Why Sorting?

- “When in doubt, sort” – one of the principles of algorithm design. Sorting used as a subroutine in many of the algorithms:
  - Searching in databases: we can do binary search on sorted data
  - A large number of computer graphics and computational geometry problems
  - Closest pair, element uniqueness

Why Sorting? (2)

- A large number of sorting algorithms are developed representing different algorithm design techniques.
- A lower bound for sorting $\Omega(n \log n)$ is used to prove lower bounds of other problems

Sorting Algorithms so far

- Insertion sort, selection sort
  - Worst-case running time $\Theta(n^2)$; in-place
- Heap sort
  - Worst-case running time $\Theta(n \log n)$.

Divide and Conquer

- *Divide-and-conquer* method for algorithm design:
  - **Divide**: if the input size is too large to deal with in a straightforward manner, divide the problem into two or more disjoint subproblems
  - **Conquer**: use divide and conquer recursively to solve the subproblems
  - **Combine**: take the solutions to the subproblems and “merge” these solutions into a solution for the original problem

Merge Sort Algorithm

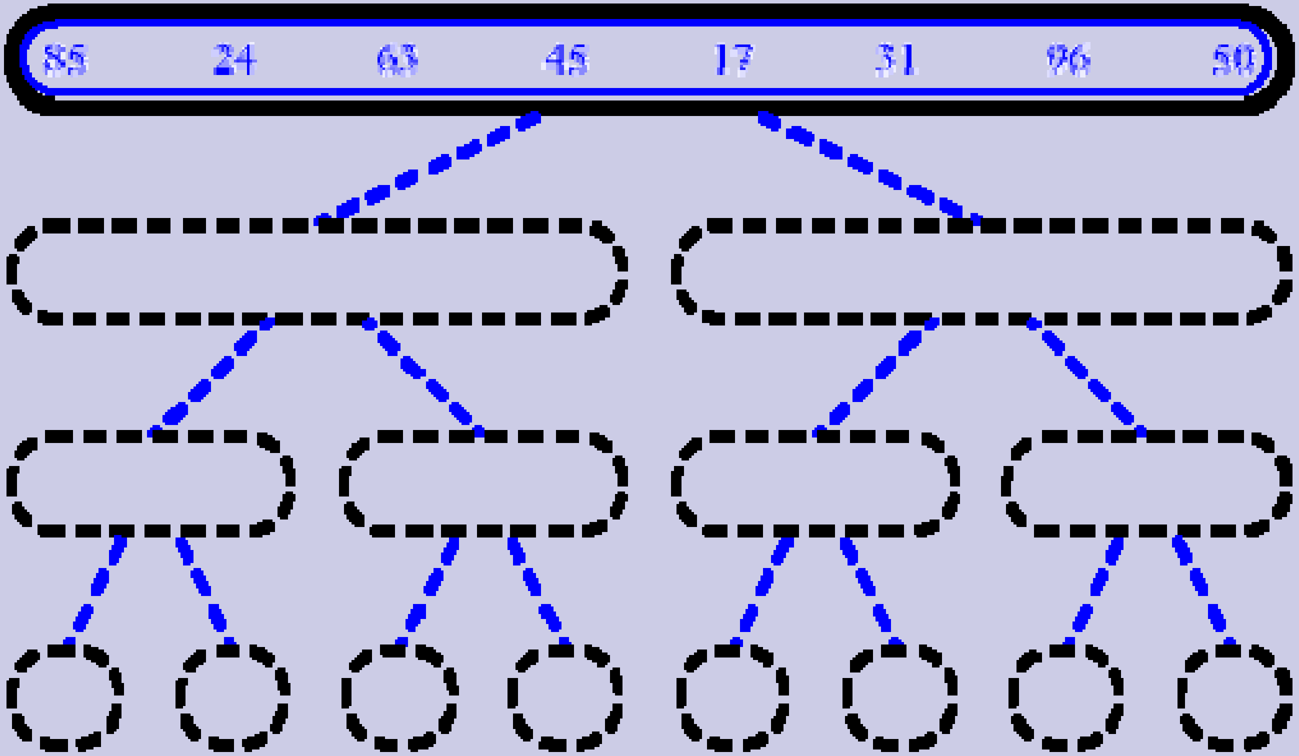
- **Divide**: If *S* has at least two elements (nothing needs to be done if *S* has zero or one elements), remove all the elements from *S* and put them into two sequences, S_1 and S_2 , each containing about half of the elements of *S*. (i.e. S_1 contains the first $\lceil n/2 \rceil$ elements and S_2 contains the remaining $\lfloor n/2 \rfloor$ elements).
- **Conquer**: Sort sequences S_1 and S_2 using Merge Sort.
- **Combine**: Put back the elements into *S* by merging the sorted sequences S_1 and S_2 into one sorted sequence

Merge Sort: Algorithm

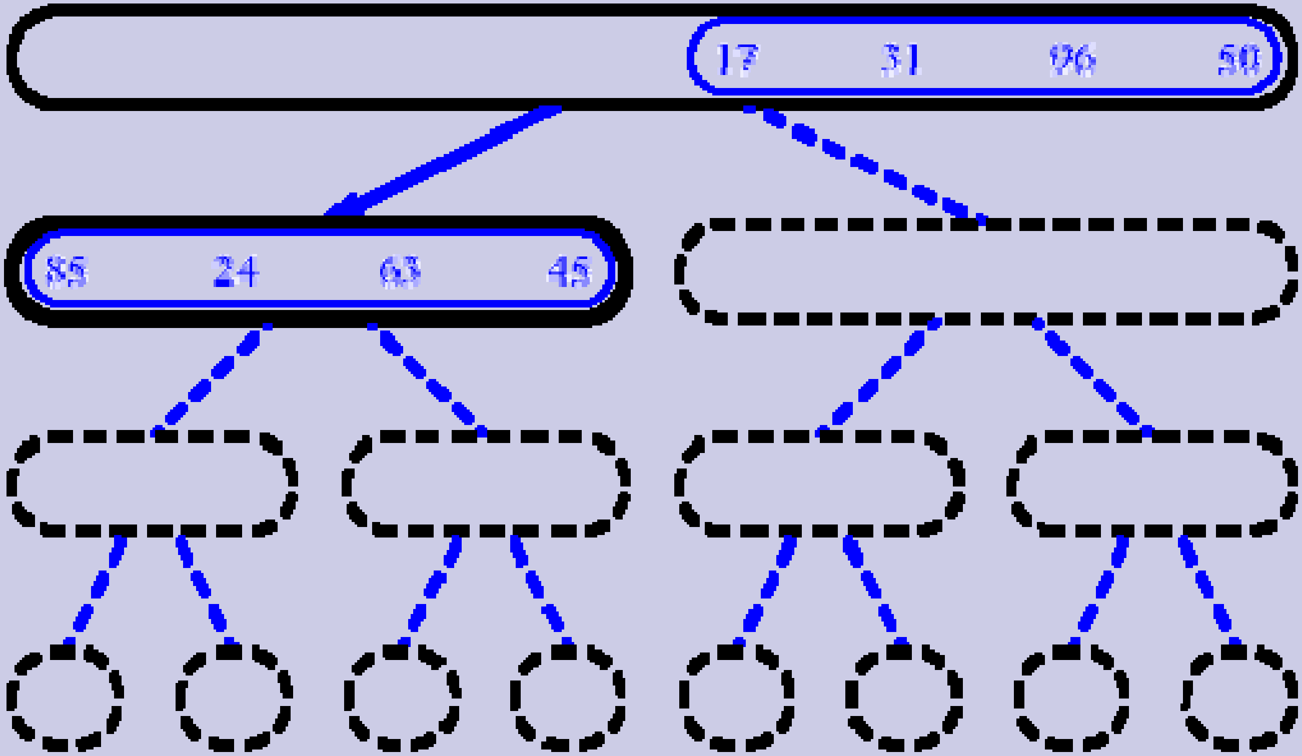
```
Merge-Sort(A, p, r)
   if p < r then
      q←(p+r)/2
      Merge-Sort(A, p, q)
      Merge-Sort(A, q+1, r)
      Merge(A, p, q, r)
```

```
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].
```

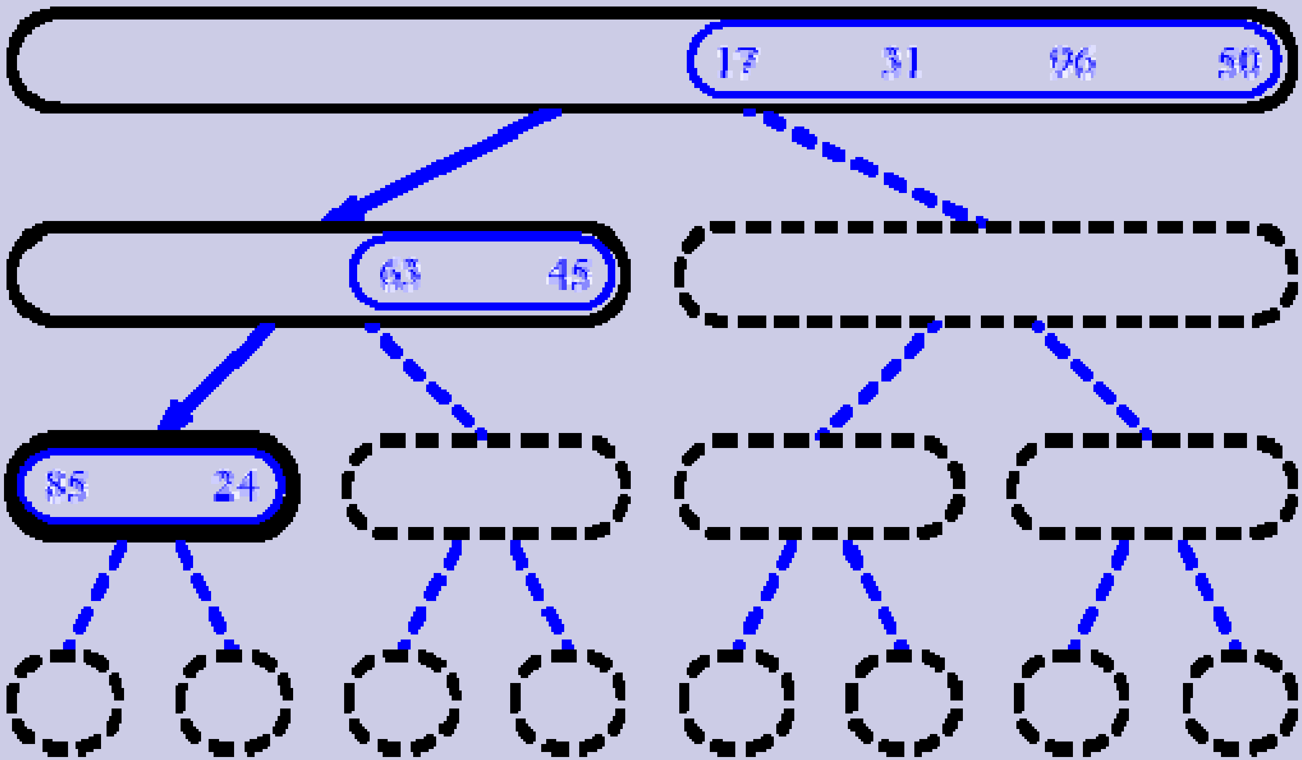
MergeSort (Example)



