CST8130: Data Structures

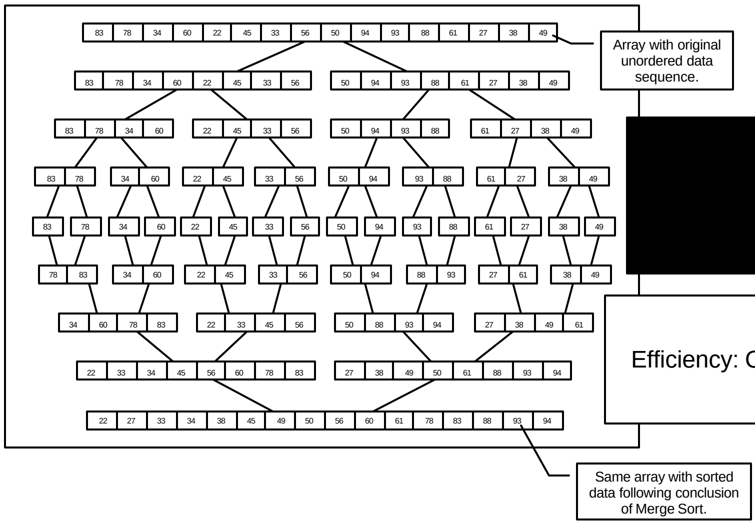
# *Merge Sort*

**Created by Rex Woollard**

**Use PageUp and PageDown to move from screen to screen. Online use arrow buttons.**

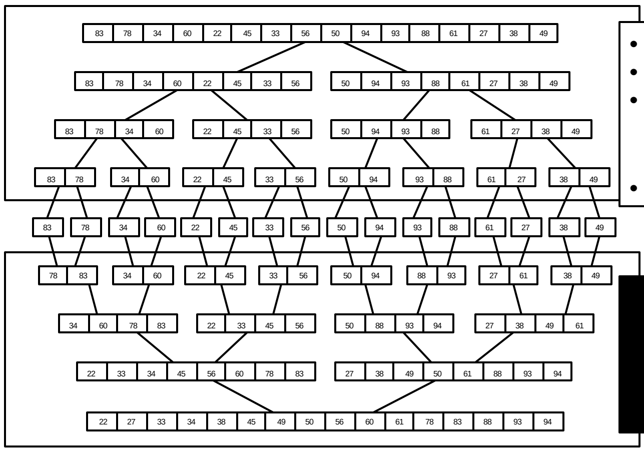
**Click on speaker to play sound.**

## *Overview: Processing*



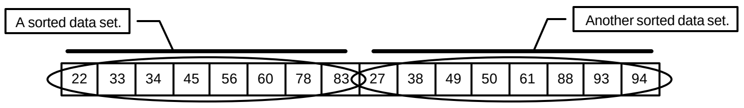
Efficiency: $O(n \log_2 n)$

## *Overview: Algorithm*



- *MergeSort()*
- Finds midpoint.
- Calls *recursively* for both sides of midpoint.
- *Merge()* result.

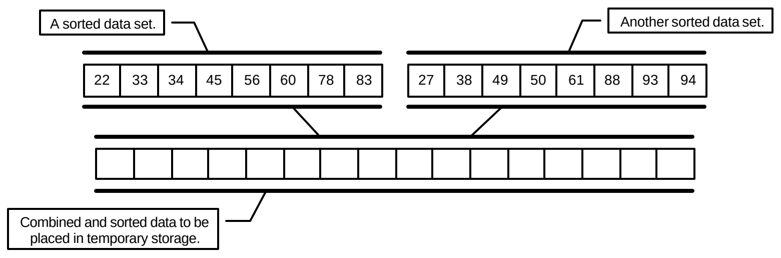
## *Merge: Overview*



Single array containing two subsets of data.

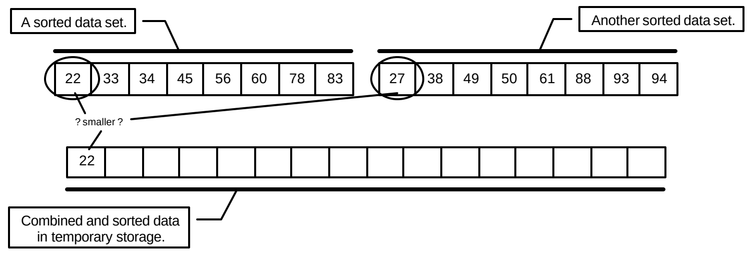
Each subset of data already sorted.

## Merge: Temporary Storage



Create the Temporary Storage

## Merge: Loop 1.1



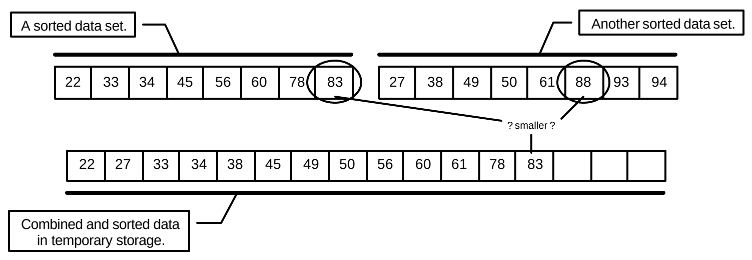
Loop 1: Compare and copy smaller item until one data set exhausted.

Loop 2: Copy remainder of left data set.

Loop 3: Copy remainder of right data set.

Loop 4: Copy data from temporary storage back to original location.

## *Merge: Loop 1.13*



Loop 1: Compare and copy smaller item until one data set exhausted.

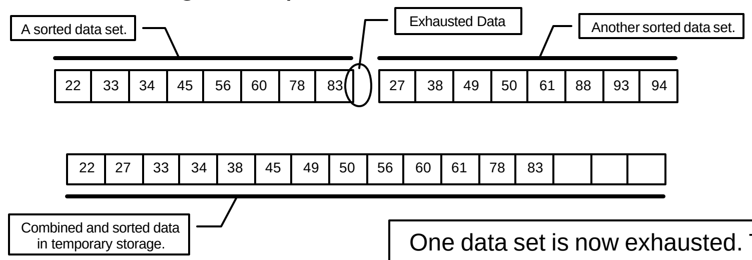
Loop 2: Copy remainder of left data set.

Loop 3: Copy remainder of right data set.

Loop 4: Copy data from temporary storage back to original location.

No sound object for this slide.

## *Merge: Loop 2: Finish Left Data Set*



One data set is now exhausted. Try loop 2, but there is nothing to do.

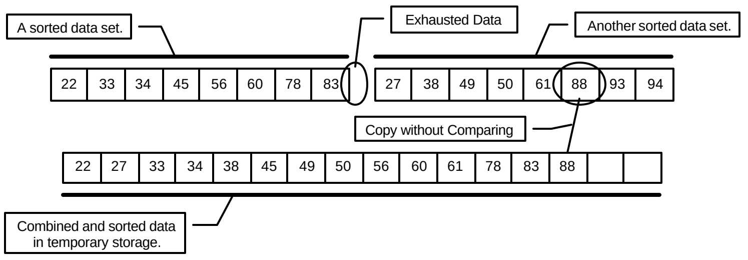
Loop 1: Compare and copy smaller item until one data set exhausted.

Loop 2: Copy remainder of left data set (but there is nothing to do).

Loop 3: Copy remainder of right data set.

Loop 4: Copy data from temporary storage back to original location.

## *Merge: Loop 3.1*



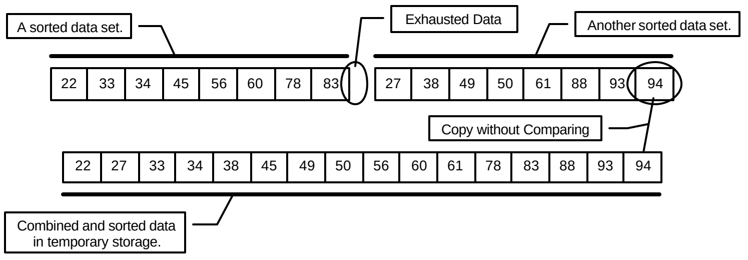
Loop 1: Compare and copy smaller item until one data set exhausted.

Loop 2: Copy remainder of left data set (but there is nothing to do).

Loop 3: Copy remainder of right data set (without comparing).

Loop 4: Copy data from temporary storage back to original location.

## *Merge: Loop 3.3*



Loop 1: Compare and copy smaller item until one data set exhausted.

