UNIVERSITY OF MICHIGAN-DEARBORN

Chapter 7 : Recurrence Relations

University of Michigan-Dearborn

Introduction

Recurrence relations

- **Definition:** A *recurrence relation* for the sequence A0, A1, … is an equation that relates An to certain of its predecessors A0, A1, …, An-1.
- *Initial conditions* for the sequence A0, A1, … are explicitly given values for a finite number of the terms of the sequence.

- **Example 1:** The *Fibonacci sequence* that we saw in the chapter 4, is defined by the recurrence relation

$$f_n = f_{n-1} + f_{n-2}$$

And initial condition

$$f_1 = 1$$

$$f_2 = 1$$

Recurrence relations

- **Example 2:** A person invests \$1000 at 12 percent interest compounded annually. If An represents the amount at the end of n years, *find a recurrence relation and initial conditions that define the sequence {An}.*

At the end of n-1 years, the amount is An-1. After one more year, we will have the amount An-1 plus the interest. Thus

$$A_n = A_{n-1} + (0.12) A_{n-1} = (1.12) A_{n-1}, \qquad n \geq 1$$

To apply the recurrence relation for n =1, we need to know the value A0. Since A0 is the beginning amount, we have the initial condition $A_0 = 1000$.

- The initial condition A0 and the recurrence relation An allow us to compute the value of An for any n. For example:

$$\mathbf{A_3 = (1.12)\ A_2 = (1.12)\ (1.12)\ A_1 = (1.12)\ (1.12)\ (1.12)\ A_0 = 1404.93}$$

Thus, at the end of the third year, the amount is \$1404.93.

Recurrence relations

- The computation A3 can be carried out for an arbitrary value of n to obtain

$$
\begin{aligned}
A_n &= (1.12)\, A_{n-1} \\
&\;\vdots \\
&= (1.12)^n\, (1000)
\end{aligned}
$$

- *Algorithm : Computing Compound interest*

This recursive algorithm computes the amount of money at the end of n years assuming an initial amount of $1000 and an interest rate of 12% compounded annually.

Input: n, the number of years

Output: The amount of money at the end of n years

```
Compound_interest(n){
    If (n == 0)
        return 1000
    return 1.12 * compound_interest(n-1)
}
```

Exercises

DEARBORN

Exercise 1: Assume that a person invests $2000 at 14 percent interest compounded annually. Let An represent the amount at the end of n years.

① Find a recurrence relation for the sequence A0, A1, …

② Find an initial condition for the sequence A0, A1, …

③ Find A1, A2, and A3.

④ Find an explicit formula for An.

⑤ How long will it take for a person to double the initial investment.

Exercise 2: Let Sn denote the number of n-bit strings that do not contain the pattern 000. Find the recurrence relation and initial conditions for the sequence {Sn}.

Applications to the Analysis of Algorithms

Recurrence relations

- In this section, we use recurrence relation to analyze *the time algorithms requires*.
- The technique is to develop a recurrence relation and initial conditions that defines a sequence A1, A2, …, where An is the time (*best-case, average case and worst-case*) required for an algorithm to execute an input of size n.

By solving the recurrence relation, we can determine the time needed by the algorithm.

Sorting Algorithms

- https://www.cs.usfca.edu/~galles/visualization/Algorithms.html

Recurrence relations

DEARBORN

- Our first algorithm is a version of *the selection sorting algorithm*. This algorithm selects the largest item and places it last, then recursively repeats this process.
- **Algorithm: Selection Sort**

This algorithm sorts the sequence *S1, S2, …, Sn* is nondecreasing order by first selecting the largest item and placing it last and then recursively sorting the remaining elements.

Input: *S1, S2, …, Sn and length n of the sequence*

Output: S1, S2, …, Sn arranged in nondecreasing order.

```
Selection_sort(S, n) {
   //base case
   if (n ==1)
      return
   //find largest
   max_index = 1 // assume initially that S1 is largest
   for i = 2 to n
       if (Si > Smax_index) // found larger, so update
           max_index = i
   // move largest to end
   swap(Sn, Smax_index)
   selection_sort(S, n-1)
}
```

Recurrence relations

- Our next algorithm is a version of *the Binary Search*. Binary search looks for a value in a sorted sequence and returns the index of the value if it is found or 0 if it is not found.
- **Algorithm: Binary Search**

This algorithm looks for a value in a nondecreasing sequence and returns the index of the value if it is found or 0 if it is not found.

Input: A sequence *Si, Si+1, …, Sj, i ≥ 1, sorted in nondecreasing order, a value key, i and j.*

Output: The output is an index k fro which Sk = key, or if key is not in the sequence, the output is the value 0.

```
Binary_search(S, i, j, key) {
    if (i > j) // not found
        return 0
     k = |_(i + j)/2_|
    if (key ==Sk) // found
         return k
    if (key < Sk) // search left half
         j = k - 1
     else // search right half
        i = k + 1
     return binary_search(S, i, j, key)
}
```

Recurrence relations

DEARBORN

- *Explanation of the Algorithm: Binary Search:* The sequence is divided into two nearly equal parts (line 4). If the item is found at the dividing point (line 5), the algorithm terminates. If the item is not found, because the sequence is sorted, additional comparison (line 7) will locate the half of the sequence in which the item appear if it is present. We then recursively invoke binary search (line 11) to continue the search.
- **Theorem:** The worst-case time for binary search for input of size n is

$$\Theta(\log_2 n)$$

- *Proof*

Recurrence relations

- **Algorithm: Merge Sort**

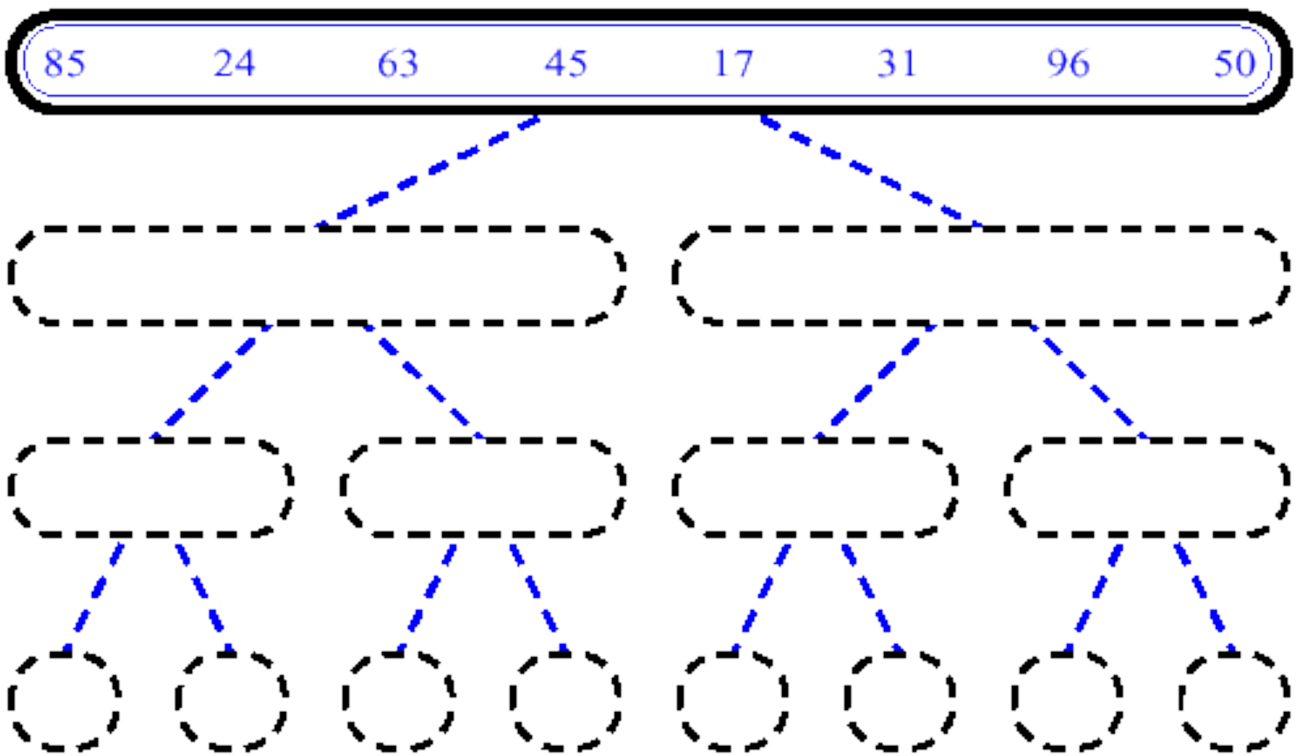
This recursive algorithm sorts a sequence into nondecreasing order by using an algorithm which merges two decreasing sequences.

Input: *S1, S2, …, Sj, i and j*

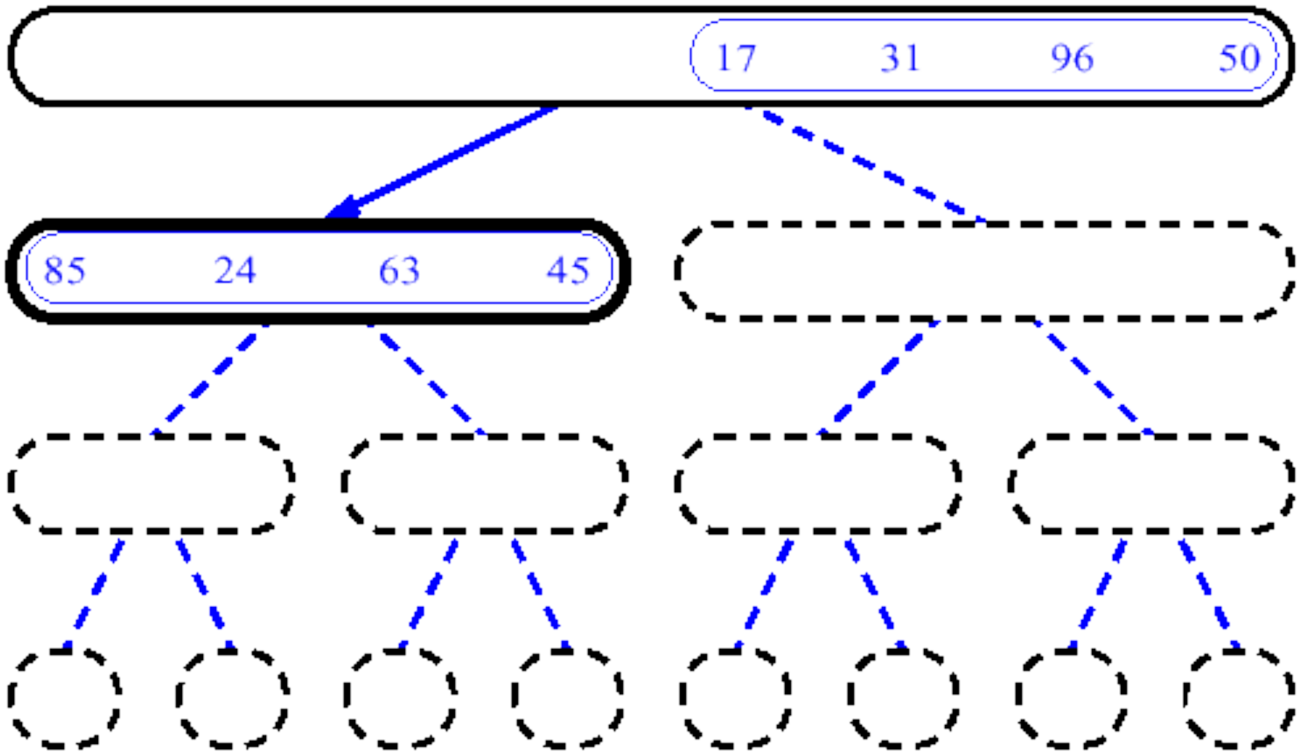
Output: S1, S2, …, Sj arranged in nondecreasing order.

```
Merge_Sort(S, n) {
   //base case: i ==j
   if (n ==1)
      return
    //divide sequence and sort
    m = |_(i+2)/2_|
    merge_Sort(s, i, m)
    merge_Sort(s, m+1, j)
     // merge
    merge(s, i, m, j, C)
    //copy C, the output of merge, into s
    for k = i to j
       Sk = Ck
}
```