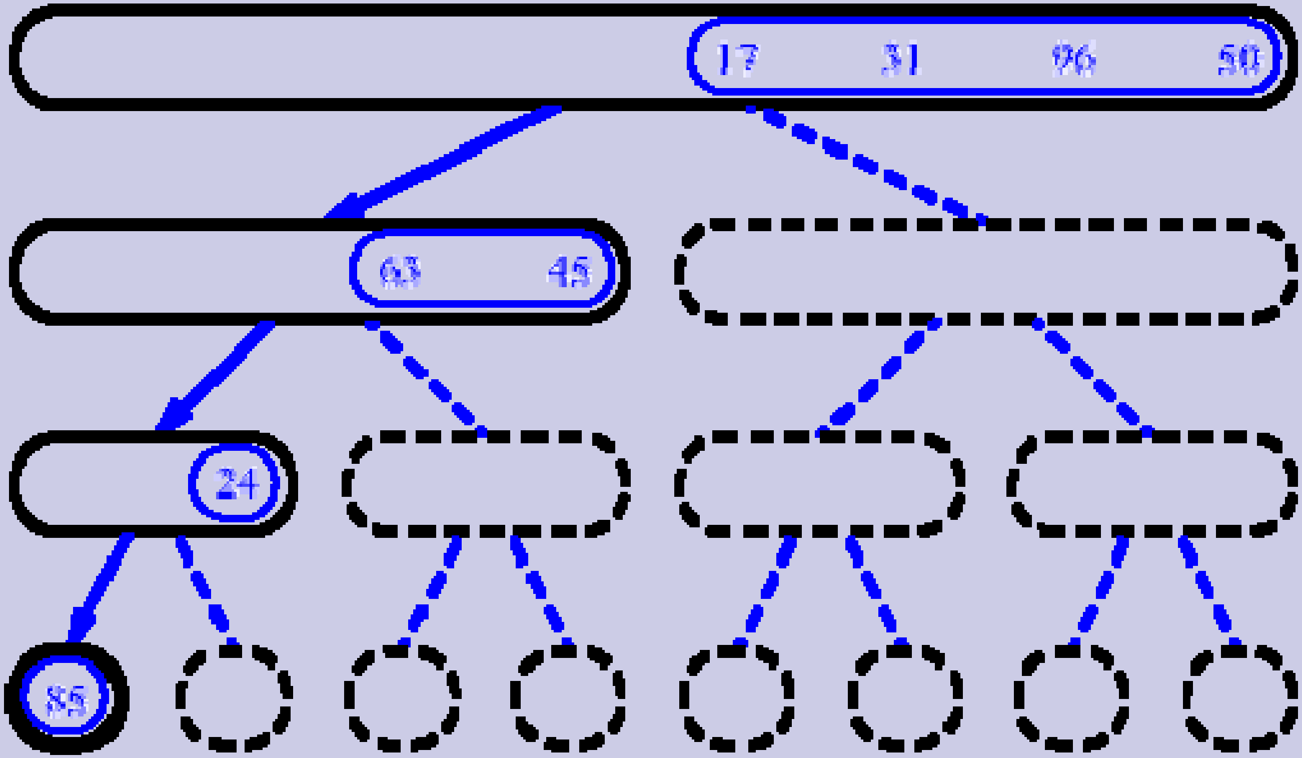
MergeSort (Example)



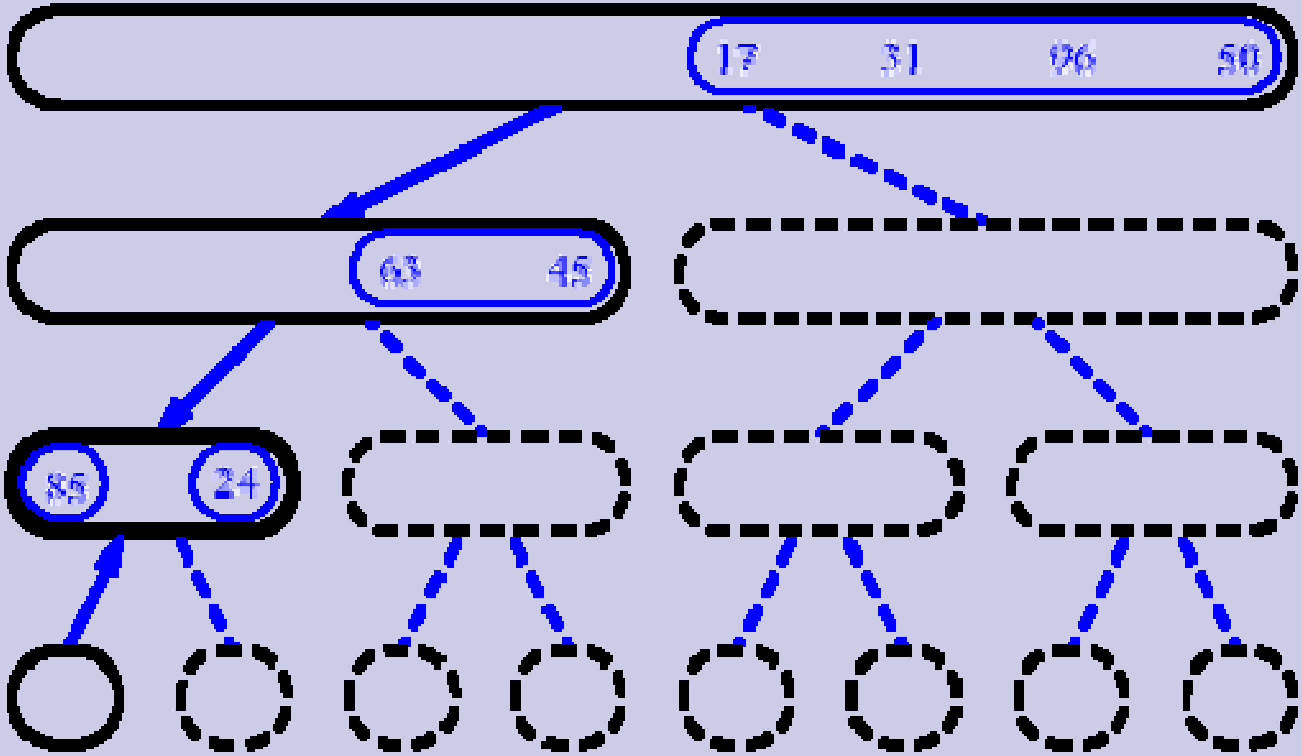
MergeSort (Example)



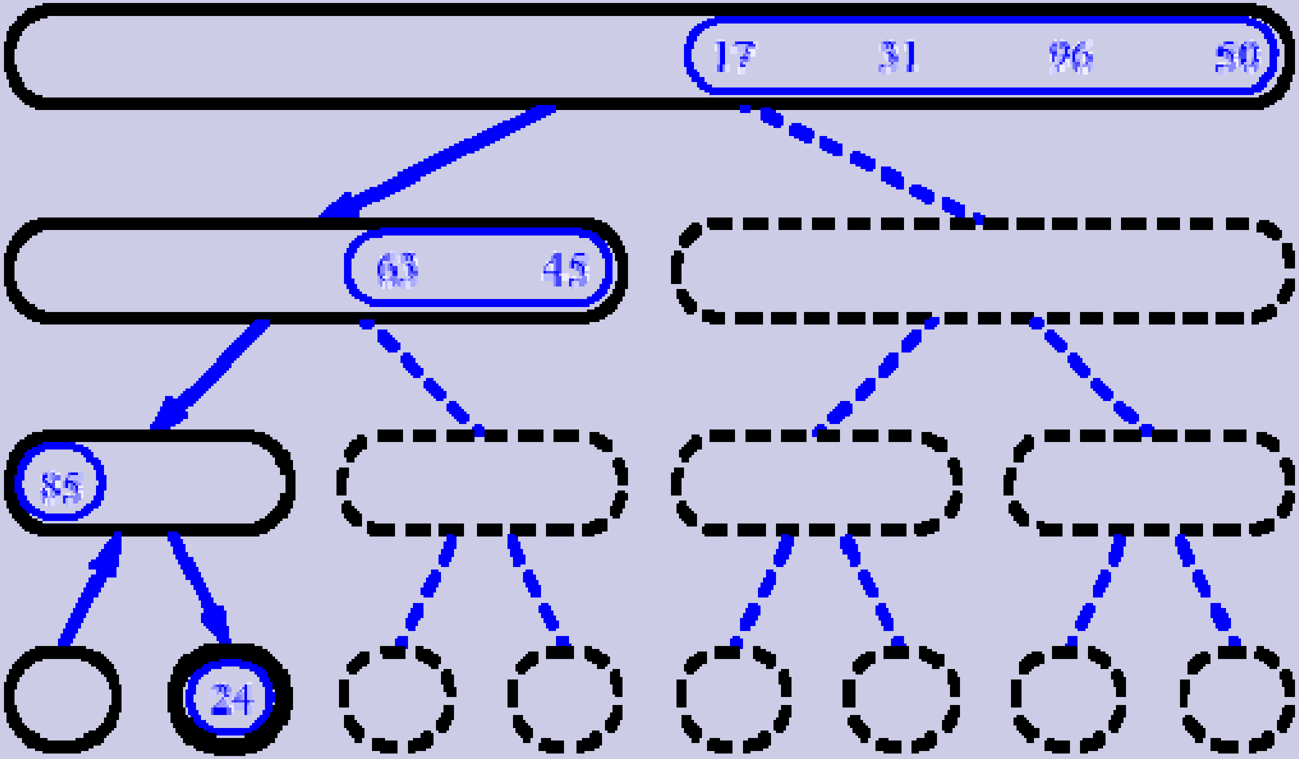
MergeSort (Example)



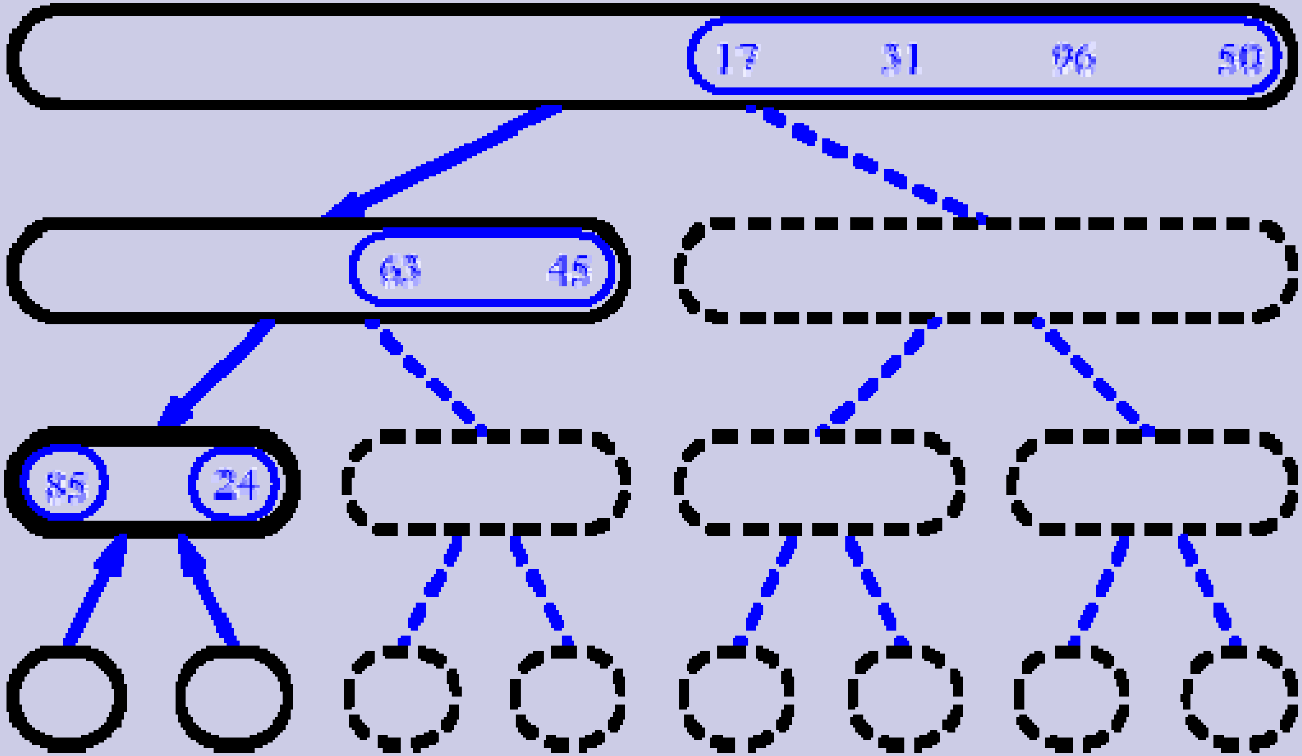
MergeSort (Example)