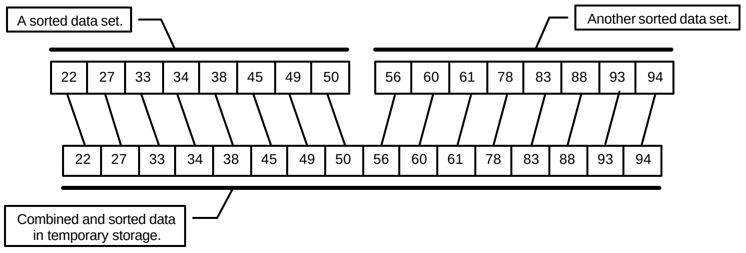
Loop 2: Copy remainder of left data set.

Loop 3: Copy remainder of right data set (without comparing).

Loop 4: Copy data from temporary storage back to original location.

No sound object for this slide.

## *Merge: Loop 4: Copy Data Back*



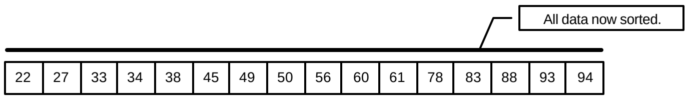
Loop 1: Compare and copy smaller item until one data set exhausted.

Loop 2: Copy remainder of left data set.

Loop 3: Copy remainder of right data set.

Loop 4: Copy data from temporary storage back to original location.

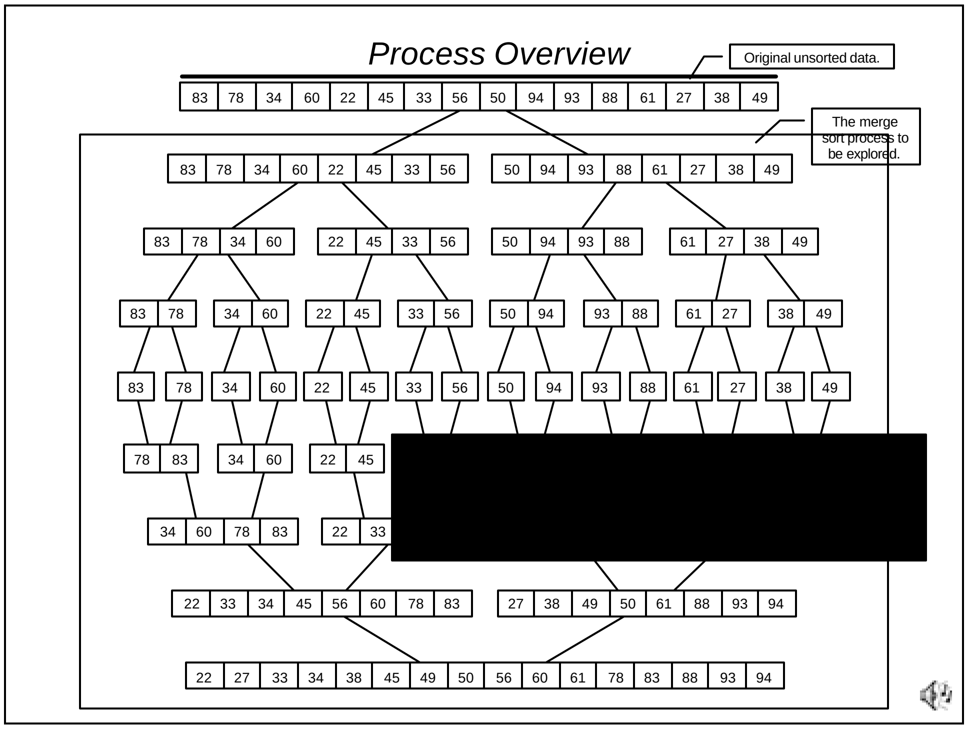
## *Merge: Free Temporary Storage*



Free the Temporary Storage

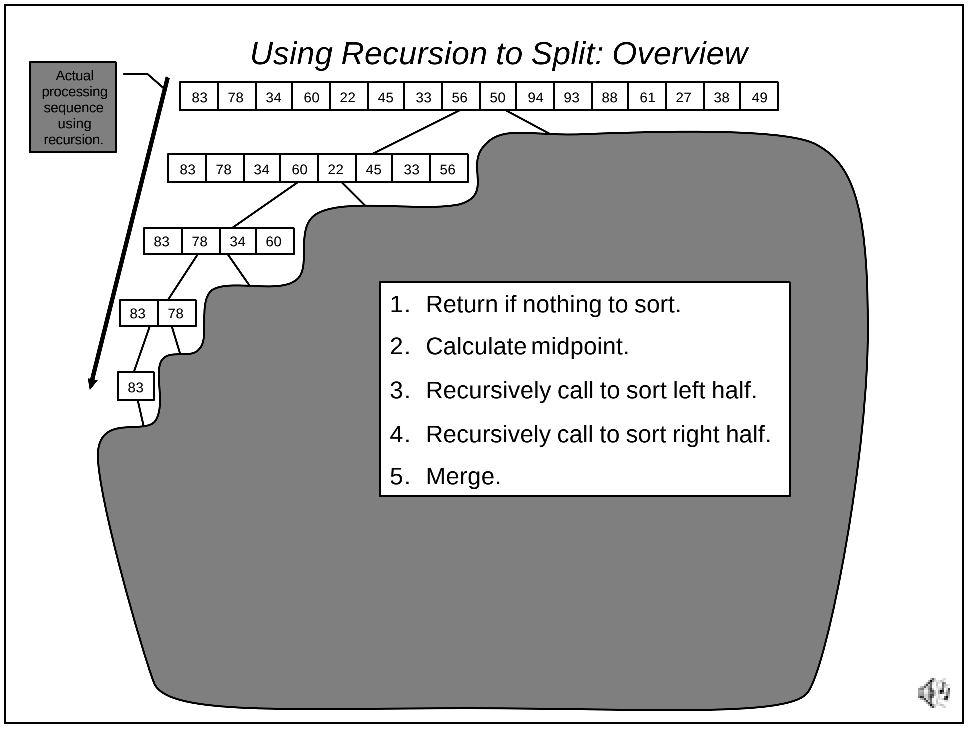
Next apply *Merge()* as part of *MergeSort()*

## Process Overview



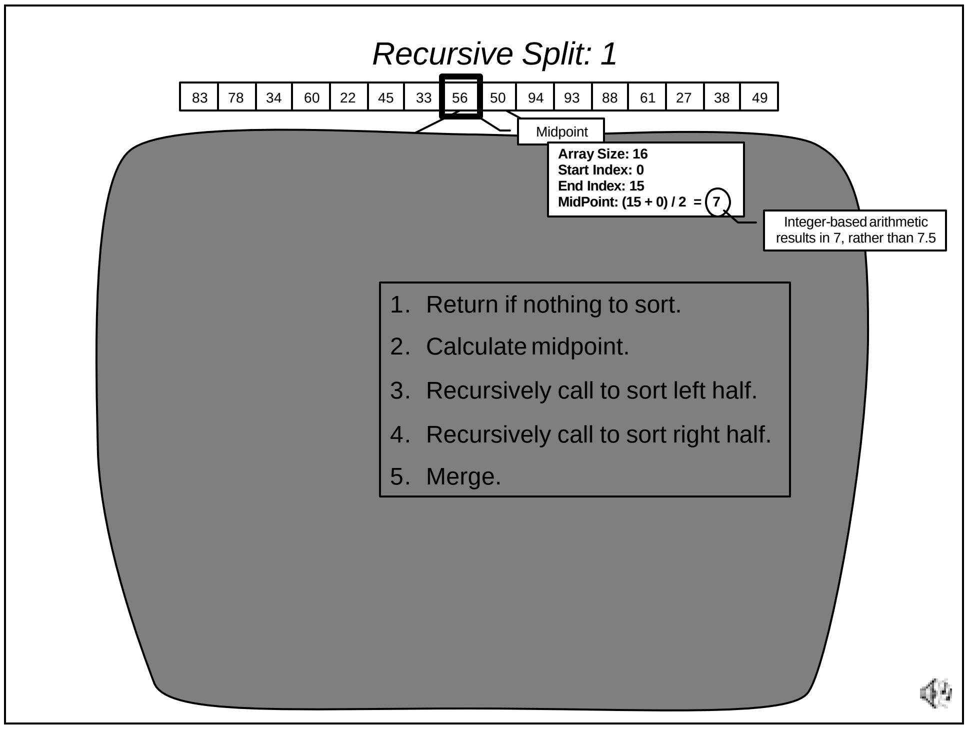
[redacted]

