Chapter 3

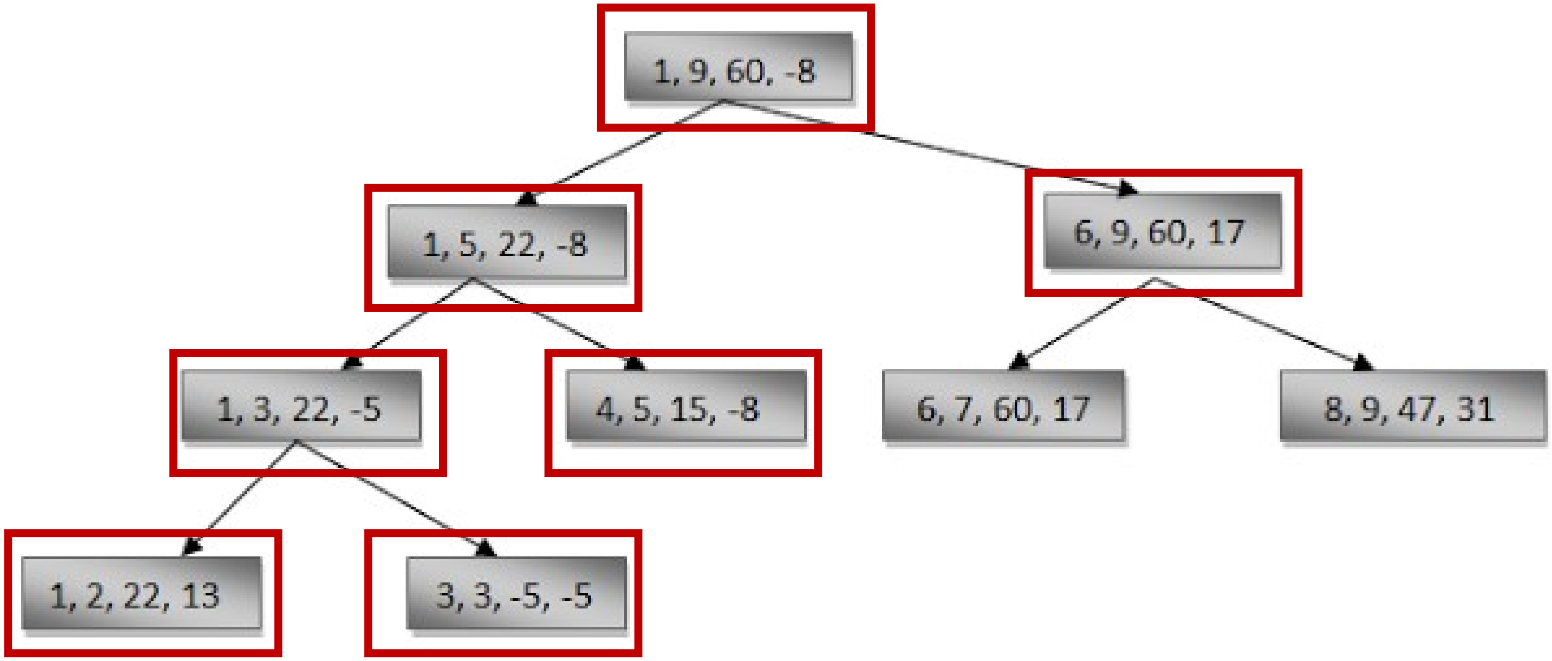
Divide-and-Conquer

MaxMin Algorithm

```
MaxMin(i, j, max, min)
{
    if (i=j) then max := min := a[i]; //Small(P)
    else if (i=j-1) then // Another case of Small(P)
        {
            if (a[i] < a[j]) then max := a[j]; min := a[i];
            else max := a[i]; min := a[j];
        }
    else
    {
        // if P is not small, divide P into sub-problems. Find where to split the set.
        mid := ( i + j )/2;     // Solve the sub-problems.
        MaxMin( i, mid, max, min );
        MaxMin( mid+1, j, max1, min1 );
        // Combine the solutions.
        if (max < max1) then max := max1;
        if (min > min1) then min := min1;
    }
}
```

A simple example

- Finding the maximum and minimum of a set **S** of **n** numbers



Time complexity

$T(n)=2T(n/2)+2T(n/2)+2 \quad \text{if } n>=2$

$\quad =1 \quad \text{if } n=2$

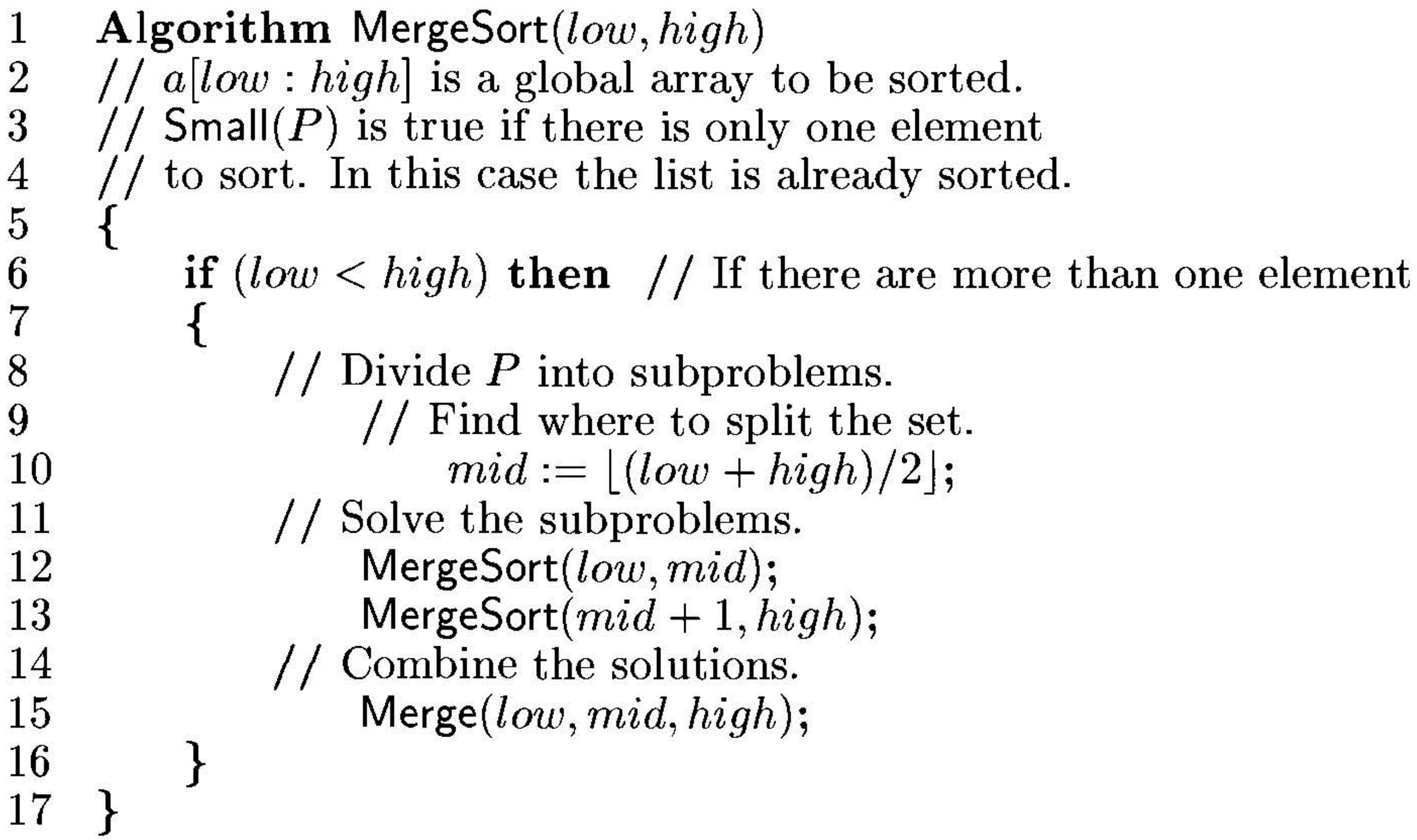
$\quad =0 \quad \text{if } n=1$

- Calculation of T(n):

Assume $n = 2^k$,

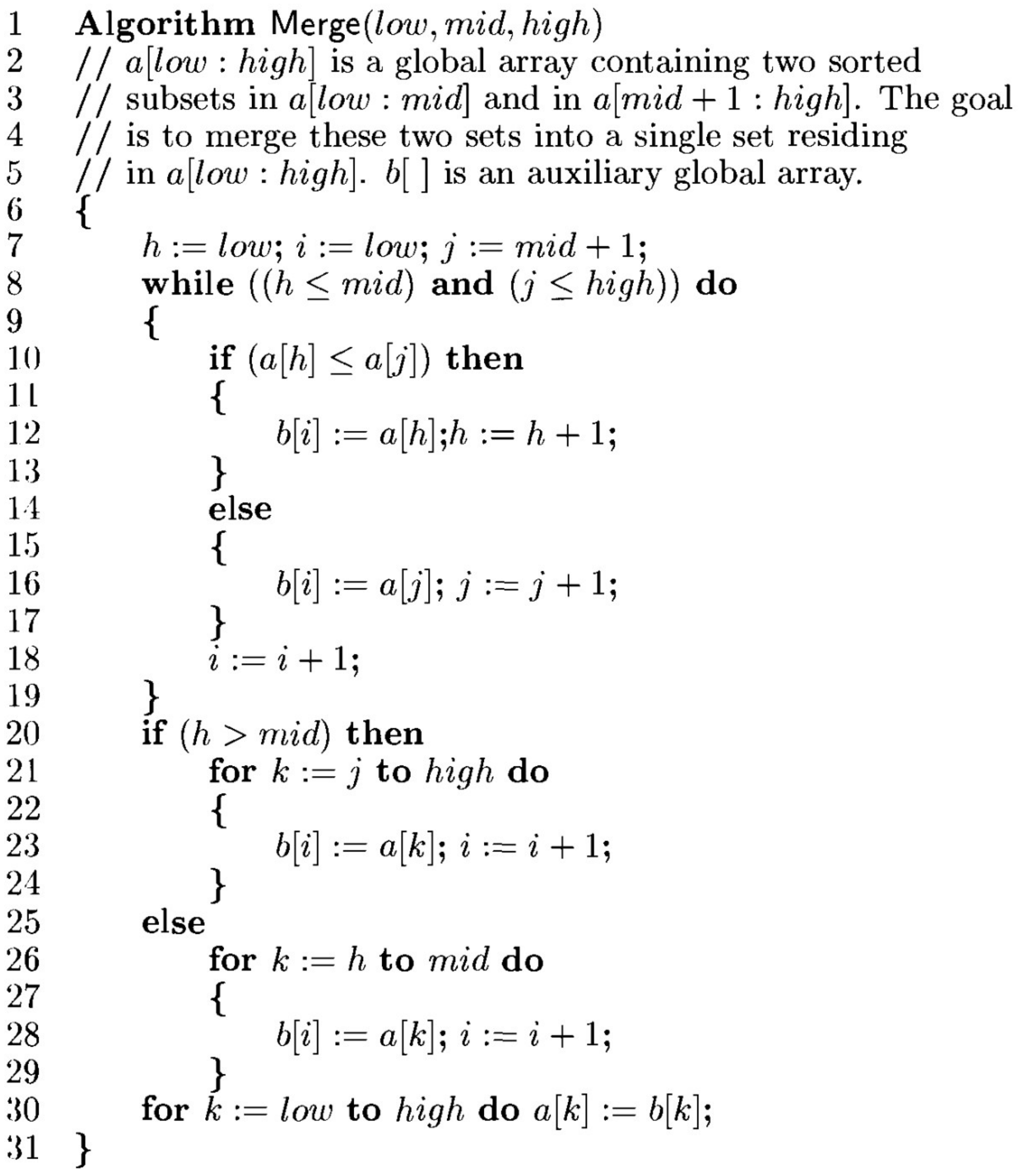
$$
\begin{aligned}
T(n) &= 2T(n/2)+2 \\
&= 2(2T(n/4)+2)+2 \\
&= 4T(n/4)+4+2 \\
&\quad \vdots \\
&=2^{k-1}T(2)+2^k-2 \\
&=3n/2-2
\end{aligned}
$$

