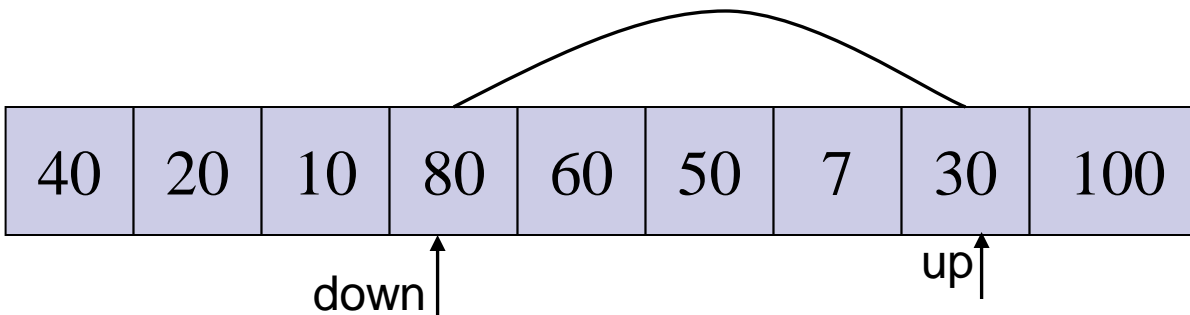
- Partition(A, lb, ub)
 - $x \leftarrow A[lb]$
 - $up \leftarrow ub$
 - $down \leftarrow lb$
 - while ($down < up$) do
 - while ($A[down] \leq x$ and $down < ub$)
 - $down \leftarrow down + 1$
 - while ($A[up] > x$)
 - $up \leftarrow up - 1$
 - if ($down < up$)
 - Swap ($A[down], A[up]$)
 - $A[lb] \leftarrow A[up]$
 - $A[up] \leftarrow x$
 - Return up



Quicksort ...contd

Partition(A, lb, ub)

- $x \leftarrow A[lb]$
- $up \leftarrow ub$
- $down \leftarrow lb$
- while ($down < up$) do
 - while ($A[down] \leq x$ and $down < ub$)
 - $down \leftarrow down + 1$
 - while ($A[up] > x$)
 - $up \leftarrow up - 1$
 - if ($down < up$)
 - Swap ($A[down], A[up]$)
- $A[lb] \leftarrow A[up]$
- $A[up] \leftarrow x$
- Return up



Quicksort ...contd

Partition(A, lb, ub)

- $x \leftarrow A[lb]$
- $up \leftarrow ub$
- $down \leftarrow lb$
- while ($down < up$) do
 - while ($A[down] \leq x$ and $down < up$)
 - $down \leftarrow down + 1$
 - while ($A[up] > x$)
 - $up \leftarrow up - 1$
 - if ($down < up$)
 - Swap ($A[down]$, $A[up]$)
- $A[lb] \leftarrow A[up]$
- $A[up] \leftarrow x$
- Return up