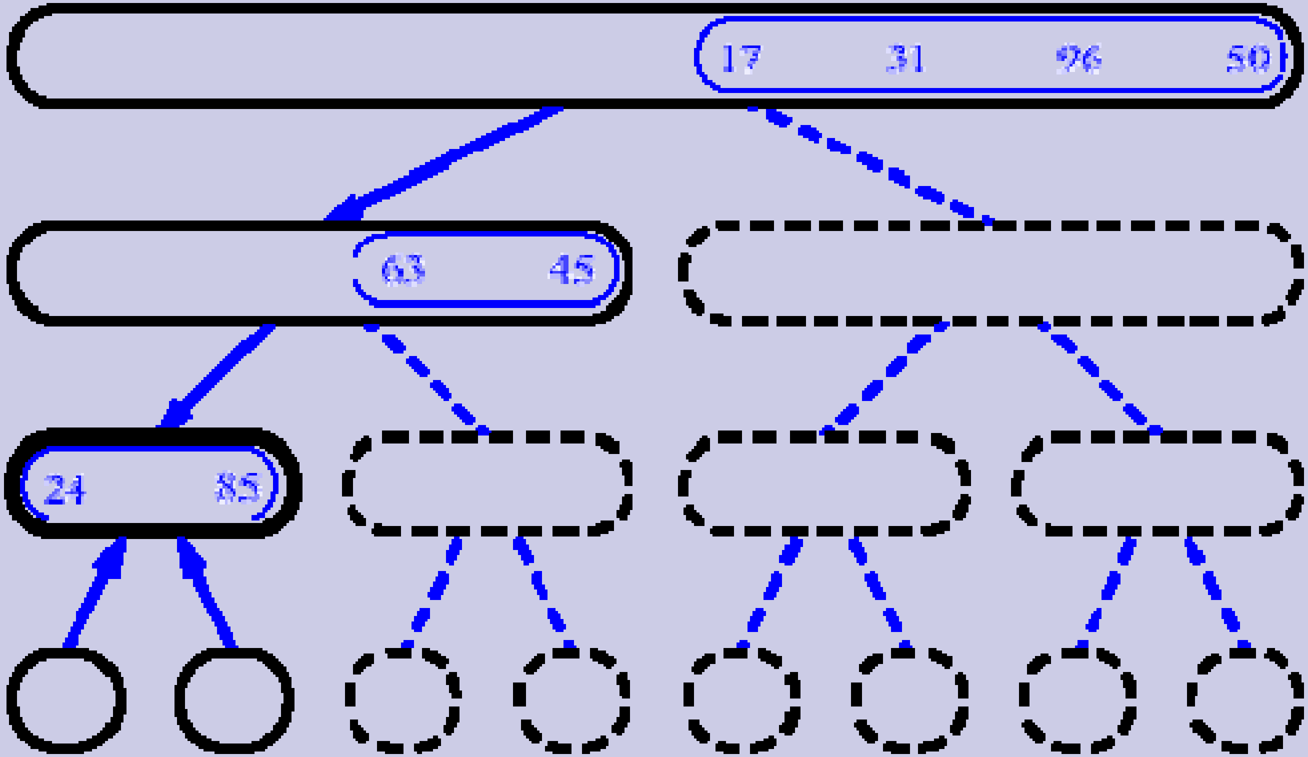
MergeSort (Example)



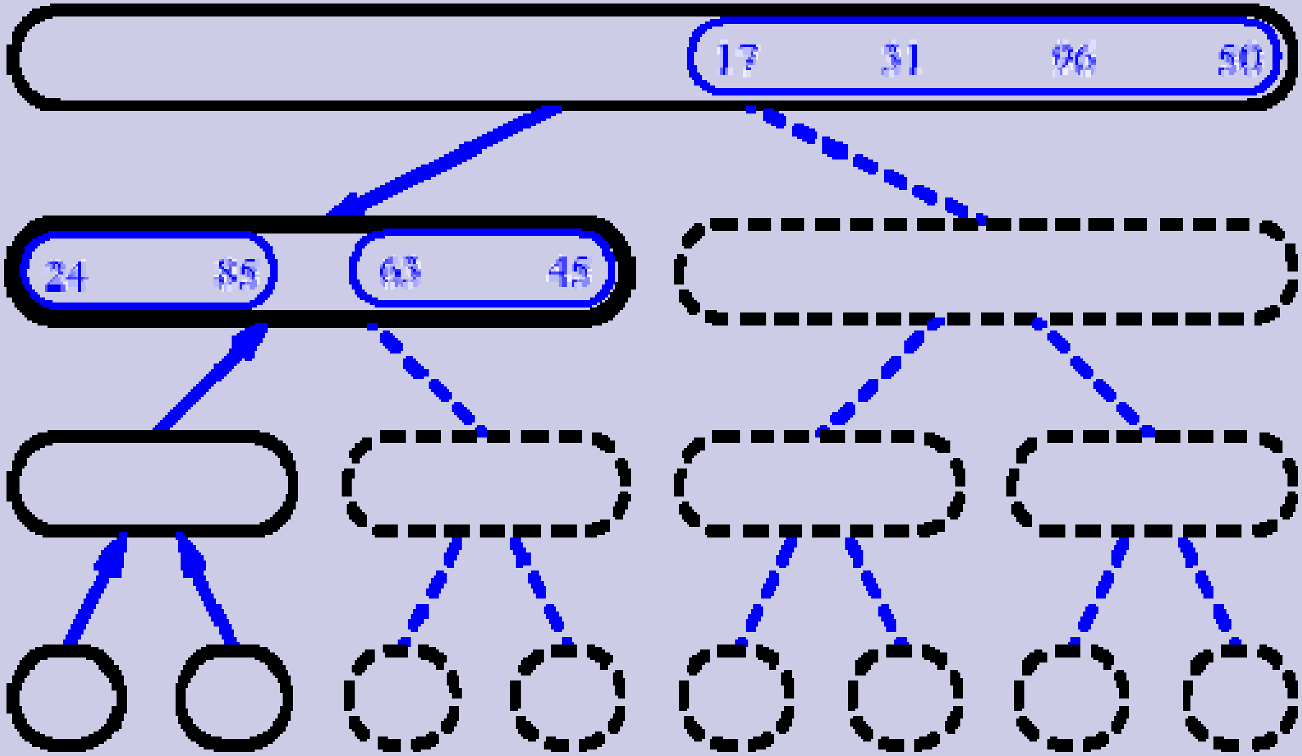
MergeSort (Example)



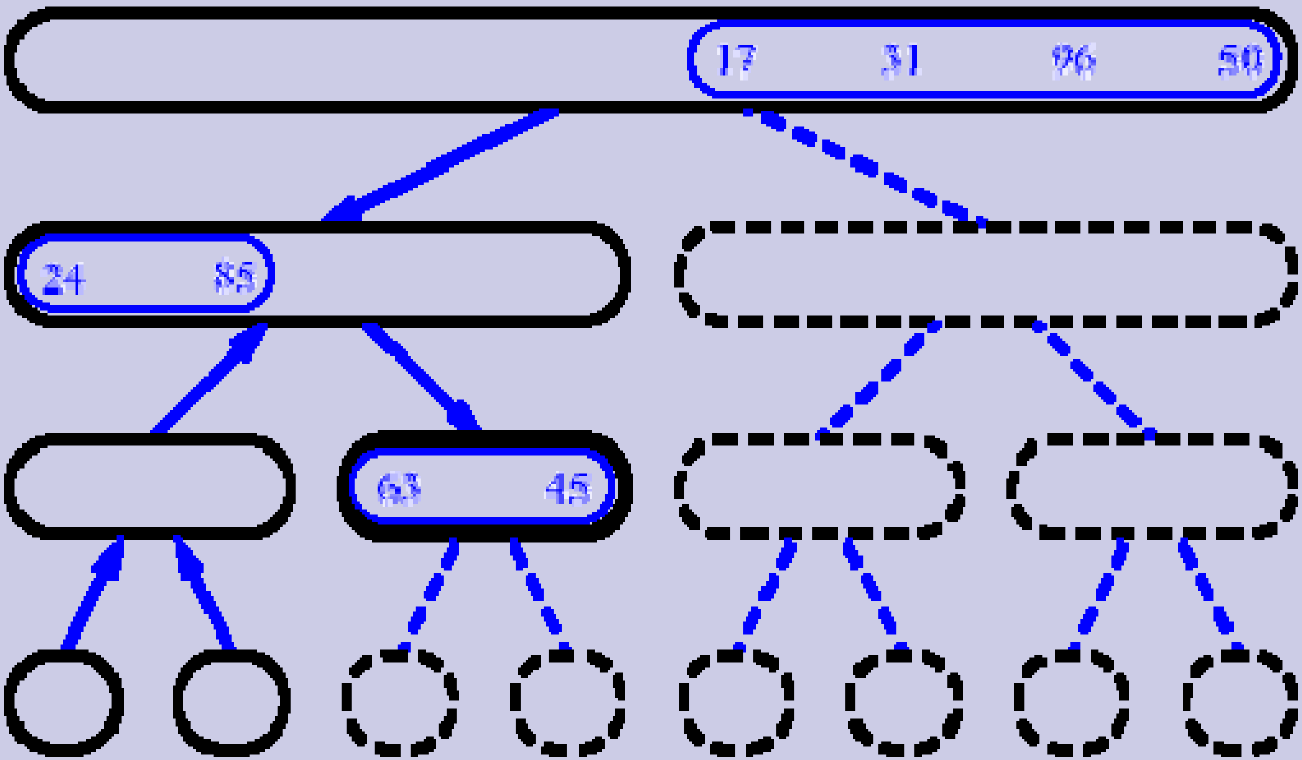
MergeSort (Example)