Merge Sort



```
Algorithm MergeSort($low, high$)
// $a[low : high]$ is a global array to be sorted.
// Small($P$) is true if there is only one element
// to sort. In this case the list is already sorted.
{
    if ($low < high$) then  // If there are more than one element
    {
        // Divide $P$ into subproblems.
            // Find where to split the set.
                $mid := \lfloor (low + high)/2 \rfloor$;
        // Solve the subproblems.
            MergeSort($low, mid$);
            MergeSort($mid + 1, high$);
        // Combine the solutions.
            Merge($low, mid, high$);
    }
}
```

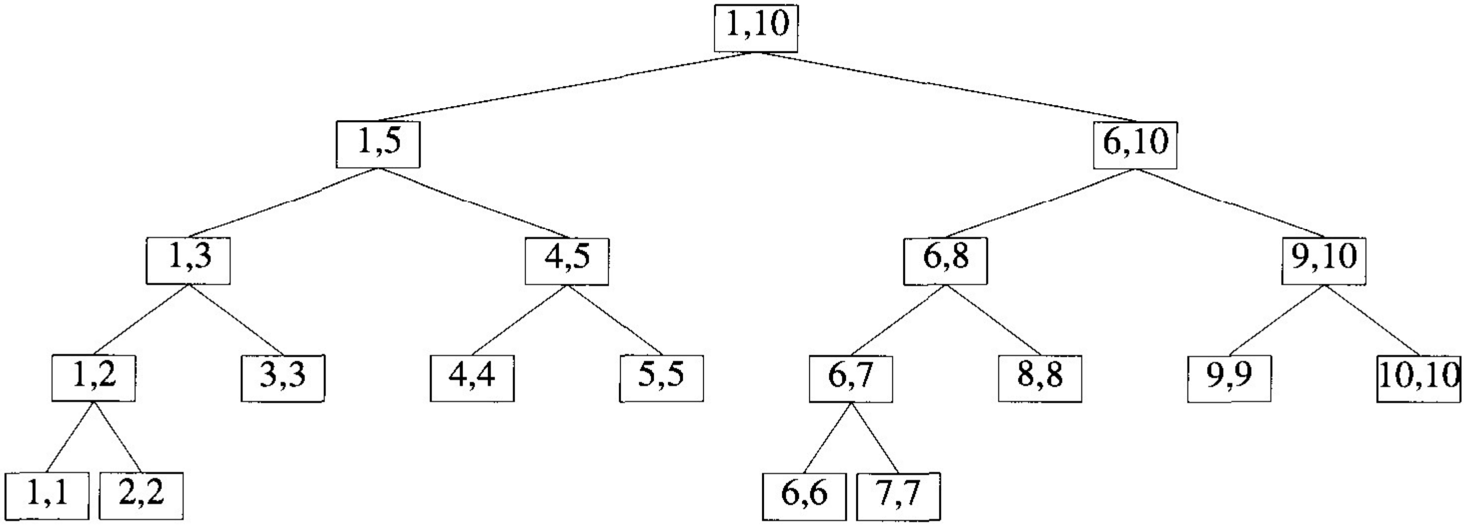
Algorithm: Merge Element



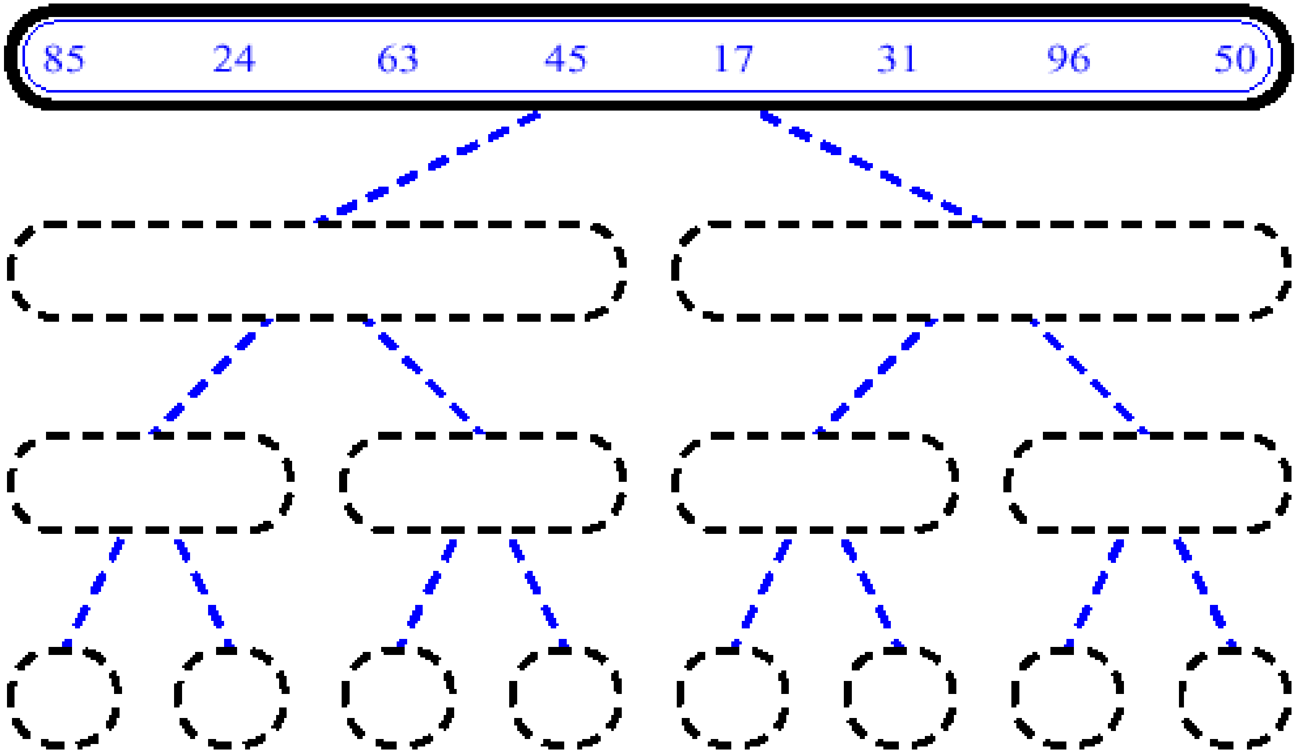
```
Algorithm Merge($low, mid, high$)
// $a[low : high]$ is a global array containing two sorted
// subsets in $a[low : mid]$ and in $a[mid + 1 : high]$. The goal
// is to merge these two sets into a single set residing
// in $a[low : high]$. $b[\ ]$ is an auxiliary global array.
{
    $h := low$; $i := low$; $j := mid + 1$;
    while (($h \leq mid$) and ($j \leq high$)) do
    {
        if ($a[h] \leq a[j]$) then
        {
            $b[i] := a[h]$;$h := h + 1$;
        }
        else
        {
            $b[i] := a[j]$; $j := j + 1$;
        }
        $i := i + 1$;
    }
    if ($h > mid$) then
        for $k := j$ to $high$ do
        {
            $b[i] := a[k]$; $i := i + 1$;
        }
    else
        for $k := h$ to $mid$ do
        {
            $b[i] := a[k]$; $i := i + 1$;
        }
    for $k := low$ to $high$ do $a[k] := b[k]$;
}
```

Tree of Calls of Merge Sort

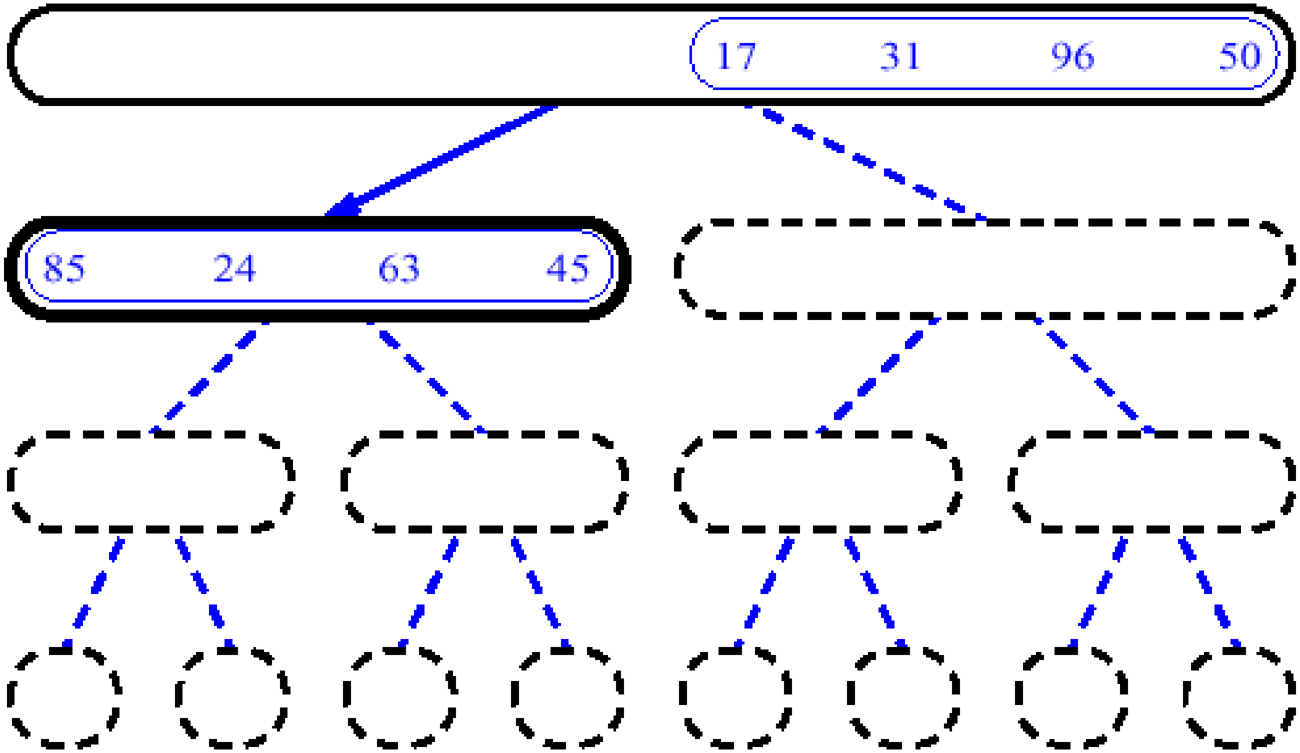
(310 | 285 | 179 | 652, 351 | 423, 861, 254, 450, 520)