40	20	10	30	60	50	7	80	100
----	----	----	----	----	----	---	----	-----

down ↑ ↑ up

40	20	10	30	7	50	60	80	100
----	----	----	----	---	----	----	----	-----

down ↑ ↑ up

40	20	10	30	7	50	60	80	100
----	----	----	----	---	----	----	----	-----

down ↑ ↑ up

40	20	10	30	7	50	60	80	100
----	----	----	----	---	----	----	----	-----

down ↑↑ up

40	20	10	30	7	50	60	80	100
----	----	----	----	---	----	----	----	-----

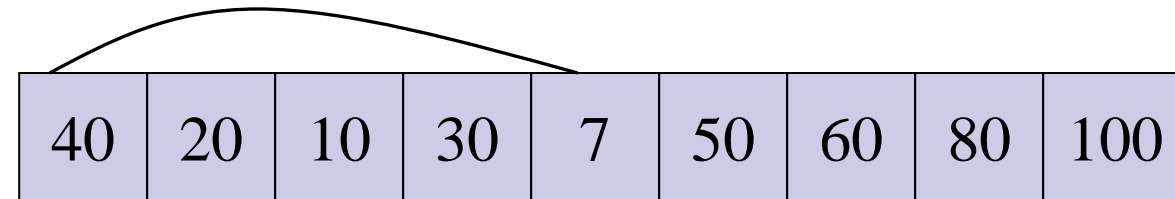
up ↑ ↓ down



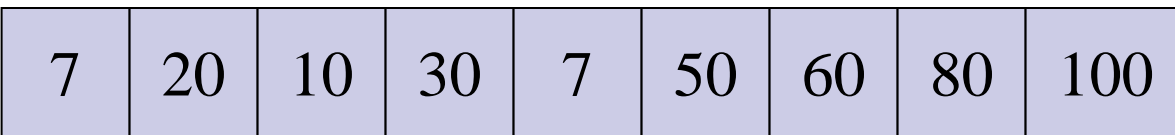
Quicksort ... contd

Partition(A, lb, ub)

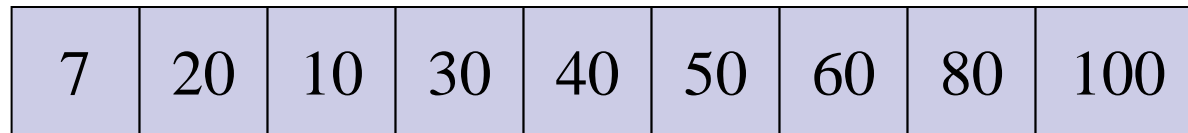
- $x \leftarrow A[lb]$
- $up \leftarrow ub$
- $down \leftarrow lb$
- while ($down < up$) do
 - while ($A[down] \leq x$ and $down < up$)
 - $down \leftarrow down + 1$
 - while ($A[up] > x$)
 - $up \leftarrow up - 1$
 - if ($down < up$)
 - Swap ($A[down]$, $A[up]$)
- $A[lb] \leftarrow A[up]$
- $A[up] \leftarrow x$
- Return up



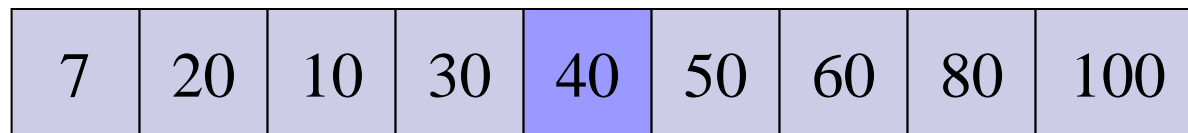
up ↑ down ↑



up ↑ down ↑



up ↑ down ↑



Quicksort... alternate algorithm

QUICKSORT(A, p, r)

```

1  if  $p < r$ 
2       $q = \text{PARTITION}(A, p, r)$ 
3      QUICKSORT( $A, p, q - 1$ )
4      QUICKSORT( $A, q + 1, r$ )

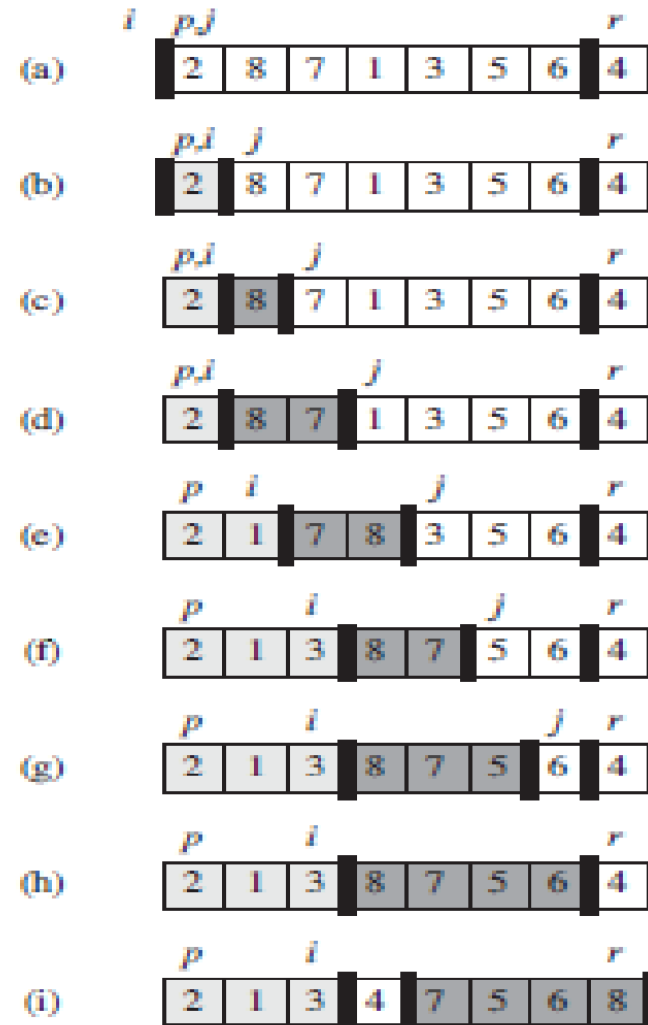
```