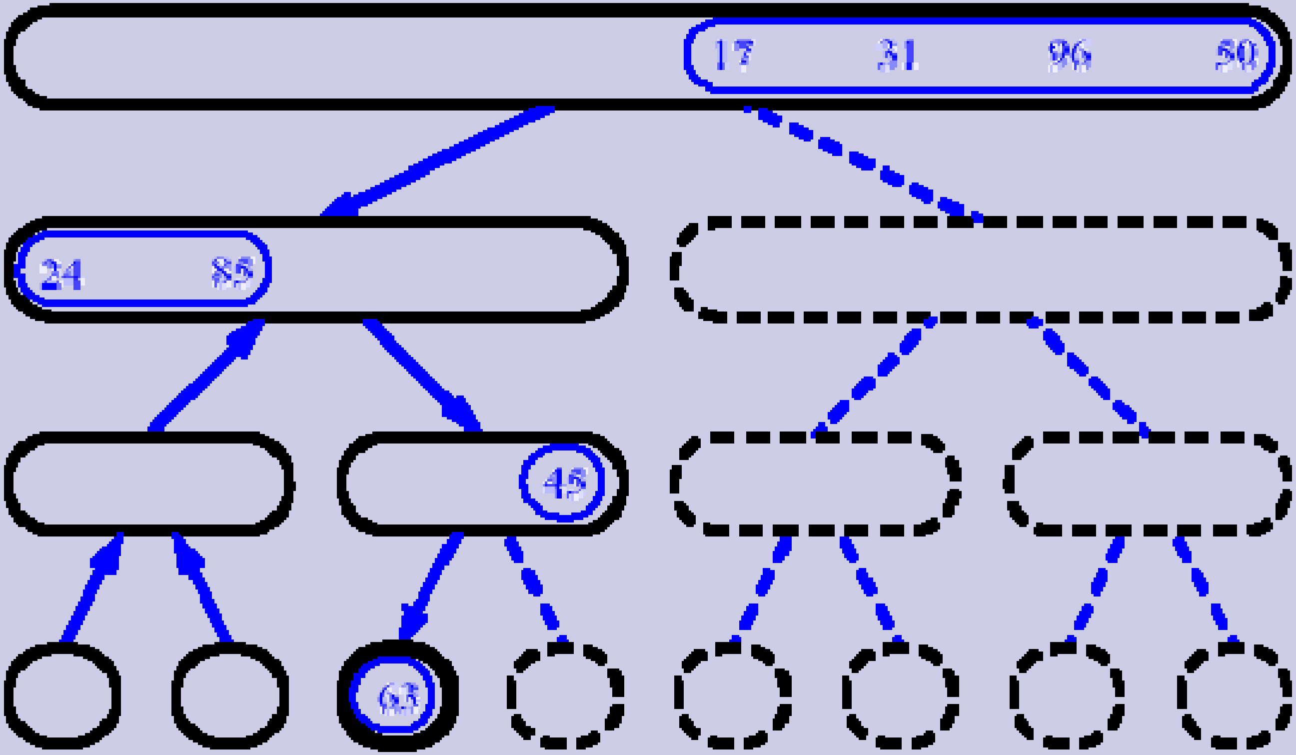
MergeSort (Example)



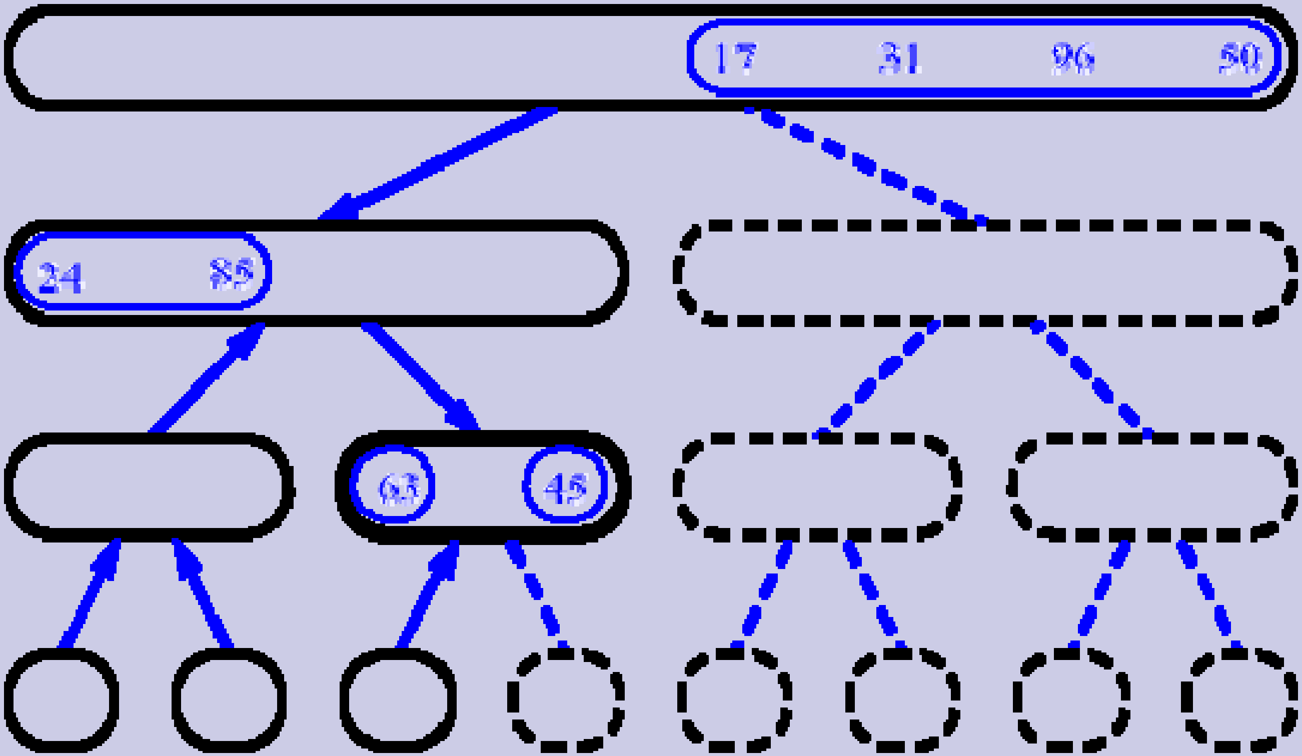
MergeSort (Example)



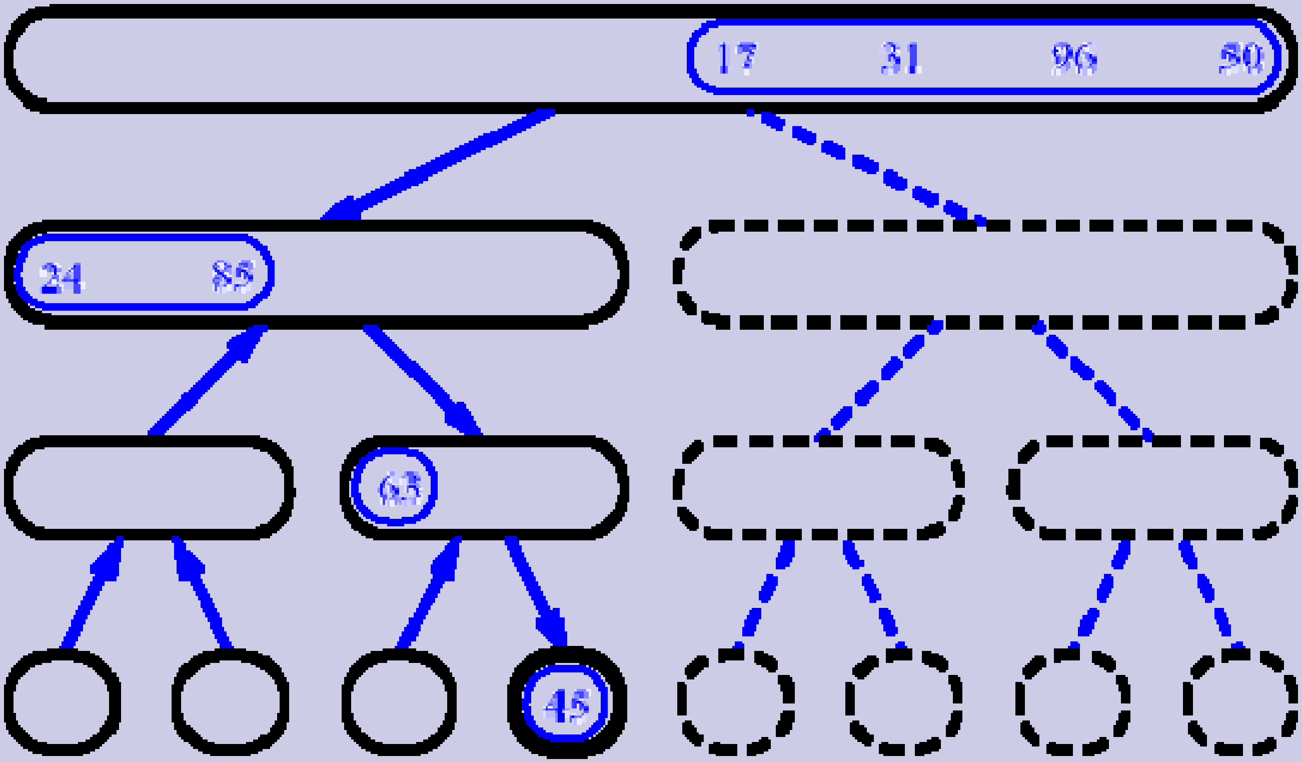
MergeSort (Example)



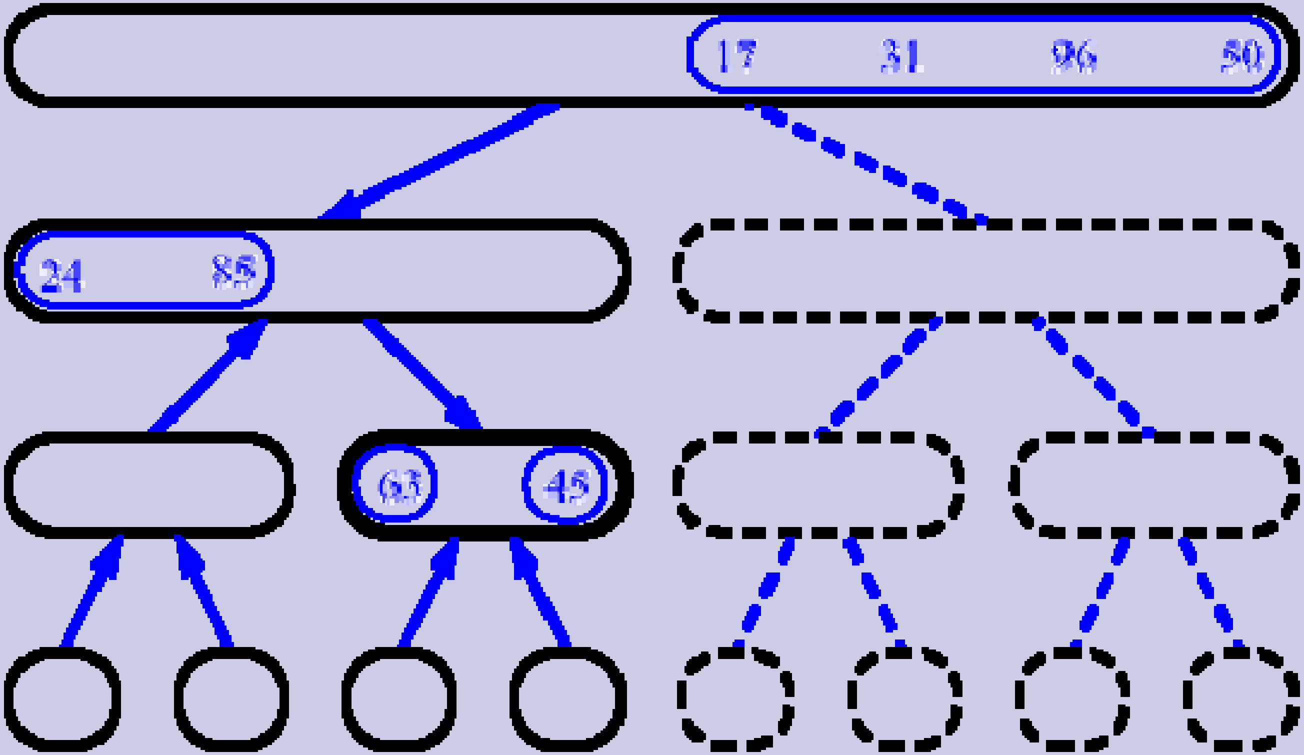
MergeSort (Example)