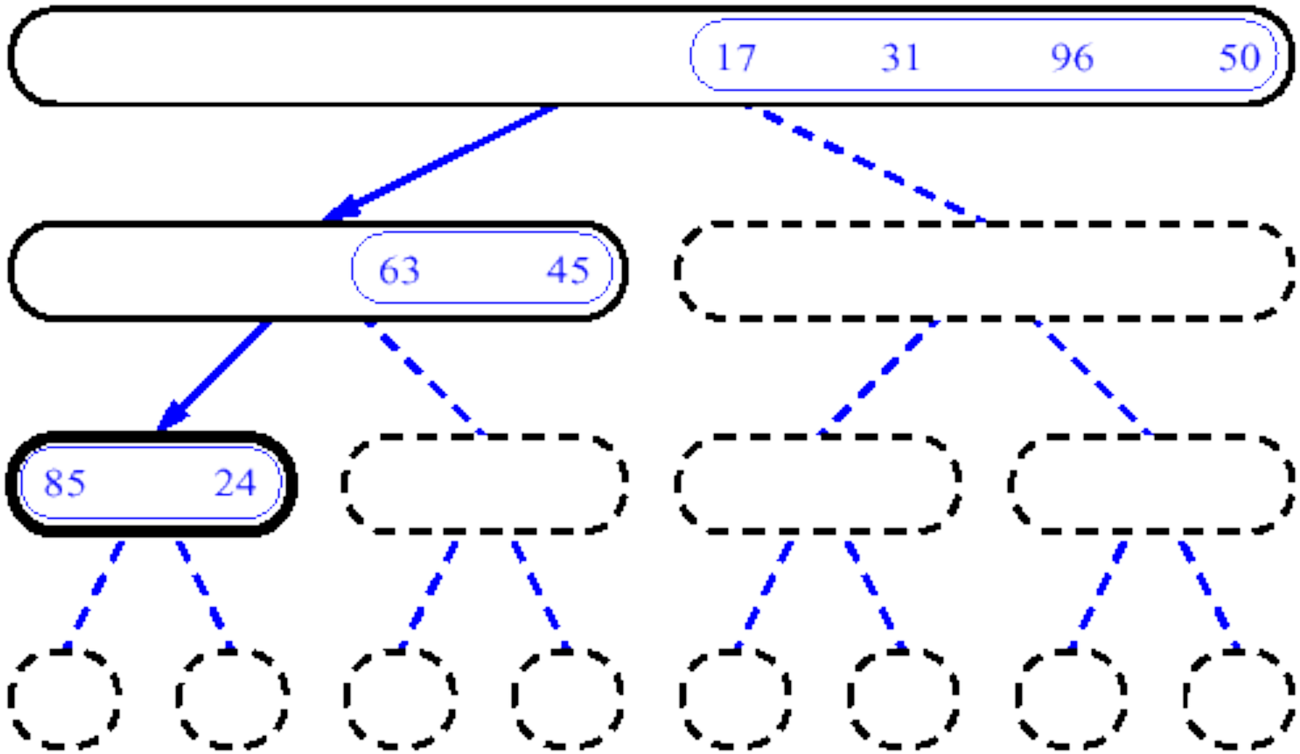
MergeSort (Example) - 1



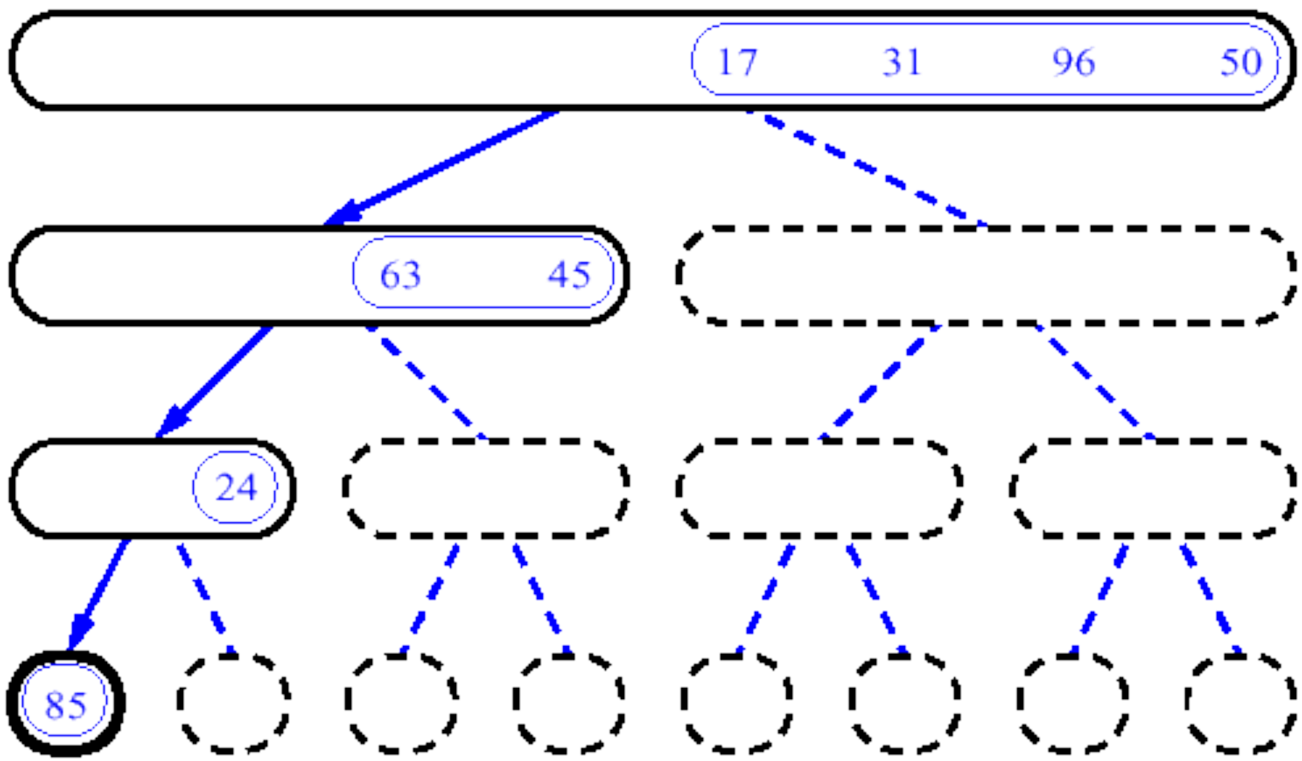
MergeSort (Example) - 2



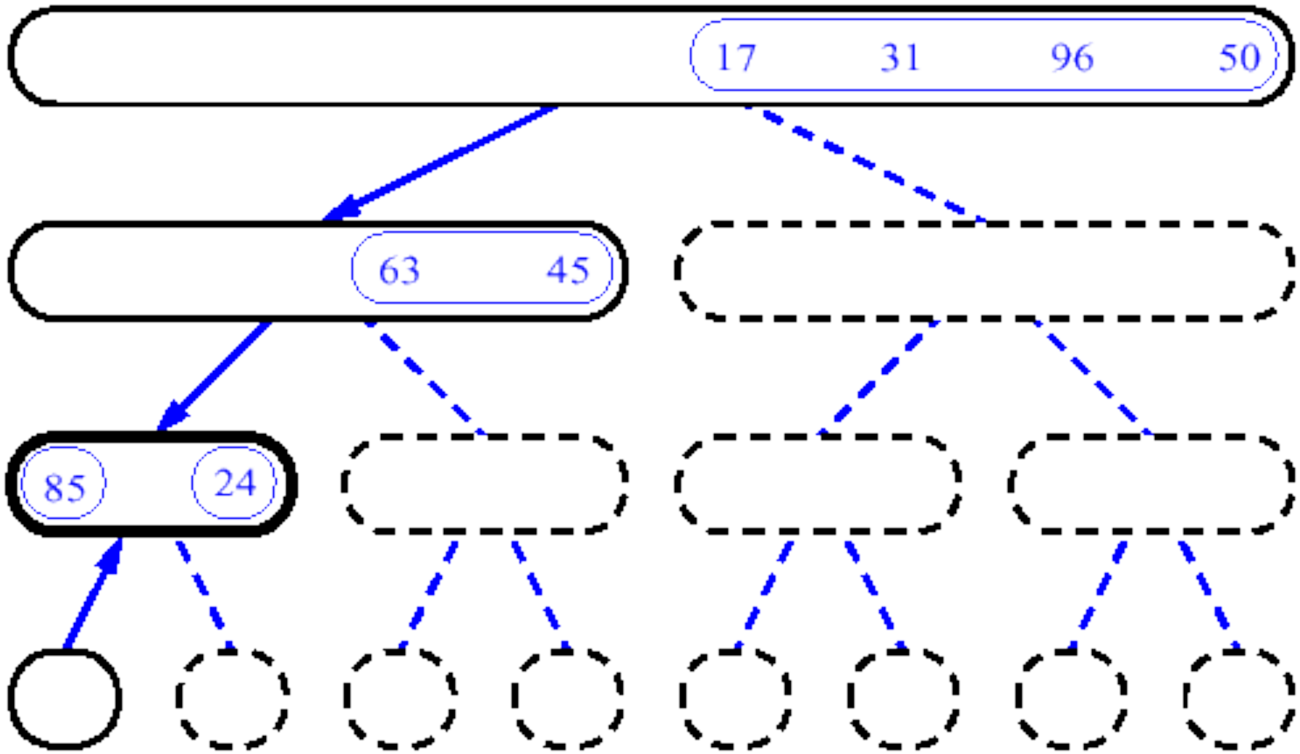
MergeSort (Example) - 3



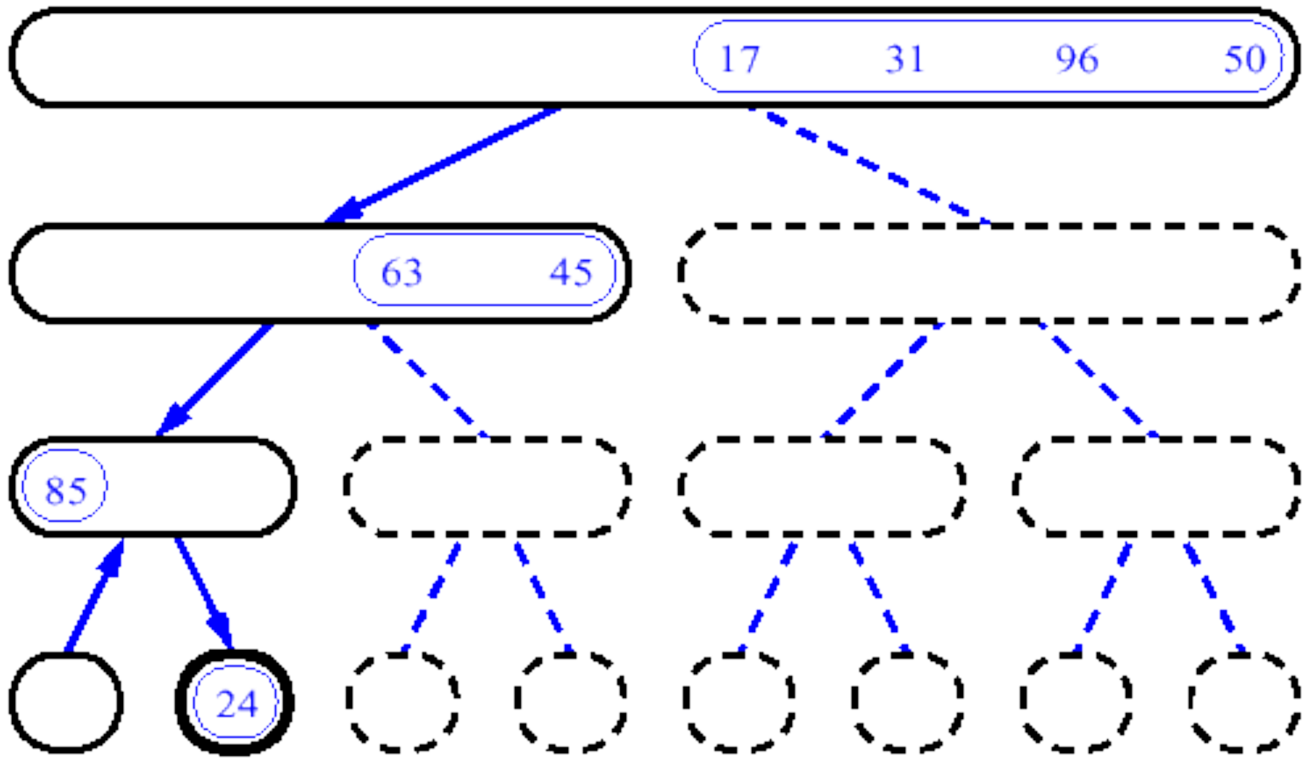
MergeSort (Example) - 4



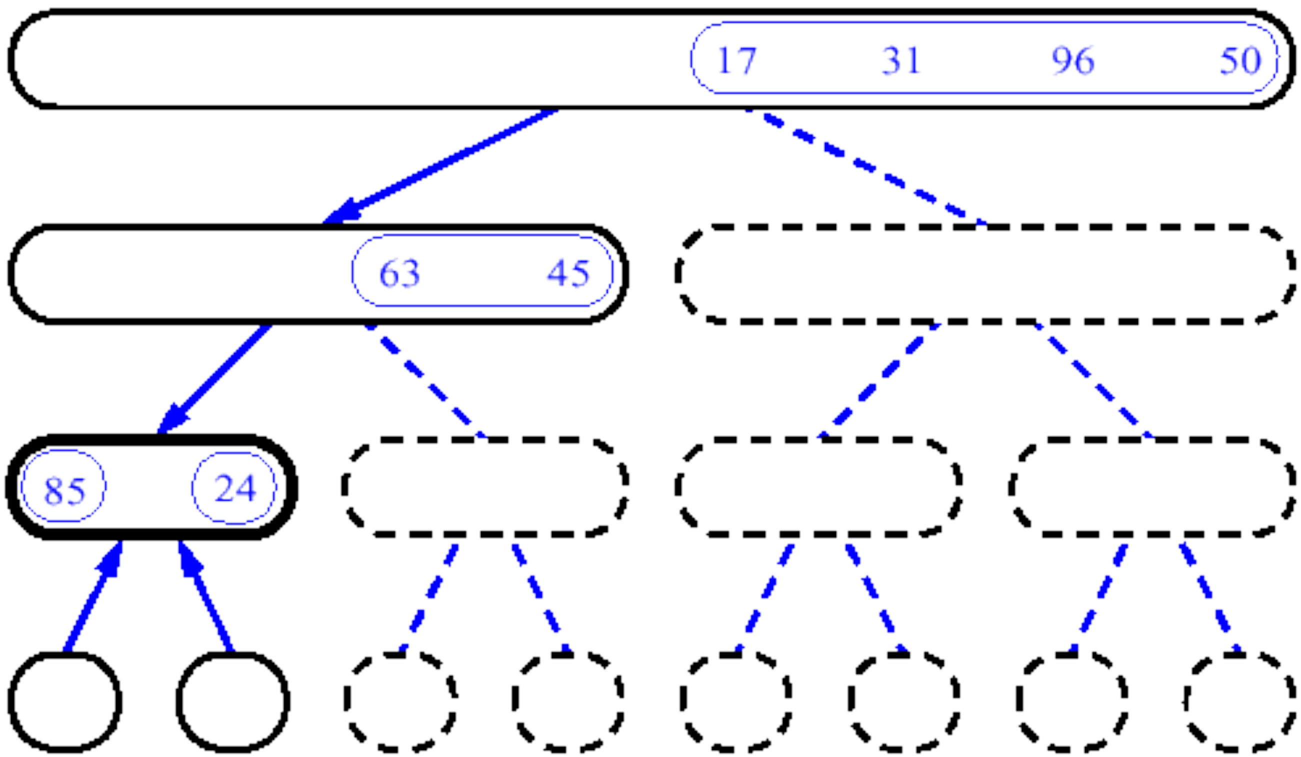
MergeSort (Example) - 5