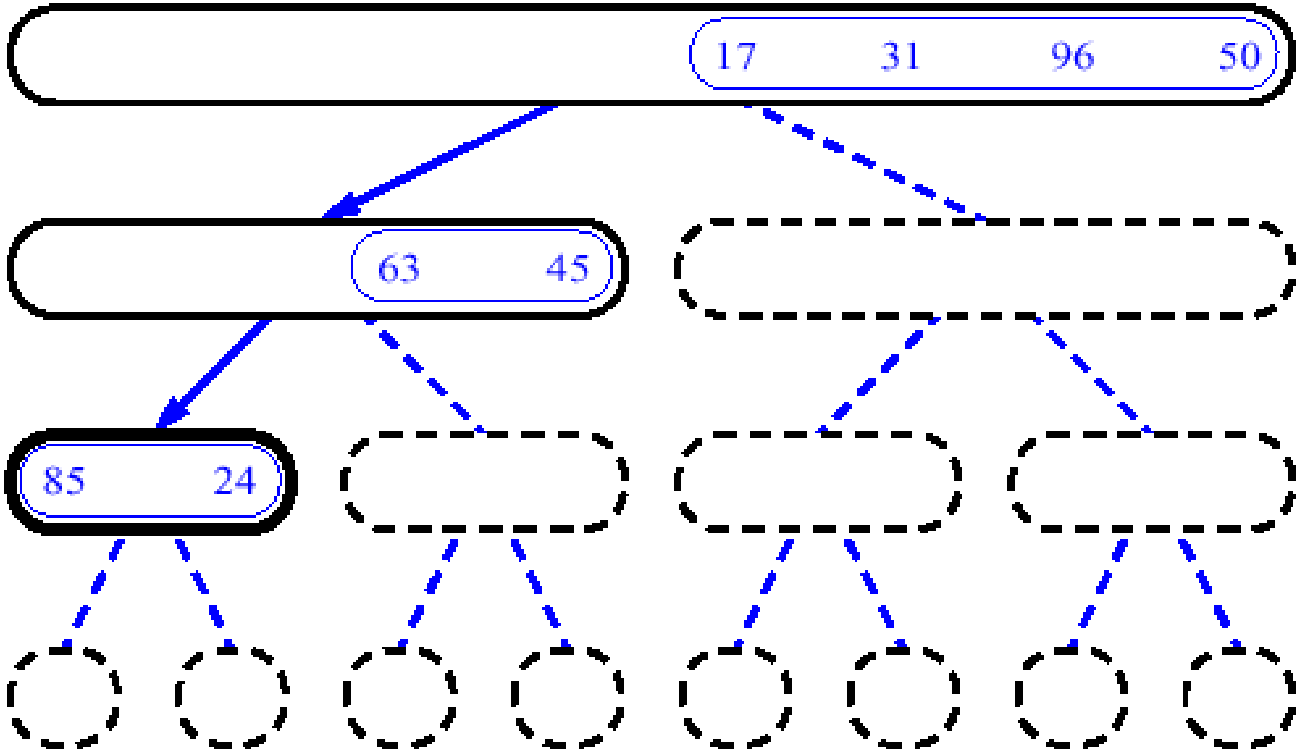
Merge Sort (Example) Cont..



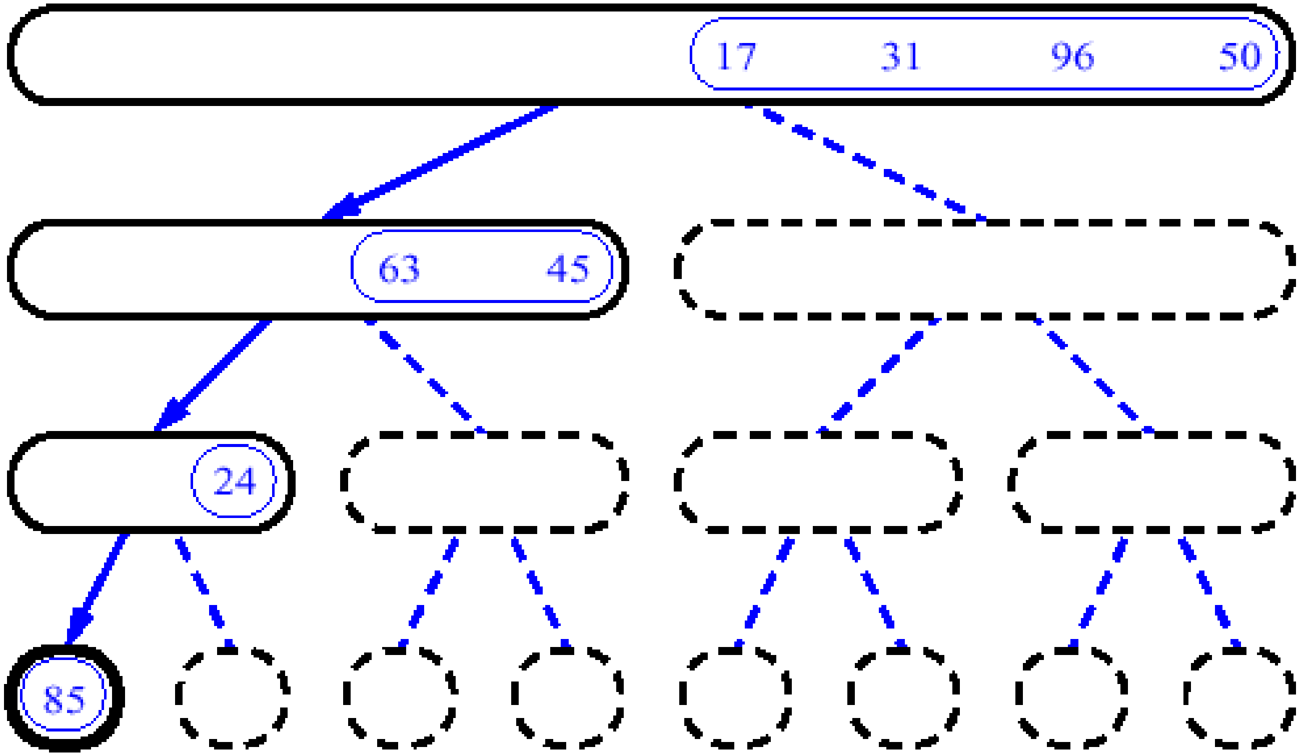
Merge Sort (Example) Cont..



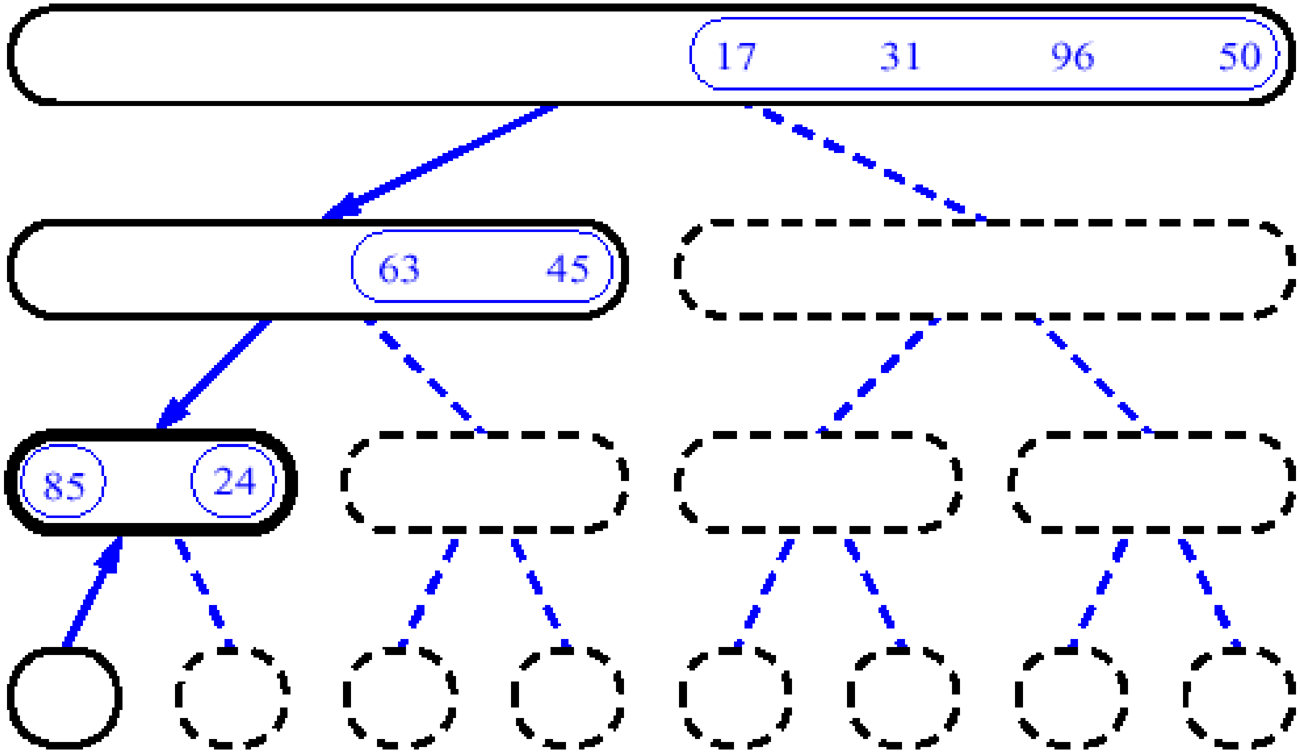
MergeSort (Example) Cont..



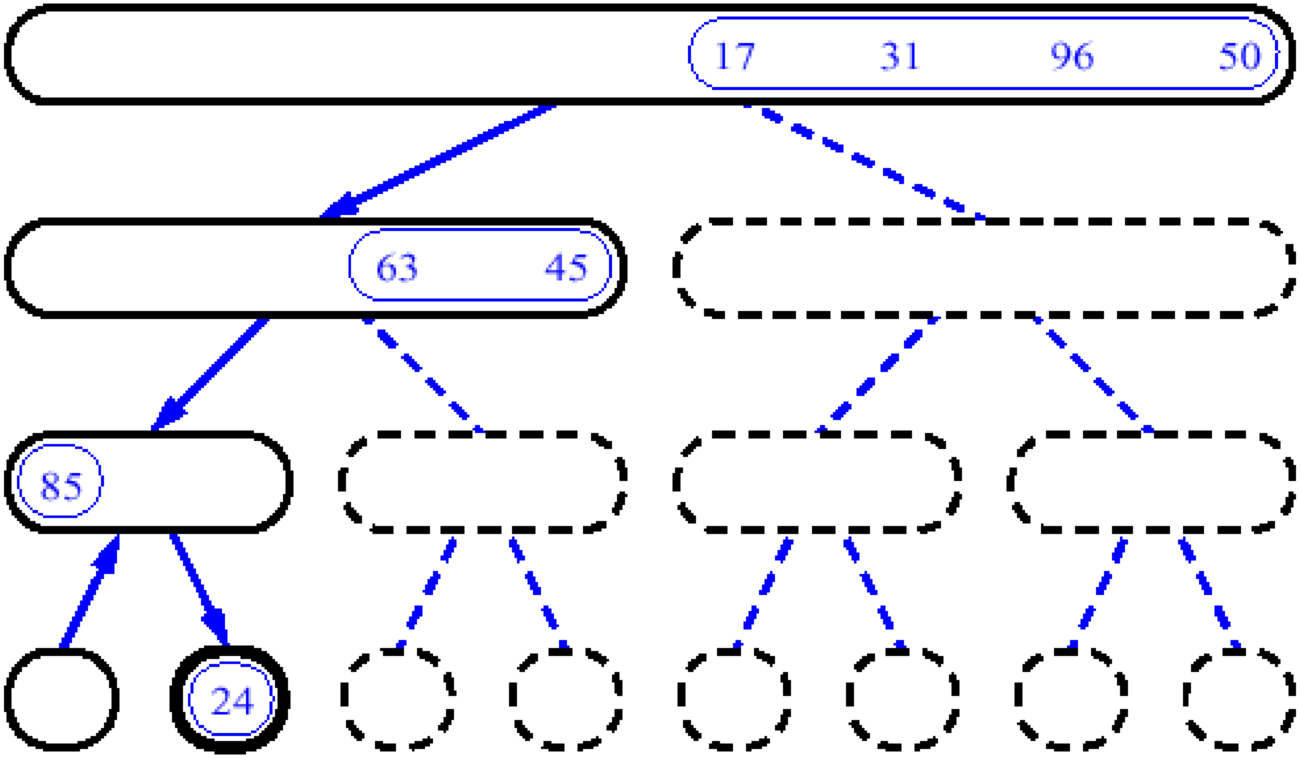
MergeSort (Example) Cont..