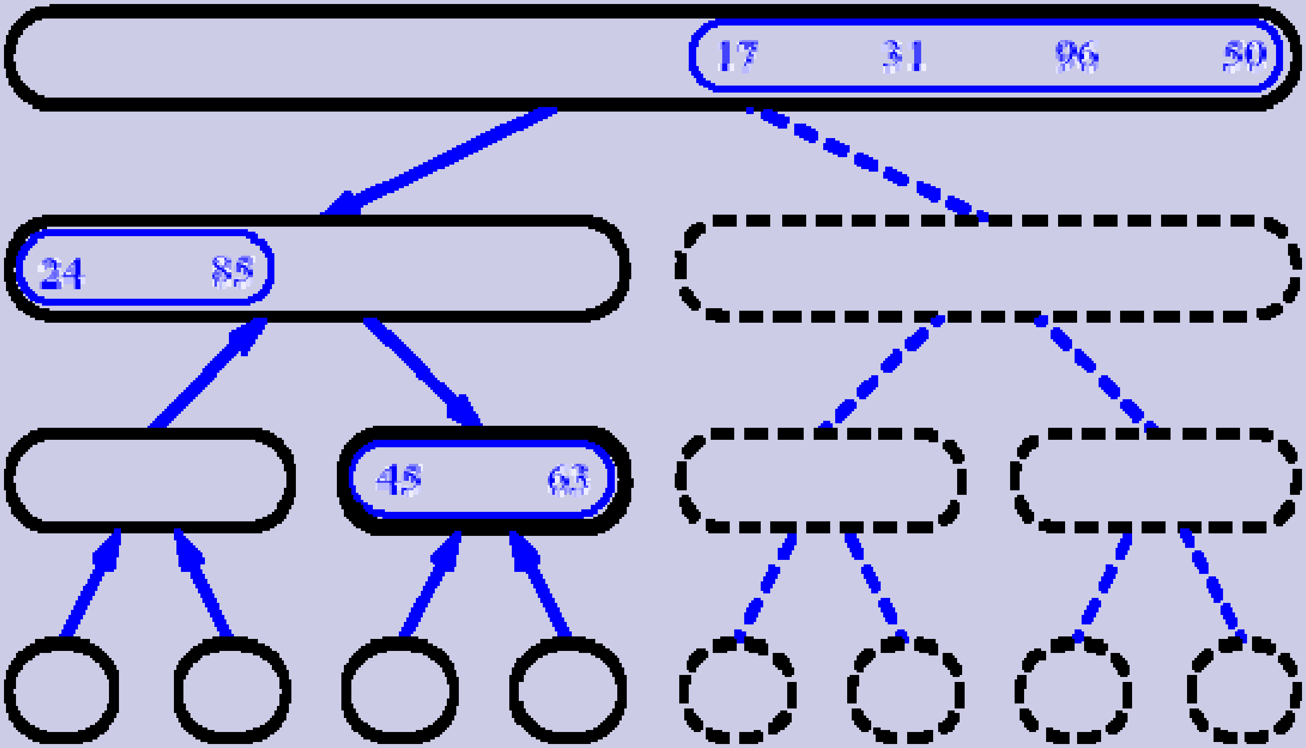
MergeSort (Example)



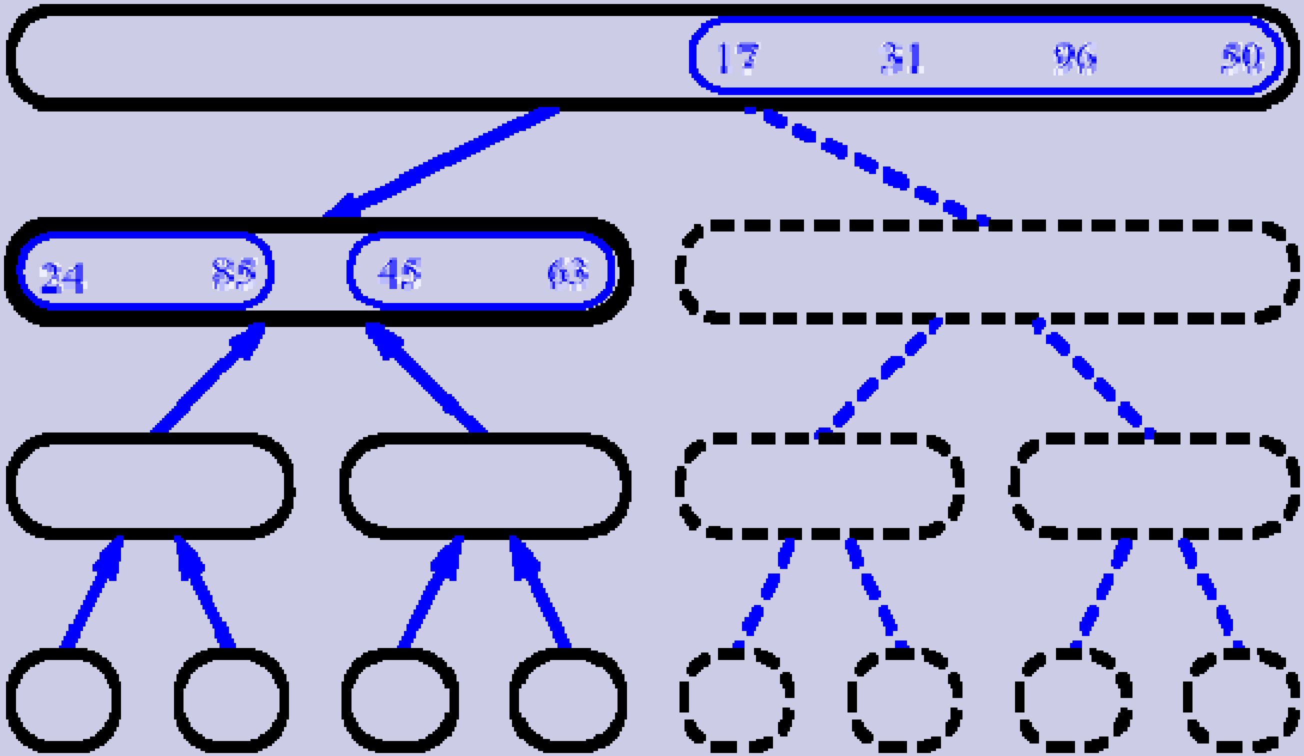
MergeSort (Example)



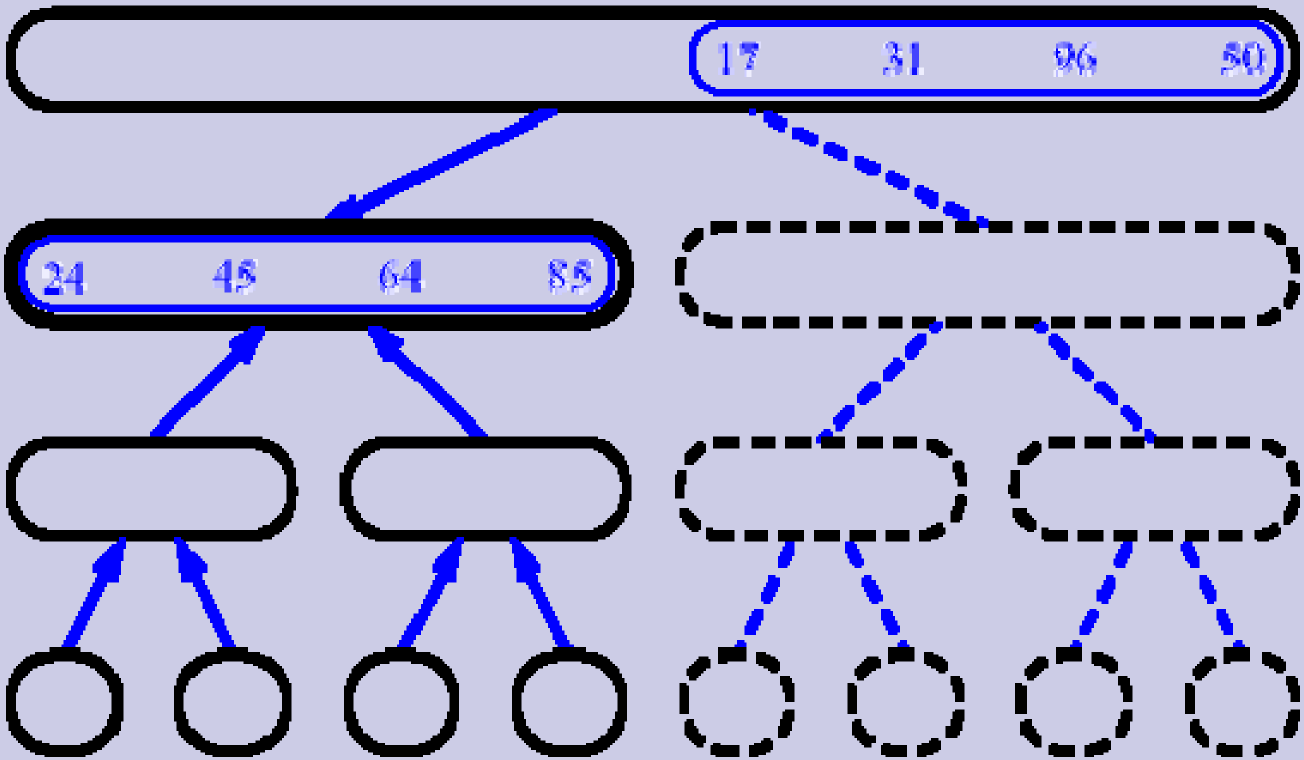
MergeSort (Example)



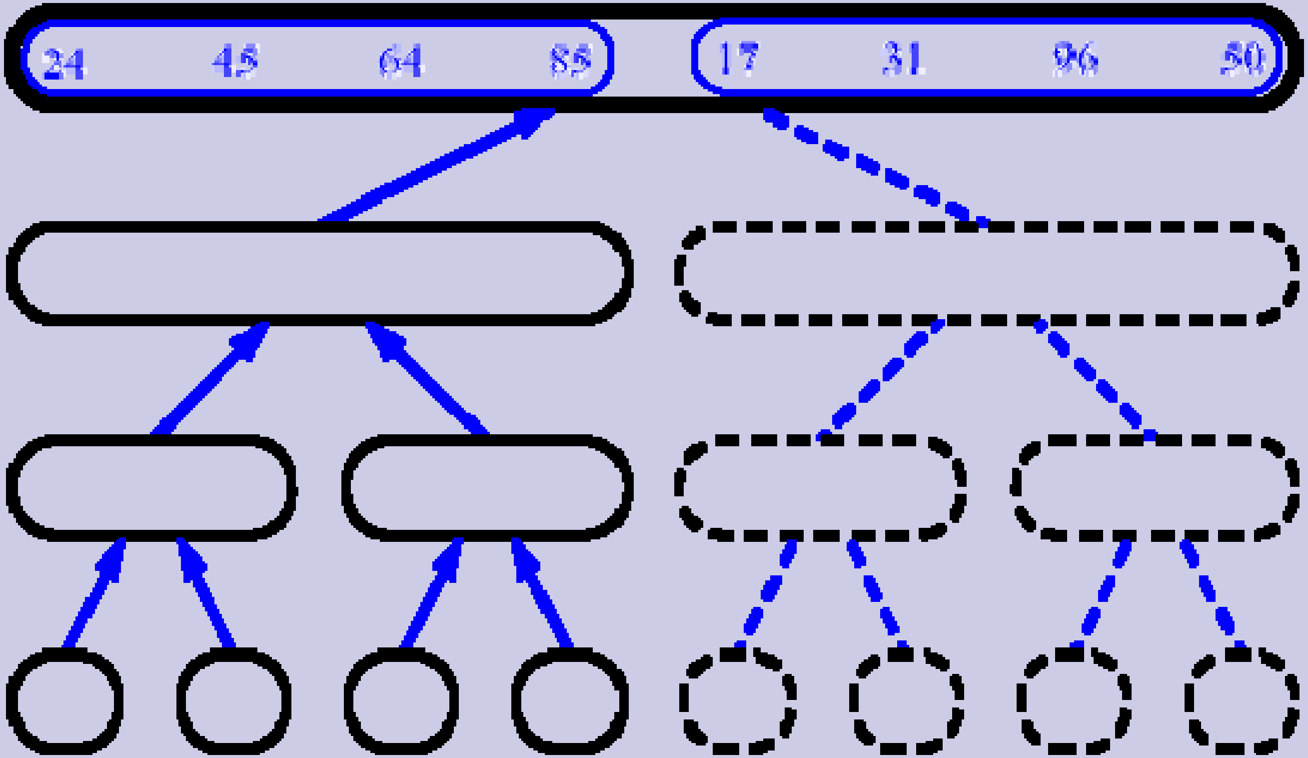
MergeSort (Example)