PARTITION(A, p, r)

```

1   $x = A[r]$ 
2   $i = p - 1$ 
3  for  $j = p$  to  $r - 1$ 
4      if  $A[j] \leq x$ 
5           $i = i + 1$ 
6          exchange  $A[i]$  with  $A[j]$ 
7  exchange  $A[i + 1]$  with  $A[r]$ 
8  return  $i + 1$ 

```



Quicksort ... Analysis

- For analyzing quicksort, we count only number of element comparisons.
- Running time
 - pivot selection: constant time, i.e. $O(1)$
 - partitioning: linear time, i.e. $O(N)$
 - running time of the two recursive calls
- $T(n) = T(i) + T(n-i-1) + cn$
 - c is a constant
 - i is the number of elements in the partition.



Quicksort... Analysis

- Worst case
 - Complexity $O(n^2)$
- Average Case
 - Complexity $O(n \log n)$
- Applications
 - Commercial Applications



Randomized Quicksort

- *Choose the pivot randomly* (or randomly permute the input array before sorting).
- The running time of the algorithm is independent of input ordering.
- No specific input elicits worst case behavior.
 - The worst case depends on the random number generator.
- Helps modify Quicksort so that it performs well on every input.



Randomized Quicksort

- R_Quicksort(A, lb, ub)
 - if (lb < ub) then
 - $i \leftarrow \text{Random}(\text{lb}, \text{ub})$
 - Swap $A[\text{up}] \leftrightarrow A[i]$
 - $\text{pivot} \leftarrow \text{Partition}(A, \text{lb}, \text{ub})$
 - Quicksort(A, lb, pivot-1)
 - Quicksort(A, pivot+1, ub)



Randomized Quicksort ... Analysis

- Assumptions: all elements are distinct
- All partitions from $0:n-1$ and $n-1:0$ equally likely
- Probability of each partition is $1/n$
- Average case complexity is $O(n \log n)$.



Posteriori Analysis

n	1000	2000	3000	4000	5000
MergeSort	72.8	167.2	275.1	378.5	500.6
QuickSort	36.6	85.1	138.9	205.7	269.0
n	6000	7000	8000	9000	10000
MergeSort	607.6	723.4	811.5	949.2	1073.6
QuickSort	339.4	411.0	487.7	556.3	645.2

Average Sorting
time on Random
inputs

Average Sorting
time on Worst
inputs

n	1000	2000	3000	4000	5000
MergeSort	105.7	206.4	335.2	422.1	589.9
QuickSort	41.6	97.1	158.6	244.9	397.8
n	6000	7000	8000	9000	10000
MergeSort	691.3	794.8	889.5	1067.2	1167.6
QuickSort	383.8	497.3	569.9	616.2	738.1



Why Priori Analysis???