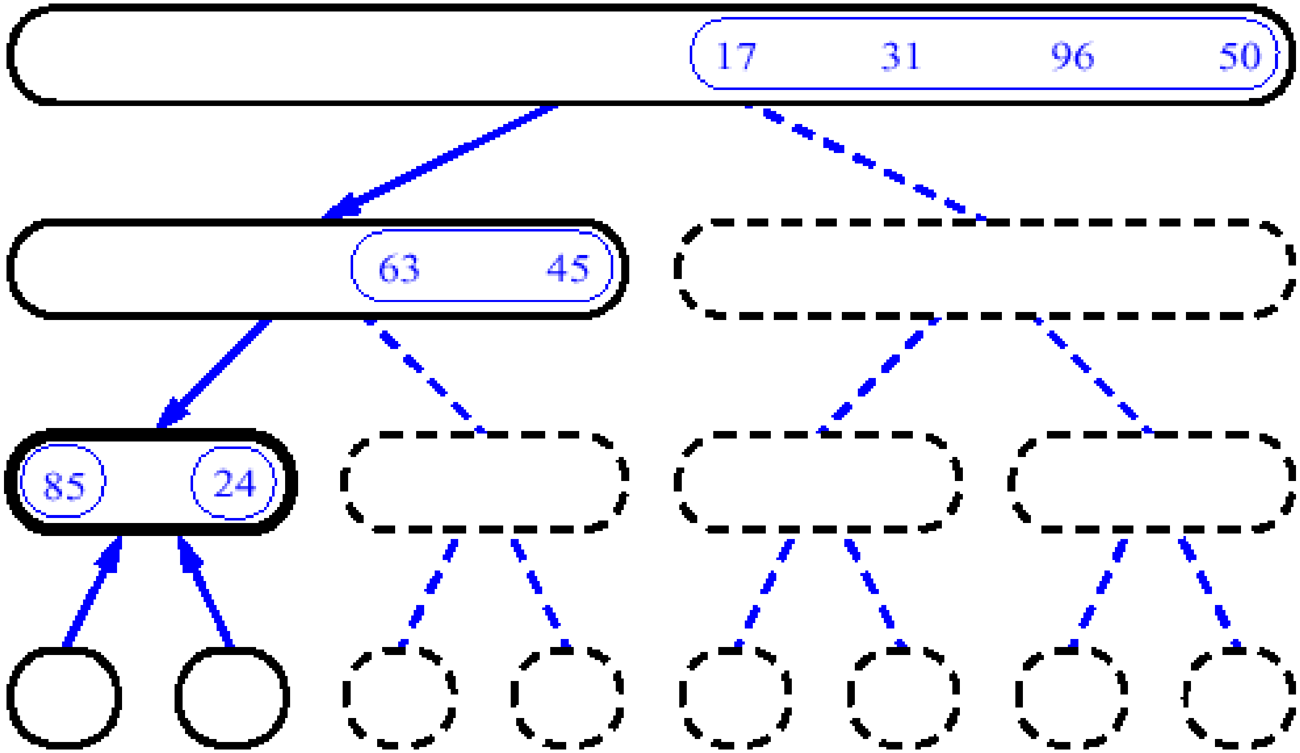
MergeSort (Example) Cont..



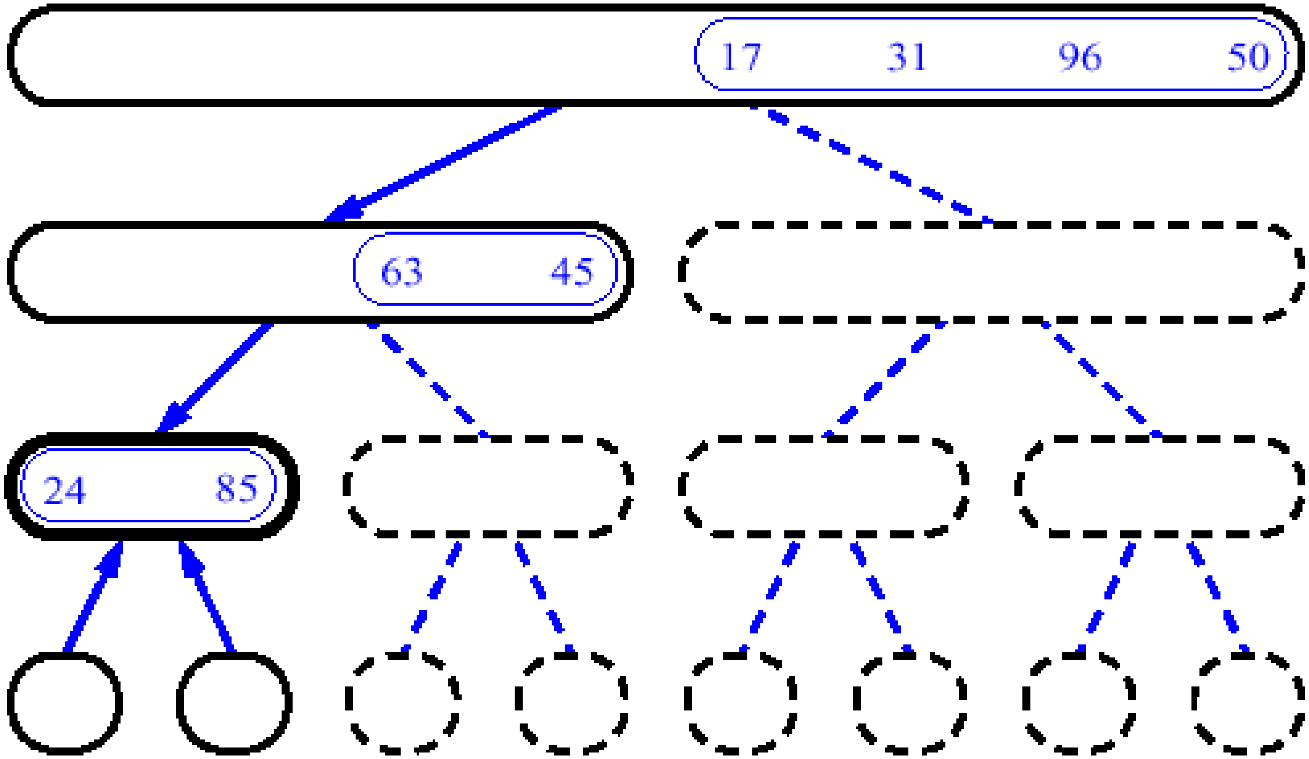
MergeSort (Example) Cont..



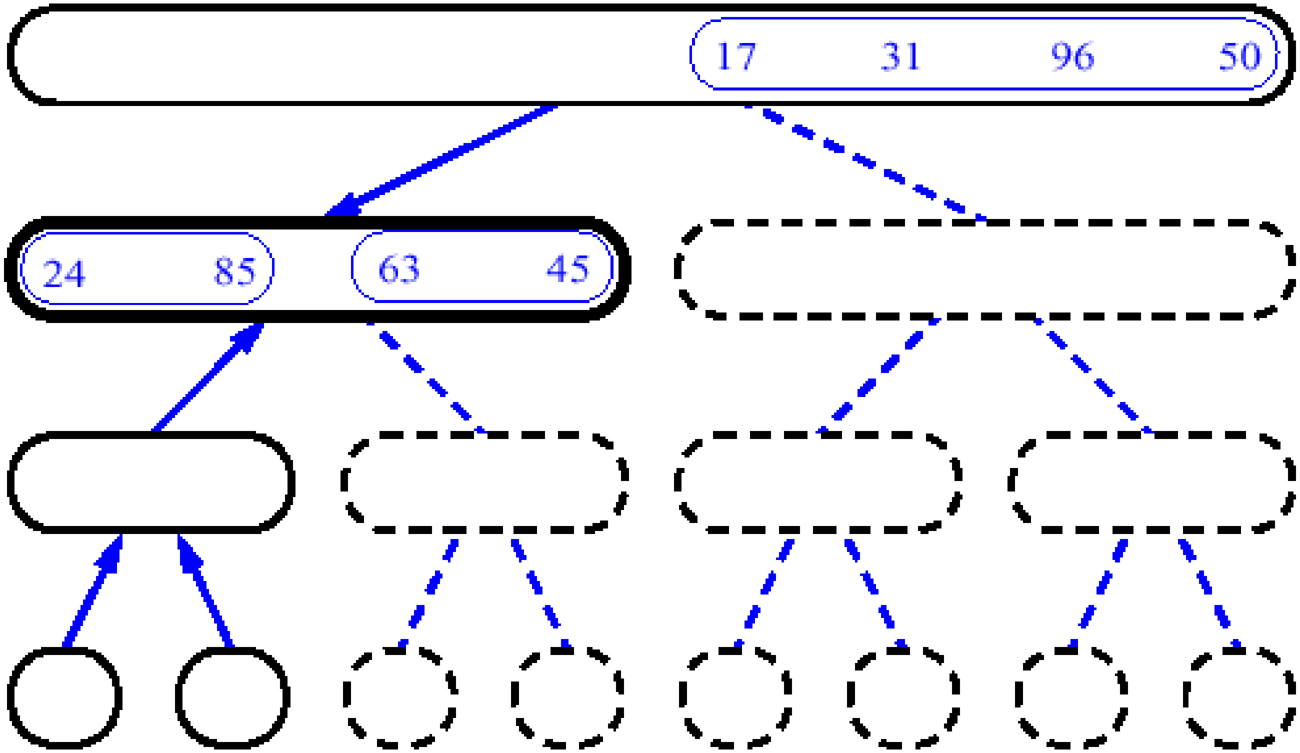
MergeSort (Example) Cont..



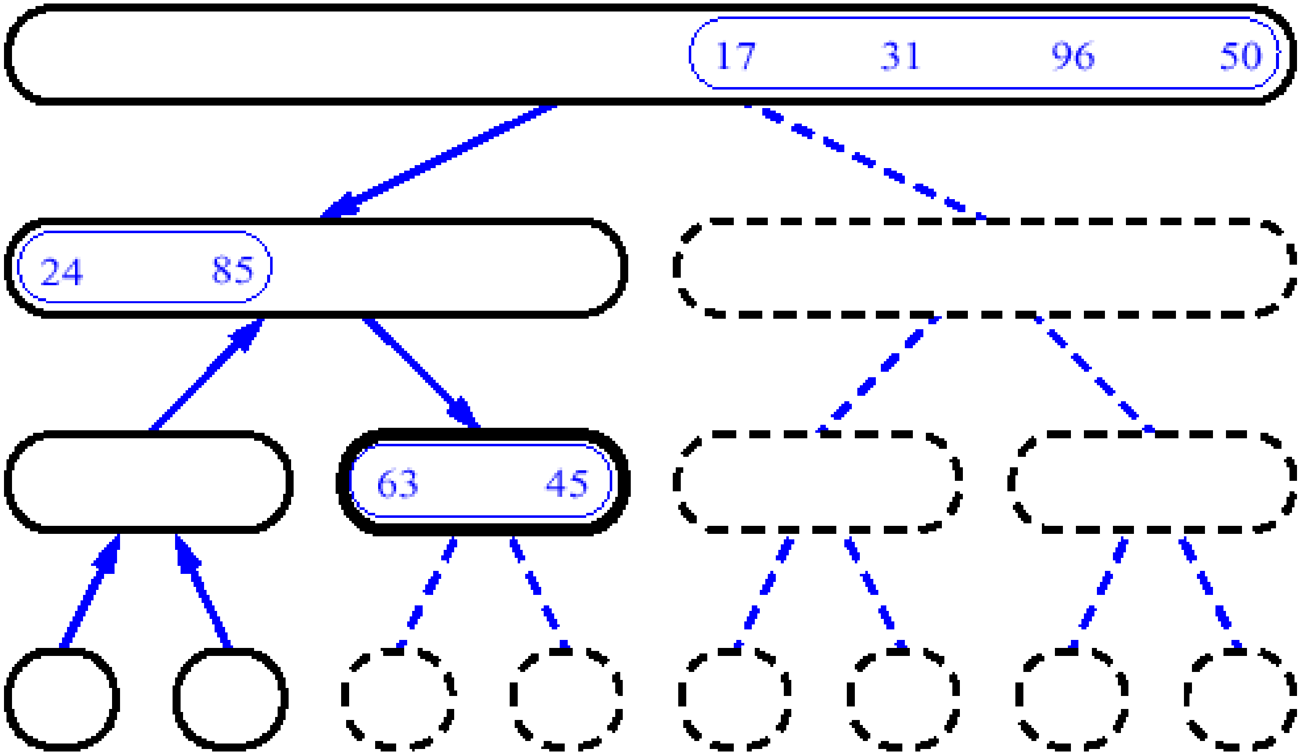
MergeSort (Example) Cont..