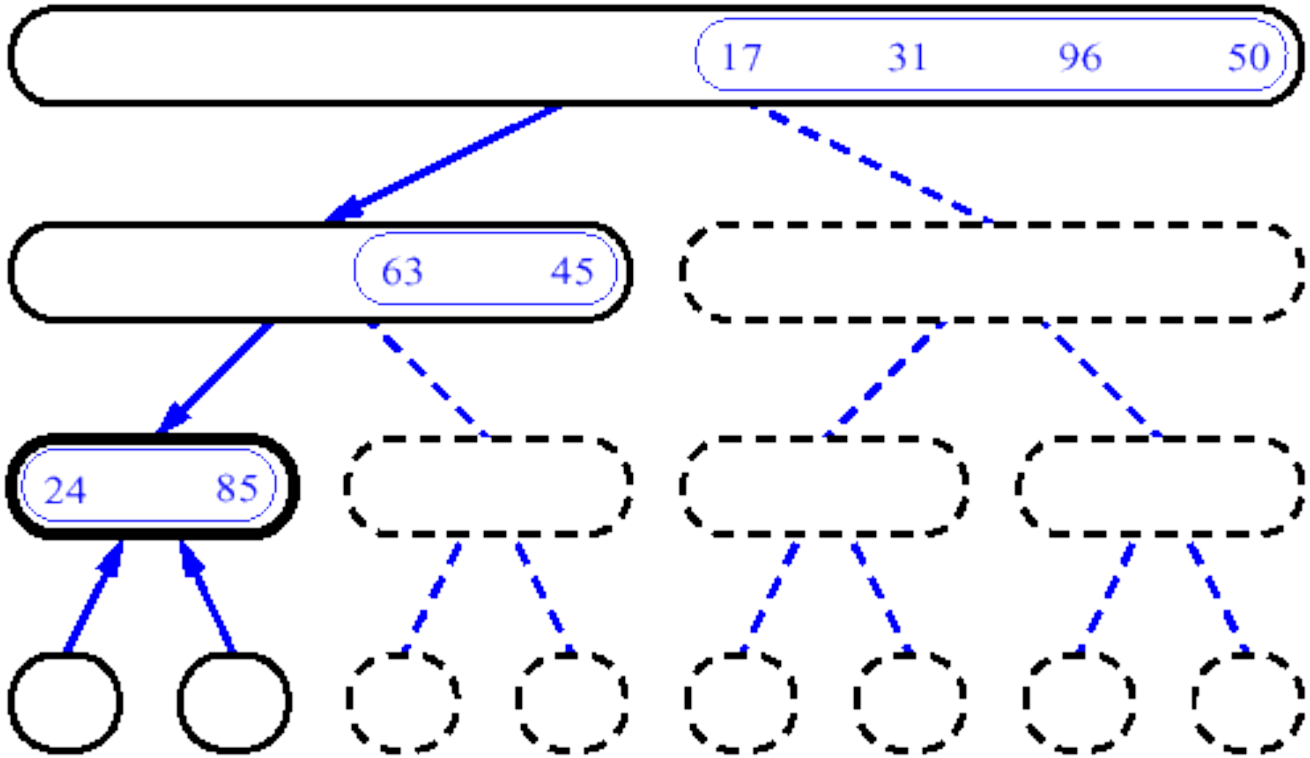
MergeSort (Example) - 6



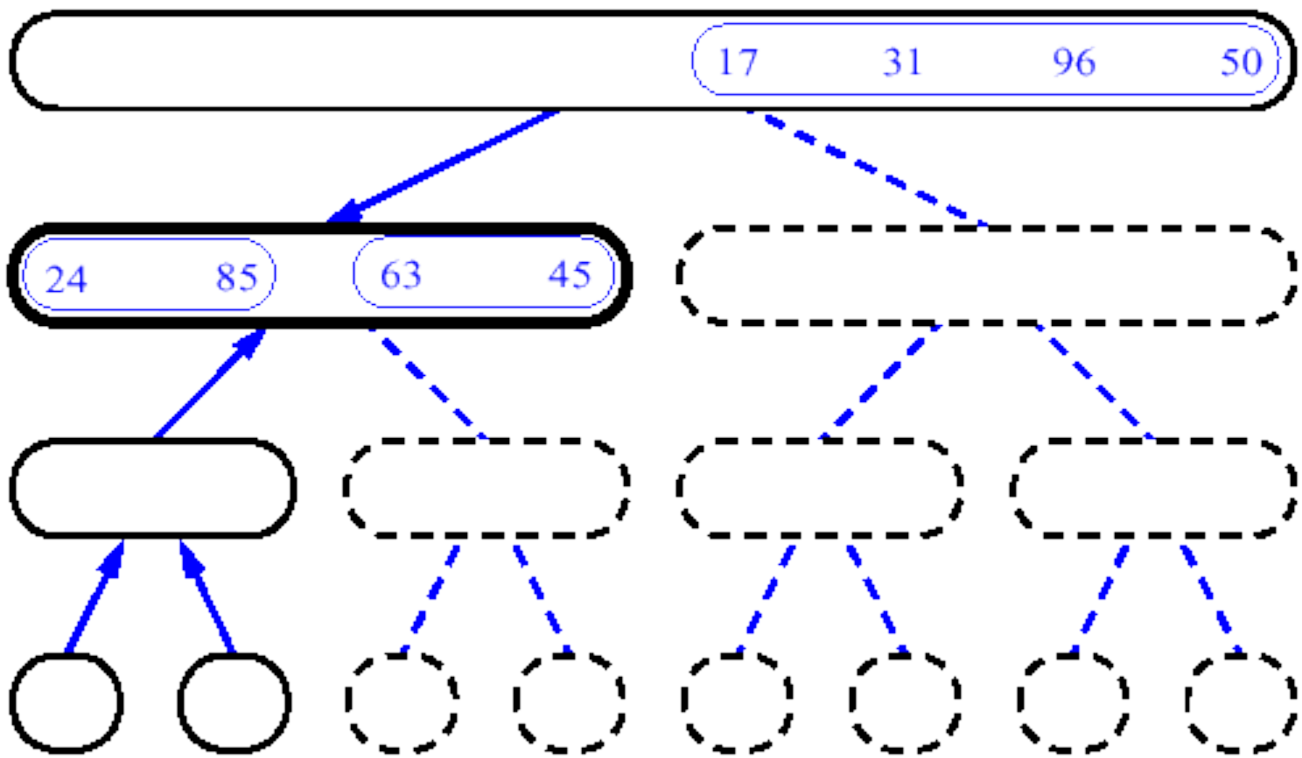
MergeSort (Example) - 7



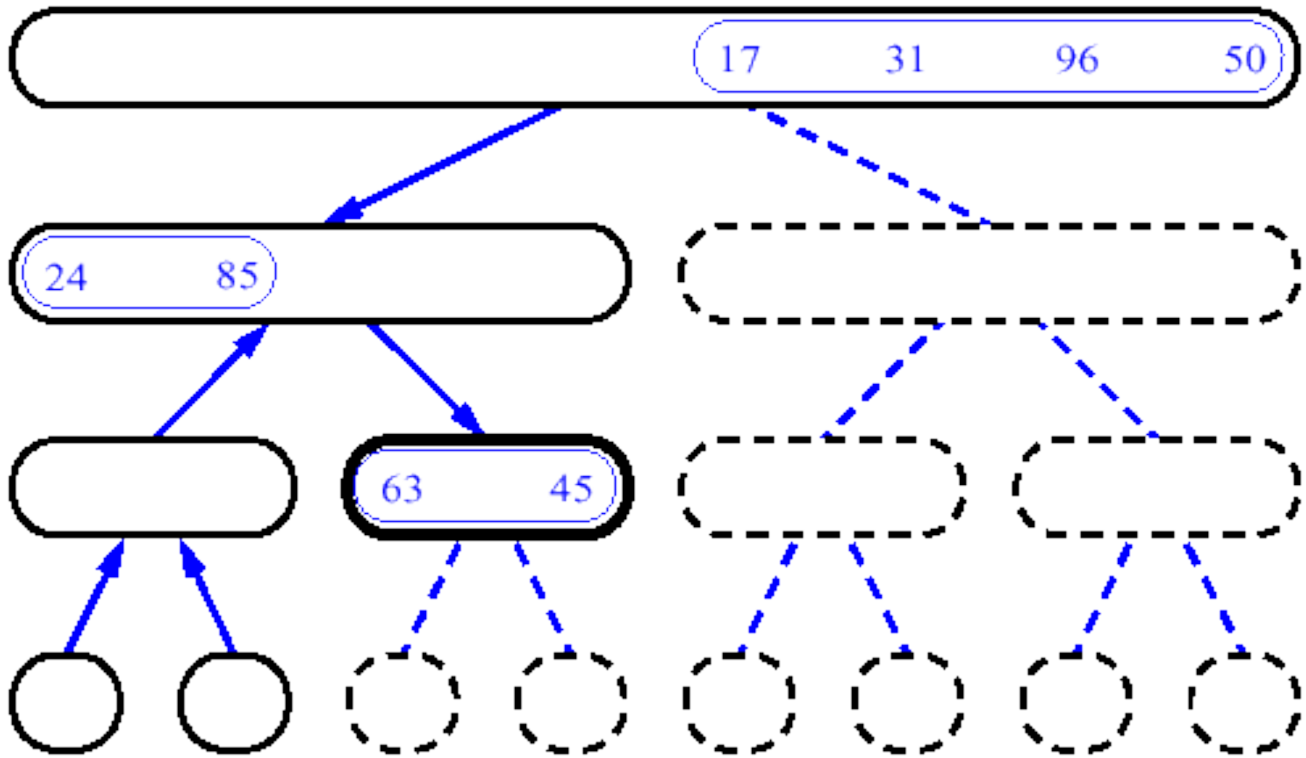
MergeSort (Example) - 8



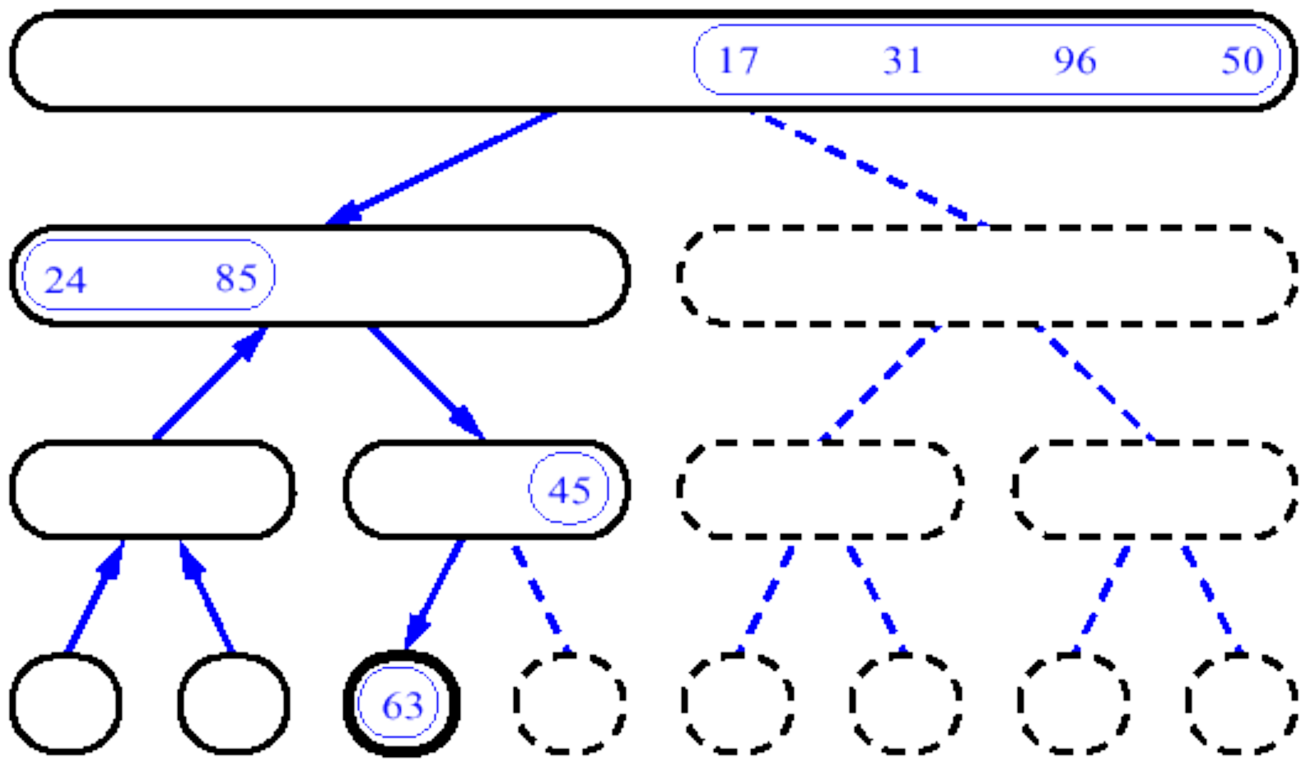
MergeSort (Example) - 9



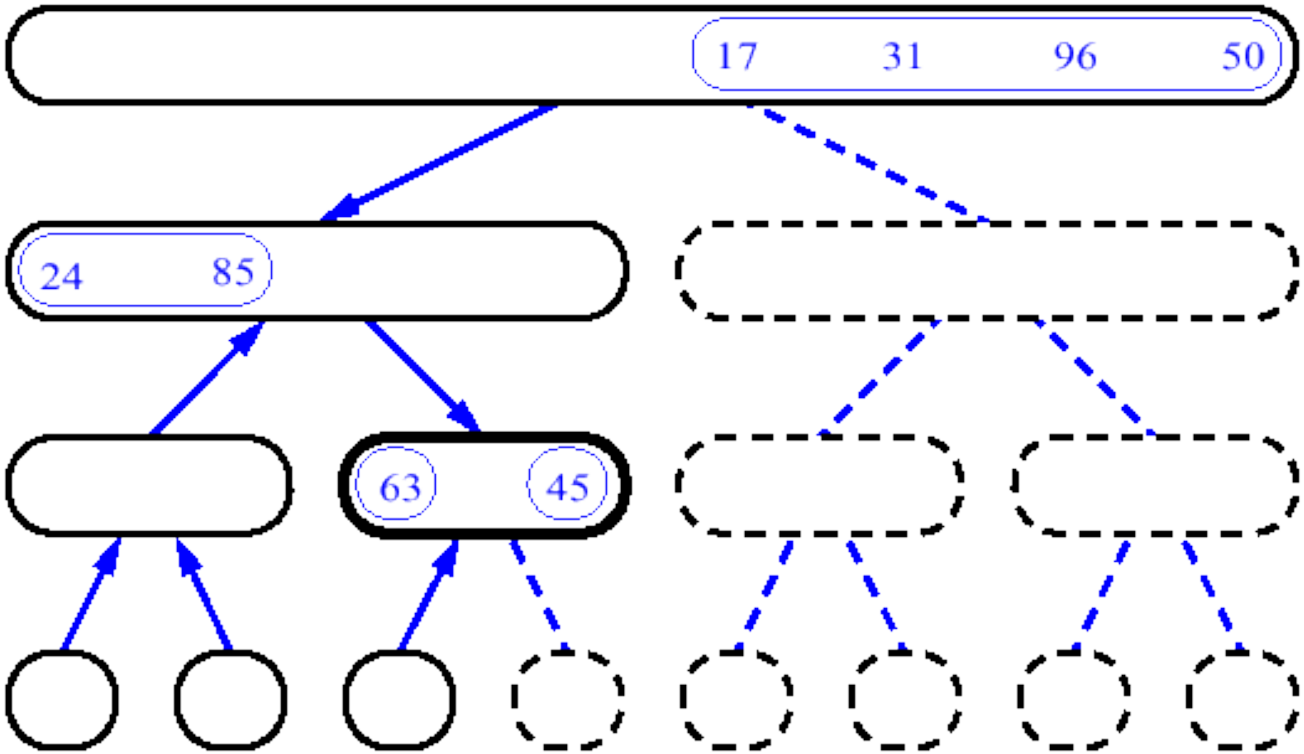
MergeSort (Example) - 10