## Using Recursion to Split: Overview



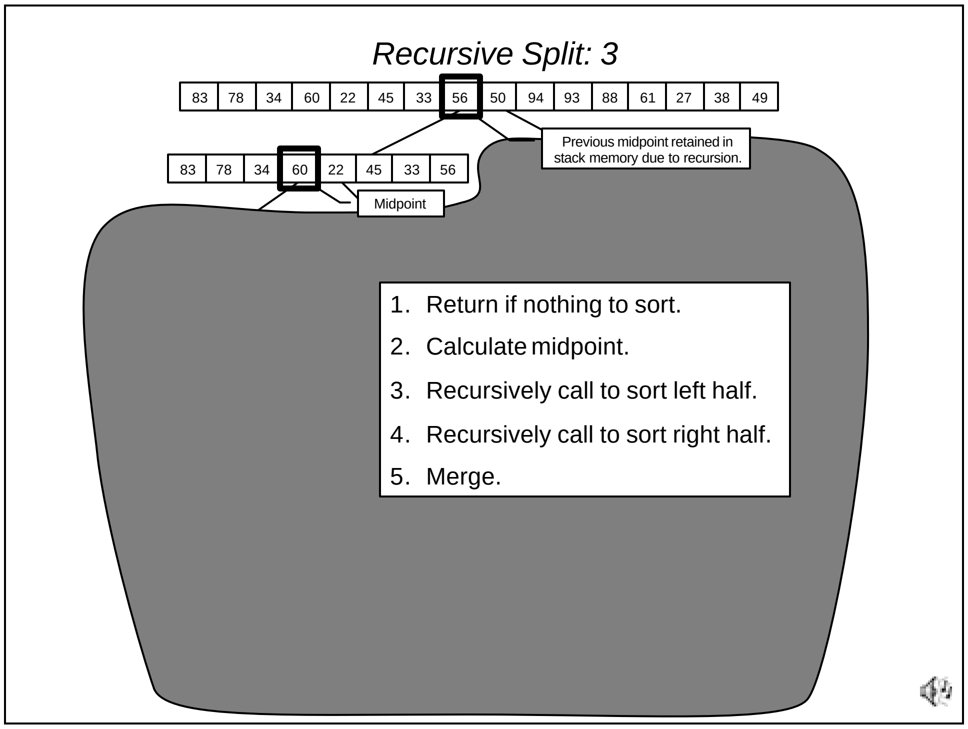
1. Return if nothing to sort.
2. Calculate midpoint.
3. Recursively call to sort left half.
4. Recursively call to sort right half.
5. Merge.

## Recursive Split: 1



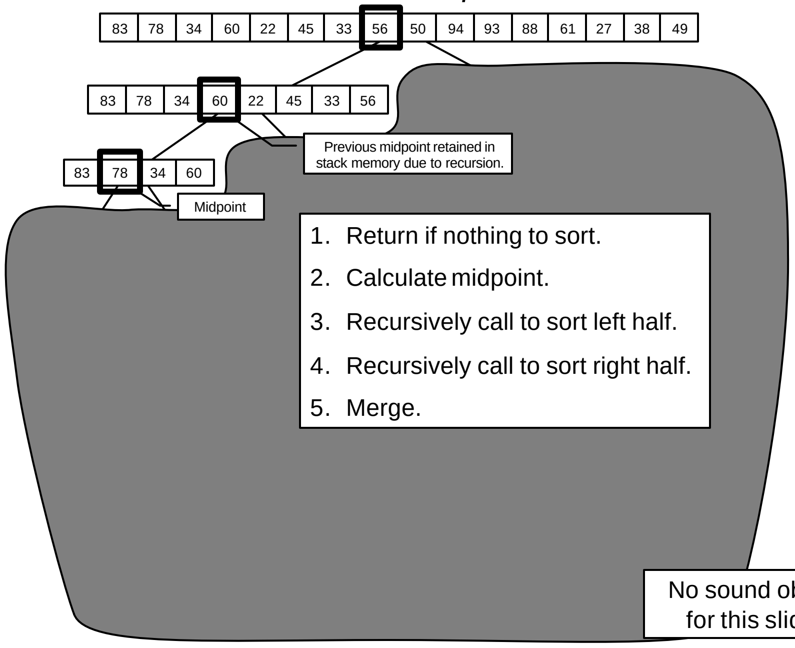
1. Return if nothing to sort.
2. Calculate midpoint.
3. Recursively call to sort left half.
4. Recursively call to sort right half.
5. Merge.

## Recursive Split: 3



1. Return if nothing to sort.
2. Calculate midpoint.
3. Recursively call to sort left half.
4. Recursively call to sort right half.
5. Merge.

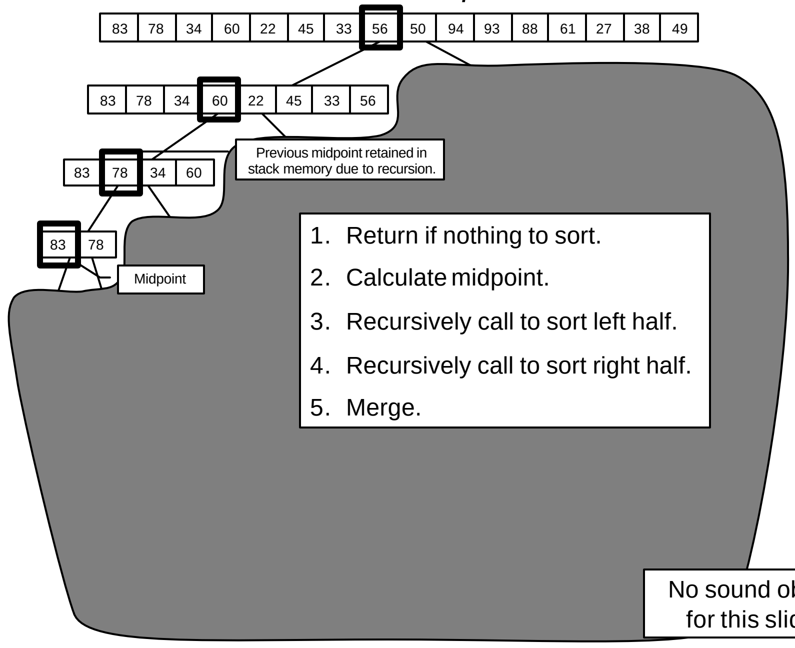
## *Recursive Split: 5*



1. Return if nothing to sort.
2. Calculate midpoint.
3. Recursively call to sort left half.
4. Recursively call to sort right half.
5. Merge.

No sound object for this slide.

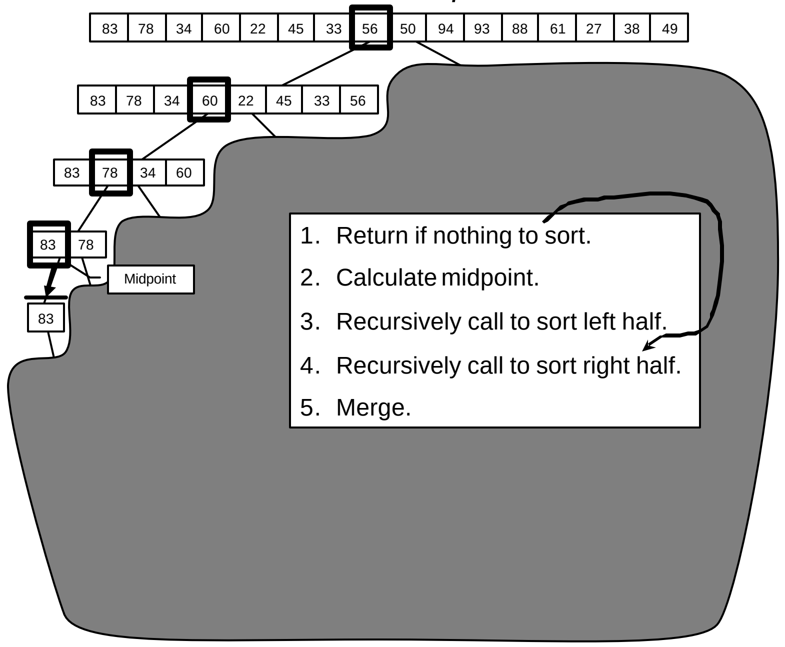
## *Recursive Split: 7*



1. Return if nothing to sort.
2. Calculate midpoint.
3. Recursively call to sort left half.
4. Recursively call to sort right half.
5. Merge.