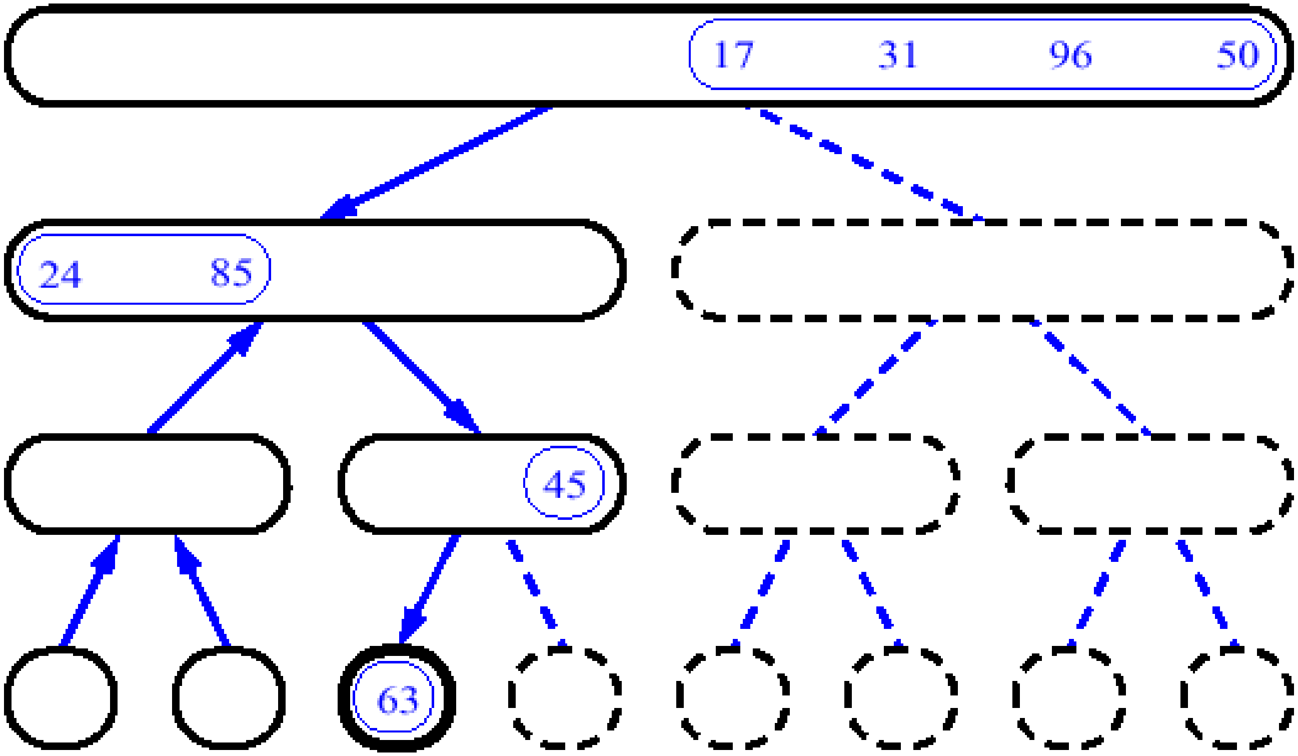
MergeSort (Example) Cont..



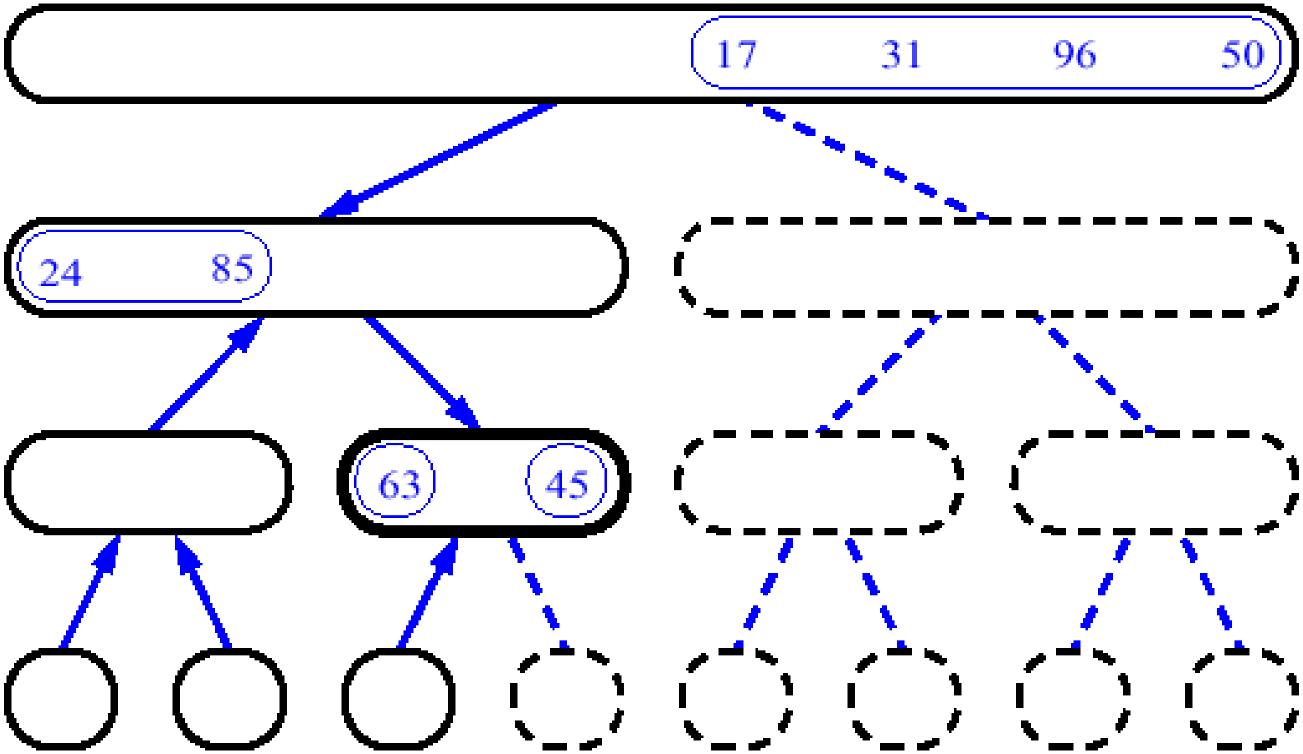
MergeSort (Example) Cont..



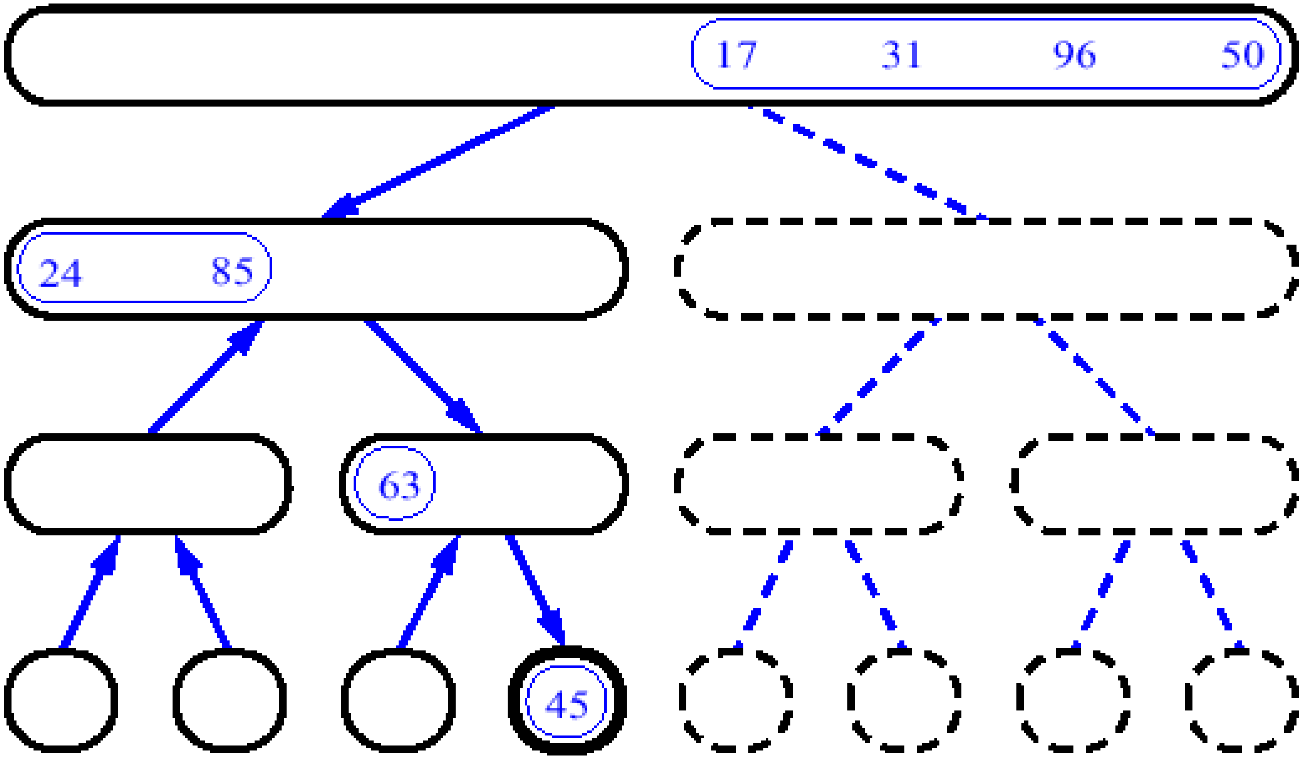
MergeSort (Example) Cont..



