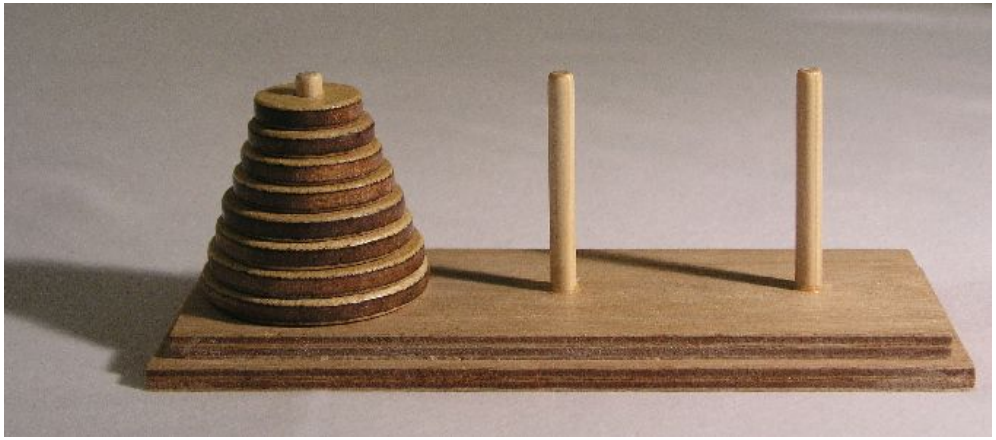
Recursion

Today:

- Ch 8.1 Applications of Recurrence relations
- Ch 8.2 Solving Recurrence relations
- Ch 8.3 Divide and conquer algorithms

Applications of Recurrence relations

The Tower of Hanoi puzzle:



- Move all but the bottom disk to peg 2
- Move bottom disk to peg 3
- Move all remaining disk to peg 3

Applications of Recurrence relations

- The Tower of Hanoi puzzle

Applications of Recurrence relations

The Tower of Hanoi puzzle:

- Move all but the bottom disk to peg 2
- Move bottom disk to peg 3
- Move all remaining disk to peg 3

H_n = number of moves needed for the n-disk problem

$H_n = 2H_{n-1} +1$

Unroll the recursion and prove by induction

Applications of Recurrence relations

Number of bit strings without 2 consecutive 0's:

- must end in 1 or 10

$a_n = a_{n-1} + a_{n-2}$

Same as Fibonacci but $a_1 = 2$

Applications of Recurrence relations

Dynamic programming algorithms:

Counting the number of paths in a rectangular lattice if you can only go right or down.

Solving Recurrence relations

General soln for linear, homogenous equations of order 2:

Put $a_n = r^n$

So the Fibonacci recurrence yields

$r^2 = r + 1$

This has solutions r_1, r_2. The solution of the Fibonacci recurrence relation is

$a_n = \alpha_1 r_1^n + \alpha_2 r_2^n$.

α_1, α_2 must be computed from first two values.

Solving Recurrence relations

Can be extended to higher order equations, non-homogenous equations.

Divide and Conquer algorithms

Example 1: Multiplication of 2 n-bit numbers

Example 2: Merge sort

Example 3:

Finding maximum and minimum of n numbers

Example 4: Binary Search

Divide and Conquer algorithms

Example 1: Multiplication of 2 n-bit numbers

Multiplication of two n-bit numbers

• X =	a	b
• Y =	c	d

- $X = a\,2^{n/2} + b \qquad Y = c\,2^{n/2} + d$

- $XY = ac\,2^{n} + (ad+bc)\,2^{n/2} + bd$

MULT(X,Y):

If |X| = |Y| = 1 then RETURN XY

Break X into a;b and Y into c;d

RETURN

MULT(a,c) 2^n + (MULT(a,d) + MULT(b,c)) $2^{n/2}$ + MULT(b,d)

Time complexity of MULT

- $T(n)$ = time taken by MULT on two n-bit numbers
- What is $T(n)$? Is it $\Theta(n^2)$?
- Hard to compute directly
- Easier to express as a **recurrence relation**!
- $T(1) = k$ for some constant k
- $T(n) = 4\,T(n/2) + c_1 n + c_2$ for some constants c_1 and c_2
- How can we get a $\Theta()$ expression for $T(n)$?

MULT(X,Y):

If |X| = |Y| = 1 then RETURN XY

Break X into a;b and Y into c;d

RETURN

MULT(a,c) 2^n + (MULT(a,d) + MULT(b,c)) $2^{n/2}$ + MULT(b,d)

Time complexity of MULT

Make it concrete

- $T(1) = 1$
- $T(n) = 4\ T(n/2) + n$

Technique 1: Guess and verify

$T(n) = 2n^2 - n$

Holds for $n=1$

$T(n) = 4\ (2(n/2)^2 - n/2 + n)$

$\quad = 2n^2 - n$

Time complexity of MULT

- $T(1) = 1$ & $T(n) = 4\,T(n/2) + n$

Technique 2: Expand recursion

$$
\begin{aligned}
T(n) &= 4\,T(n/2) + n \\
&= 4\,(4T(n/4) + n/2) + n = 4^2T(n/4) + n + 2n \\
&= 4^2(4T(n/8) + n/4) + n + 2n \\
&= 4^3T(n/8) + n + 2n + 4n \\
&= \ldots\ldots\ldots \\
&= 4^kT(1) + n + 2n + 4n + \ldots + 2^{k-1}n \text{ where } 2^k = n
\end{aligned}
$$

GUESS

$$
\begin{aligned}
&= n^2 + n\,(1 + 2 + 4 + \ldots + 2^{k-1}) \\
&= n^2 + n\,(2^k - 1) \\
&= 2\,n^2 - n \quad [\text{NOT FASTER THAN BEFORE}]
\end{aligned}
$$

Gaussified MULT (Karatsuba 1962)

MULT(X,Y):

If $|X| = |Y| = 1$ then RETURN XY

Break X into a;b and Y into c;d

e = MULT(a,c) and f =MULT(b,d)

RETURN $e2^n + (\text{MULT}(a+b, c+d) - e - f)\, 2^{n/2} + f$

- $T(n) = 3\, T(n/2) + n$
- Actually: $T(n) = 2\, T(n/2) + T(n/2 + 1) + kn$

Time complexity of Gaussified MULT

- $T(1) = 1$ & $T(n) = 3\,T(n/2) + n$

Technique 2: Expand recursion

$$
\begin{aligned}
T(n) &= 3\,T(n/2) + n \\
&= 3\,(3T(n/4) + n/2) + n = 3^2T(n/4) + n + 3/2n \\
&= 3^2(3T(n/8) + n/4) + n + 3/2n \\
&= 3^3T(n/8) + n + 3/2n + (3/2)^2n \\
&= \ldots\ldots\ldots \\
&= 3^kT(1) + n + 3/2n + (3/2)^2n + \ldots + (3/2)^{k-1}n \text{ where } 2^k = n \\
&= 3^{\log_2 n} + n(1 + 3/2 + (3/2)^2 + \ldots + (3/2)^{k-1}) \\
&= n^{\log_2 3} + 2n\,((3/2)^k - 1) \\
&= n^{\log_2 3} + 2n\,(n^{\log_2 3}/n - 1) \\
&= 3n^{\log_2 3} - 2n
\end{aligned}
$$

Not just 25% savings!