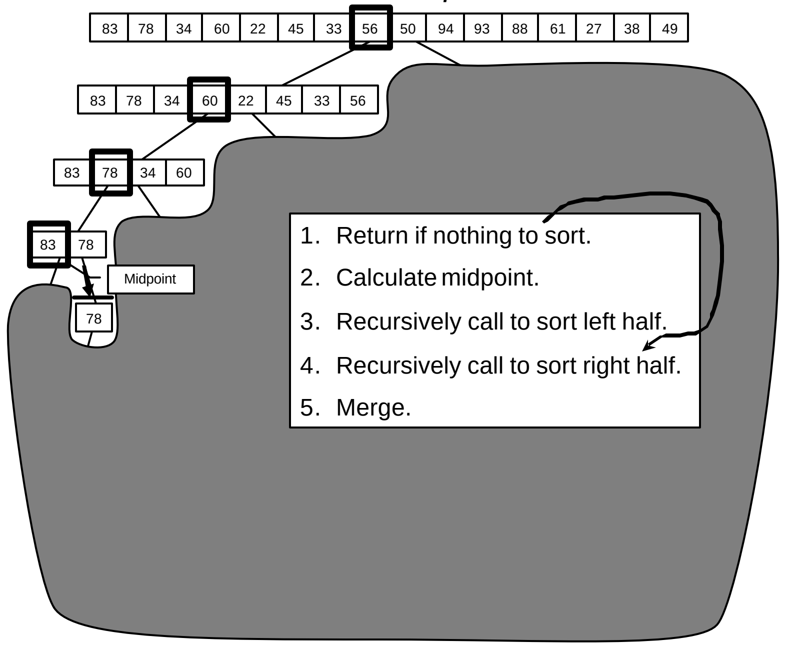
No sound object for this slide.

## *Recursive Split: 9*



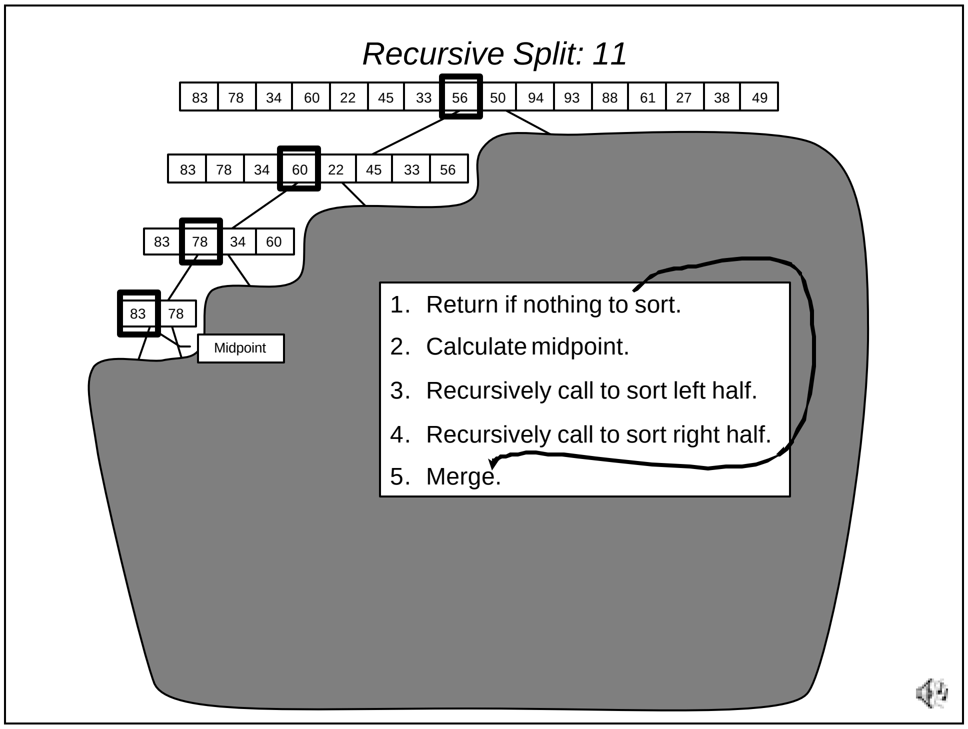
1. Return if nothing to sort.
2. Calculate midpoint.
3. Recursively call to sort left half.
4. Recursively call to sort right half.
5. Merge.

## *Recursive Split: 10*



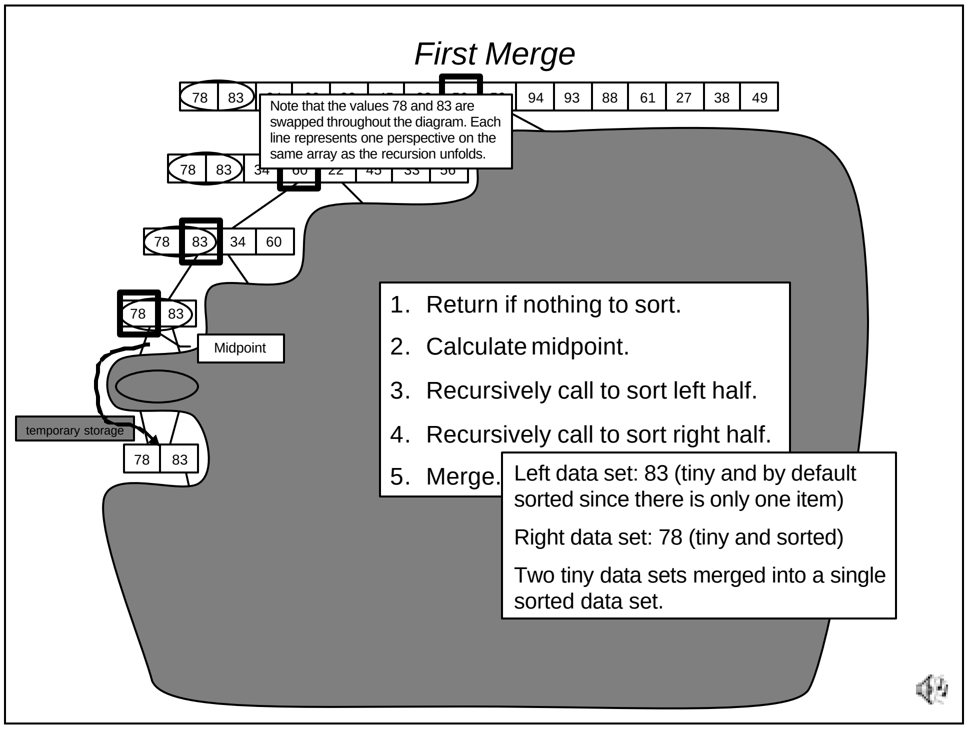
1. Return if nothing to sort.
2. Calculate midpoint.
3. Recursively call to sort left half.
4. Recursively call to sort right half.
5. Merge.

## *Recursive Split: 11*



1. Return if nothing to sort.
2. Calculate midpoint.
3. Recursively call to sort left half.
4. Recursively call to sort right half.
5. Merge.

## *First Merge*



Note that the values 78 and 83 are swapped throughout the diagram. Each line represents one perspective on the same array as the recursion unfolds.

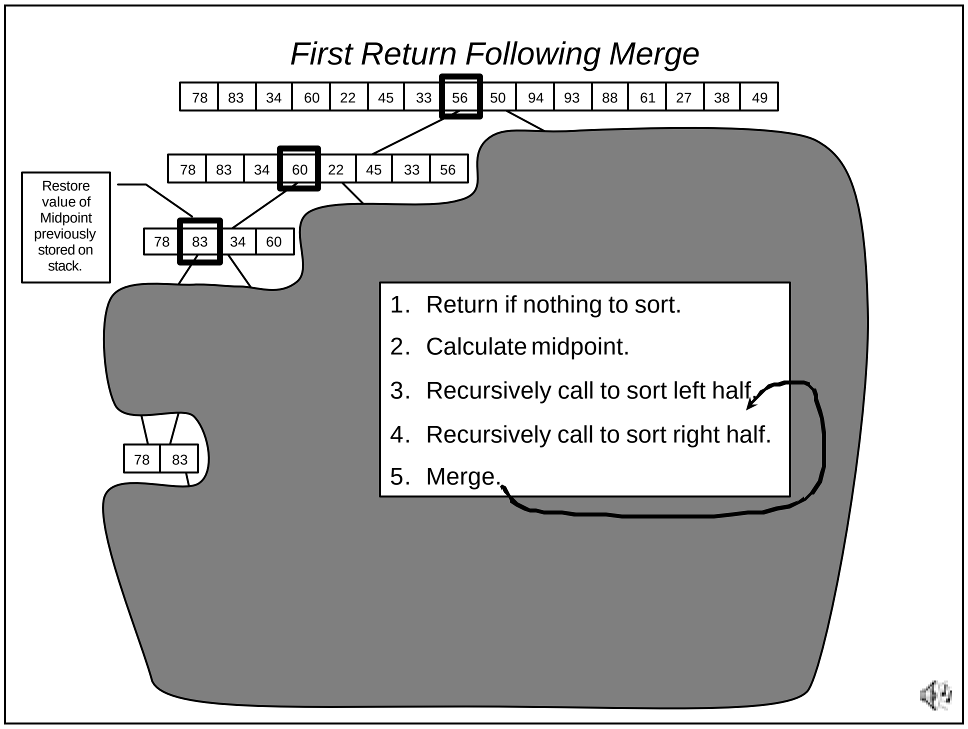
1. Return if nothing to sort.
2. Calculate midpoint.
3. Recursively call to sort left half.
4. Recursively call to sort right half.
5. Merge.