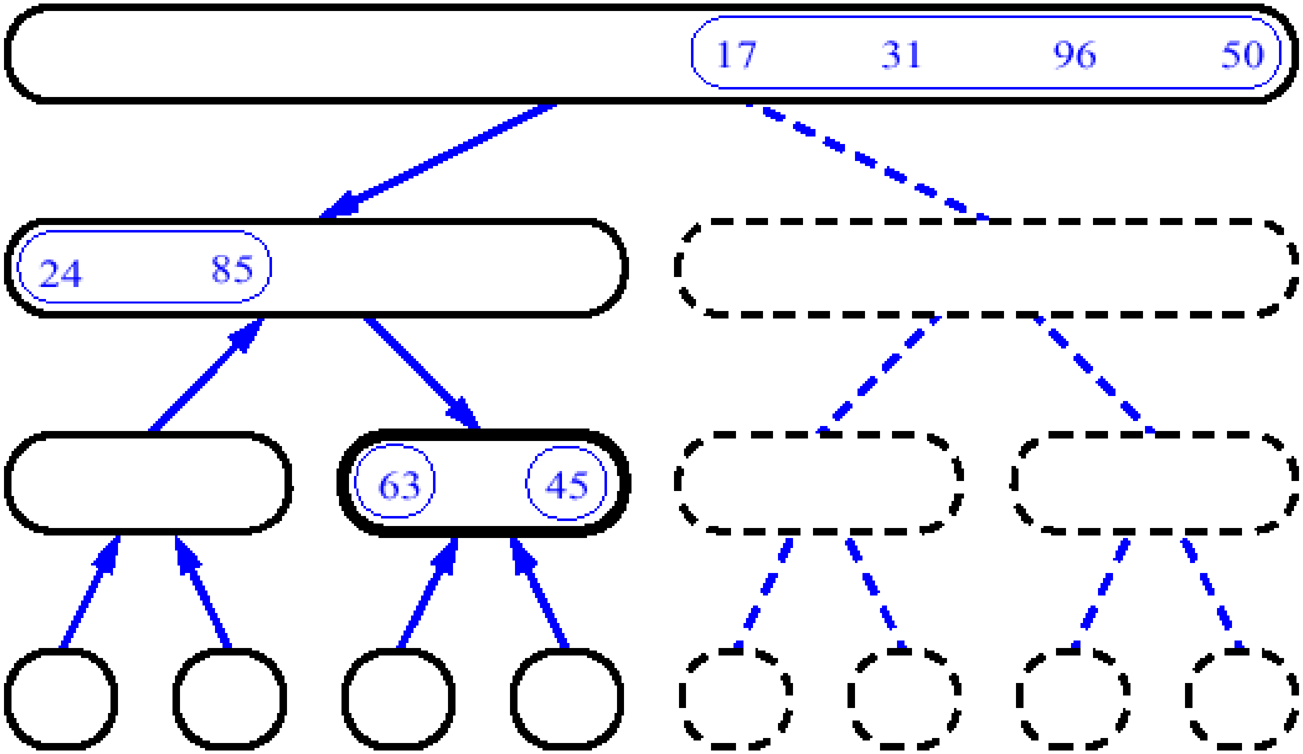
MergeSort (Example) Cont..



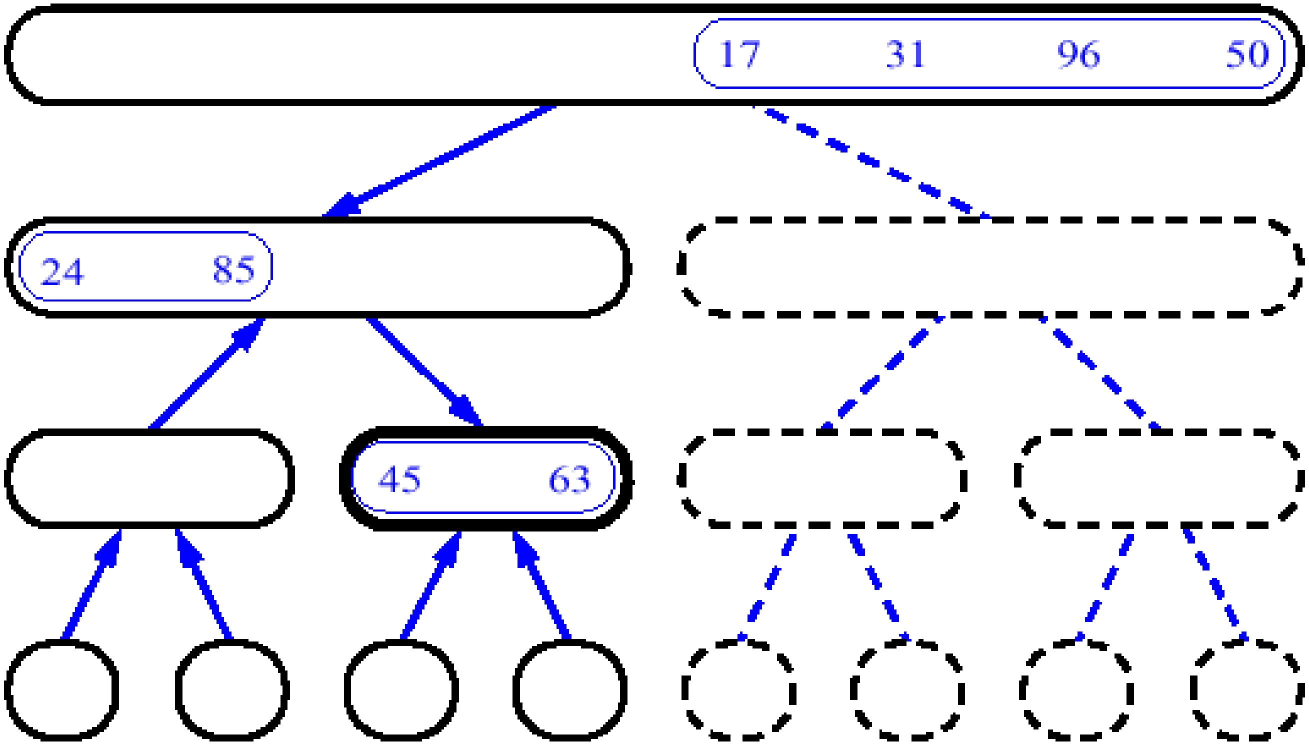
MergeSort (Example) Cont..



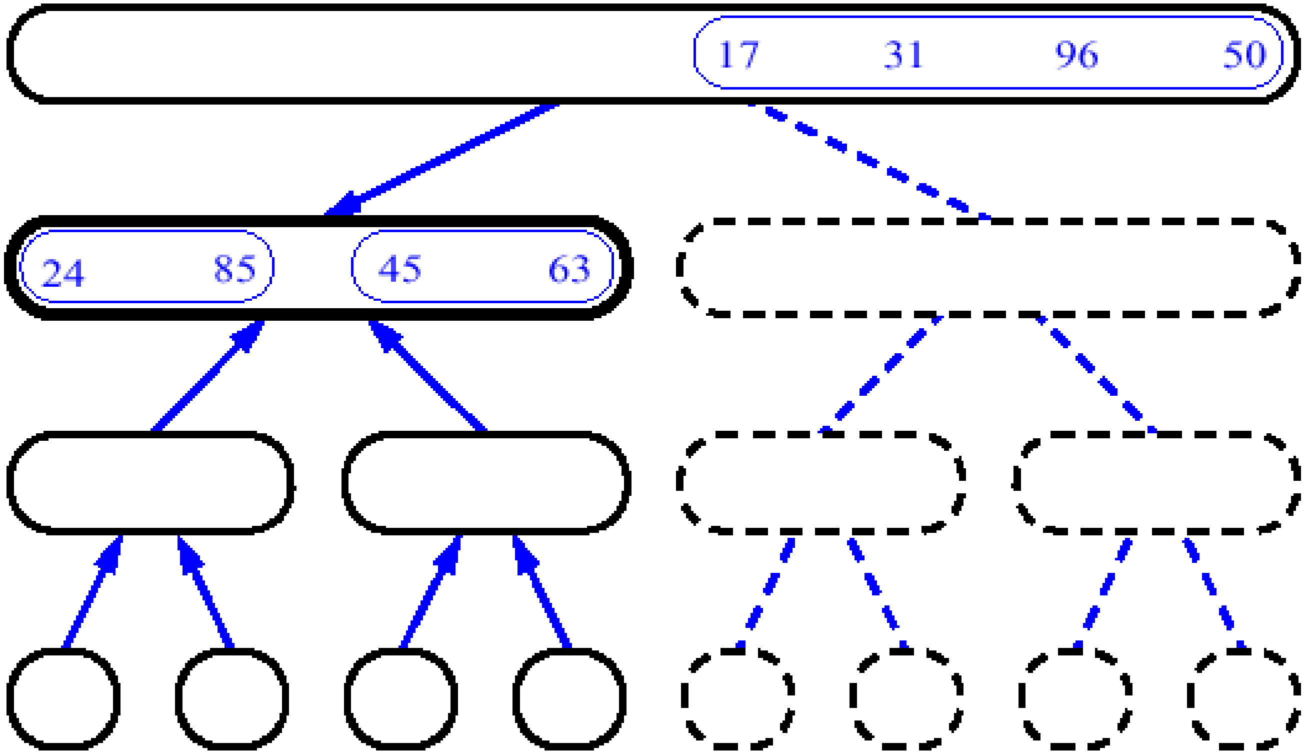
MergeSort (Example) Cont..



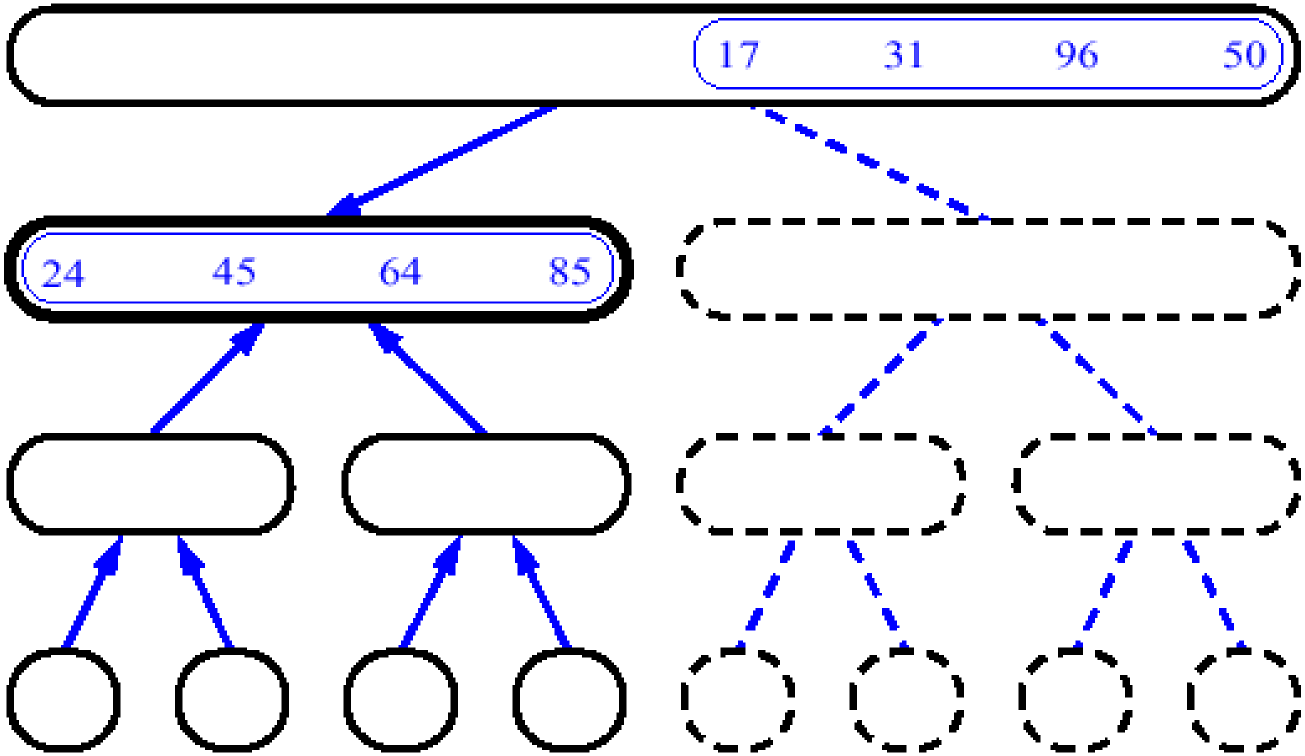
MergeSort (Example) Cont..