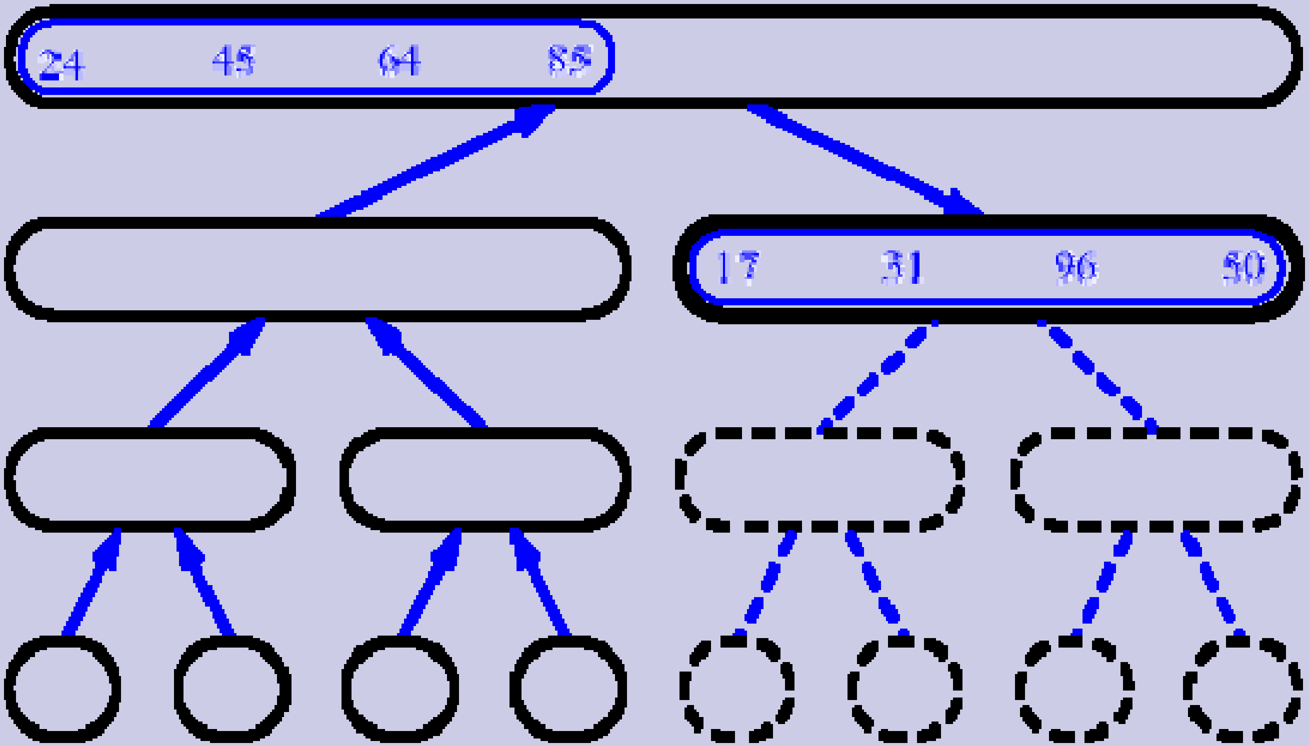
MergeSort (Example)



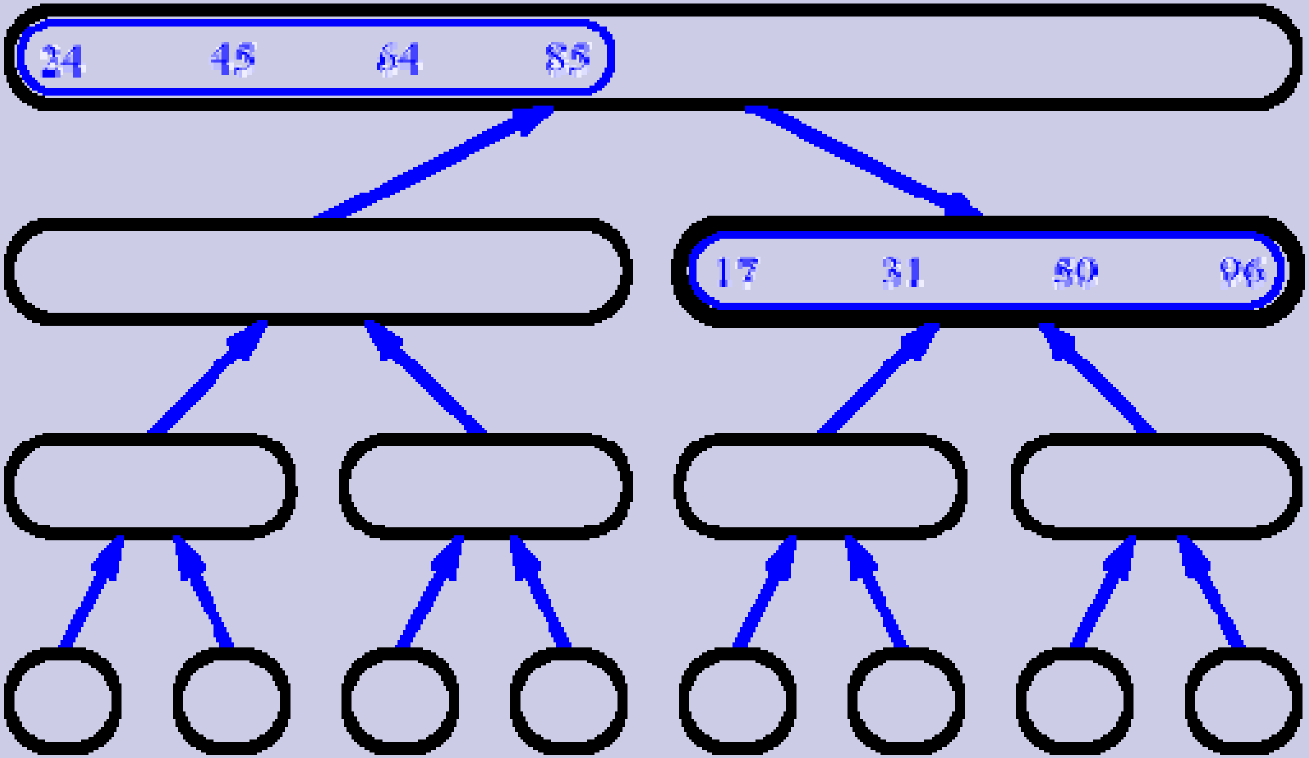
MergeSort (Example)



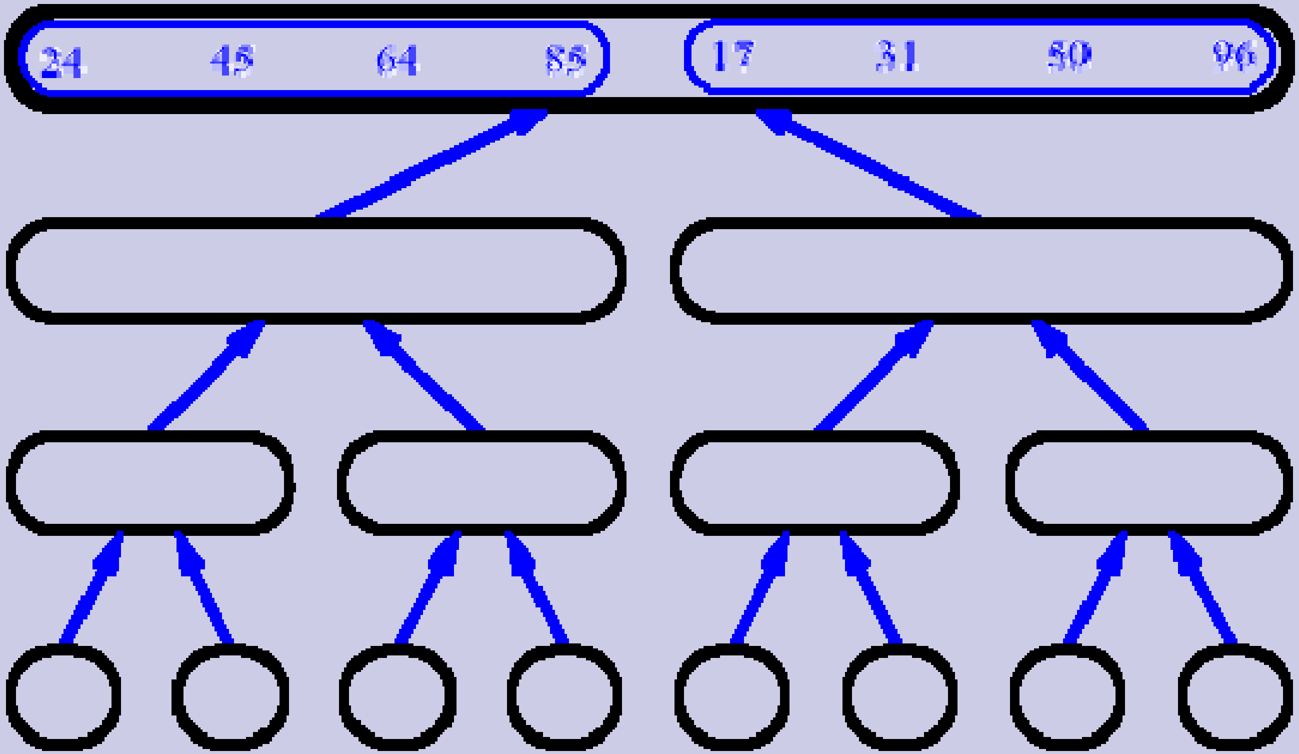
MergeSort (Example)