Left data set: 83 (tiny and by default sorted since there is only one item)

Right data set: 78 (tiny and sorted)

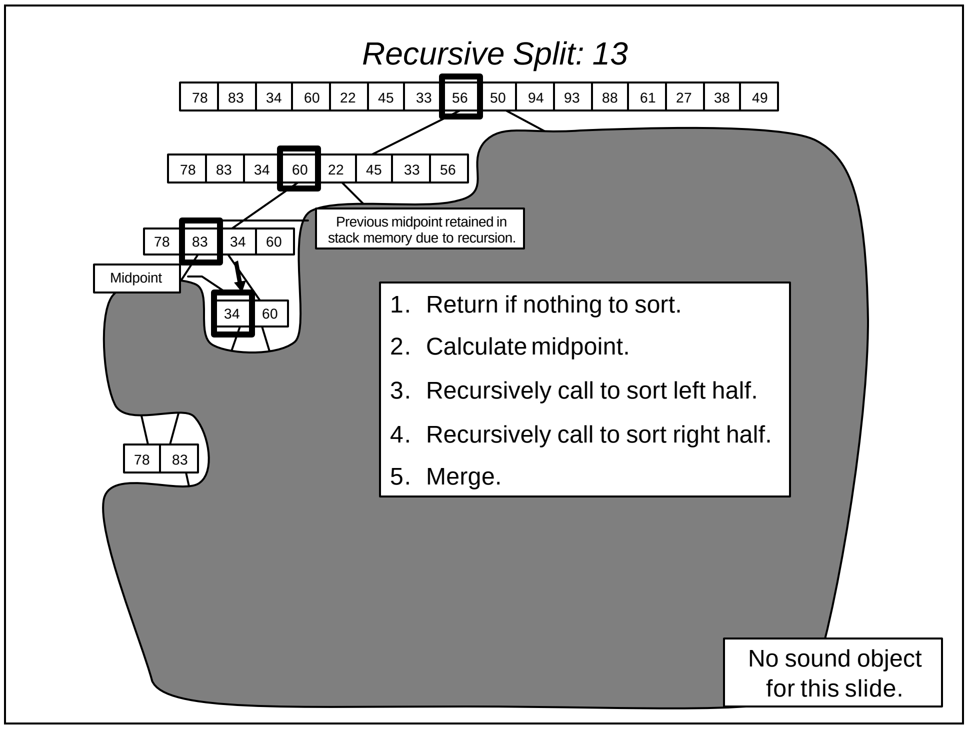
Two tiny data sets merged into a single sorted data set.

## *First Return Following Merge*



1. Return if nothing to sort.
2. Calculate midpoint.
3. Recursively call to sort left half.
4. Recursively call to sort right half.
5. Merge.

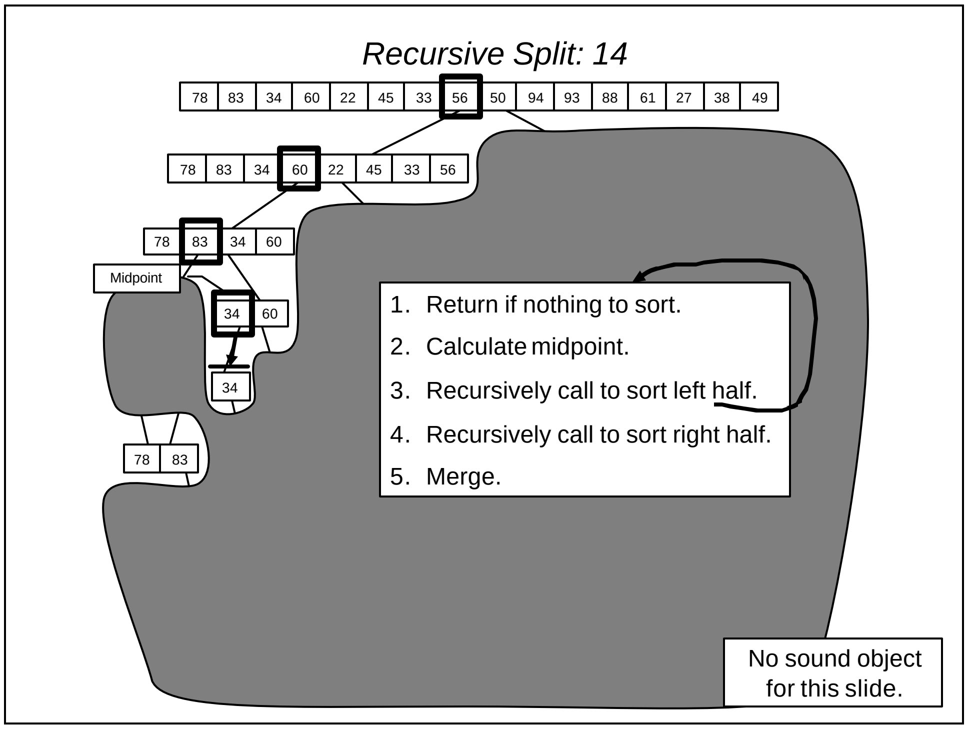
## *Recursive Split: 13*



1. Return if nothing to sort.
2. Calculate midpoint.
3. Recursively call to sort left half.
4. Recursively call to sort right half.
5. Merge.

No sound object for this slide.

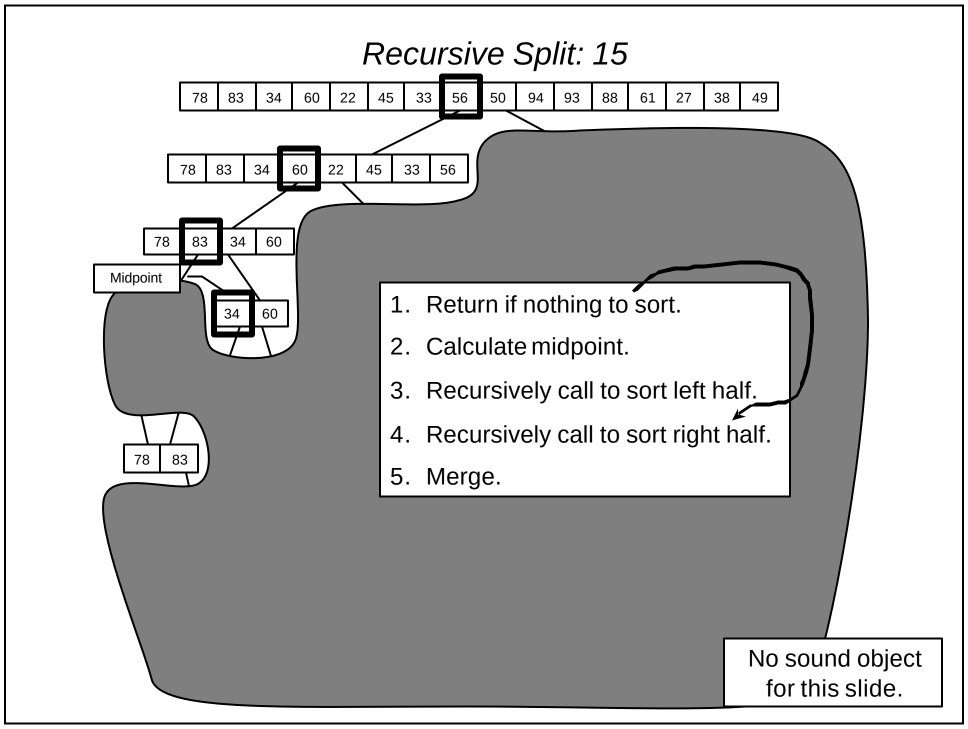
## *Recursive Split: 14*



1. Return if nothing to sort.
2. Calculate midpoint.
3. Recursively call to sort left half.
4. Recursively call to sort right half.
5. Merge.

No sound object for this slide.