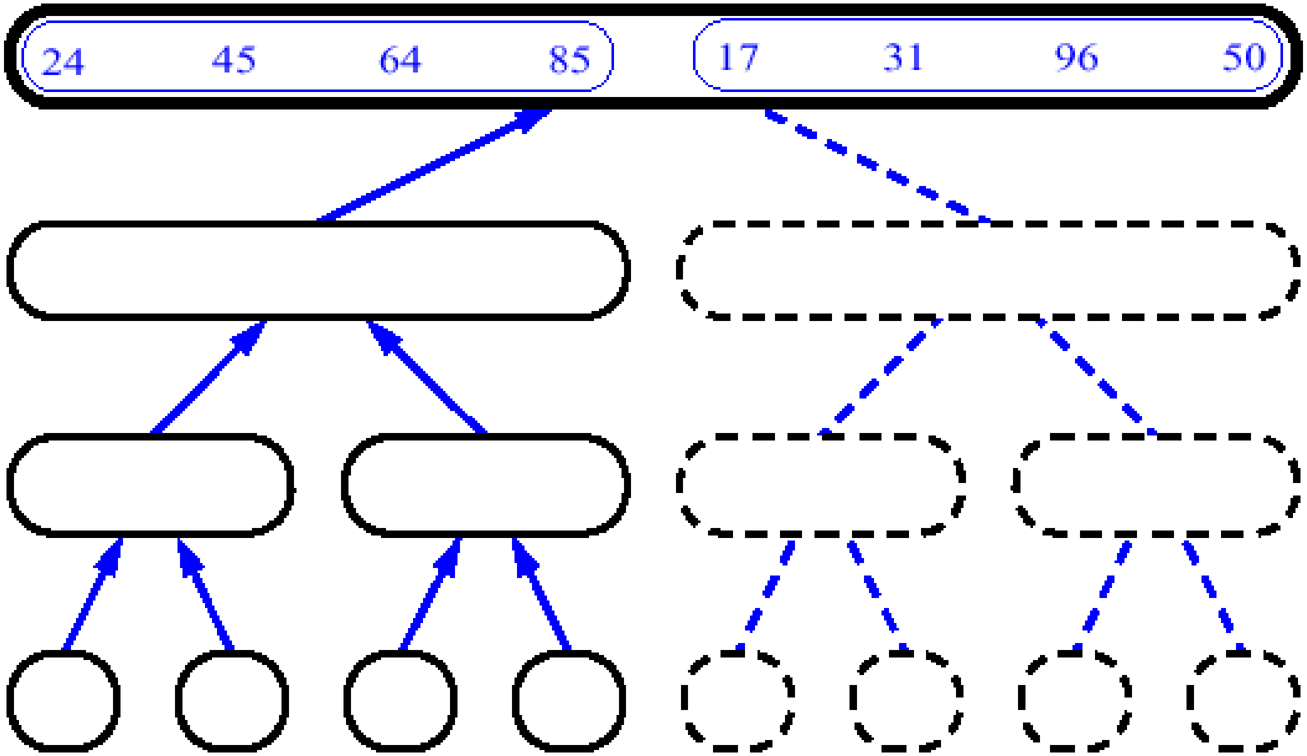
MergeSort (Example) Cont..



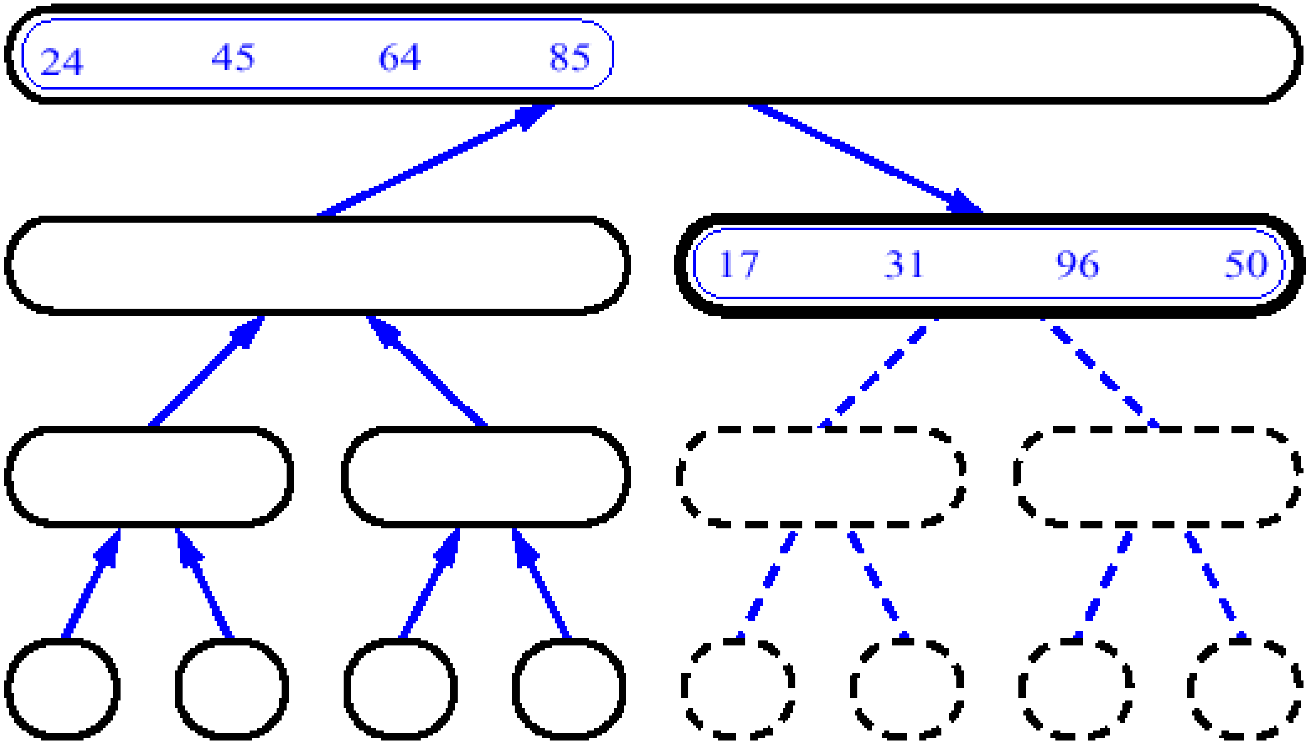
MergeSort (Example) Cont..



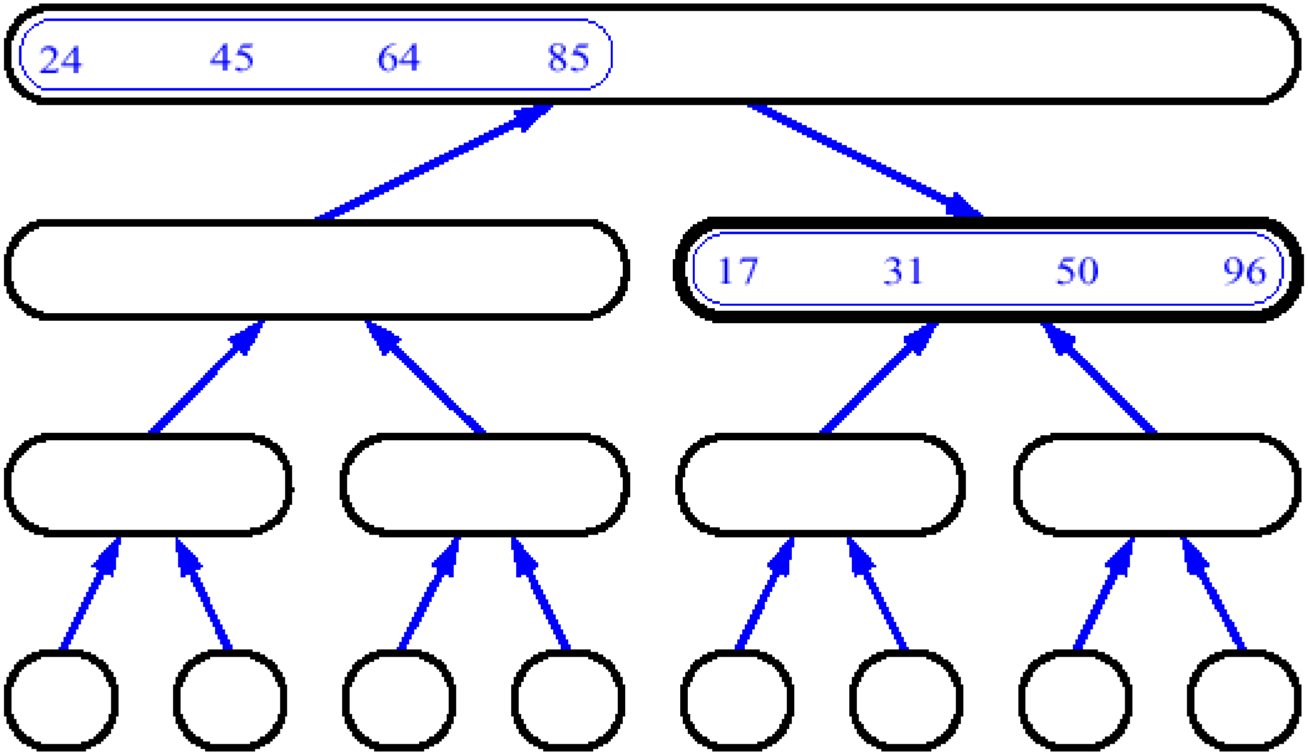
MergeSort (Example) Cont..



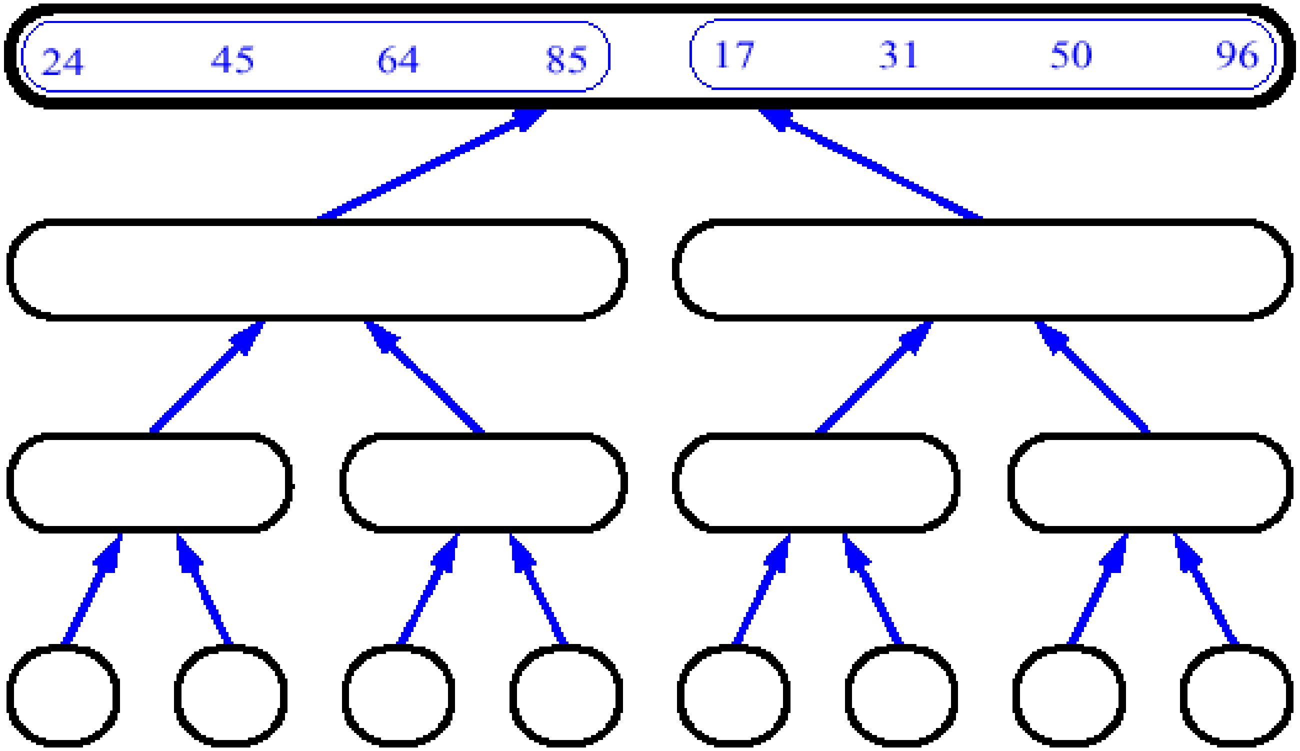
MergeSort (Example) Cont..