- Telephone directory for mobile users in India
 - 1 billion $\approx 10^9$ phones
- n^2 sorting algorithm requires 10^{18} operations
- CPU cycle: 10^8 operations per second i.e. 10^{10} seconds
 - 2778000 hours \rightarrow 11570 days \rightarrow 300 years
- $n \log n$ algorithm takes only 3×10^{10} operations
 - 300 seconds or 5 minutes



Binary Search

■ Basics

- ☐ Sequential search not very efficient for large lists.
 - ☐ Binary search faster
 - ☐ Works only on sorted list
 - ☐ E.g. searching for some number in a phone book based on person's name.
-
- Logic: compares the target (search key) with the middle element of the array and terminates if it is the same, else it divides the array into two sub arrays and continues the search in left/right sub array if the target is less/greater than the middle element of the array.



Binary Search ... Recursive

■ Bin_Search(A, target, low, high)

- If (high < low)
 - return not found
- mid \leftarrow (low + high) / 2
- if (A[mid] > target)
 - return Bin_Search(A, target, low, mid-1)
- if (A[mid] < target)
 - return Bin_Search(A, target, mid+1, high)

□ else

–15, –6, 0, 7, 9, 23, 54, 82, 101, 112, 125, 131, 142, 151

■ return mid

$x = 151$	<i>low</i>	<i>high</i>	<i>mid</i>
	1	14	7
	8	14	11
	12	14	13
	14	14	14
			found

$x = -14$	<i>low</i>	<i>high</i>	<i>mid</i>
	1	14	7
	1	6	3
	1	2	1
	2	2	2
	2	1	not found

$x = 9$	<i>low</i>	<i>high</i>	<i>mid</i>
	1	14	7
	1	6	3
	4	6	5
			found



Binary Search ...recursive

- Average and worst case complexity is $O(\log n)$
- Easier to implement.
- Note:
 - Successful searches: Best: $\Theta(1)$, Worst and Average: $\Theta(\log_2 n)$
 - Unsuccessful searches: Best, Worst and Average: $\Theta(\log_2 n)$



Binary Search ... non recursive

■ Bin_search (A, n, target)

-15, -6, 0, 7, 9, 23, 54, 82, 101, 112, 125, 131, 142, 151

- left \leftarrow 1
- right \leftarrow n
- While (left \leq right) do
 - mid \leftarrow (left + right)/2
 - If (target = A[mid])
 - then return mid
 - If (target < A[mid])
 - then right \leftarrow mid -1
 - If (target > A[mid])
 - then left \leftarrow mid+1
- Return element not found

$x = 151$	low	high	mid
	1	14	7
	8	14	11
	12	14	13
	14	14	14
			found

$x = -14$	low	high	mid
	1	14	7
	1	6	3
	1	2	1
	2	2	2
	2	1	not found

$x = 9$	low	high	mid
	1	14	7
	1	6	3
	4	6	5
			found



Binary Search ... non recursive

- Complexity order same as non recursive search i.e. $O(\log n)$
- Vs. Linear Search
 - Linear search is of order $O(n)$.
 - Benefit of binary search over linear search becomes significant for lists over about 100 elements.
 - For smaller lists linear search may be faster because of the speed of the simple increment compared with the divisions needed in binary search.