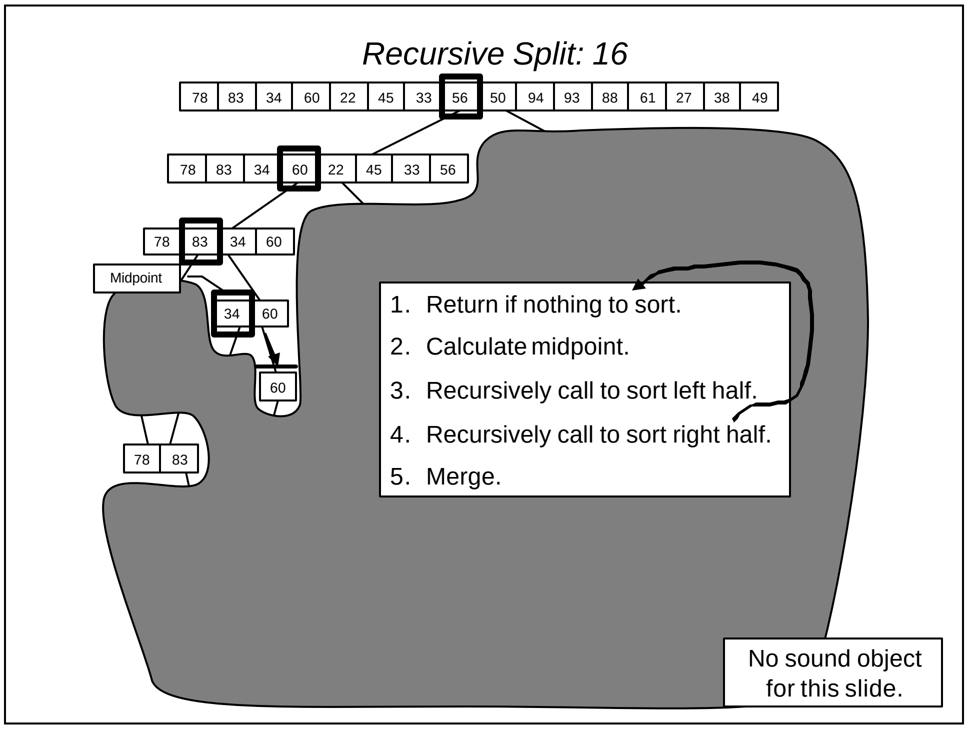
## *Recursive Split: 15*



1. Return if nothing to sort.
2. Calculate midpoint.
3. Recursively call to sort left half.
4. Recursively call to sort right half.
5. Merge.

No sound object for this slide.

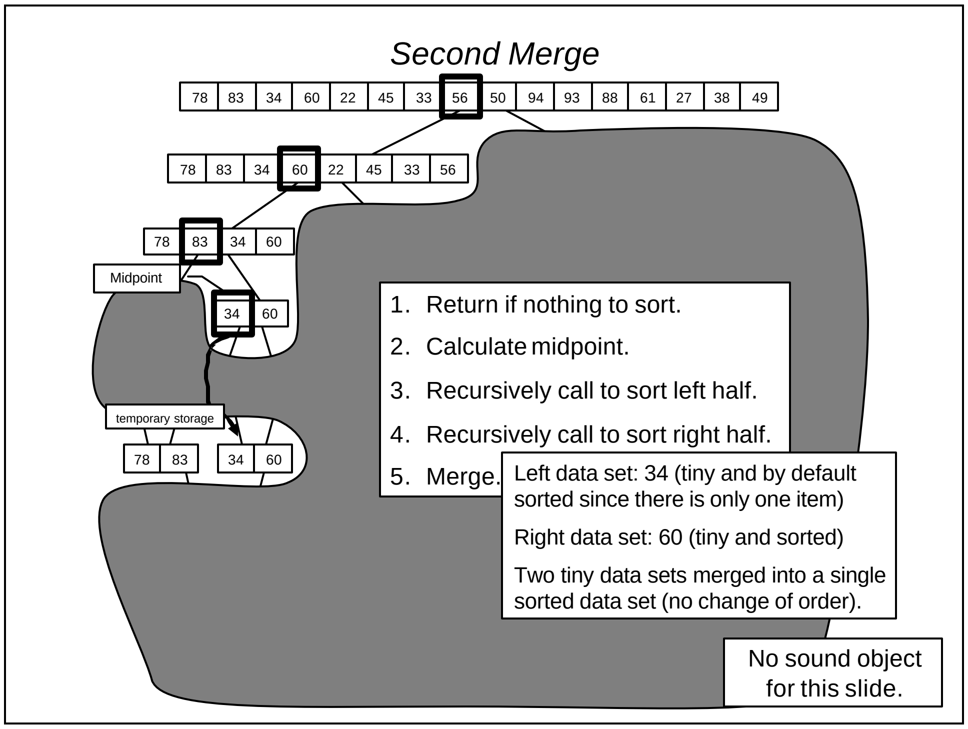
## *Recursive Split: 16*



1. Return if nothing to sort.
2. Calculate midpoint.
3. Recursively call to sort left half.
4. Recursively call to sort right half.
5. Merge.

No sound object for this slide.

## *Second Merge*



1. Return if nothing to sort.
2. Calculate midpoint.
3. Recursively call to sort left half.
4. Recursively call to sort right half.
5. Merge.

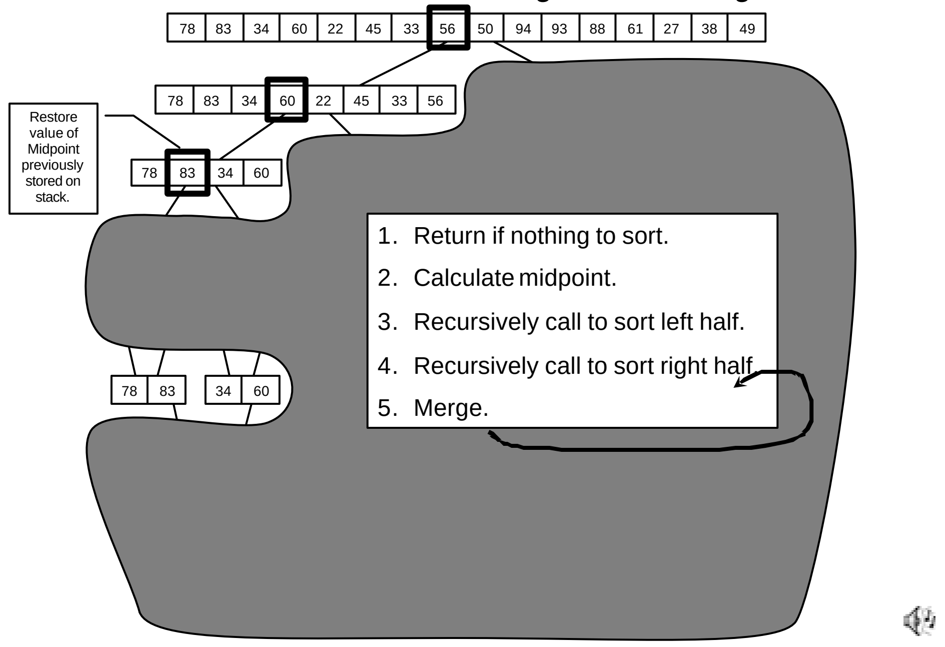
Left data set: 34 (tiny and by default sorted since there is only one item)

Right data set: 60 (tiny and sorted)

Two tiny data sets merged into a single sorted data set (no change of order).

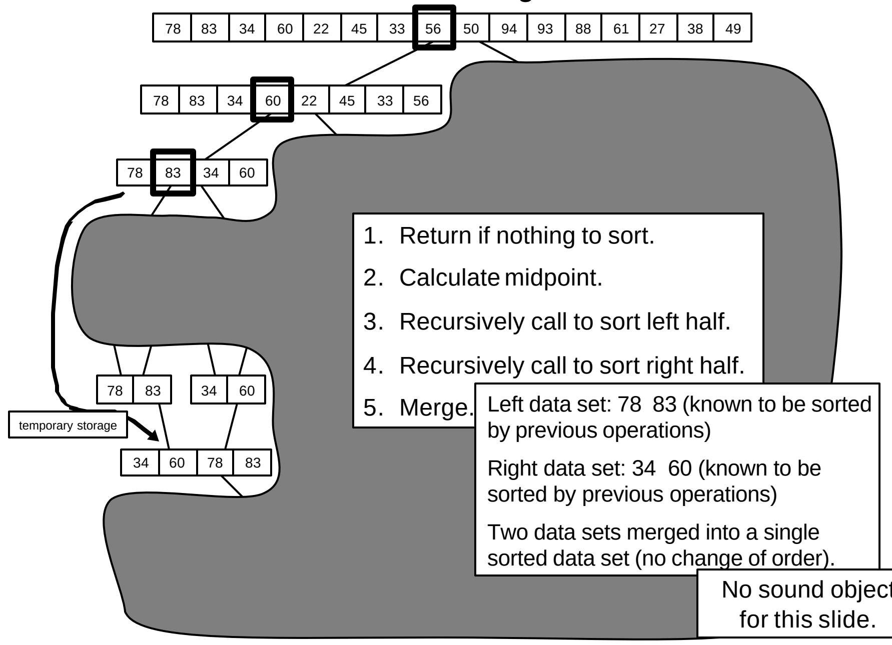
No sound object for this slide.

## *Second Return Following Second Merge*



1. Return if nothing to sort.
2. Calculate midpoint.
3. Recursively call to sort left half.
4. Recursively call to sort right half.
5. Merge.