$\theta(n^2)$ vs $\theta(n^{1.58..})$

Divide and Conquer algorithms

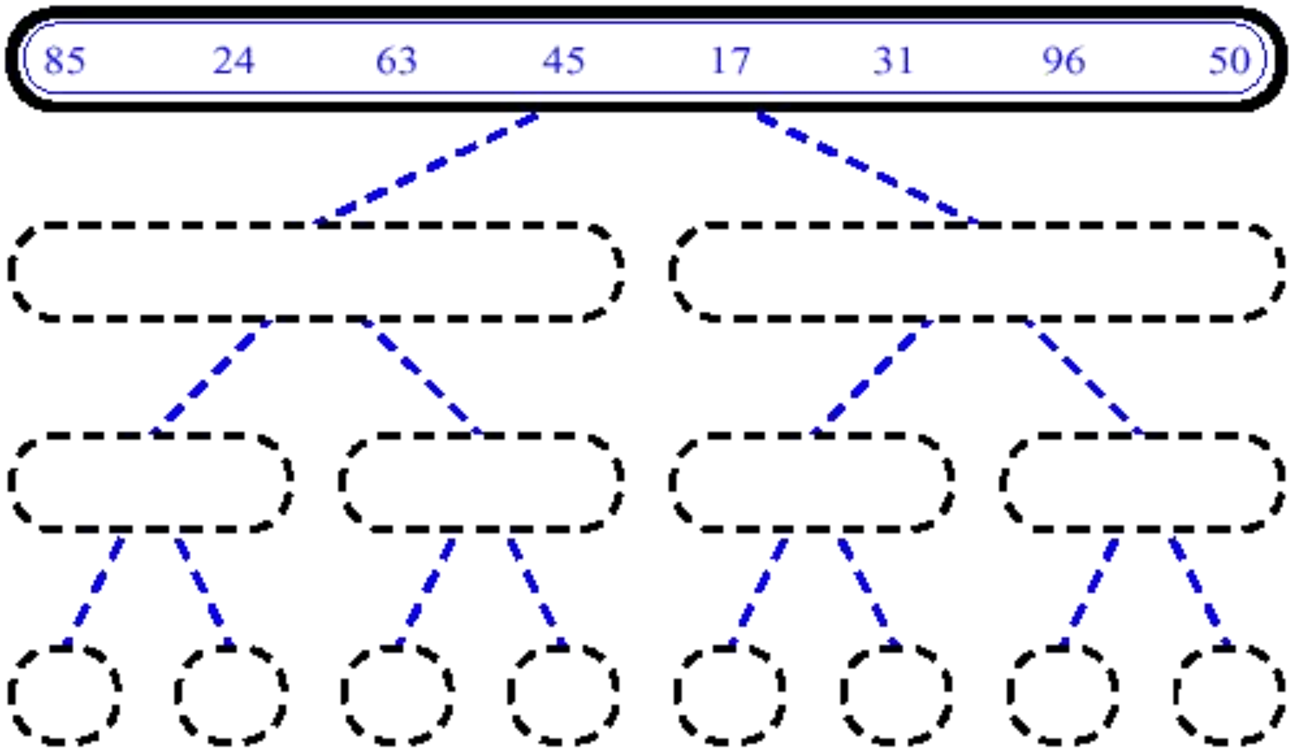
Example 2: Merge Sort

Merge Sort: Algorithm

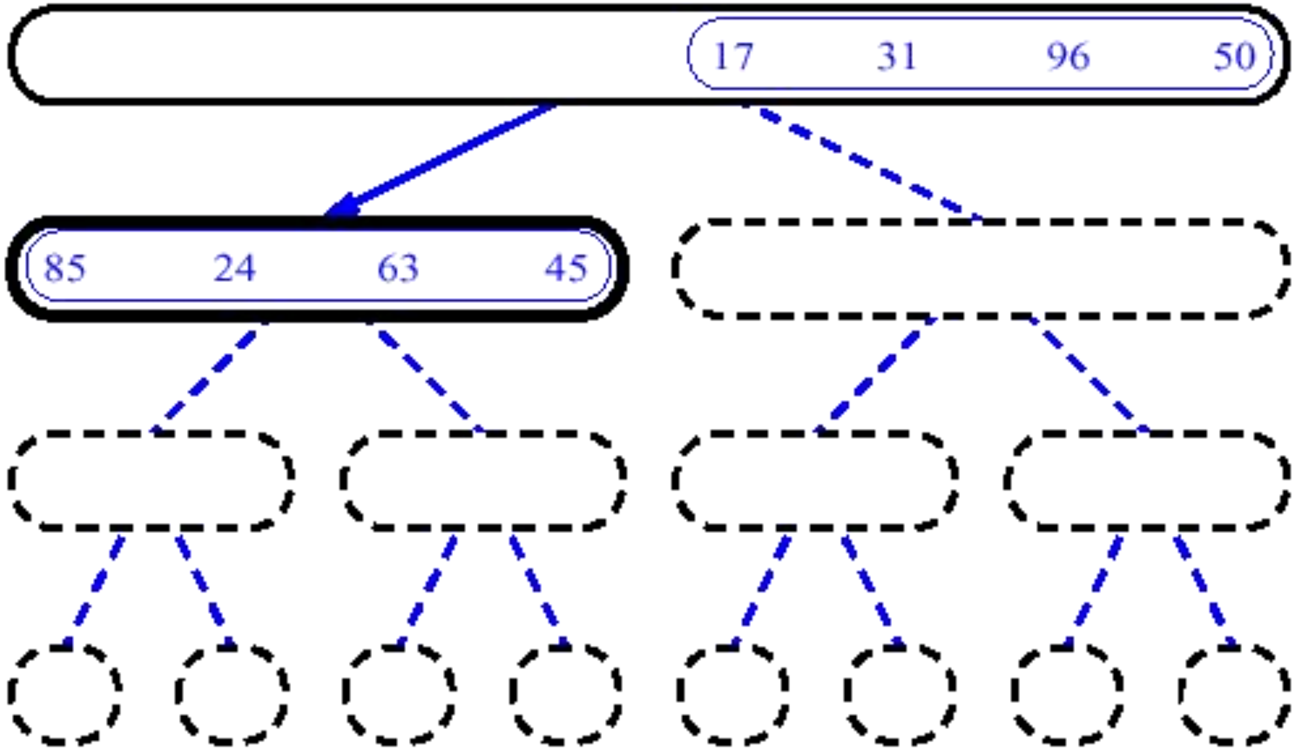
```
Merge-Sort(A, p, r)
   if p < r then
      q←(p+r)/2
      Merge-Sort(A, p, q)
      Merge-Sort(A, q+1, r)
      Merge(A, p, q, r)
```

```
Merge(A, p, q, r)
   Take the smallest of the two topmost elements of
sequences A[p..q] and A[q+1..r] and put into the
resulting sequence. Repeat this, until both sequences
are empty. Copy the resulting sequence into A[p..r].
```