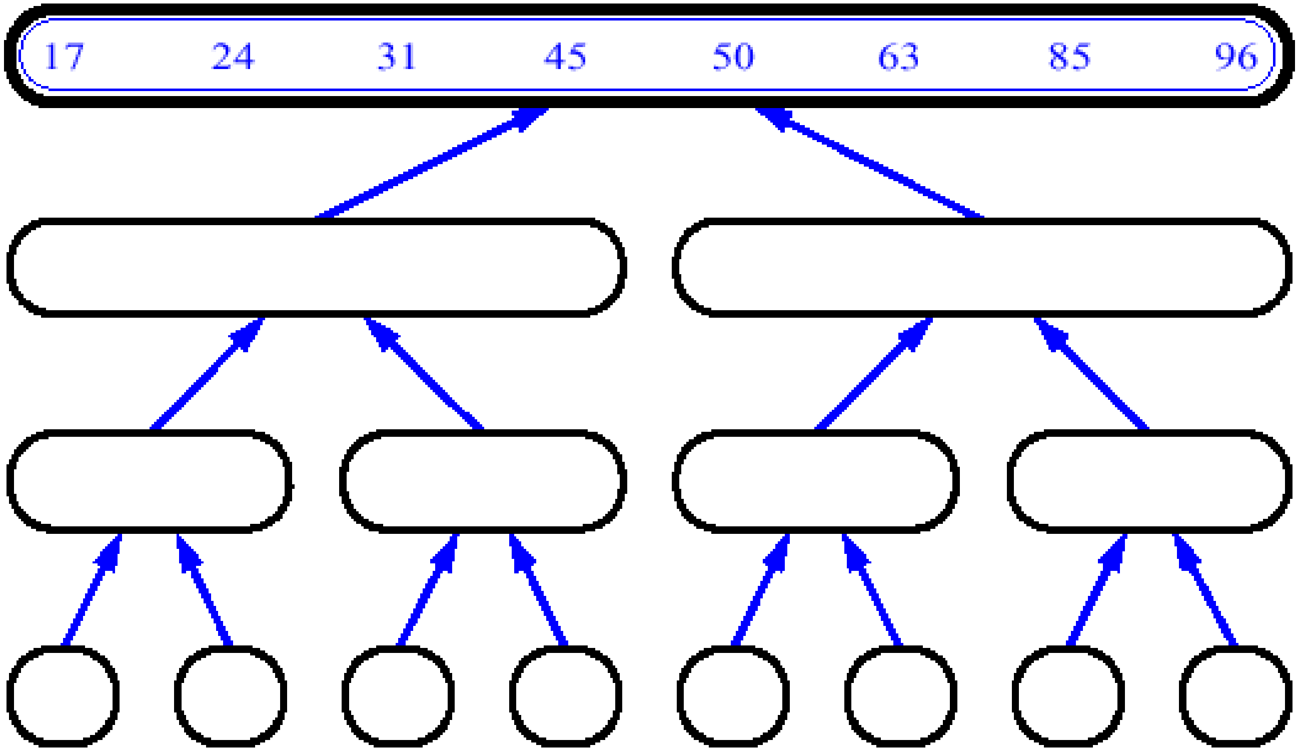
MergeSort (Example) Cont..



MergeSort (Example) Cont..



MergeSort (Example) Cont..

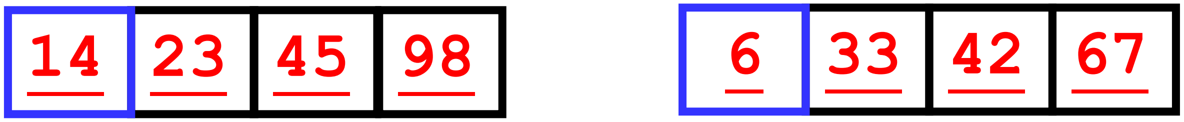


Merge

14	23	45	98

6	33	42	67

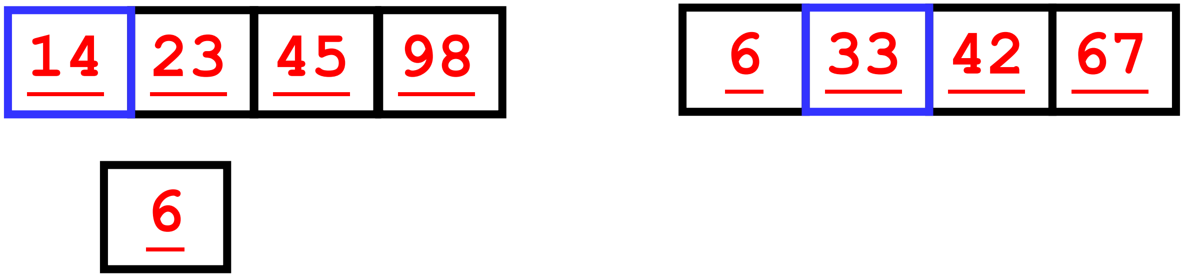
Merge



14	23	45	98

6	33	42	67

Merge

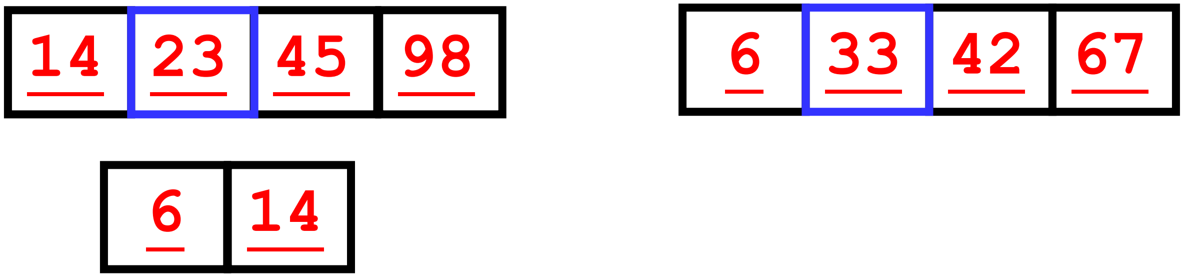


14 23 45 98

6 33 42 67

6

Merge

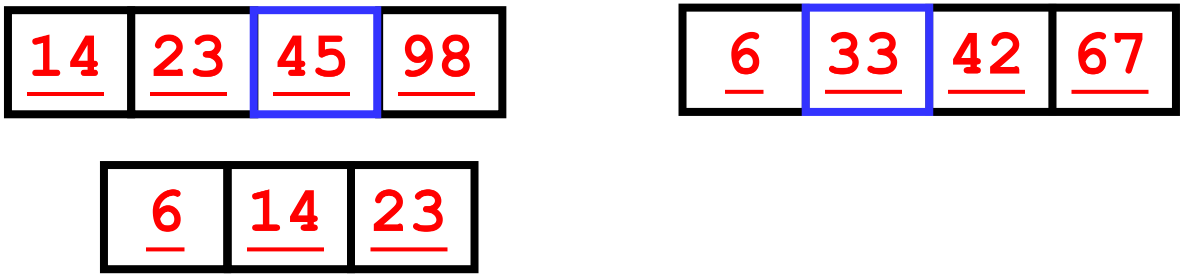


14 23 45 98

6 33 42 67

6 14

Merge

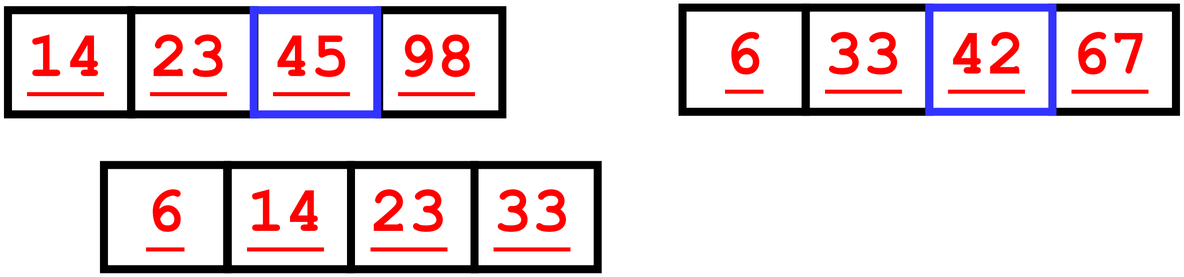


14 23 45 98

6 33 42 67

6 14 23

Merge

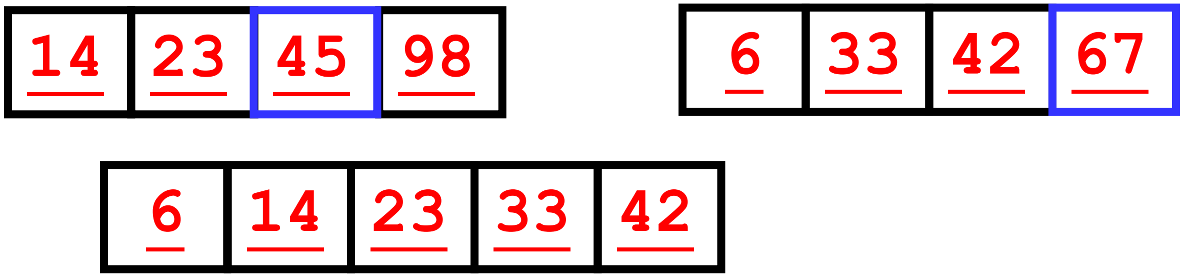


14 23 45 98

6 33 42 67

6 14 23 33

Merge

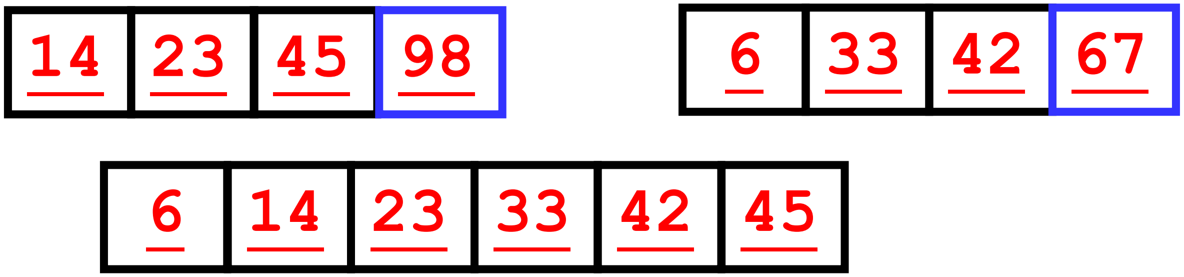


14 23 45 98

6 33 42 67

6 14 23 33 42

Merge

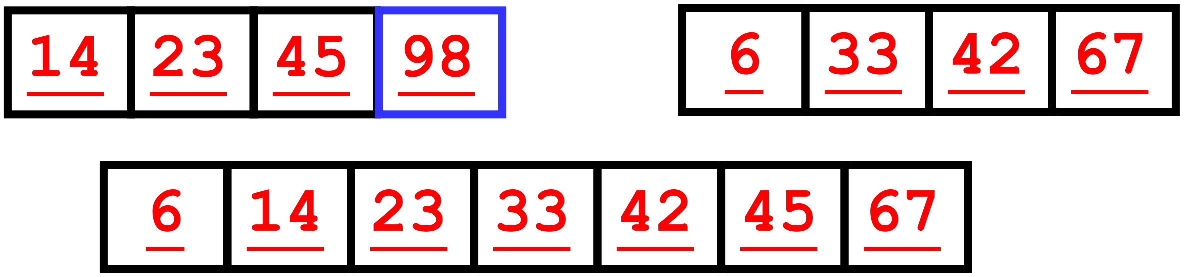


14 23 45 98

6 33 42 67

6 14 23 33 42 45

Merge



14 23 45 98

6 33 42 67

6 14 23 33 42 45 67

Merge

14	23	45	98

6	33	42	67

6	14	23	33	42	45	67	98