## *Third Merge*



1. Return if nothing to sort.
2. Calculate midpoint.
3. Recursively call to sort left half.
4. Recursively call to sort right half.
5. Merge.

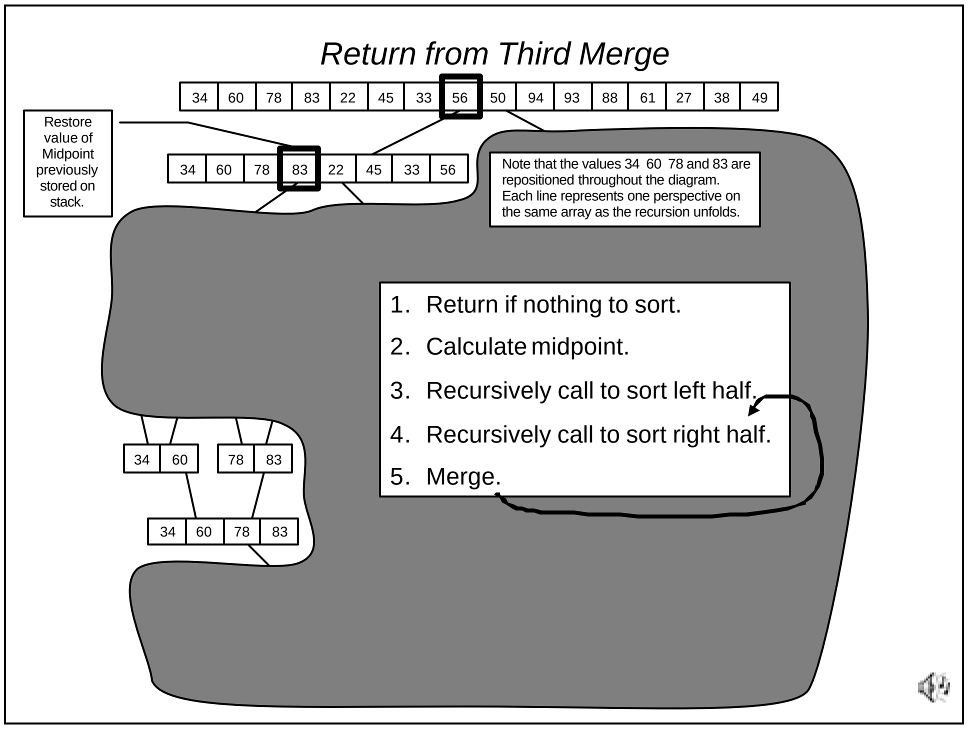
Left data set: 78 83 (known to be sorted by previous operations)

Right data set: 34 60 (known to be sorted by previous operations)

Two data sets merged into a single sorted data set (no change of order).

No sound object for this slide.

## *Return from Third Merge*

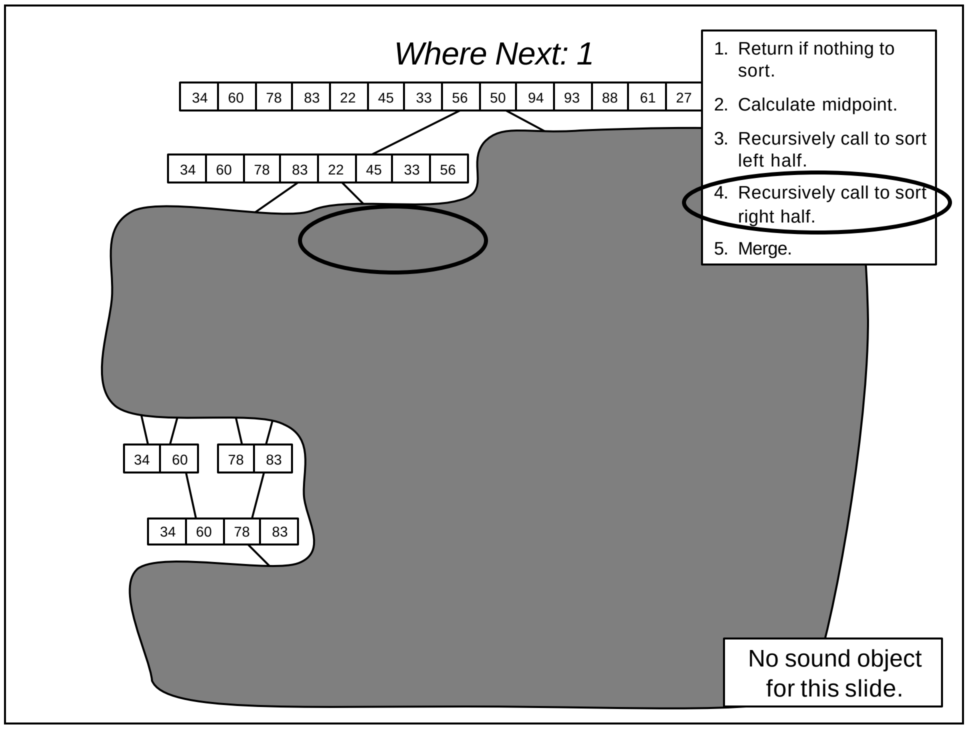


Restore value of Midpoint previously stored on stack.

Note that the values 34 60 78 and 83 are repositioned throughout the diagram. Each line represents one perspective on the same array as the recursion unfolds.

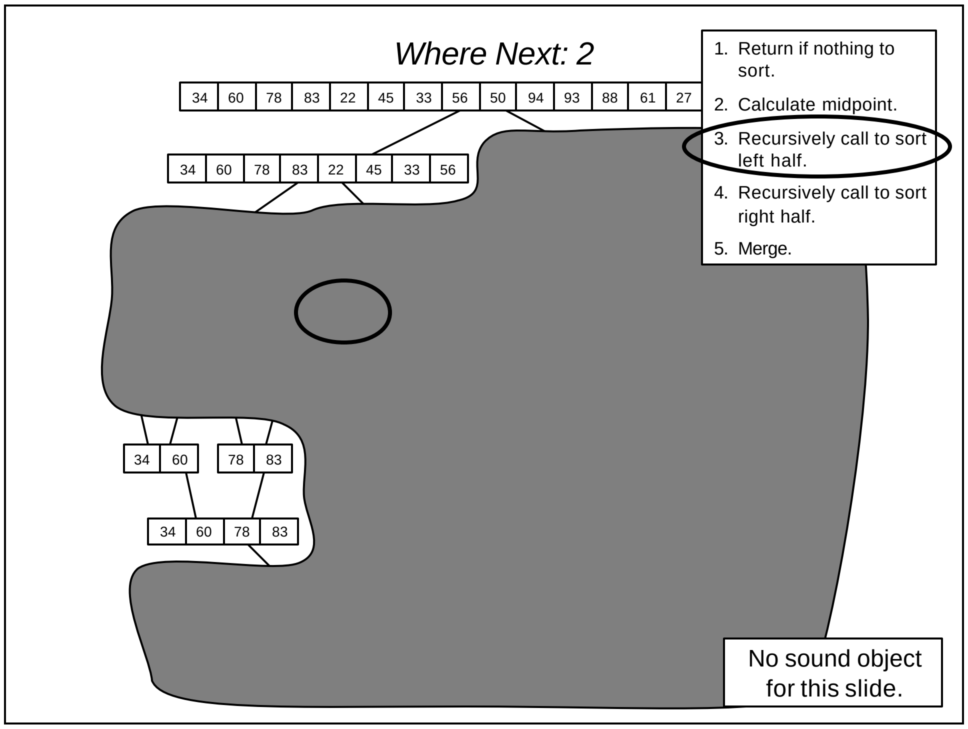
1. Return if nothing to sort.
2. Calculate midpoint.
3. Recursively call to sort left half.
4. Recursively call to sort right half.
5. Merge.

## *Where Next: 1*



1. Return if nothing to sort.
2. Calculate midpoint.
3. Recursively call to sort left half.
4. Recursively call to sort right half.
5. Merge.