Minimum and Maximum in a list

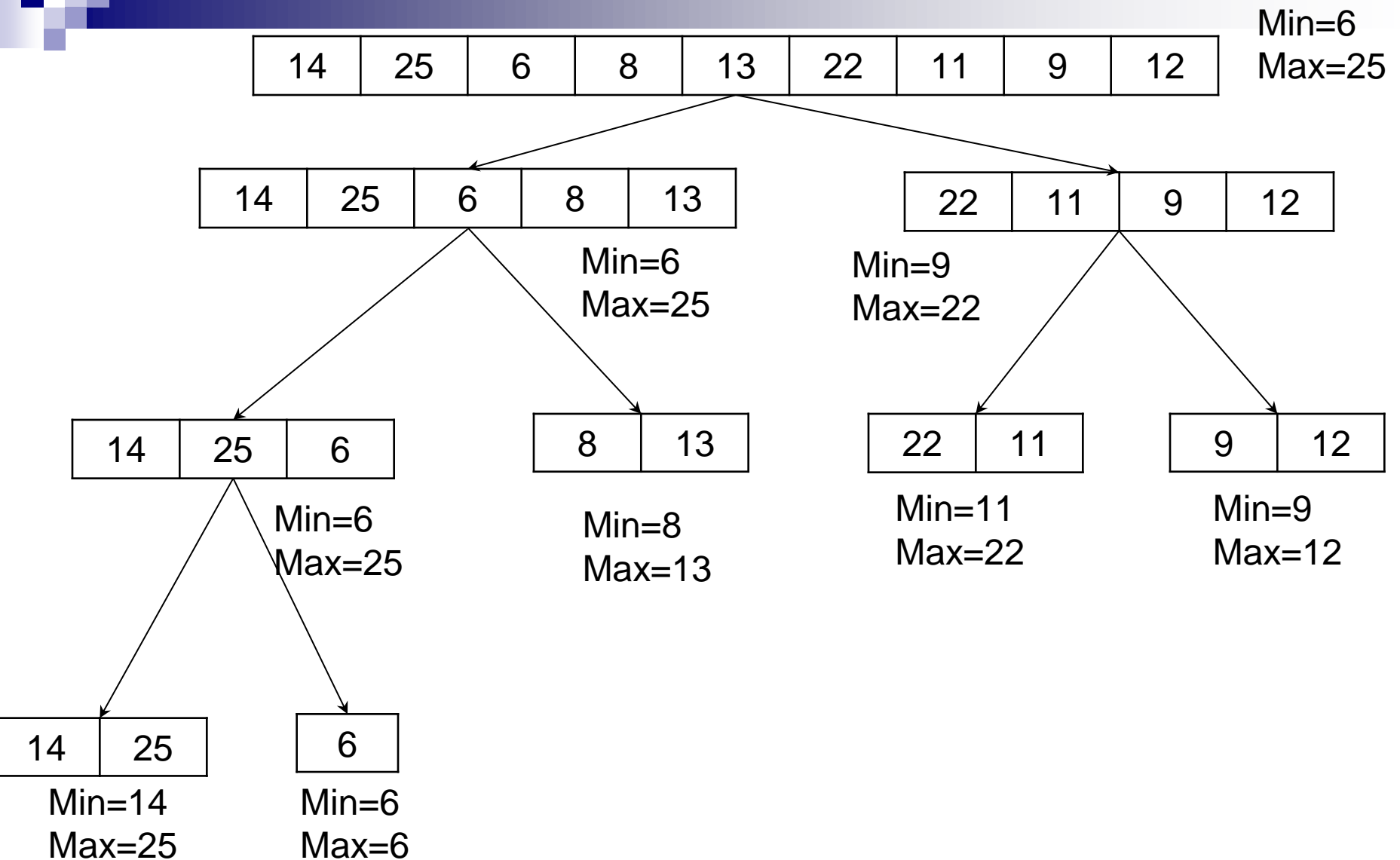
- Find the minimum and maximum in a list of n elements
- Divide the problem into $n/2$ subproblems and find their minimum and maximum recursively
- Combine the solutions



Maxmin(i,j,max,min)

- if $i=j$
 - $\max \leftarrow \min \leftarrow A[i]$
- else if $(i=j-1)$
 - if $(A[i] < A[j])$
 - $\min \leftarrow A[i]$
 - $\max \leftarrow A[j]$
 - else
 - $\max \leftarrow A[i]$
 - $\min \leftarrow A[j]$
- else
 - $\text{mid} \leftarrow (i+j)/2$
 - $\text{Maxmin}(i, \text{mid}, \max, \min)$
 - $\text{Maxmin}(\text{mid}+1, j, \max1, \min1)$
 - if $(\max < \max1)$
 - $\max \leftarrow \max1$
 - if $(\min > \min1)$
 - $\min \leftarrow \min1$





Time Complexity

- $T(n) = 2T(n/2) + d$, $n > 2$
 $= 2$ or d , $n = 2$
 $= 1$ or c , $n = 1$
- $T(n) = O(n)$



Next

- Greedy Method