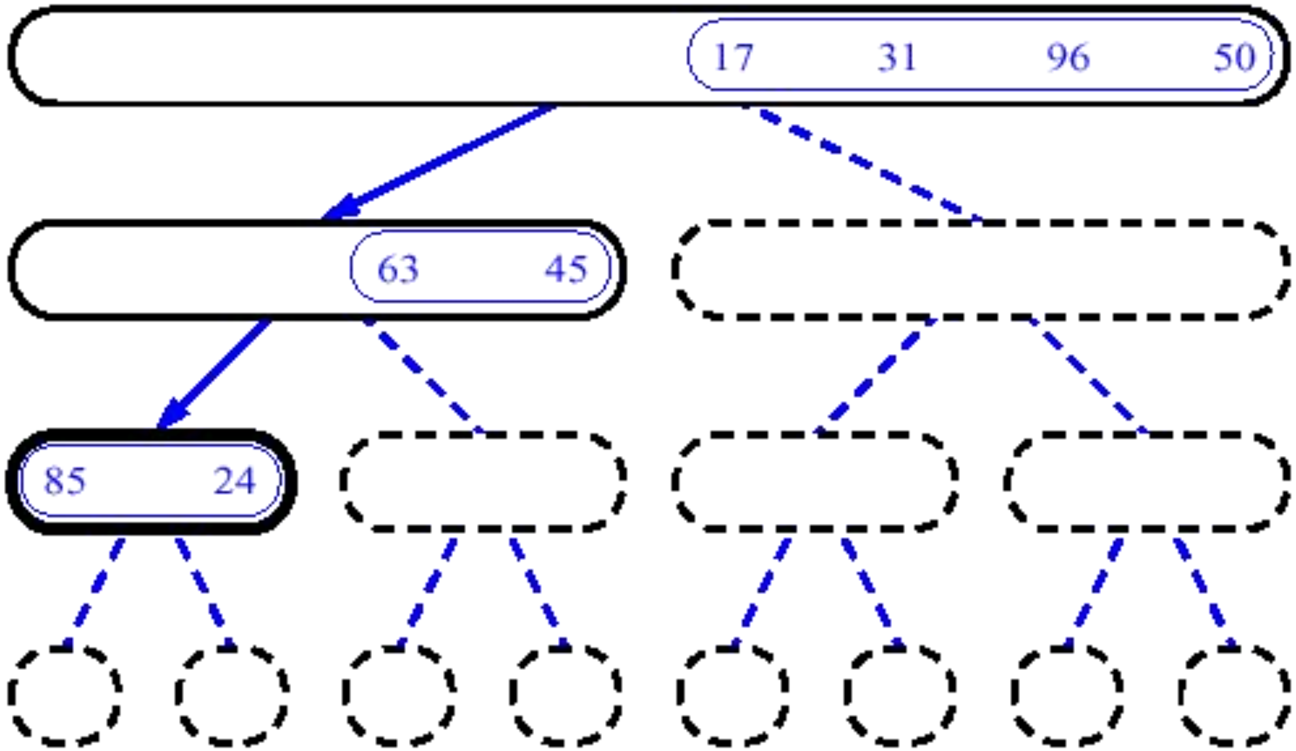
Merge Sort: Example



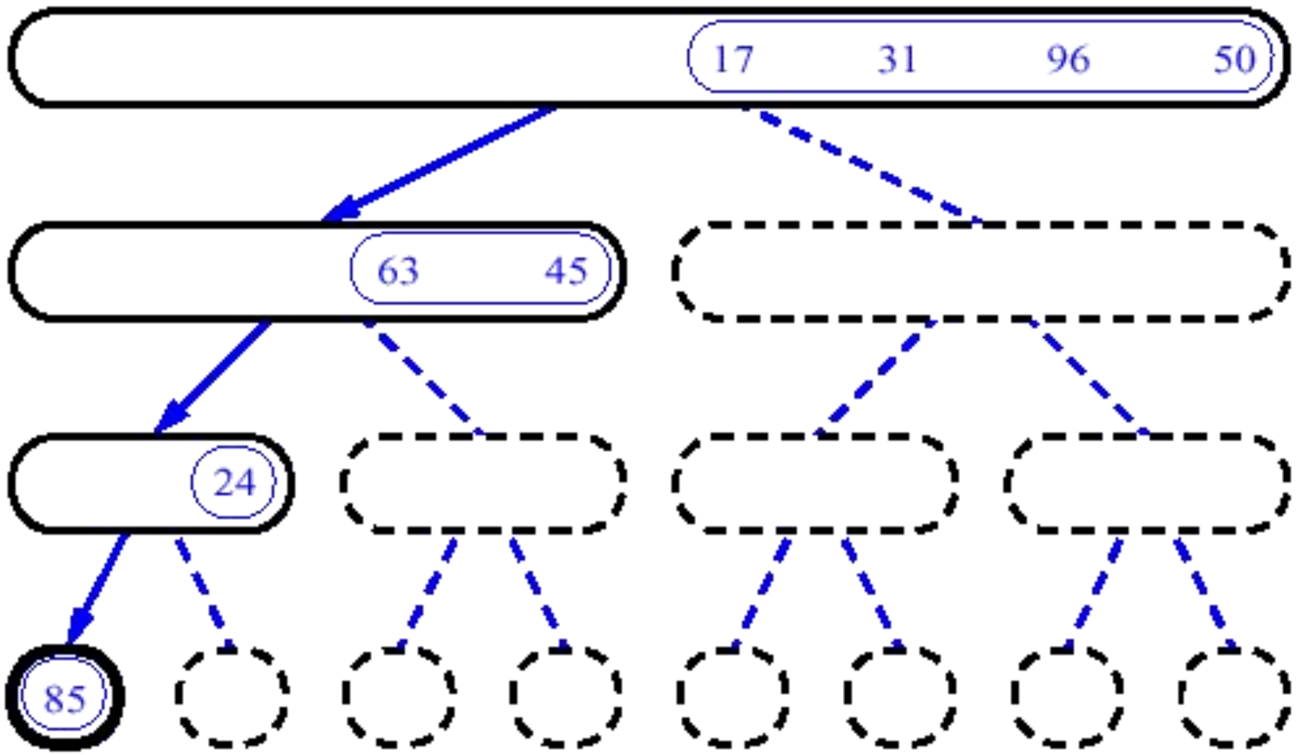
Merge Sort: Example



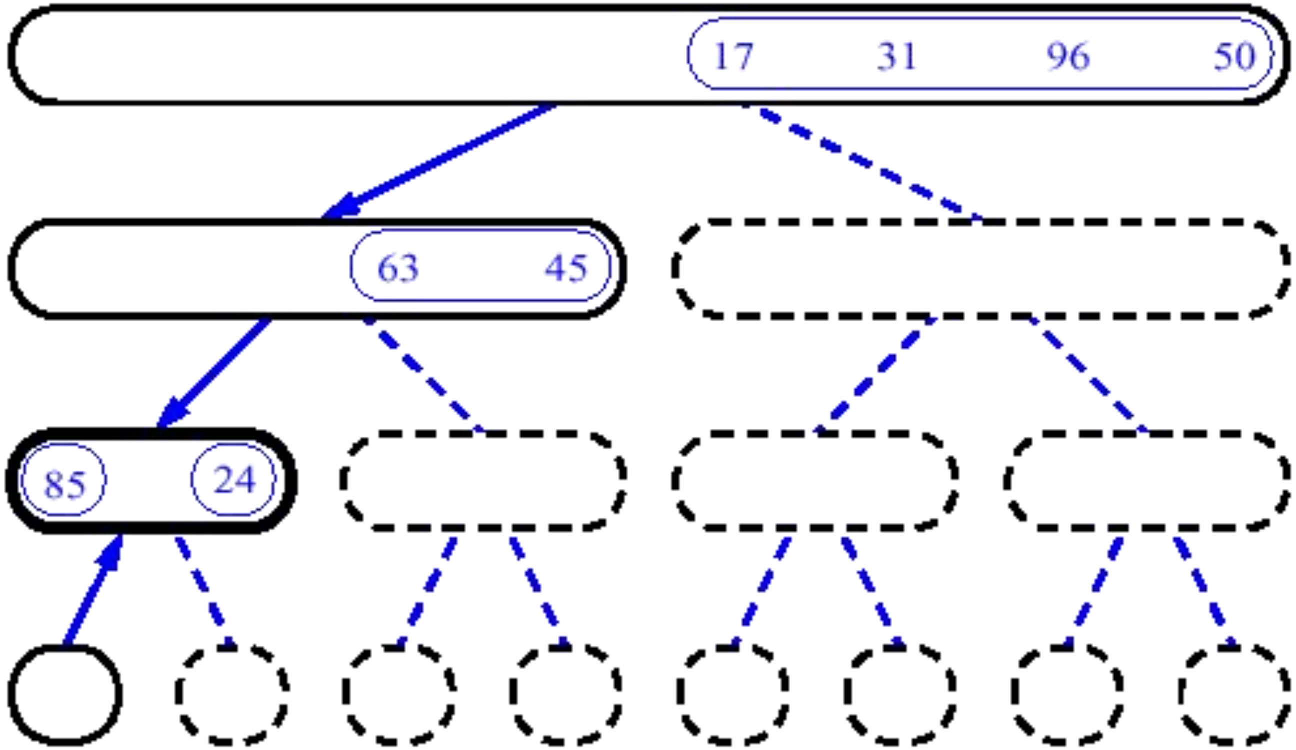
Merge Sort: Example



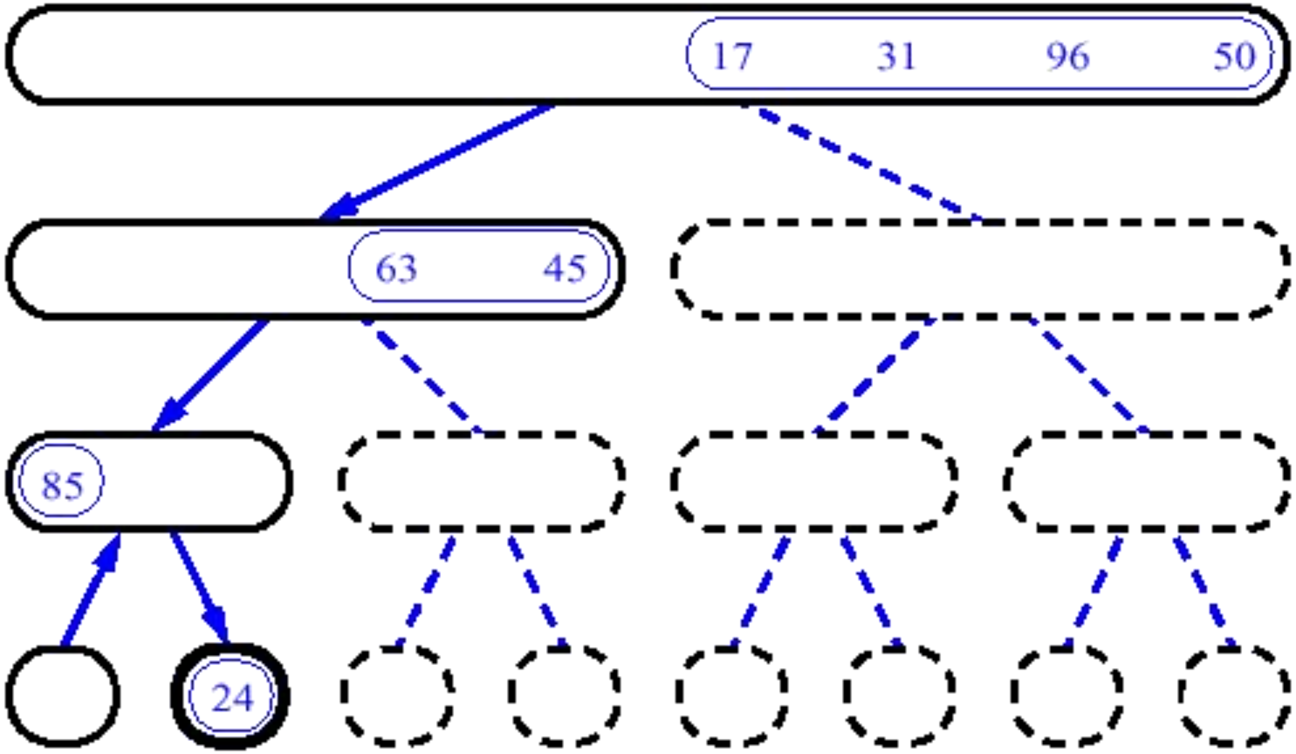
Merge Sort: Example



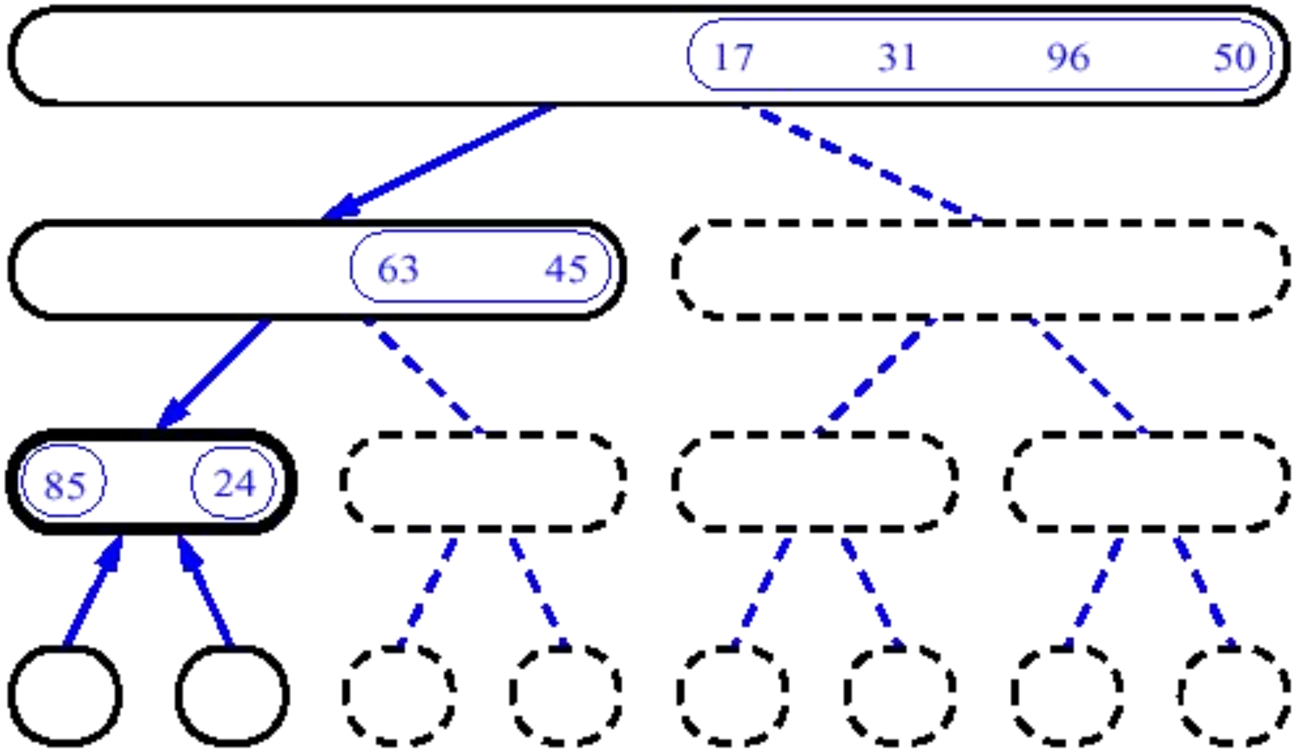
Merge Sort: Example



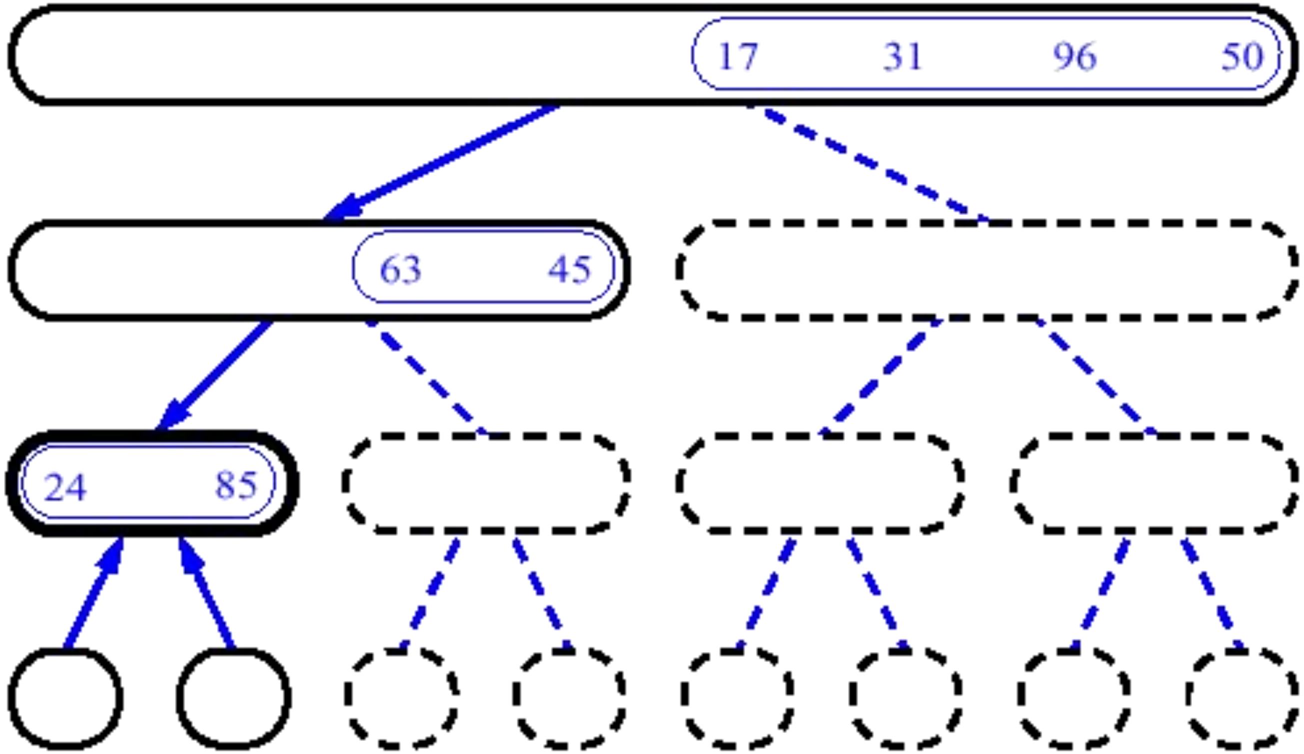
Merge Sort: Example



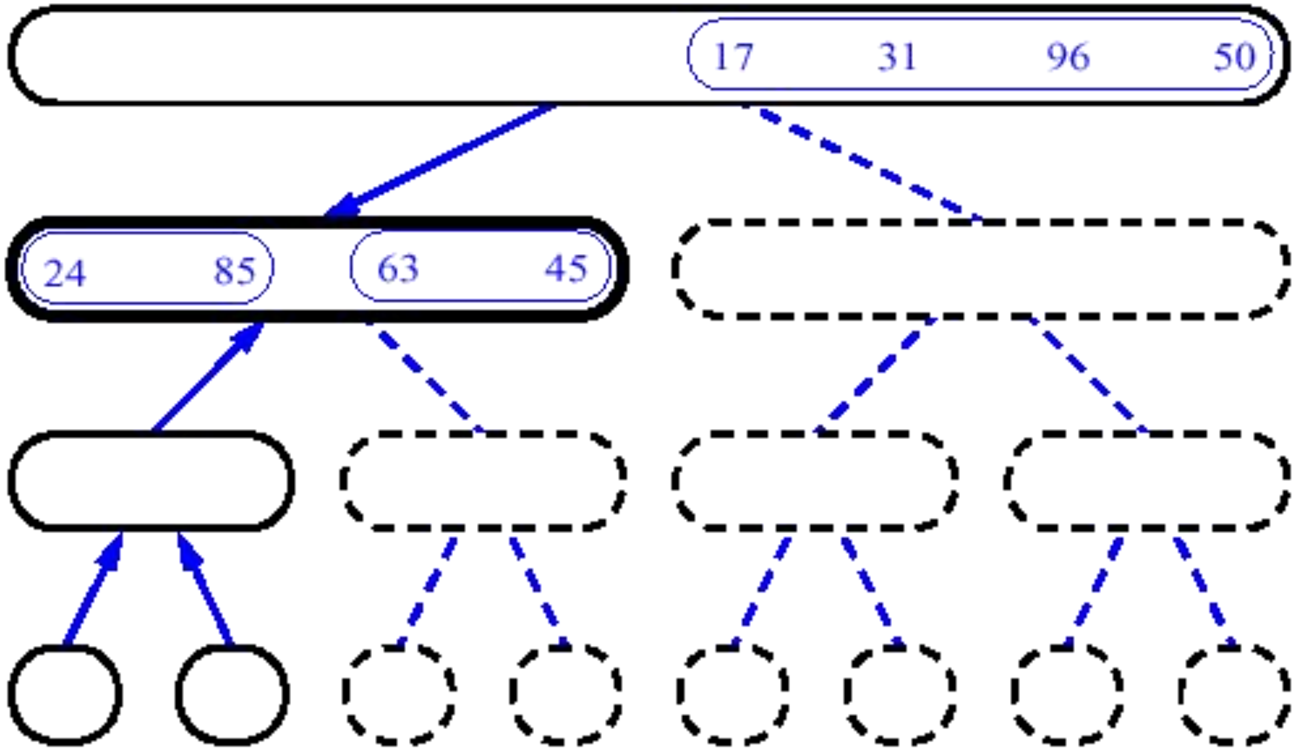
Merge Sort: Example



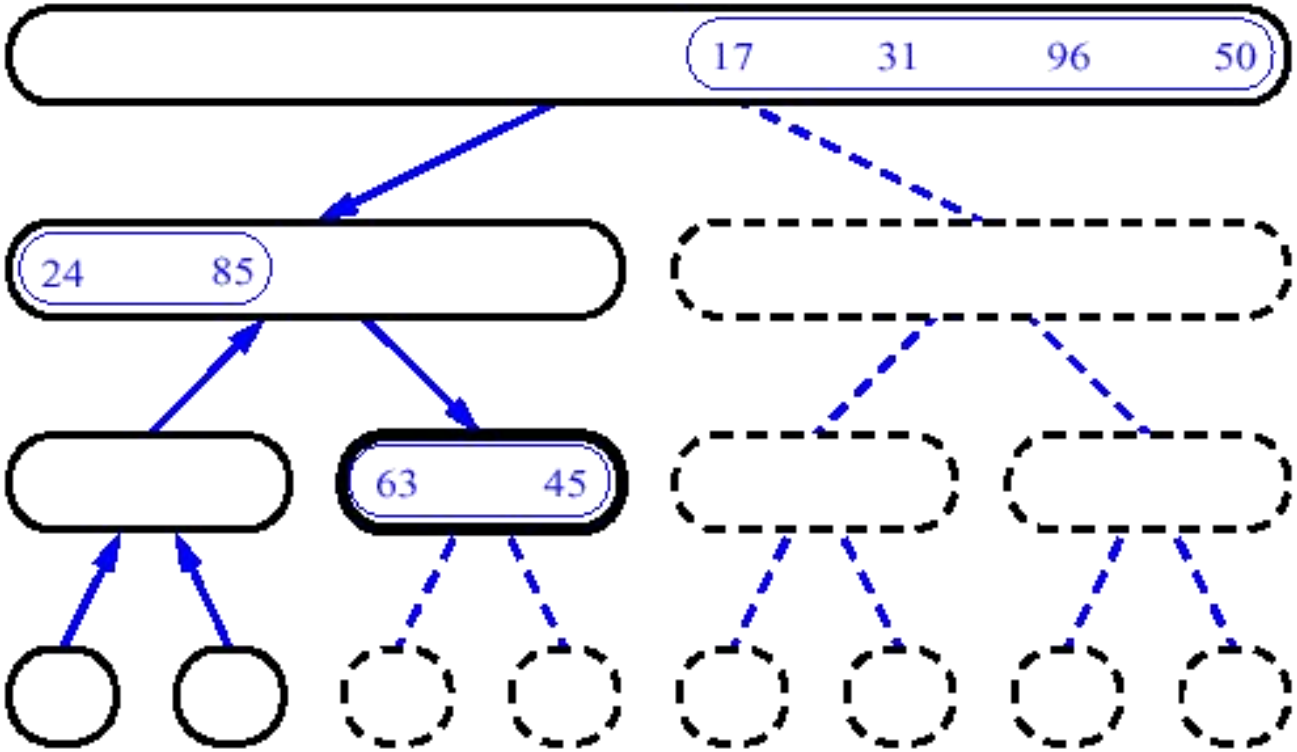
Merge Sort: Example



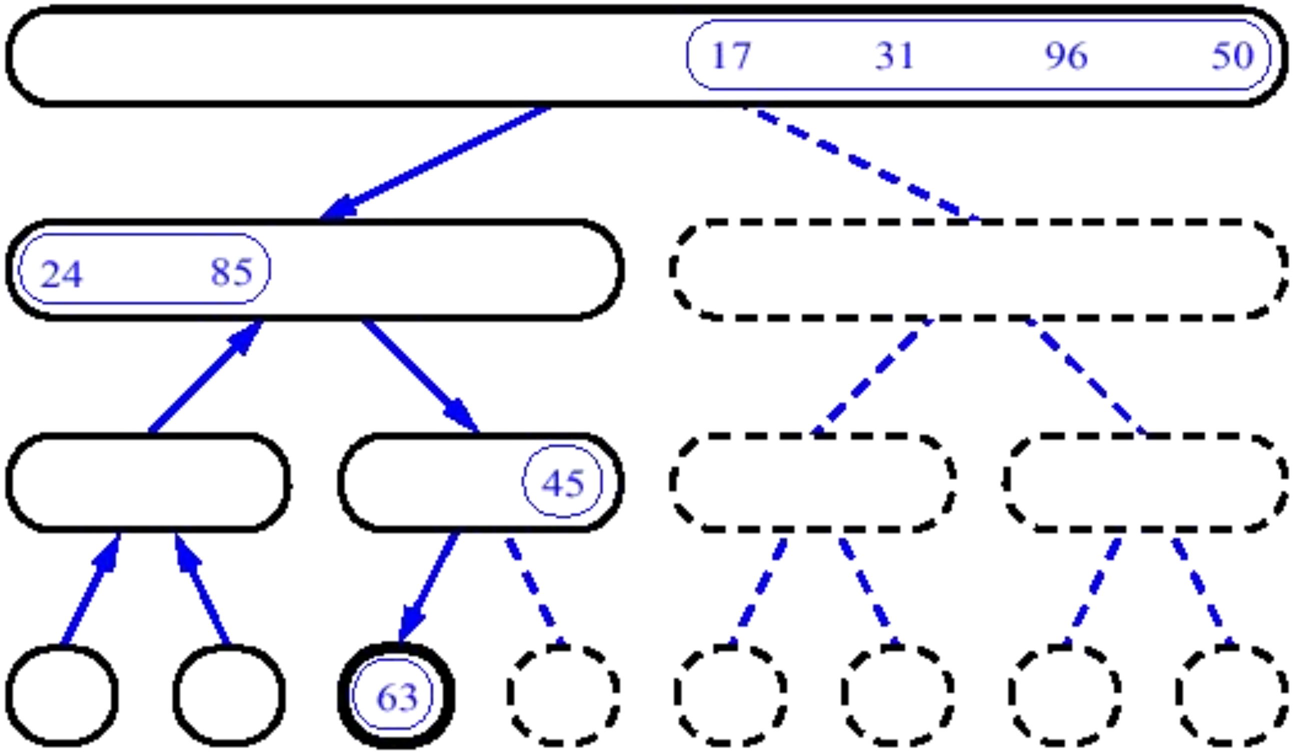
Merge Sort: Example



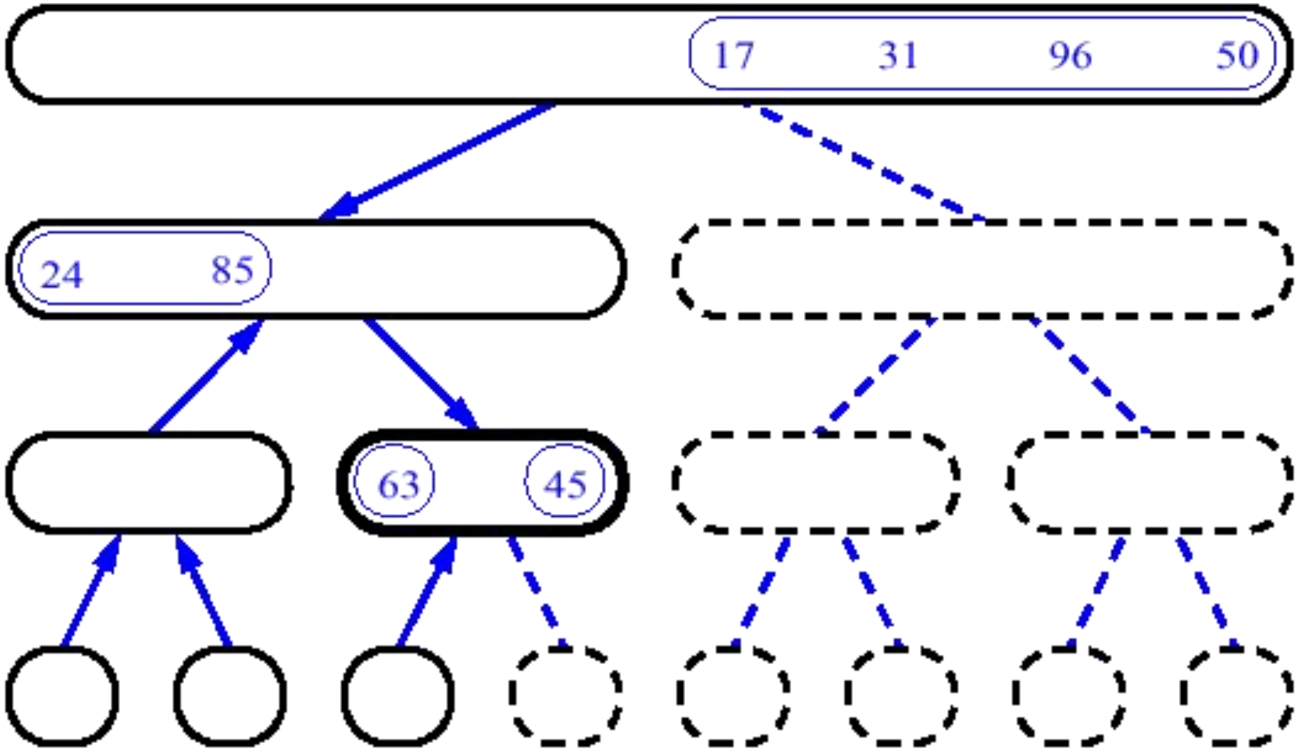
Merge Sort: Example



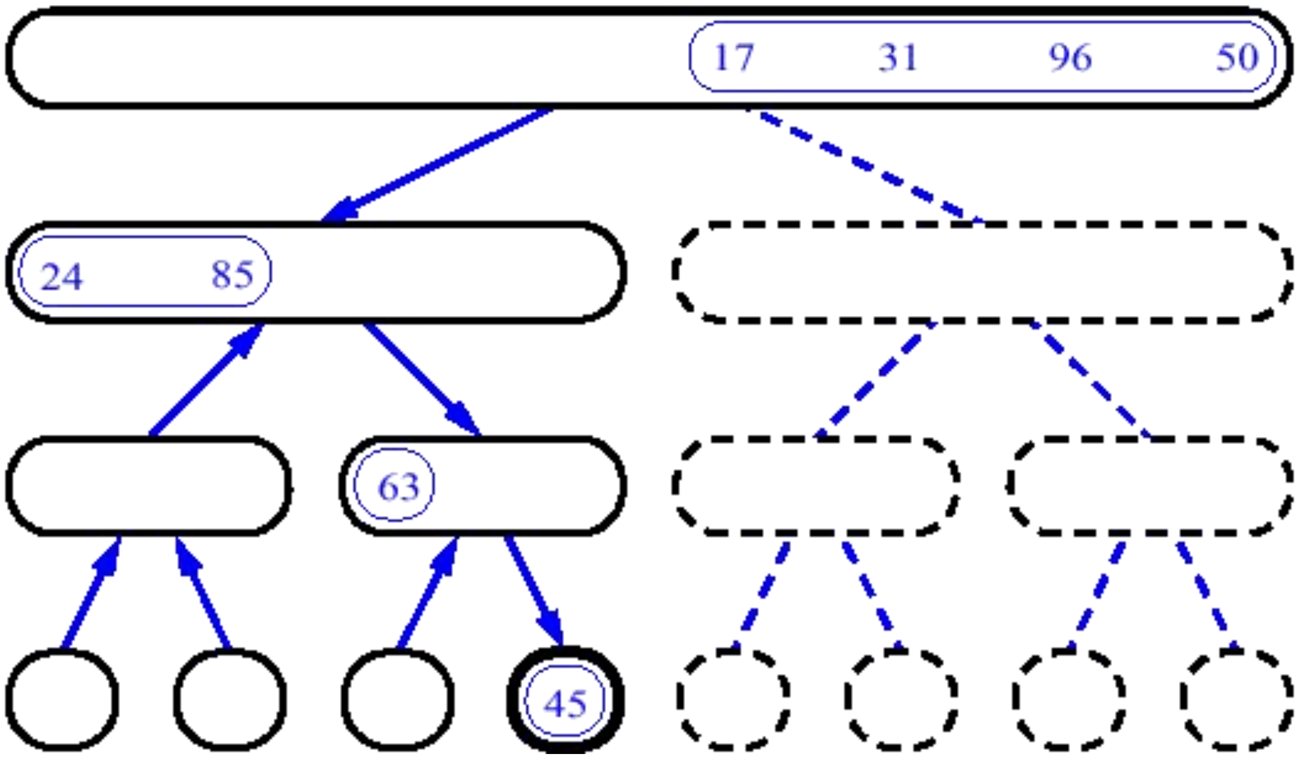
Merge Sort: Example



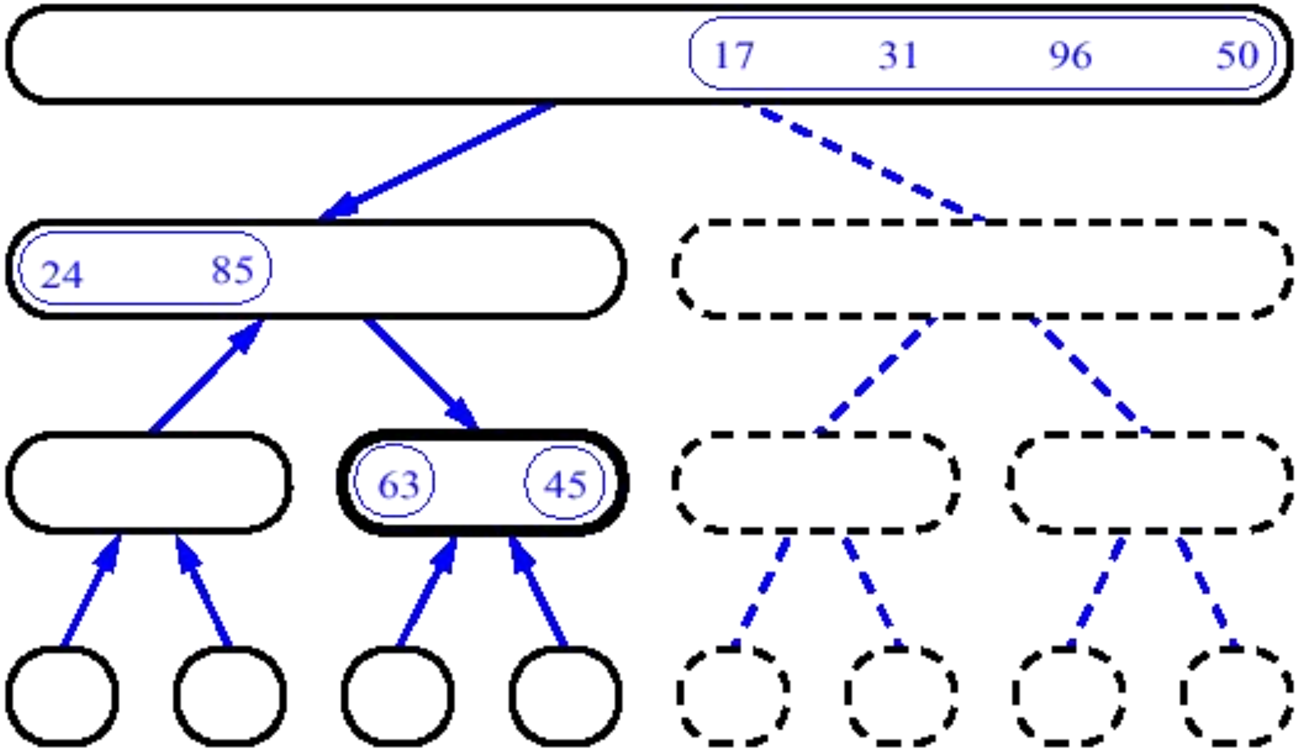