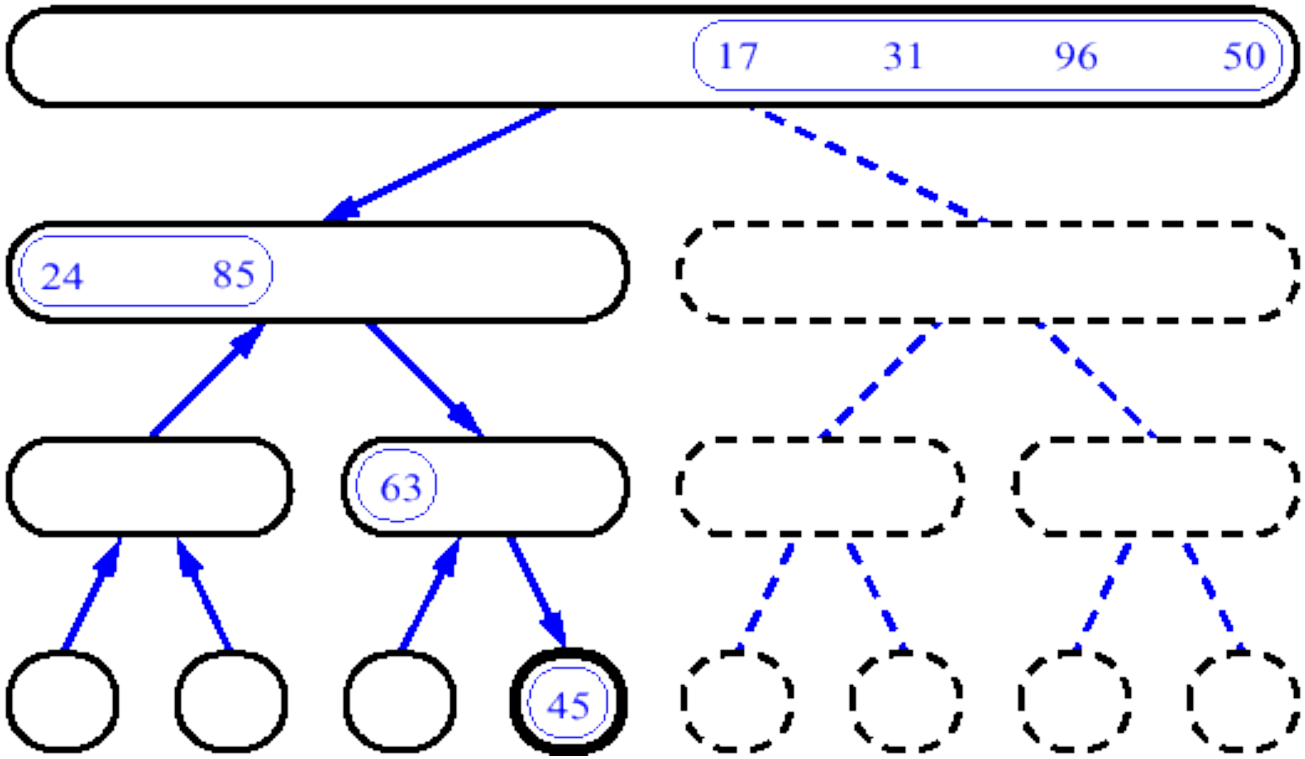
MergeSort (Example) - 11



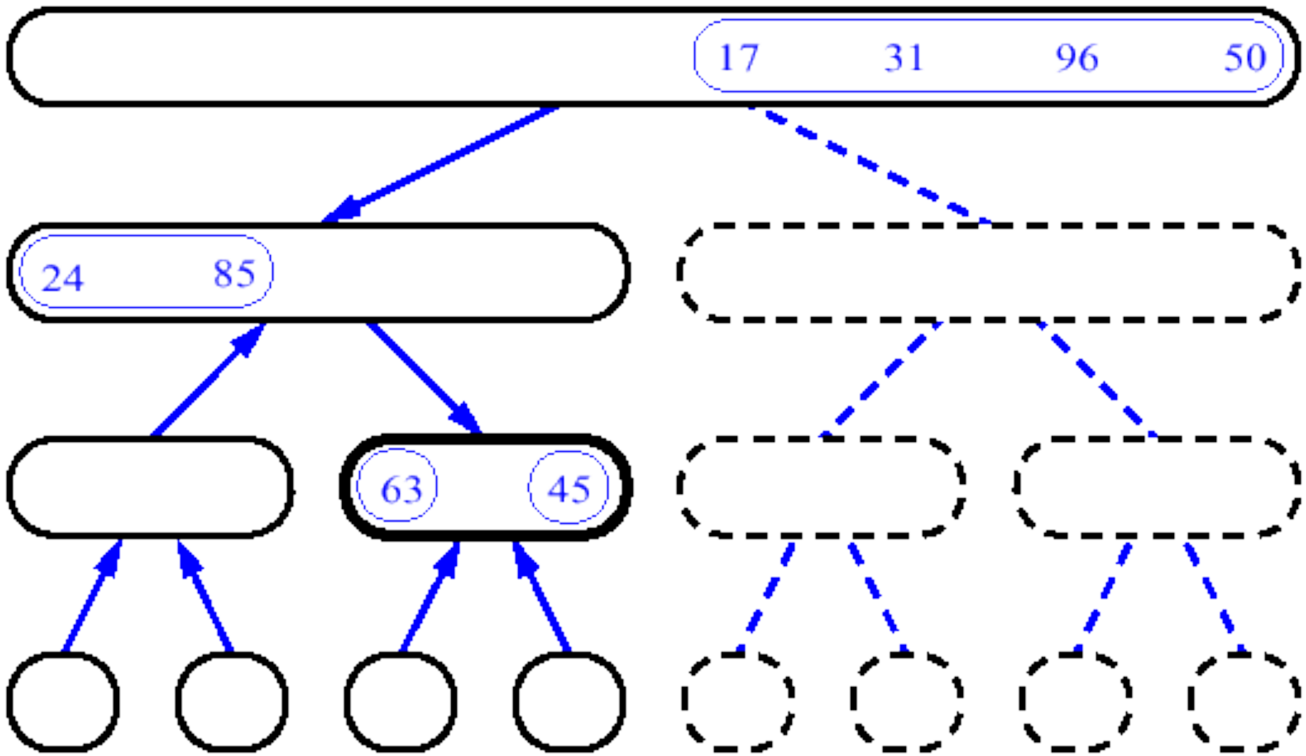
MergeSort (Example) - 12



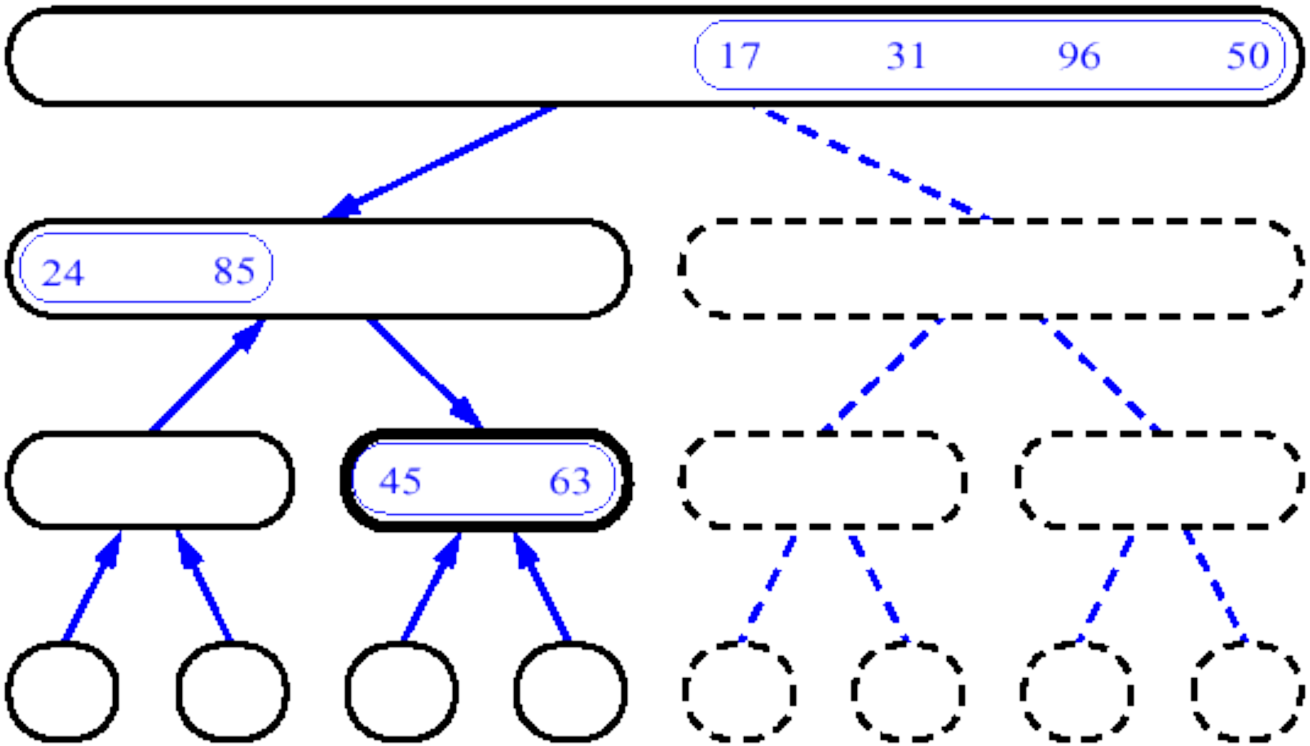
MergeSort (Example) - 13



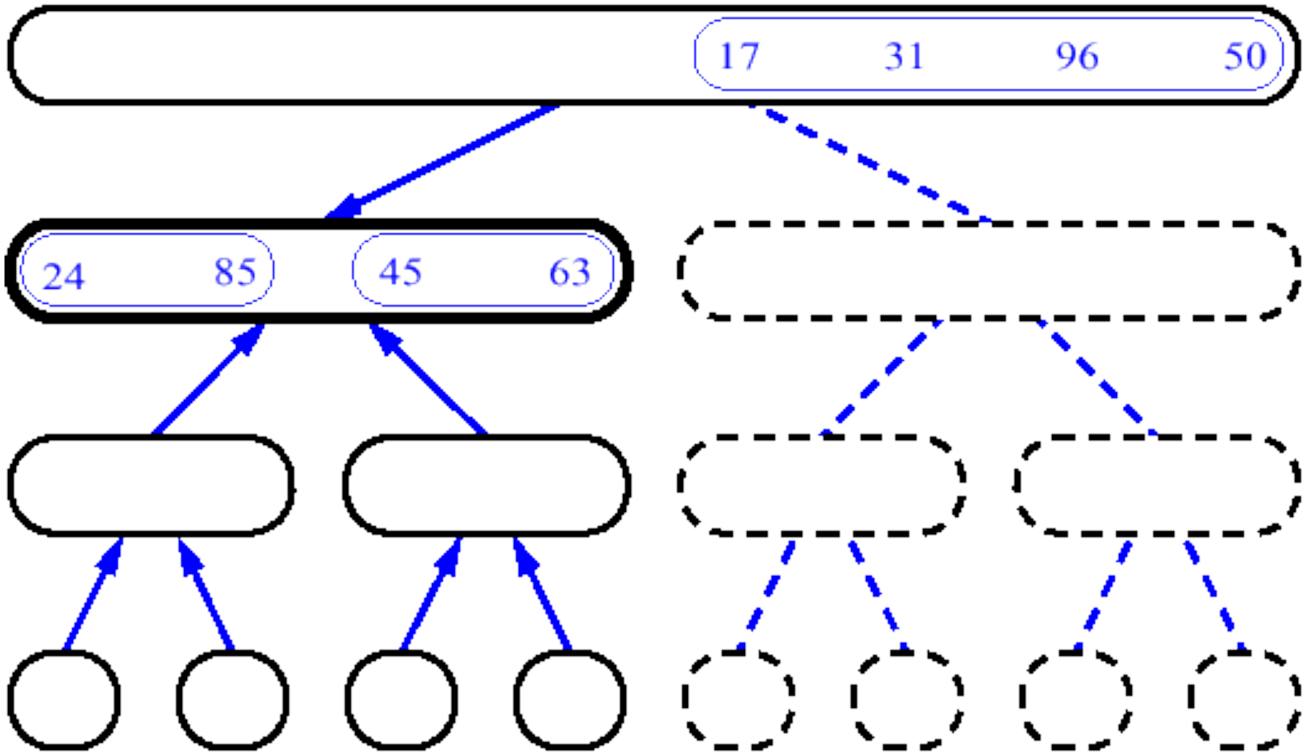
MergeSort (Example) - 14



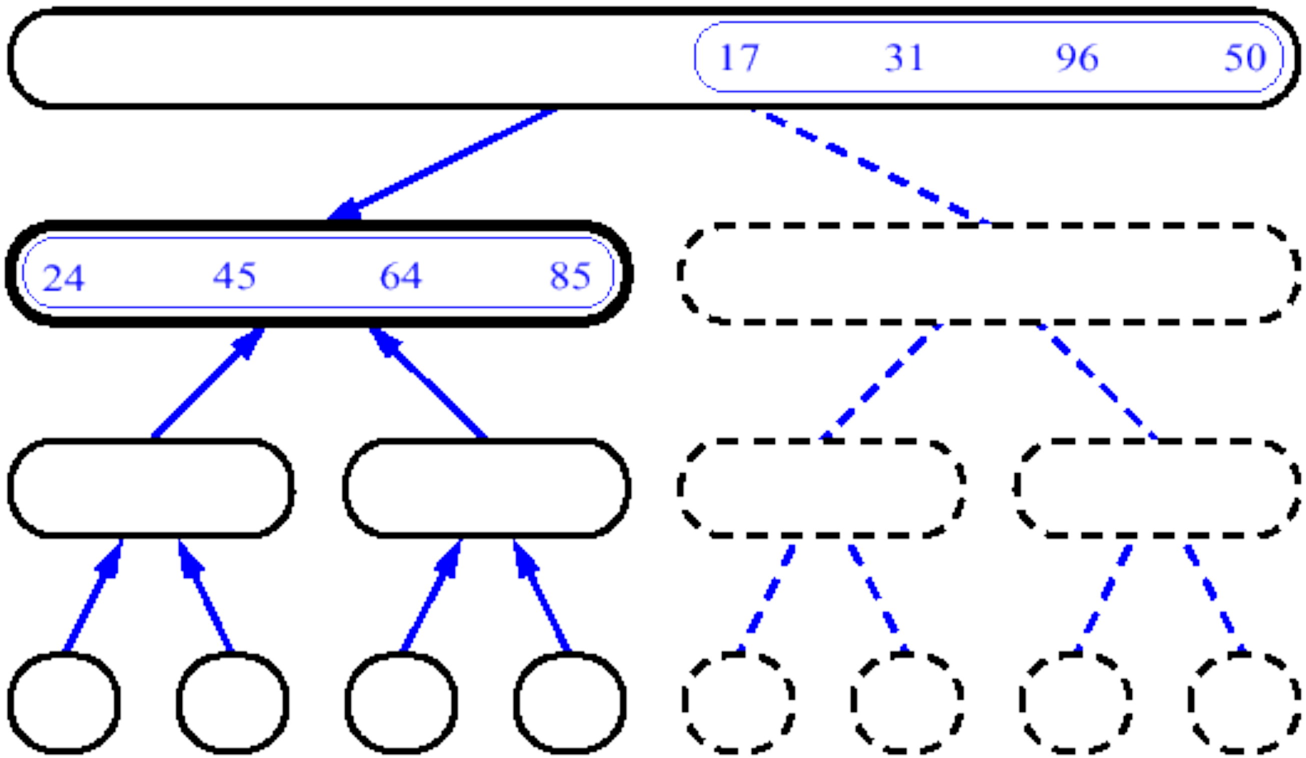
MergeSort (Example) - 15