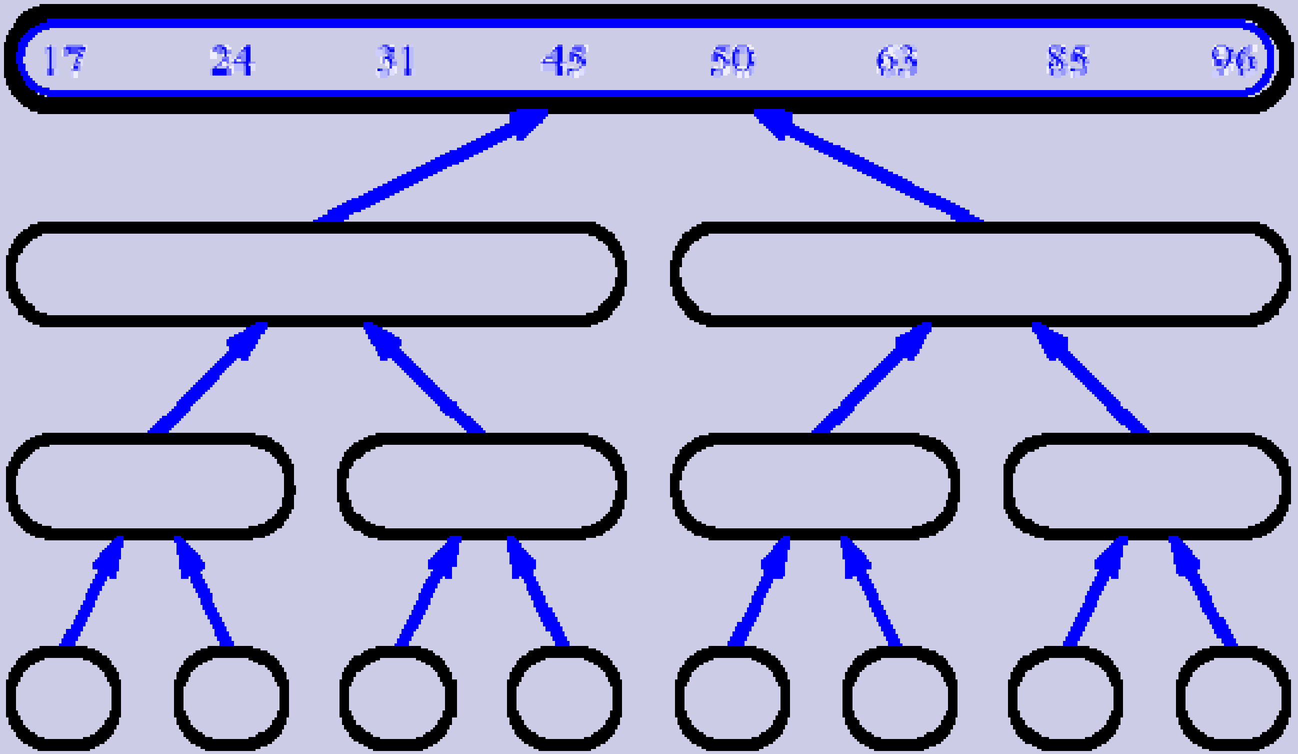
MergeSort (Example)



MergeSort (Example)



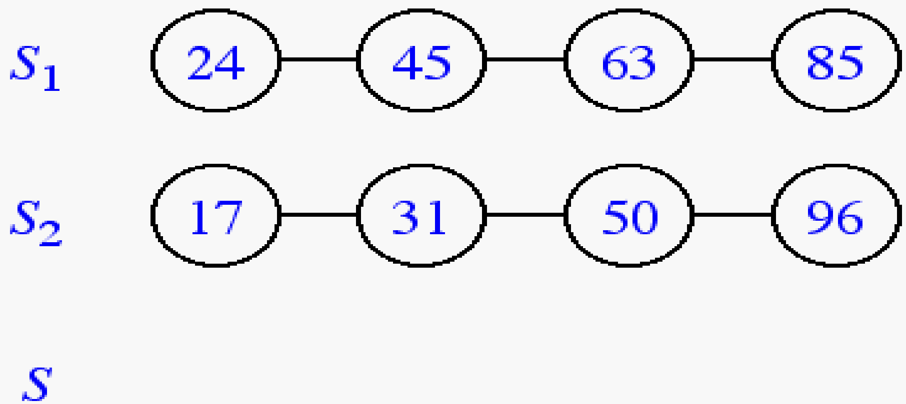
MergeSort (Example)



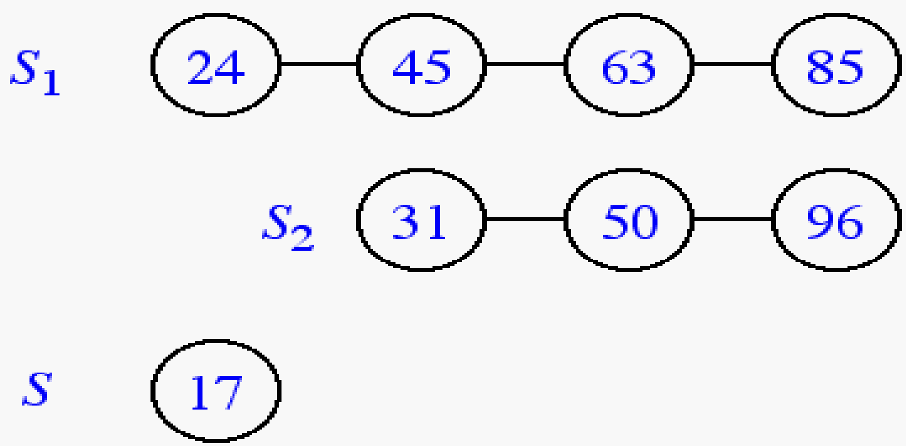
Merging Two Sequences (cont.)

- Some pictures:

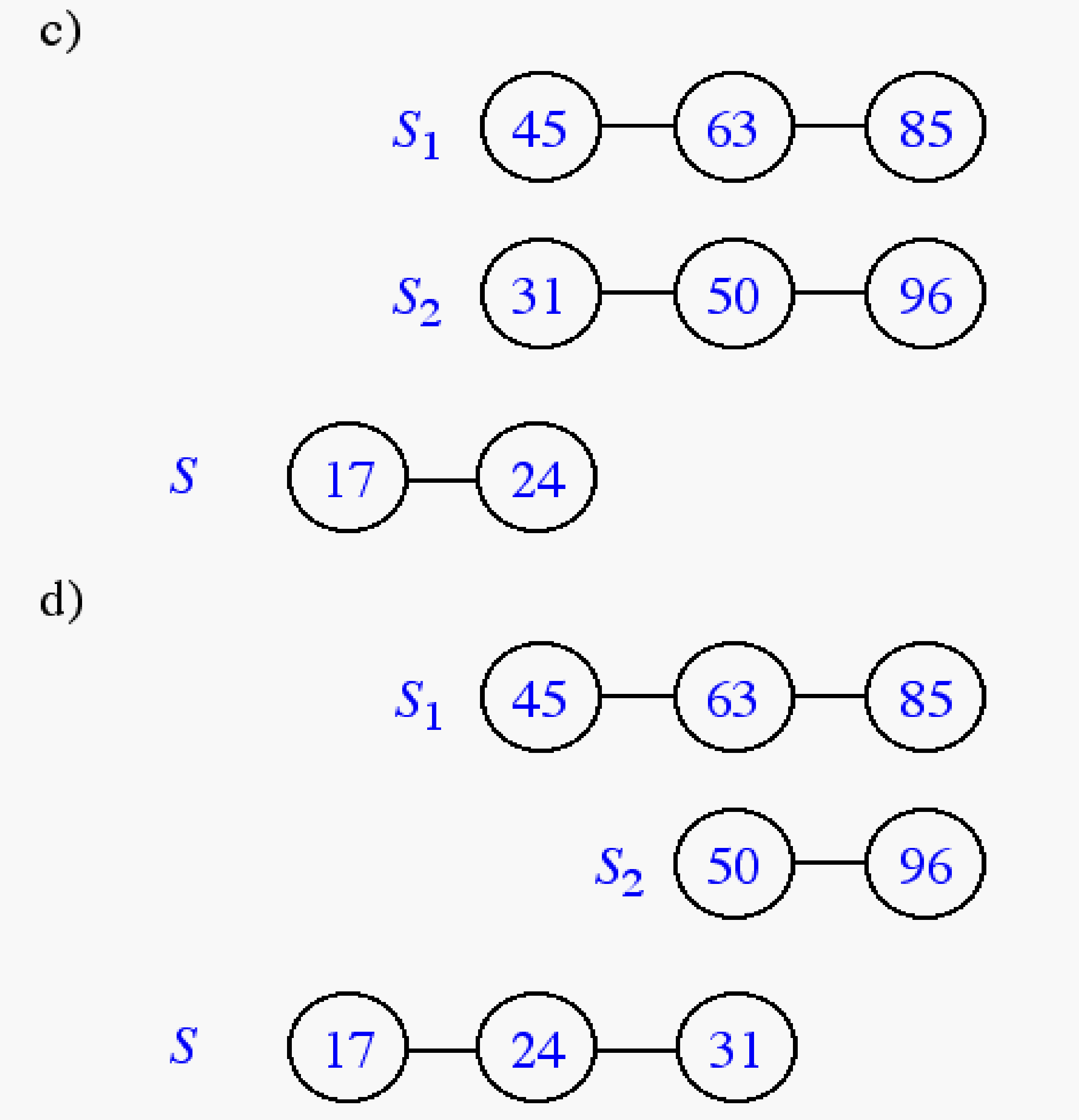
a)



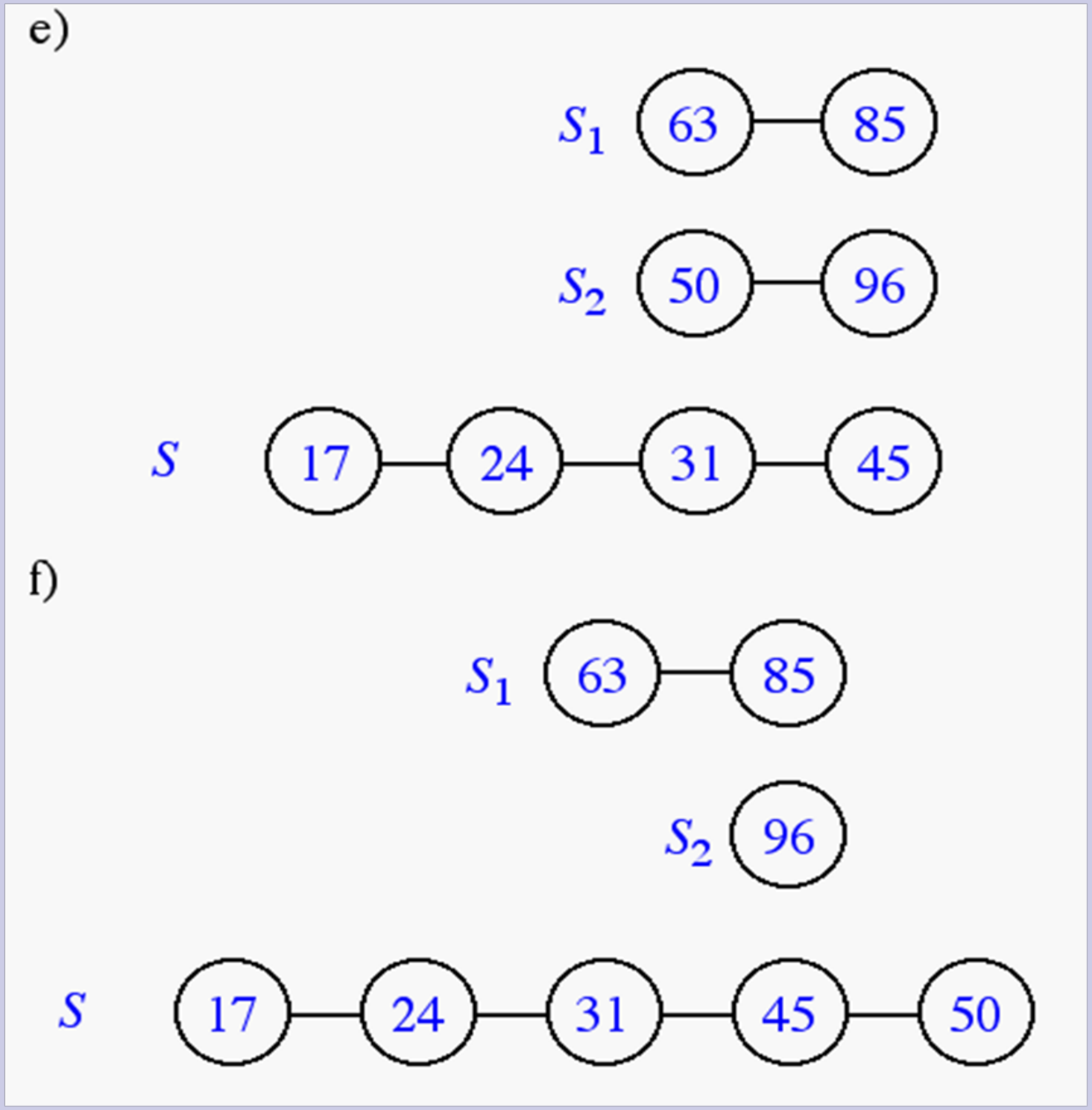
b)