Merge Sort: Example



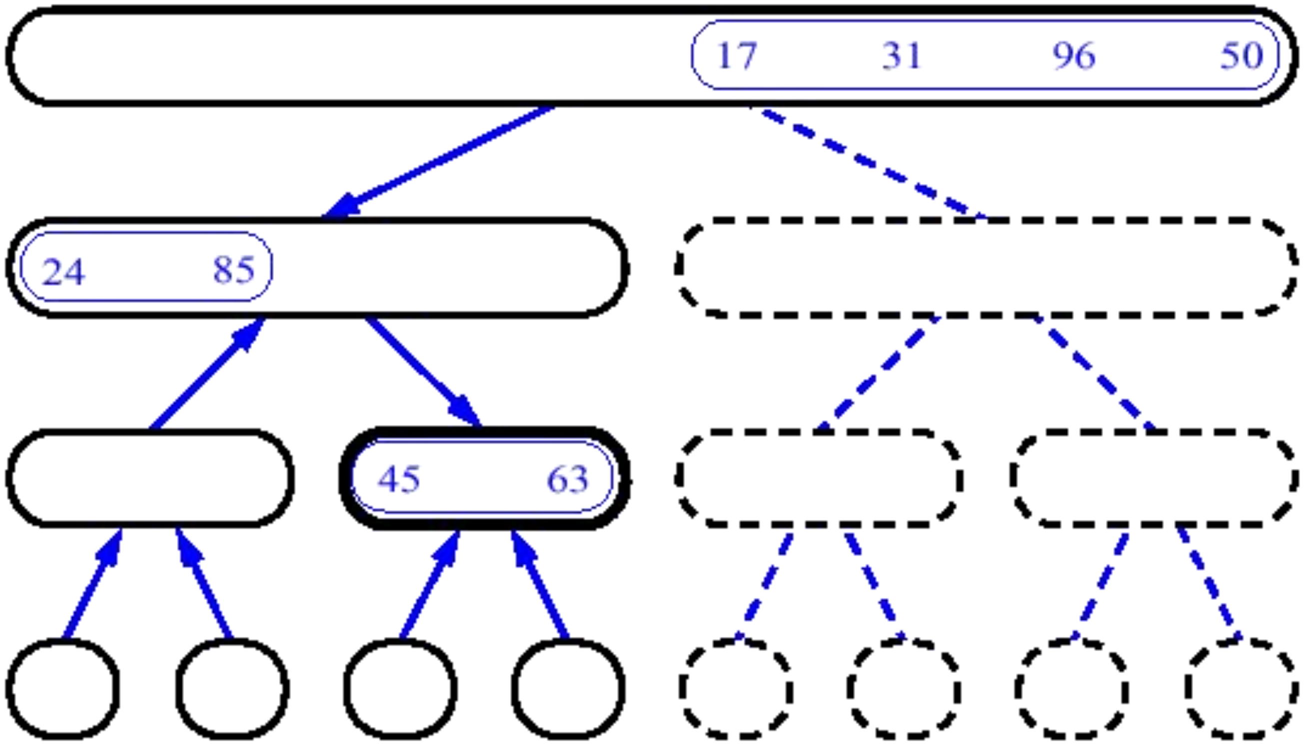
Merge Sort: Example



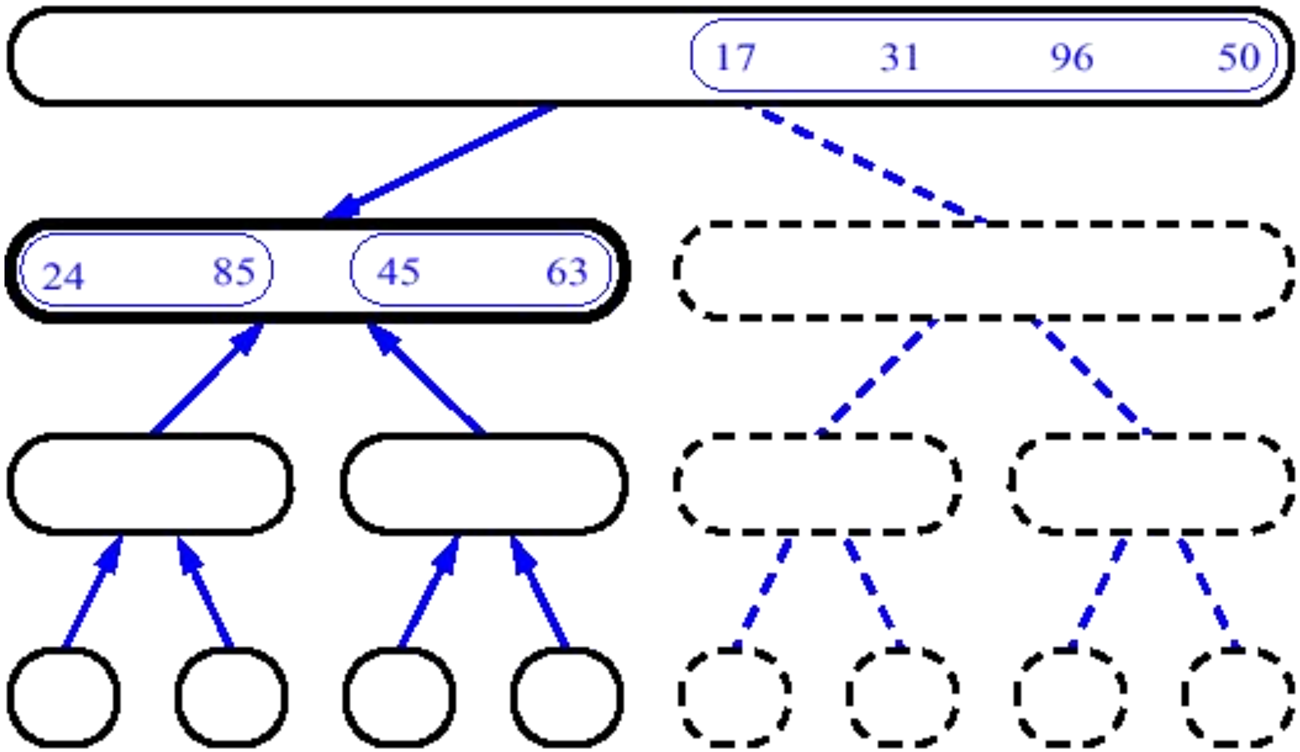
Merge Sort: Example



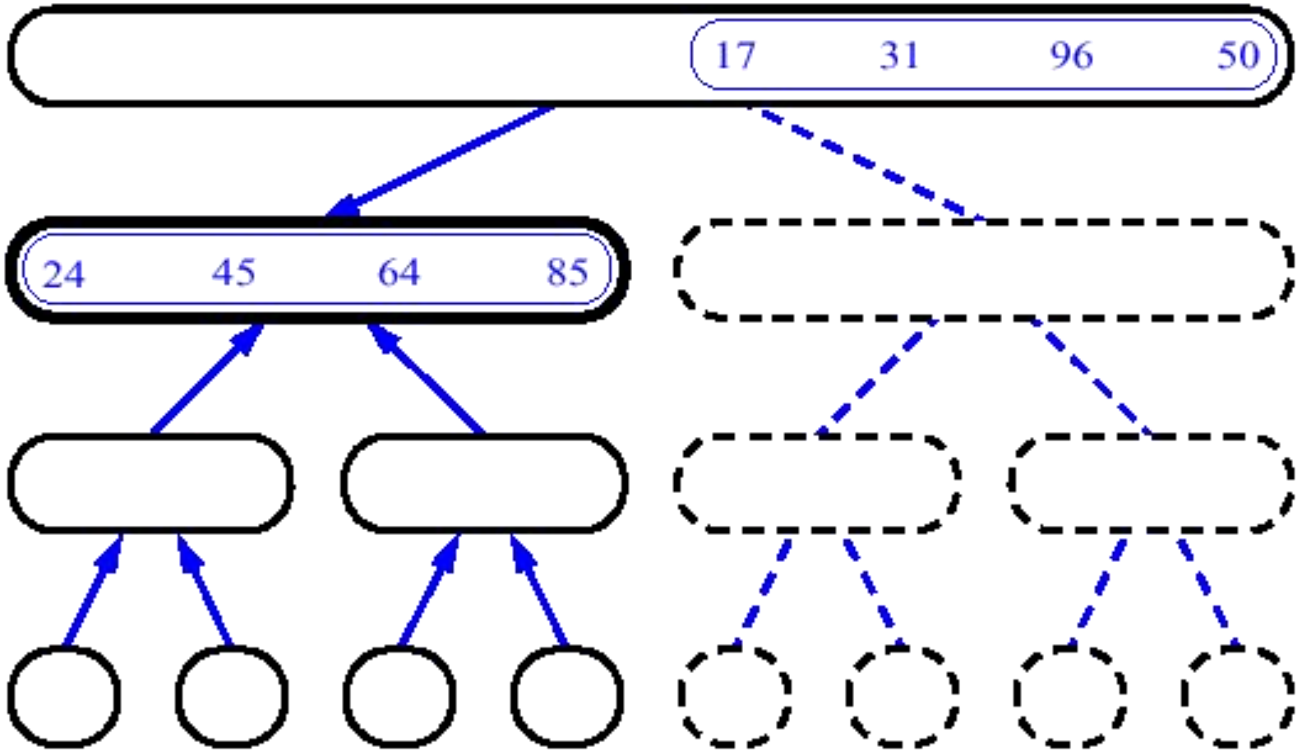
Merge Sort: Example



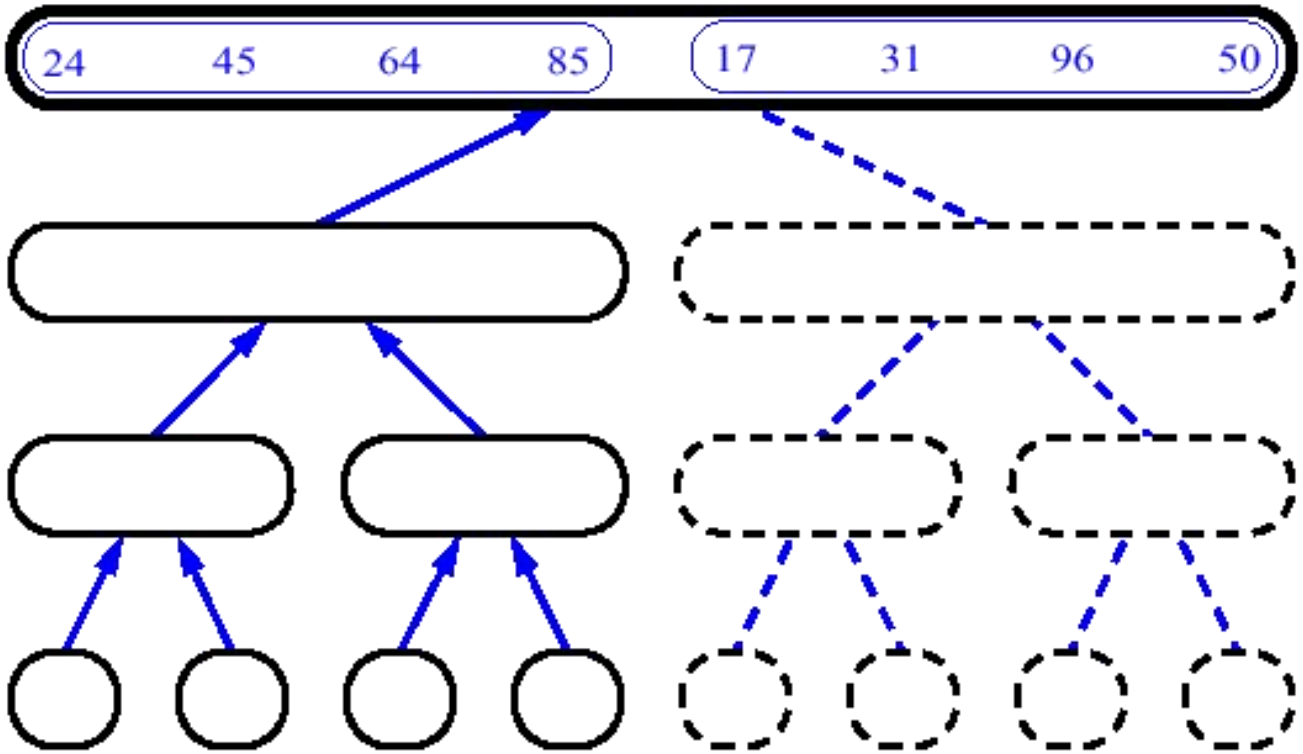
Merge Sort: Example



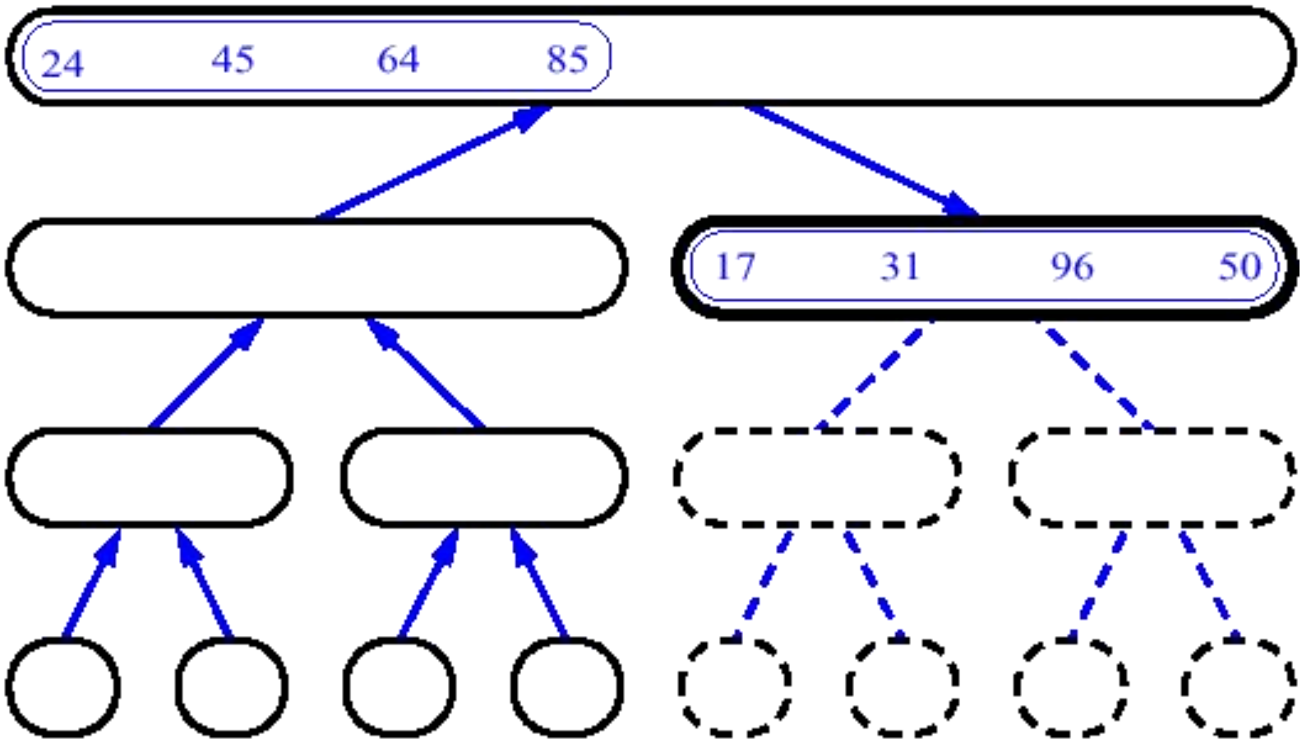
Merge Sort: Example



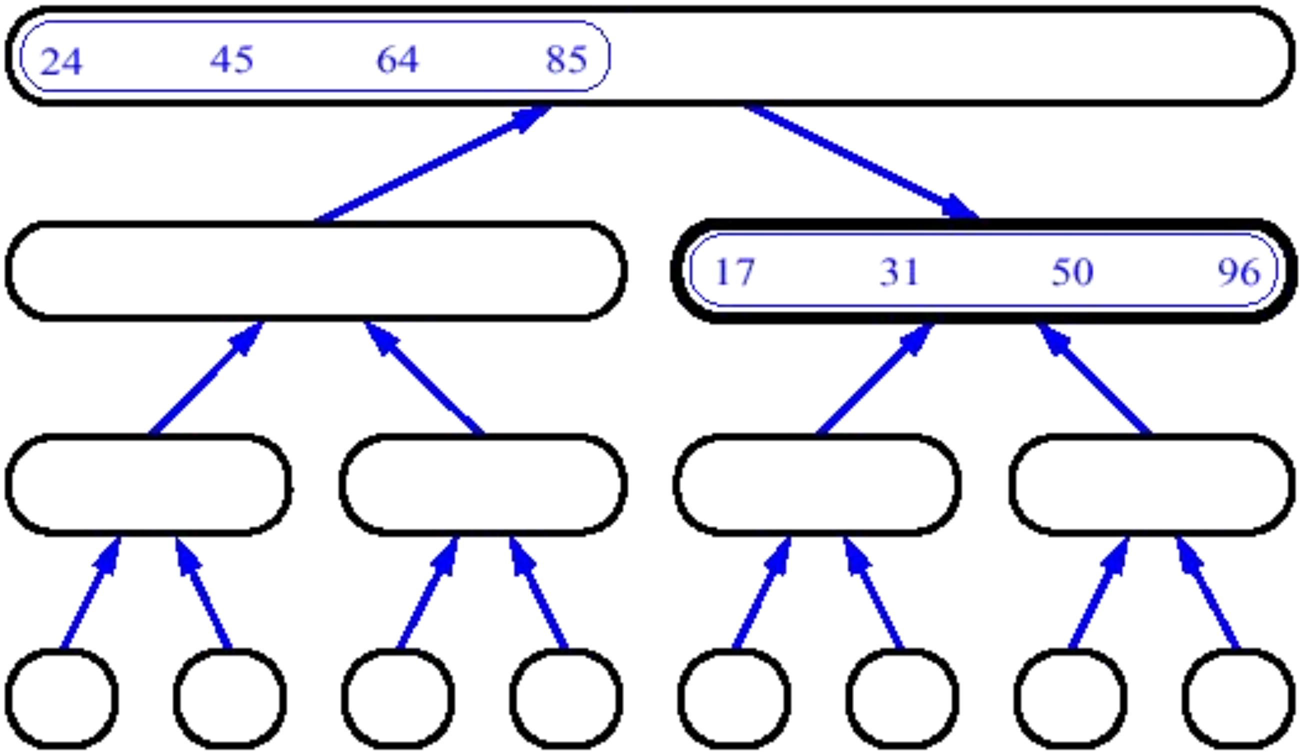
Merge Sort: Example



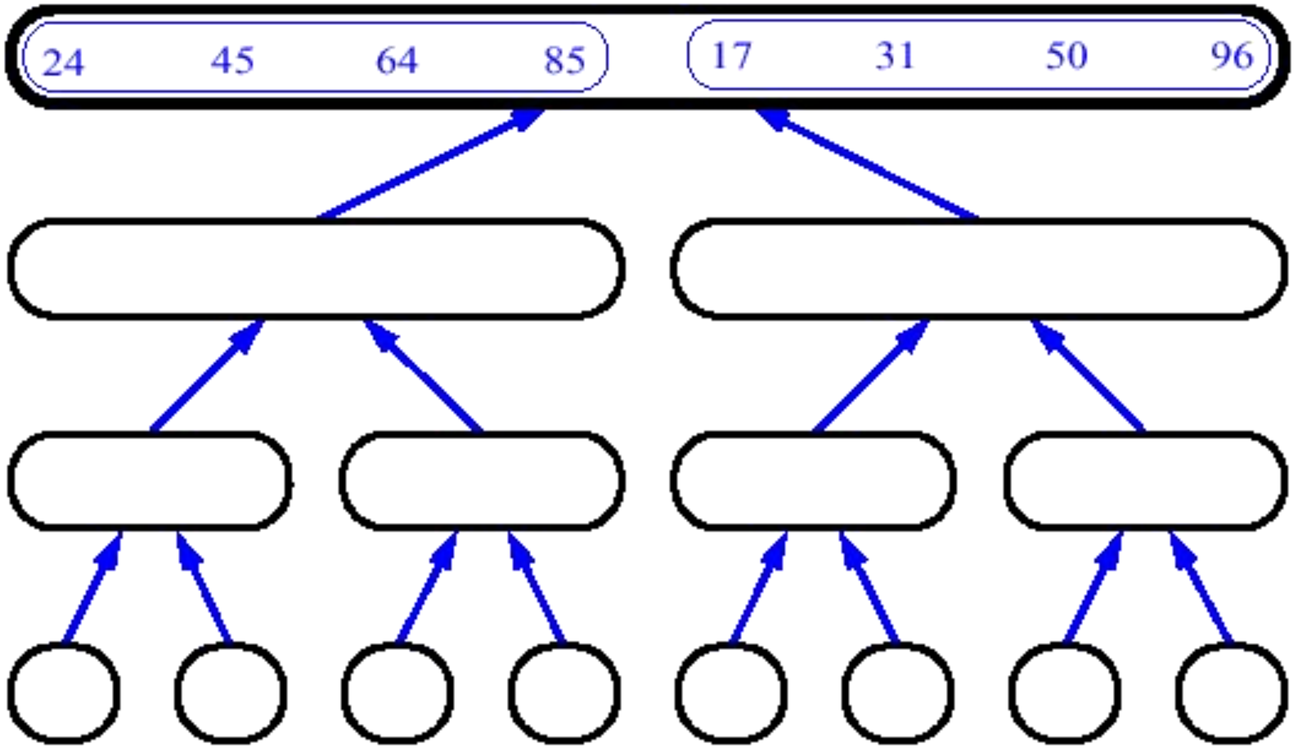
Merge Sort: Example



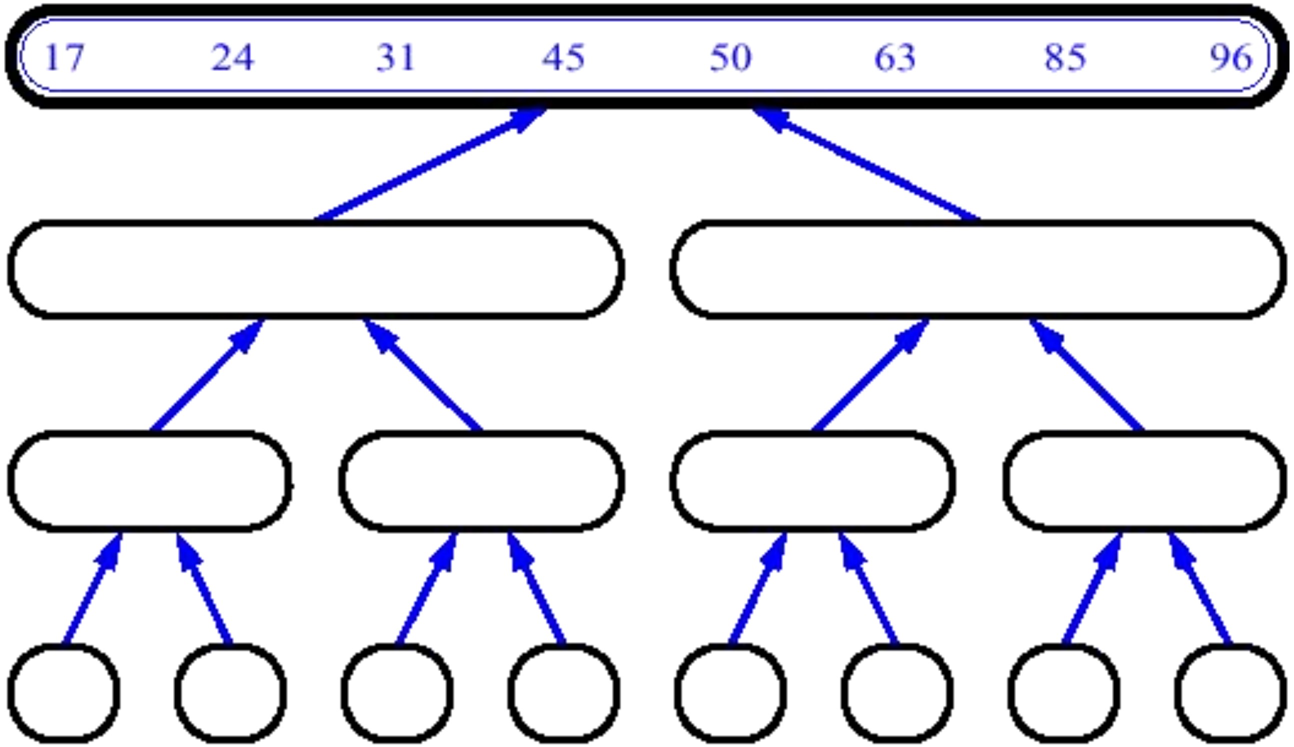
Merge Sort: Example



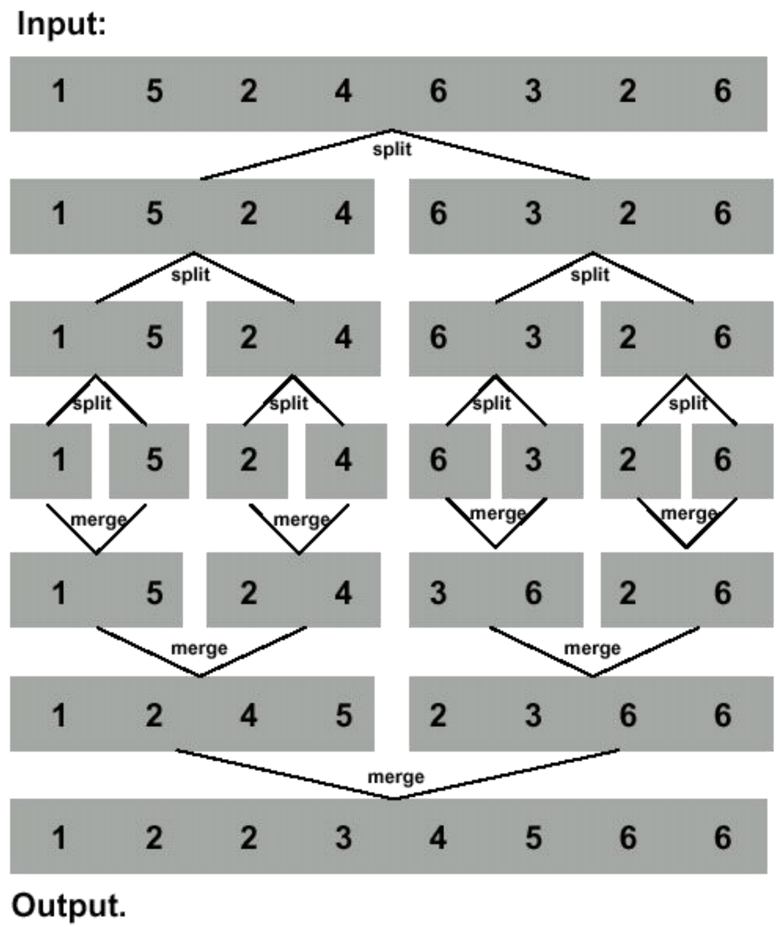
Merge Sort: Example



Merge Sort: Example



Merge Sort: Example



Recurrences

- Running times of algorithms with **Recursive calls** can be described using recurrences