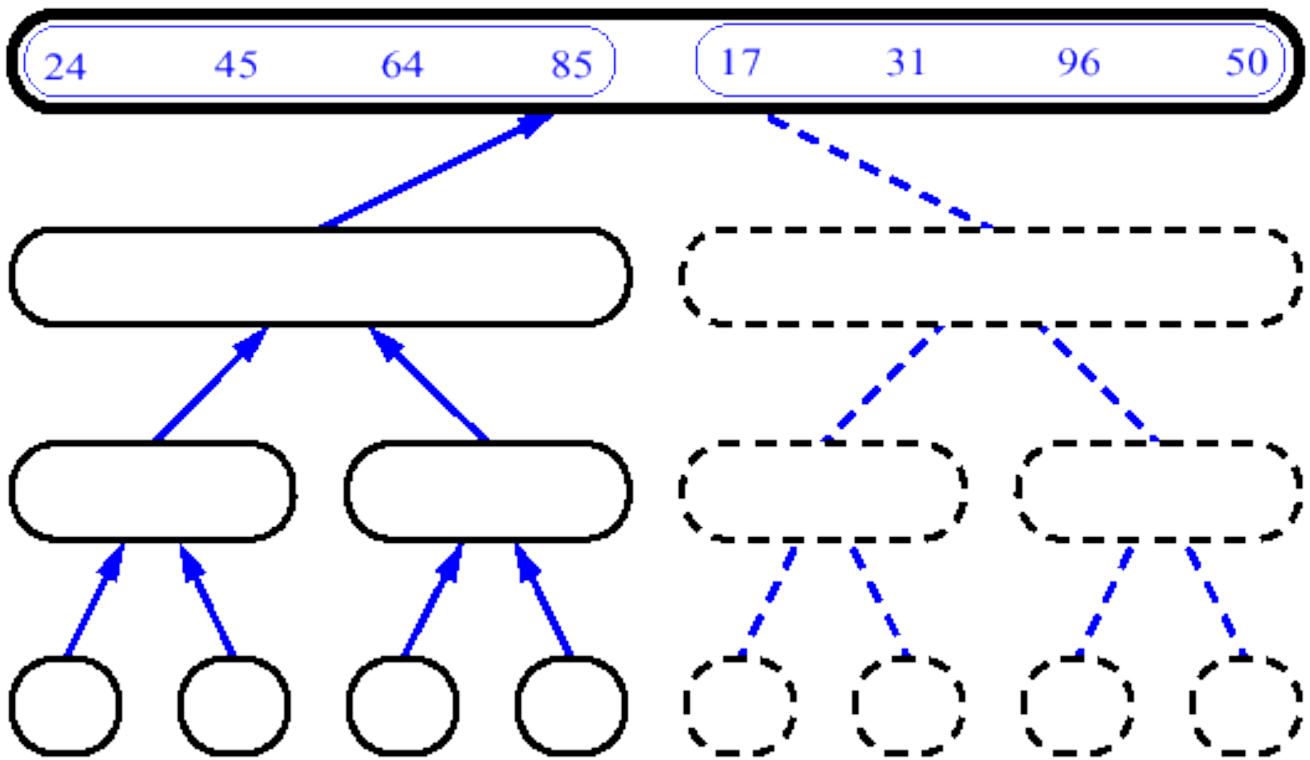
MergeSort (Example) - 16



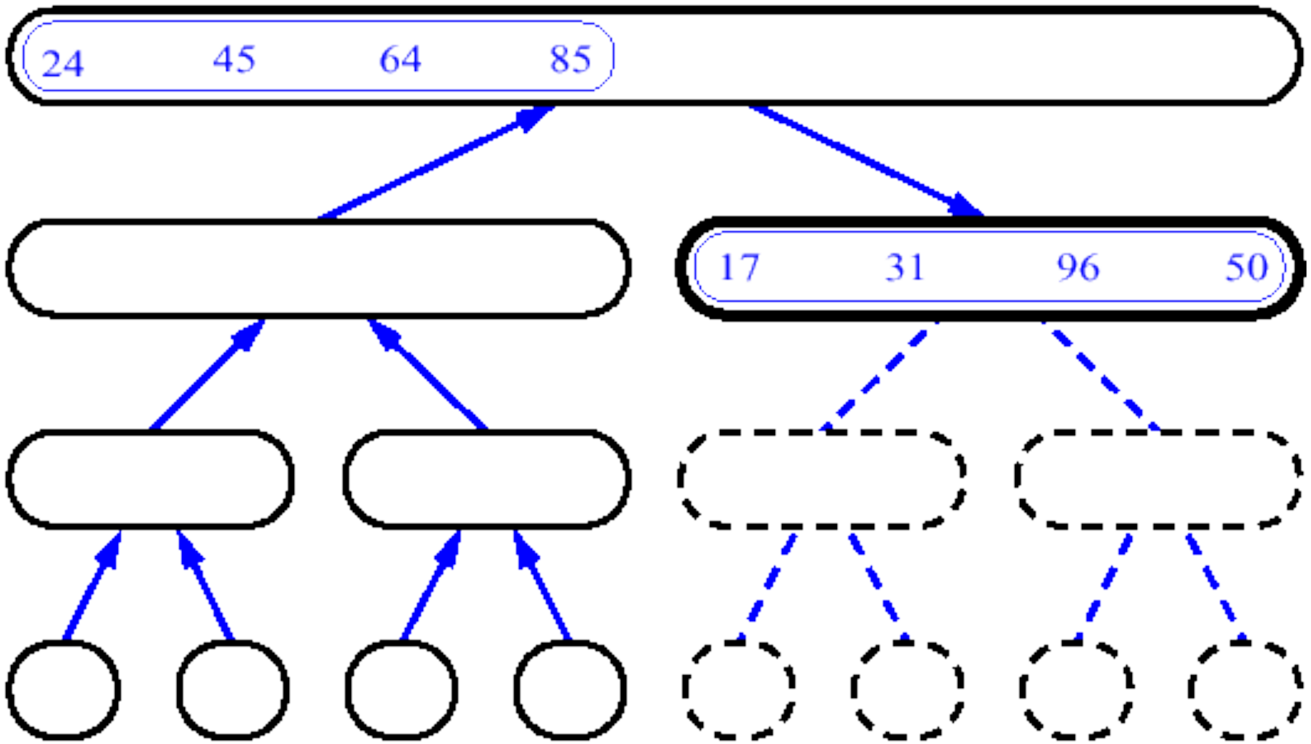
MergeSort (Example) - 17



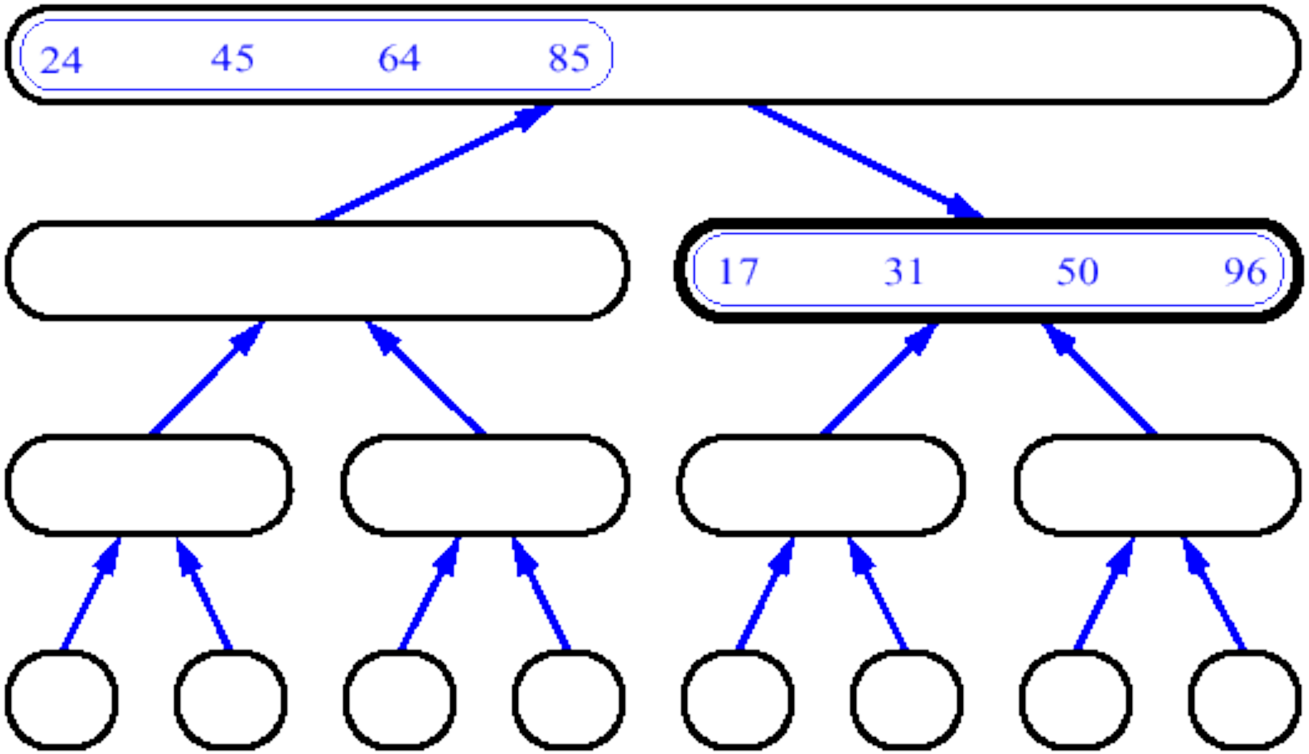
MergeSort (Example) - 18



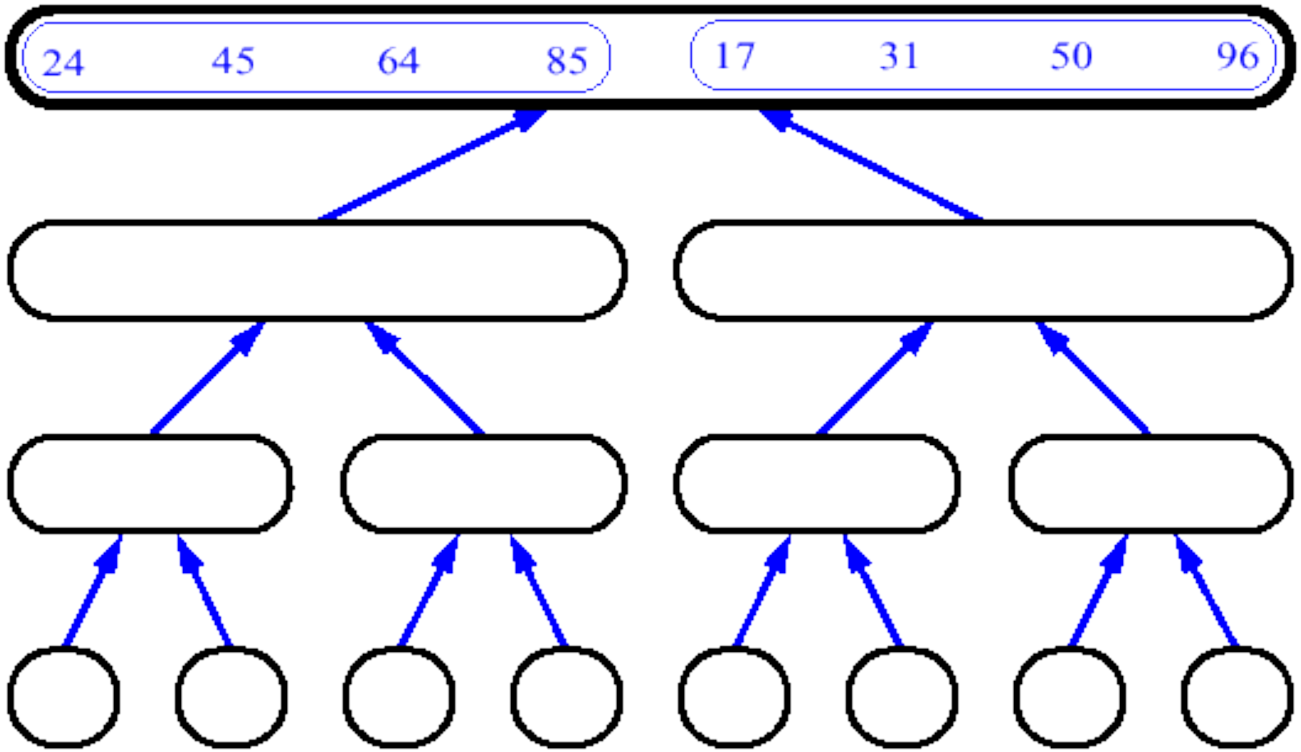
MergeSort (Example) - 19



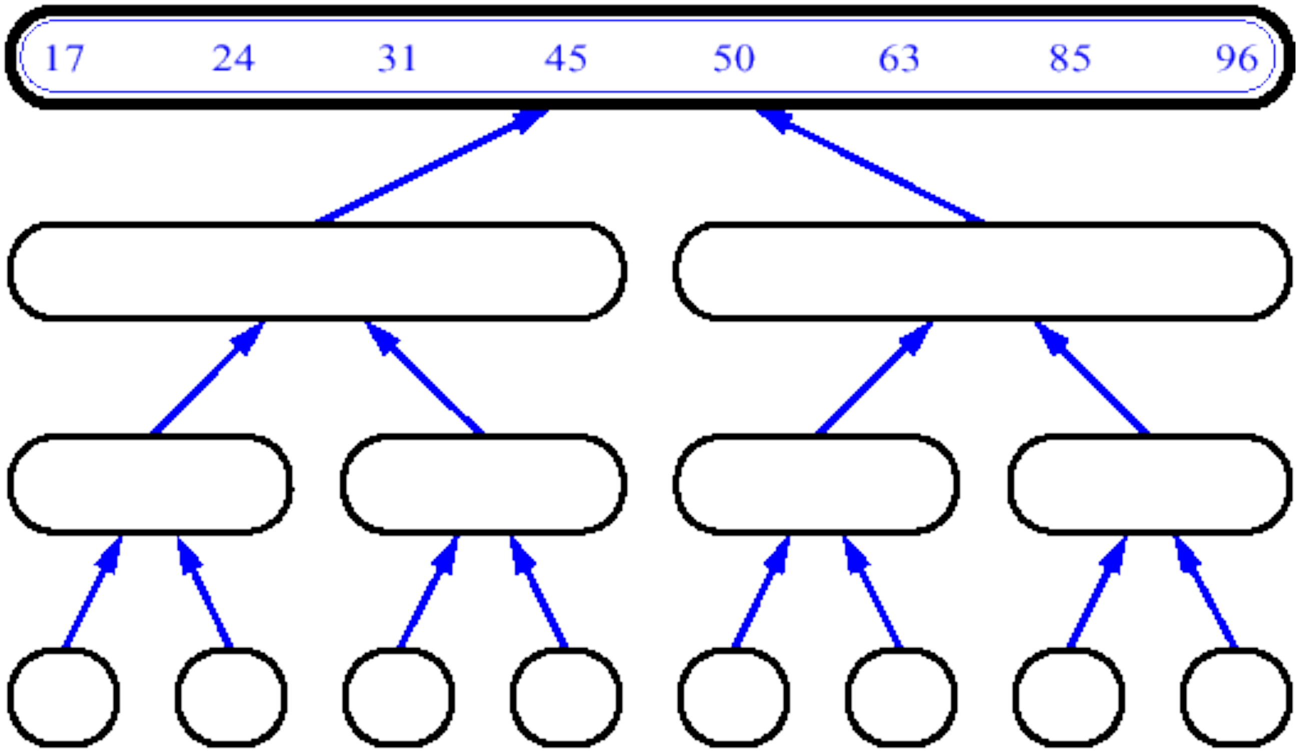
MergeSort (Example) - 20