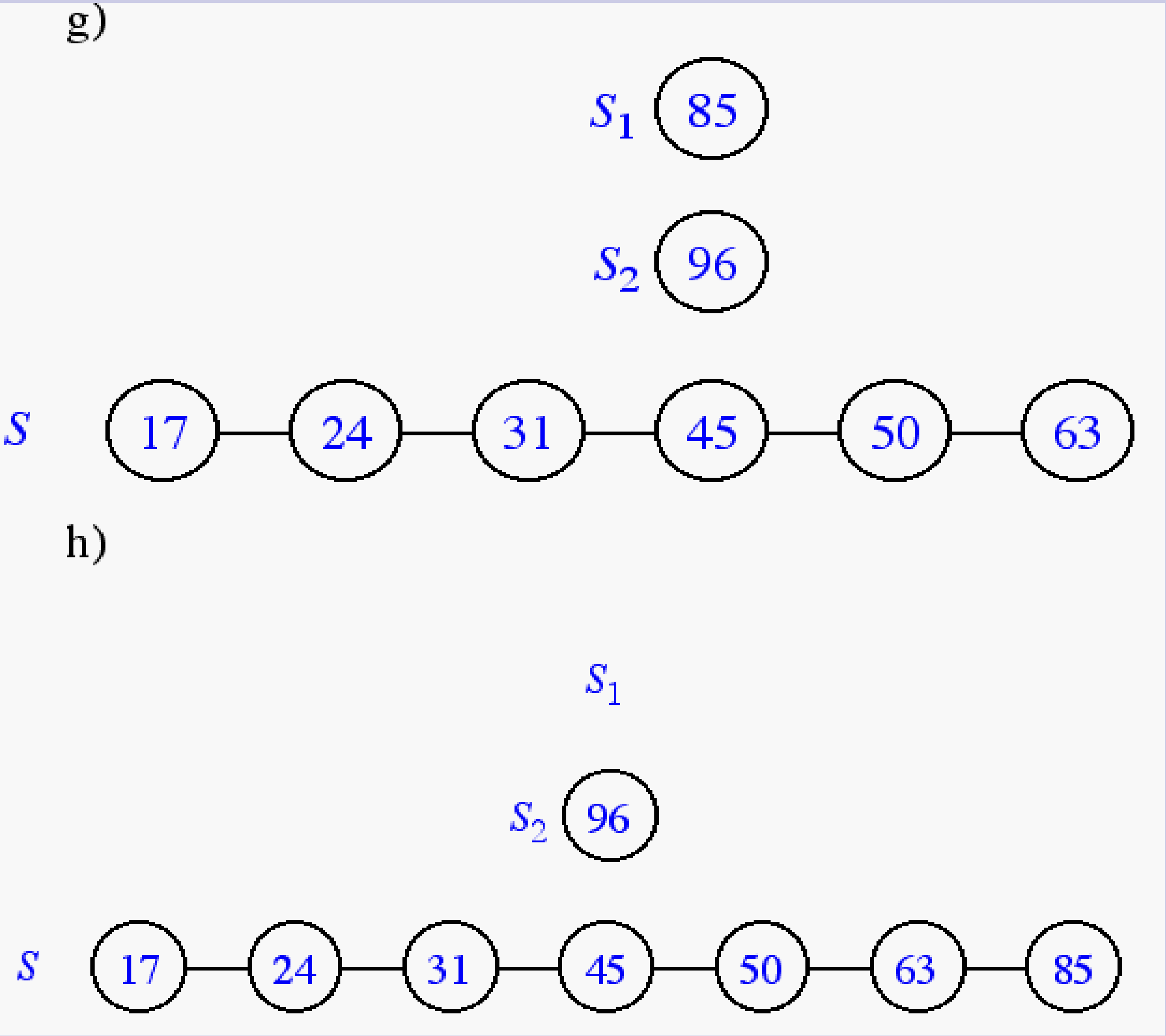
Merging Two Sequences (cont.)



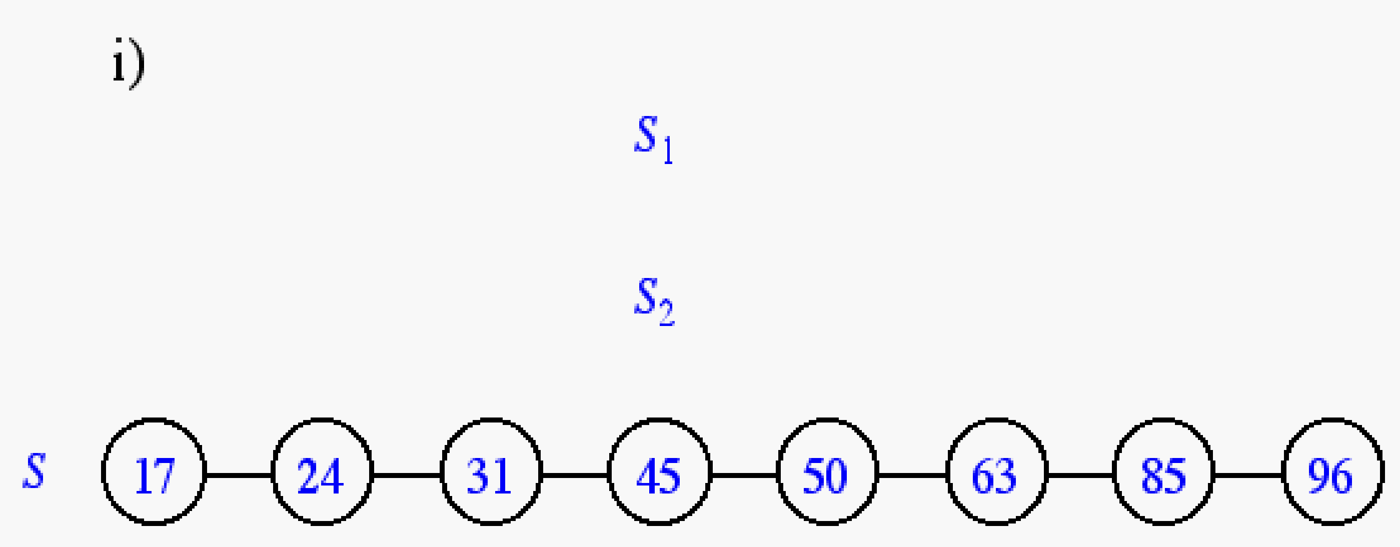
Merging Two Sequences (cont.)



Merging Two Sequences (cont.)

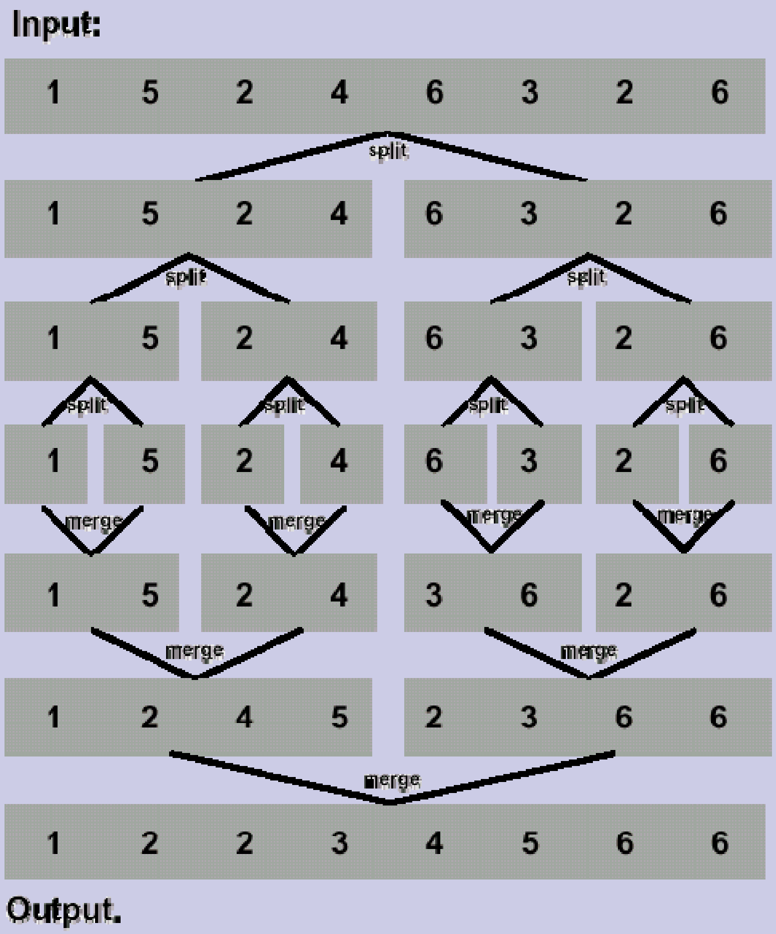


Merging Two Sequences (cont.)



Merge Sort Revisited

- To sort n numbers
  - if n=1 done!
  - recursively sort 2 lists of numbers $\lfloor n/2 \rfloor$ and $\lceil n/2 \rceil$ elements
  - merge 2 sorted lists in $\Theta(n)$ time
- Strategy
  - break problem into similar (smaller) subproblems
  - recursively solve subproblems
  - combine solutions to answer



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

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

- Example: Merge Sort

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

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).

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

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

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

Java Implementation of Merge-Sort

```
public interface SortObject {
  //sort sequence S in nondecreasing order
  using compartor c
  public void sort (Sequence S, Comparator c);
}
```

Java Implementation of MergeSort (cont.)

```
public class ListMergeSort implements SortObject {

public void sort(Sequence S, Comparator c) {
   int n = S.size();
   if (n < 2) return; //sequence with 0/1 element is sorted.
   // divide
   Sequence S1 = (Sequence)S.newContainer();
   // put the first half of S into S1
   for (int i=1; i <= (n+1)/2; i++) {
      S1.insertLast(S.remove(S.first()));
      }
   Sequence S2 = (Sequence)S.newContainer();
   // put the second half of S into S2
   for (int i=1; i <= n/2; i++) {
      S2.insertLast(S.remove(S.first()));
      }
   sort(S1,c); // recur
   sort(S2,c);
   merge(S1,S2,c,S); // conquer
  }
```

Java Implementation of MergeSort (cont.)

```
public void merge(Sequence S1, Sequence S2, Comparator c, Sequence S) {
        while(!S1.isEmpty() && !S2.isEmpty()) {
                if(c.isLessThanOrEqualTo(S1.first().element(),
                        S2.first().element())) {
                        // S1's 1st elt <= S2's 1st elt
                        S.insertLast(S1.remove(S1.first()));
                }else { // S2's 1st elt is the smaller one
                        S.insertLast(S2.remove(S2.first()));
                }
        }if(S1.isEmpty()) {
                while(!S2.isEmpty()) {
                        S.insertLast(S2.remove(S2.first()));
                }
        }if(S2.isEmpty()) {
                while(!S1.isEmpty()) {
                        S.insertLast(S1.remove(S1.first()));
                }}}
```