- A **recurrence** is an equation or inequality that describes a function in terms of its value on smaller inputs

Example: Merge Sort

$$T(n) = \begin{cases} \text{solving\_trivial\_problem} & \text{if } n = 1 \\ \text{num\_pieces } T(n/\text{subproblem\_size\_factor}) + \text{dividing} + \text{combining} & \text{if } n > 1 \end{cases}$$

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

Solving recurrences

- Repeated substitution method
  - Expanding the recurrence by substitution and noticing patterns
- Substitution method
  - guessing the solutions
  - verifying the solution by the mathematical induction
- Recursion-trees
- Master method
  - templates for different classes of recurrences

Repeated Substitution Method

- Let's find the running time of merge sort (let's assume that $n=2^b$, for some b).

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

$$\begin{aligned}
T(n) &= 2T\left(n/2\right) + n \quad \text{substitute} \\
&= 2\left(2T\left(n/4\right) + n/2\right) + n \quad \text{expand} \\
&= 2^2 T(n/4) + 2n \quad \text{substitute} \\
&= 2^2(2T(n/8) + n/4) + 2n \quad \text{expand} \\
&= 2^3 T(n/8) + 3n \quad \text{observe the pattern} \\
T(n) &= 2^i T(n/2^i) + in \\
&= 2^{\lg n} T(n/n) + n \lg n = n + n \lg n
\end{aligned}$$

Repeated Substitution Method

- The procedure is straightforward:
  - Substitute
  - Expand
  - Substitute
  - Expand
  - …
  - Observe a pattern and write how your expression looks after the *i*-th substitution
  - Find out what the value of i (e.g., $\lg n$) should be to get the base case of the recurrence (say $T(1)$)
  - Insert the value of $T(1)$ and the expression of i into your expression

Substitution method

Solve $T(n) = 4T(n/2) + n$

1) Guess that $T(n) = O(n^3)$, i.e., that T of the form cn^3

2) Assume $T(k) \leq ck^3$ for $k \leq n/2$ and

3) Prove $T(n) \leq cn^3$ by induction

$$\begin{aligned}
T(n) &= 4T(n/2) + n \text{ (recurrence)} \\
&\leq 4\mathrm{c}(\mathrm{n}/2)^3 + n \text{ (ind. hypoth.)} \\
&= \frac{c}{2}n^3 + n \text{ (simplify)} \\
&= cn^3 - \left(\frac{c}{2}n^3 - n\right) \text{ (rearrange)} \\
&\leq cn^3 \text{ if } c \geq 2 \text{ and } n \geq 1 \text{ (satisfy)}
\end{aligned}$$

Thus $T(n) = O(n^3)$!