MergeSort (Example) - 21

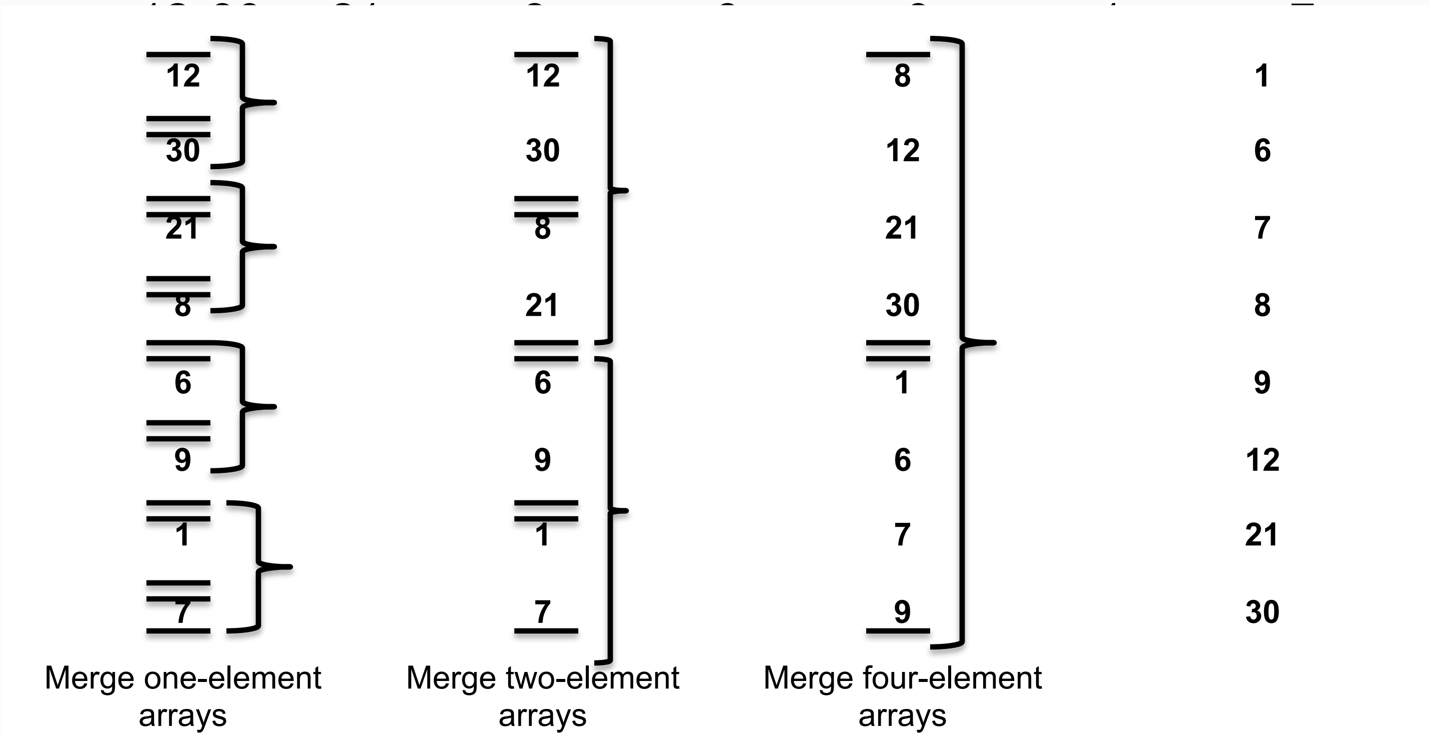


MergeSort (Example) - 22



Recurrence relations

- **Example:** This figure shows how the Algorithm of Merge Sort, sorts the sequence:



We conclude by showing that merge sort is ***$\Theta(n \log_2 n)$*** *is the worst case. The method of proof is the same as we used to show that binary search is* ***$\Theta(\log_2 n)$ in the worst case.***

Exercises

DEARBORN

Exercise 1: Refer to the sequence

$S1 = C, \quad S2 = G, \quad S3 = J, \quad S4 = M, \quad S5 = X$

① Show how the algorithm of the binary search (slide 27) executes in case key = G

② Show how the algorithm of the Binary search (slide 27) executes in case key = Z.