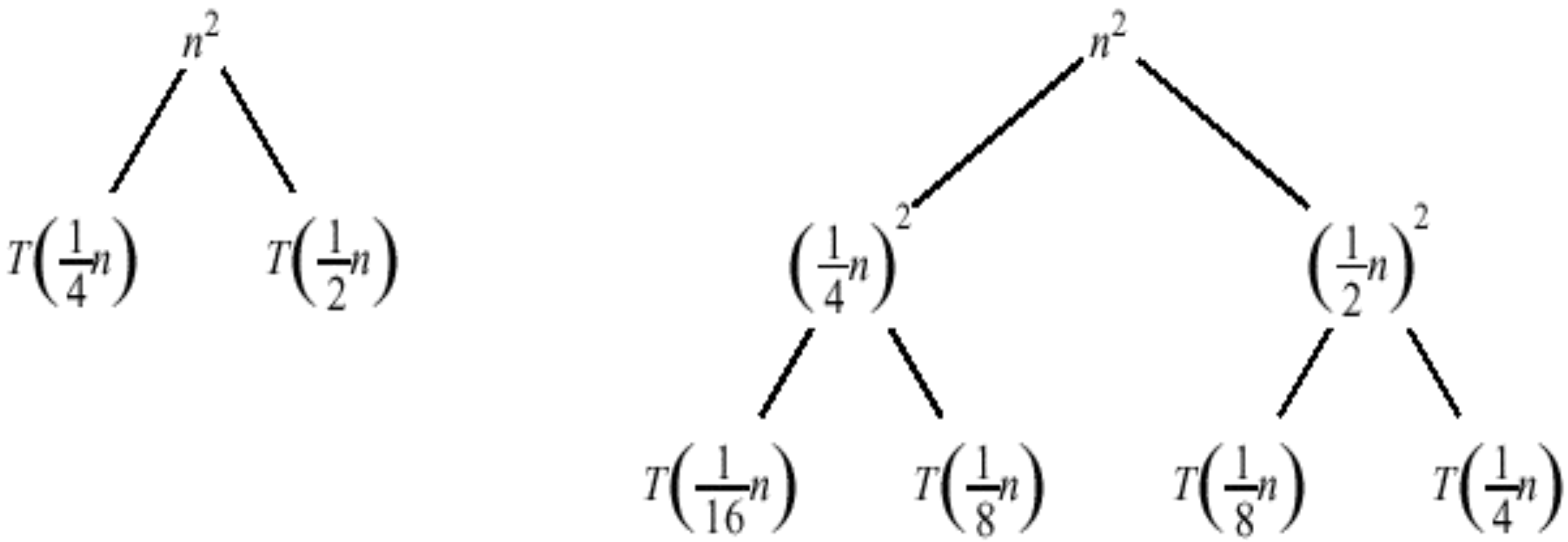
Subtlety: Must choose c big enough to handle

$T(n) = \Theta(1)$ for $n < n_0$ for some n_0

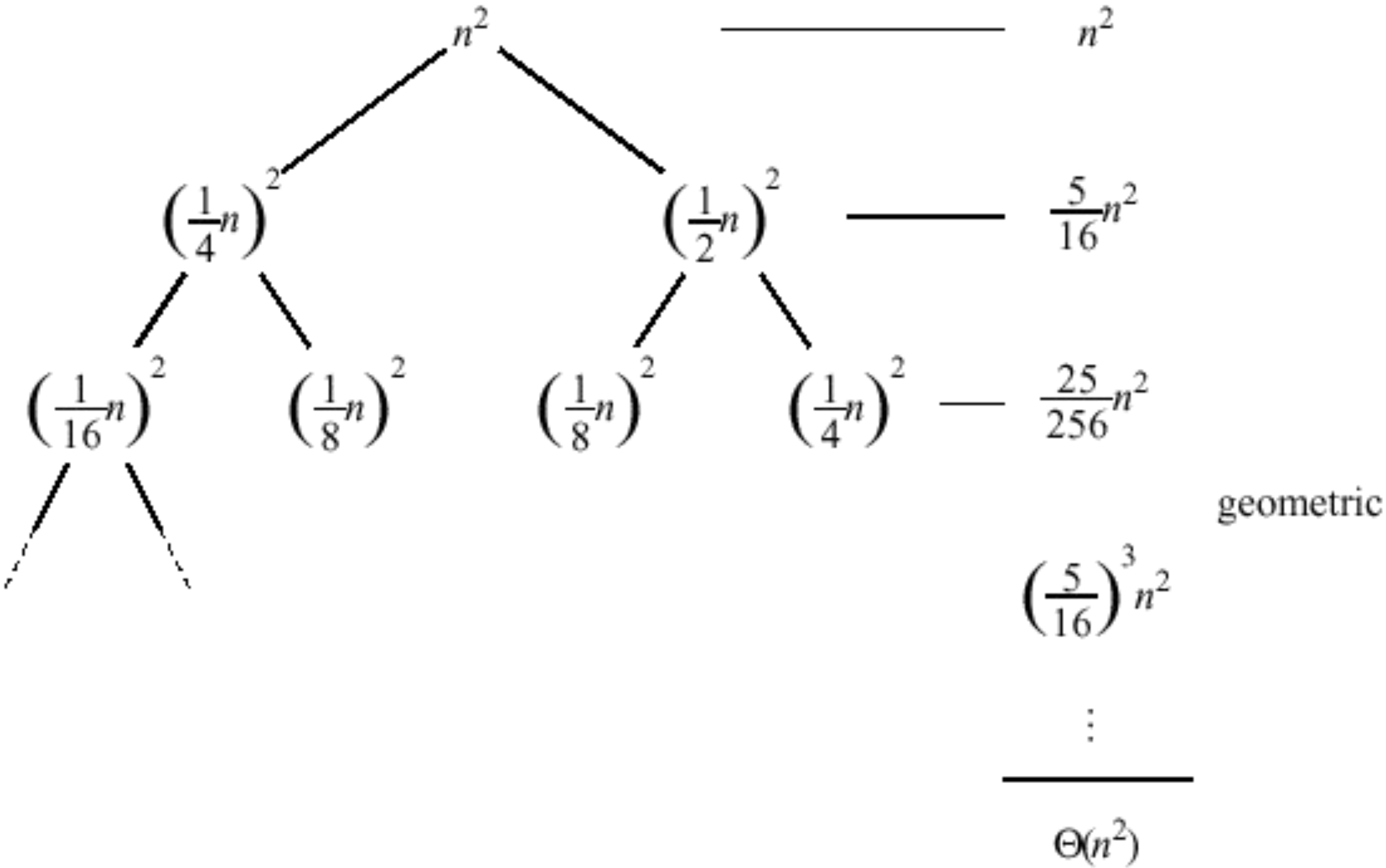
Recursion Tree

- A recursion tree is a convenient way to visualize what happens when a recurrence is iterated
- Construction of a recursion tree

$$T(n) = T(n/4) + T(n/2) + n^2$$

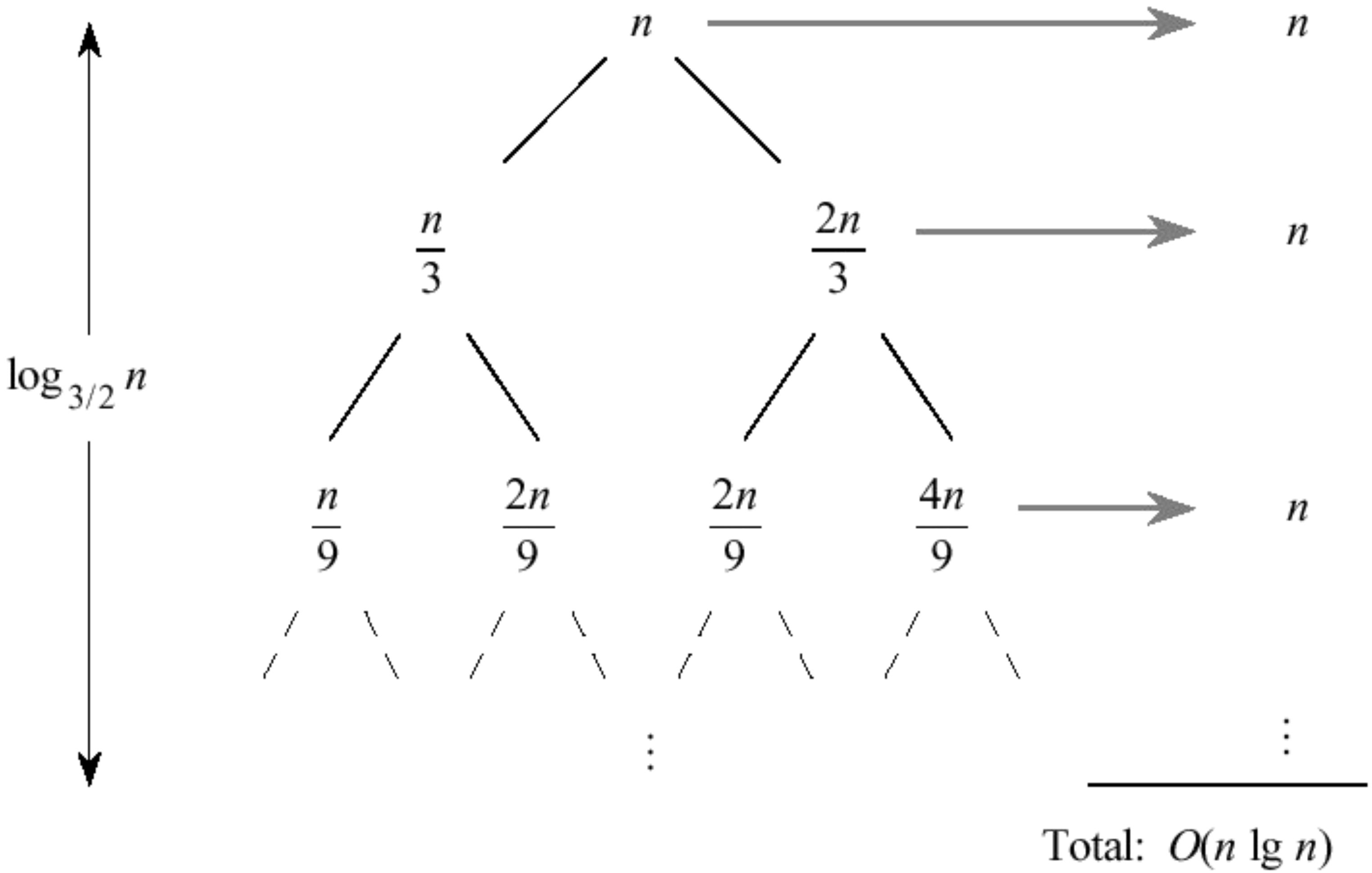


Recursion Tree



Recursion Tree

$$T(n) = T(n/3) + T(2n/3) + n$$



Master Method

- The idea is to solve a class of recurrences that have the form
$$T(n) = aT(n/b) + f(n)$$
- a $\geq$ 1 and b $>$ 1, and f is asymptotically positive!
- Abstractly speaking, T(n) is the runtime for an algorithm and we know that
  - a subproblems of size n/b are solved recursively, each in time T(n/b)
  - f(n) is the cost of dividing the problem and combining the results. In merge-sort

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

Master Theorem Summarized

- Given a recurrence of the form $T(n) = aT(n/b) + f(n)$

  1. $f(n) = O\left(n^{\log_b a - \varepsilon}\right)$

     $\Rightarrow T(n) = \Theta\left(n^{\log_b a}\right)$

  2. $f(n) = \Theta\left(n^{\log_b a}\right)$

     $\Rightarrow T(n) = \Theta\left(n^{\log_b a} \lg n\right)$

  3. $f(n) = \Omega\left(n^{\log_b a + \varepsilon}\right)$ and $af(n/b) \le cf(n)$, for some $c < 1, n > n_0$

     $\Rightarrow T(n) = \Theta\left(f(n)\right)$

- The master method cannot solve every recurrence of this form; there is a gap between cases 1 and 2, as well as cases 2 and 3

Using the Master Theorem

- Extract a, b, and f(n) from a given recurrence
- Determine $n^{\log_b a}$
- Compare f(n) and $n^{\log_b a}$ asymptotically
- Determine appropriate MT case, and apply
- Example merge sort

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

$$a = 2,\ b = 2;\ n^{\log_b a} = n^{\log_2 2} = n = \Theta(n)$$

$$\text{Also } f(n) = \Theta(n)$$

$$\Rightarrow \text{Case 2: } T(n) = \Theta\left(n^{\log_b a} \lg n\right) = \Theta\left(n \lg n\right)$$

Example 3: Binary Search

```
Binary-search(A, p, r, s):
    q←(p+r)/2
    if A[q]=s then return q
    else if A[q]>s then
         Binary-search(A, p, q-1, s)
    else Binary-search(A, q+1, r, s)
```

$T(n) = T(n/2) + 1$

$a=1$, $b=2$, $c = \log_b a = 0$, $n^c = 1$

$f(n) = 1$

Case 2: $T(n) = \Theta(\log_2 n)$

Example 3: Binary Search

```
Binary-search(A, p, r, s):
    q←(p+r)/2
    if A[q]=s then return q
    else if A[q]>s then
         Binary-search(A, p, q-1, s)
    else Binary-search(A, q+1, r, s)
```

$T(n) = T(n/2) + 1$

$a=1,\ b=2,\ c = \log_b a = 0,\ n^c = 1$

$f(n) = 1$

Case 2: $T(n) = \Theta(\log_2 n)$

Example 4: Max and min

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

$a=2$, $b=2$, $c = \log_b a = 2$, $n^c = n$

$f(n) = 1$

Case 1: $T(n) = \Theta(n)$