Exercise 2: Professor Larry proposes the following version of binary search:

```
binary_search3(s, i, j, key){
   while ( i ≤ j) {
      k = |_(i+j)/2_|
      if (key==Sk)
         return k
           if (key < Sk)
              j = k
     else
        i = k
        }
        return 0
     }
```

Is professor’s version correct (does it find key if it is present and return 0 if it is not present? If the professor’s version is correct, what is the worst-case time?

Solving Recurrence Relations

Recurrence relations

- To *solve a recurrence relation* involving the sequence A0, A1, … is to find an explicit formula for the general term An.

- In this section, we discuss two methods of solving recurrence relation:
  - ❖ an *iteration*
  - ❖ a special method that applies to *linear homogeneous recurrence relations with constant coefficients*.

Recurrence relations

- To solve a recurrence relation involving the sequence A0, A1, … by *iteration*, we use the recurrence relation to write the nth term An of certain of its predecessors An-1, …, A0.
- We then successively use the recurrence relation to replace each of An-1, … by certain of their predecessors. We continue until an explicit formula is obtained.
- **Example 1:** we can solve the recurrence relation

$$A_n = A_{n-1} + 3$$

The initial condition $A_1 = 2$

By iteration. Replacing n by n-1 in An, we obtain $A_{n-1} = A_{n-2} + 3$

If we substitute this expression for An-1 into An, we obtain

$$A_n = \underbrace{A_{n-1}}_{} + 3$$

$$= \overbrace{\mathbf{A_{n-2} + 3}}^{} + 3 = A_{n-2} + 2 \times 3$$

Replacing n by n-2, we obtain $A_{n-2} = A_{n-3} + 3$

Recurrence relations

If we substitute this expression for An-2 into An, we obtain

$$An = \underbrace{An\text{-}2}_{} + 2 \times 3$$

$$= \overbrace{\mathbf{An\text{-}3 + 3}}^{} + 2 \times 3 = An\text{-}3 + 9$$

In general, we have **An = An-k + k × 3**

If we set k = n – 1 in this expression , we have

$$An = A1 + (n - 1) \times 3$$

Since A1 = 2, we obtain the explicit formula

$$An = 2 + 3\,(n - 1)$$

For the sequence A.

Recurrence relations

- **Example 2:** We can solve the recurrence relation

$$S_n = 2 S_{n-1}$$

The initial $S_0 = 1$

By iteration : $S_n = 2 S_{n-1} = 2 (2 S_{n-2}) = \ldots = 2^n S_0 = 2^n$

- **Example 3:** Find an explicit formula for C_n, the minimum number of moves in which the n-disk tower of Hanoi puzzle can be solved.

Let consider the recurrence relation $C_n = 2 C_{n-1} + 1$

And the initial condition $C_1 = 1$.

Applying the iterative method, we obtain

$$C_n = 2 C_{n-1} + 1 = C_n = 2 (2 C_{n-2} + 1) + 1 = 2^2 C_{n-2} + 2 + 1 = 2^2 (2 C_{n-3} + 1) + 2 + 1$$

$$= 2^3 C_{n-3} + 2^2 + 2 + 1 = \ldots = 2^{n-1} C_{n-1} + 2^{n-2} + 2^{n-3} + \ldots + 2 + 1$$

$$= 2^{n-1} + 2^{n-2} + 2^{n-3} + \ldots + 2 + 1$$

Recurrence relations

- **Definition:** A *linear homogeneous recurrence relation of order k* with constant coefficients is a recurrence relation of the form

$$A_n = C_1 A_{n-1} + C_2 A_{n-2} + \dots + C_k A_{n-k}, \qquad C_k \neq 0$$

Notice that a linear homogeneous recurrence relation of order k with constant coefficient An, together with the k initial conditions

$$A_0 = C_0, A_1 = C_1, \dots \qquad A_{k-1} = C_{k-1}$$

Uniquely defines a sequence A0, A1, …

- **Example 1:** The recurrence relations

$$S_n = 2 S_{n-1}$$

And $$f_n = f_{n-1} + f_{n-2}$$

Which defines the Fibonacci sequence, are both linear homogeneous recurrence relations with constant coefficients.

The recurrence relation of Sn is of order 1 and the recurrence relation of f_n is of order 2.

Recurrence relations

- **Example 2:** The recurrence relation

$$An = 3\ An\text{-}1\ An\text{-}2$$

Is not linear homogeneous recurrence relation with constant coefficient. In a linear homogeneous recurrence relation with constant coefficients, each term is of the form C Ak. Terms such as An-1 An-2 are not permitted.

Recurrence relations such as An are said nonlinear.

- **Example 3:** The recurrence relation

$$An – An\text{-}1 = 2n$$

Is not linear homogeneous recurrence relation with constant coefficient because the expression on the right side of the equation is not zero.

Such an equation is said to be inhomogeneous.

Recurrence relations

- **Example 4:** The recurrence relation

$$A_n = 3n A_{n-1}$$

Is not linear homogeneous recurrence relation with constant coefficient because the coefficient 3n is not constant.

It is a linear homogeneous recurrence relation with non-constant coefficients

Recurrence relations

Theorem: Let

$$a_n = c_1 a_{n-1} + c_2 a_{n-2}$$

be a second order linear homogeneous recurrence relation with constant coefficients.

- If S and T are solution a_n , then

$$U = bS + dT$$

Is also a solution a_n .

- If r_1 and r_2 are solutions of

$$t^2 + c_1 t + c_2 = 0,$$

then the sequence r^n, n = 0, 1, … is a solution of a_n.

- If a is a sequence defined by a_n

$$a_0 = C_0 \qquad \text{and } a_1 = C_1$$

- And r_1 and r_2 are roots of $t^2 + c_1 t + c_2 = 0$ with $r_1 \neq r_2$, then there exist constant b and d such that]

$$a_n = br_1^n + dr_2^n \qquad n = 0, 1, \ldots$$

Recurrence relations

- **Example 1:** Find an explicit formula for *the Fibonacci sequence*

The Fibonacci sequence is defined by the linear homogeneous, second-order recurrence relation $f_n - f_{n-1} - f_{n-2} = 0$ for $n \geq 3$

And the initial conditions $f_1 = 1$, $f_2 = 1$

Let $f_n = t^n$ for some t.

$$t^n = t^{n-1} + t^{n-2}$$

$$t^n - t^{n-1} - t^{n-2} = 0$$

$$t^{n-2}(t^2 - t - 1) = 0.$$

We must use the quadratic formula to solve: $t^2 - t - 1 = 0$

$$t = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} = \frac{1 \pm \sqrt{5}}{2}$$

where a = 1, b = -1, c = -1

Recurrence relations

Let $S_n = \{ (1+\sqrt{5})/2 \}^n$ and $T_n = \{ (1-\sqrt{5})/2 \}^n$

Then $U_n = b\, S_n + d\, T_n$ is a solution.

$b\, S_0 + d\, T_0 = 0$ for $f_0 = 0$

$b\, S_1 + d\, T_1 = 1$ for $f_1 = 1$

For $f_0 = 0$ and $f_1 = 1$, We get **$b = 1/\sqrt{5}$**

$d = -1/\sqrt{5}$

So, $\mathbf{f_n = b\, S_n + d\, T_n}$

Recurrence relations

- **Example 2:** Solve $d_n = 3\, d_{n-1} - 2\, d_{n-2}$

with initial conditions $d_0 = 200$, $d_1 = 220$

Let $d_n = t^n$. Then

$$d_n - 3\, d_{n-1} + 2\, d_{n-2} = t^n - 3\, t^{n-1} + 2\, t^{n-2}$$

$$= t^{n-2}\, (t^2 - 3t + 2) = 0$$

We must use the quadratic formula to solve: $t^2 - 3t + 2 = 0$

$$t = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

$$t = 1 \text{ or } t = 2$$

$$\mathbf{d_n = b*1^n + d*2^n}$$

Recurrence relations

Using the initial conditions,

$d_n = b*1^n + d*2^n$ becomes

$d_0 = b + d = 200$

$d_1 = b*1 + d*2 = 220$

b=180 and d=20.

So, $d_n = 180*1^n + 20*2^n$

Recurrence relations

- **Theorem:**

Let $a_n = c_1 a_{n-1} + c_2 a_{n-2}$.

Be a *second-order linear homogeneous recurrence relation* with constant coefficients.

Let a be the sequence satisfying a_n and

$$a_0 = c_0 \qquad a_1 = c_1$$

If both roots of

$$t^2 - c_1 t - c_2 = 0$$

Are equal to r, then there exist constants b and d such that

$$a_n = br^n + dnr^n, \qquad n=0, 1, 2, ,,,,, v$$

Recurrence relations

DEARBORN

- **Example:** Solve the recurrence relation

$$\mathbf{d_n = 4\,(d_{n-1} - d_{n-2})}$$

Subject to the initial conditions $d_0 = d_1 = 1$

According to the previous theorem, $S_n = r^n$ is a solution of d_n where r is a solution of

$$t^2 - 4t + 4 = 0$$

$(t-2)(t-2) = 0$

$t = 2$

$$d_n = a2^n + bn2^n$$

$$d_0 = a2^0 + bn2^0 = a = 1$$

$$d_1 = a2^1 + b2^1 = 1$$

$2a + 2b = 1$

$2 + 2b = 1$

$a = 1$ **$b = -1/2$**

So, $d_n = 2^n - n2^{n-1}$

Exercises

Exercise 1: Tell whether or not each relation is linear homogeneous recurrence relaton with constant coefficients. Give the order of each linear homogeneous recurrence relations with constant coefficients.

① An = -3 An-1

② An = An-1 + n

③ An = (lg2n) An-1 – [lg(n-1)] An-2

④ An = - An-1 + 5 An-2 – 3 An-3

Exercise 2: Solve the recurrence relation for the initial condition given.

① An = -3 An-1; A0 = 2

② An = 6 An-1 – 8 An-2; A0 = 1 and A1 = 0

③ 2An = 7 An-1 – 3 An-2; A0 = A1 = 1

④ An = -8 An-1 – 16 An-2; A0 =2 and A1 = -20

Exercise 3: Show that $f_{n+1} \geq \left(\frac{1+\sqrt{5}}{2}\right)^{n-1}$ n ≥ 1

Where f denotes the Fibonacci sequence.