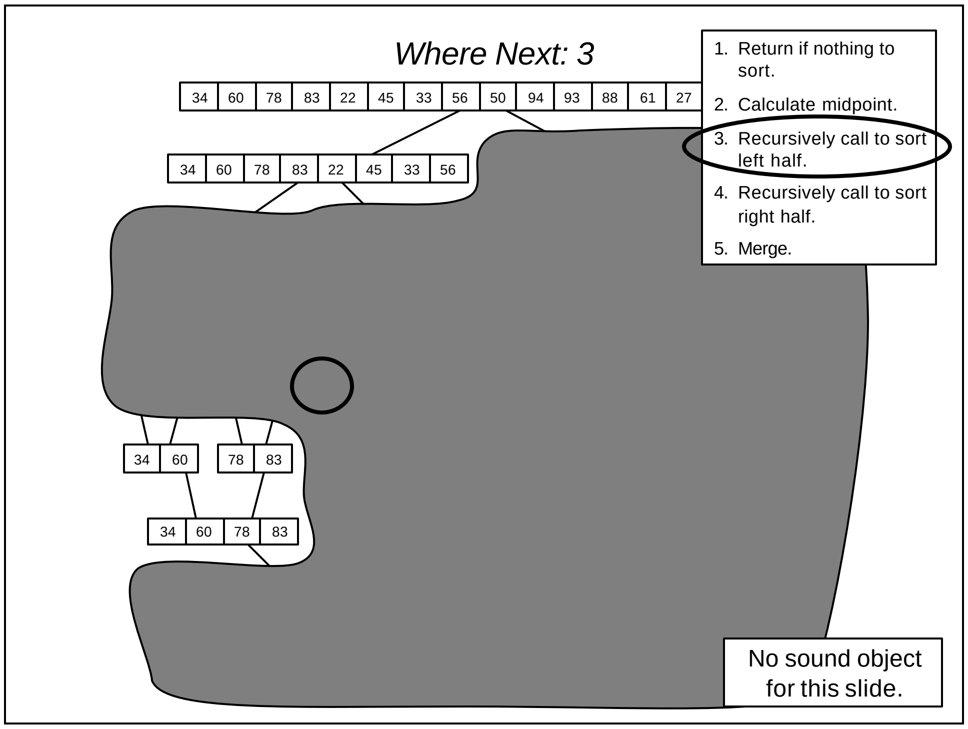
No sound object for this slide.



## *Where Next: 2*

1. Return if nothing to sort.
2. Calculate midpoint.
3. Recursively call to sort left half.
4. Recursively call to sort right half.
5. Merge.

No sound object for this slide.

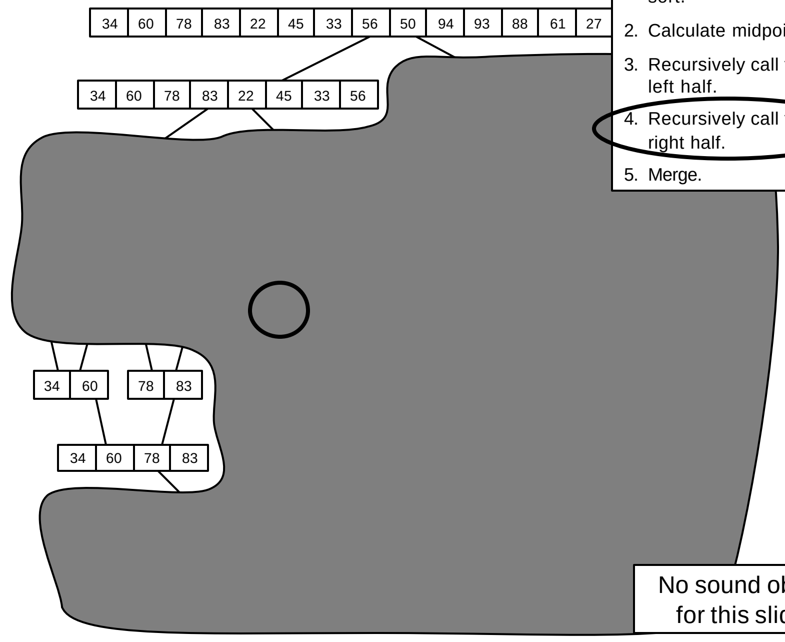


## *Where Next: 3*

1. Return if nothing to sort.
2. Calculate midpoint.
3. Recursively call to sort left half.
4. Recursively call to sort right half.
5. Merge.

No sound object for this slide.

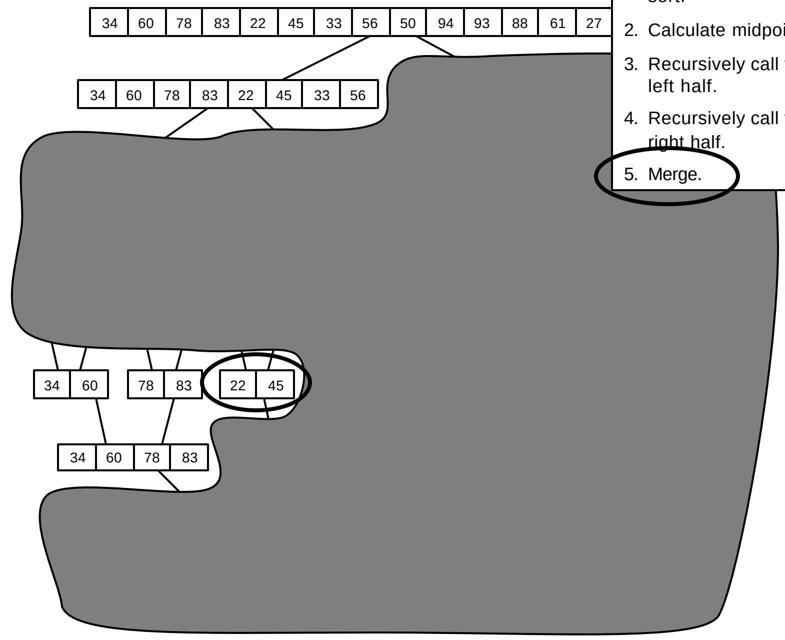
## *Where Next: 4*



1. Return if nothing to sort.
2. Calculate midpoint.
3. Recursively call to sort left half.
4. Recursively call to sort right half.
5. Merge.

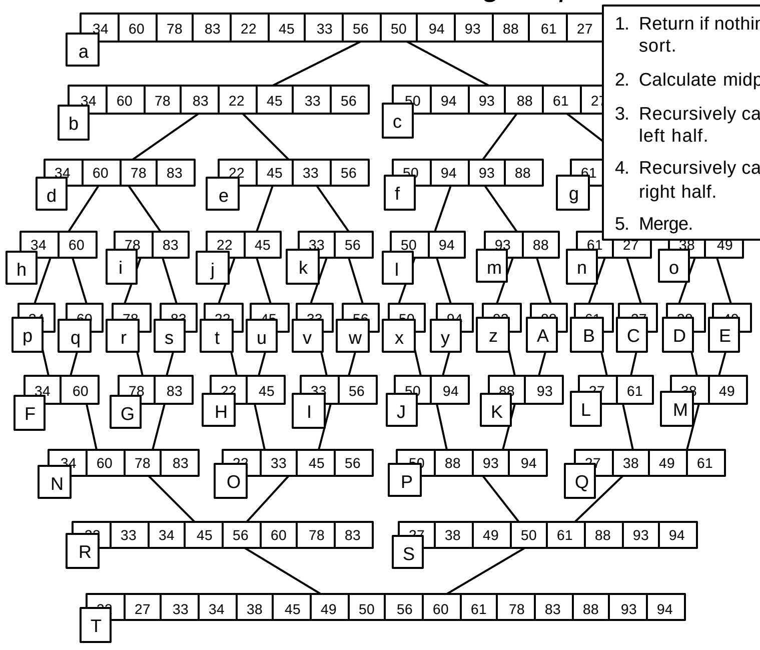
No sound object for this slide.

## *Where Next: 5*



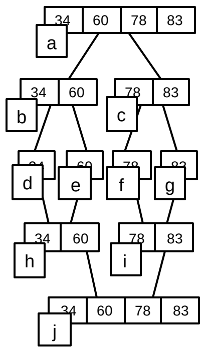
1. Return if nothing to sort.
2. Calculate midpoint.
3. Recursively call to sort left half.
4. Recursively call to sort right half.
5. Merge.

## What's the Full Processing Sequence?



1. Return if nothing to sort.
2. Calculate midpoint.
3. Recursively call to sort left half.
4. Recursively call to sort right half.
5. Merge.

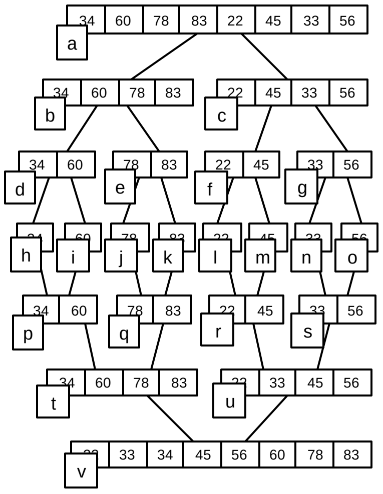
## Sequence: Simple



1. Return if nothing to sort.
2. Calculate midpoint.
3. Recursively call to sort left half.
4. Recursively call to sort right half.
5. Merge.

Identify order in which each lettered node is visited?

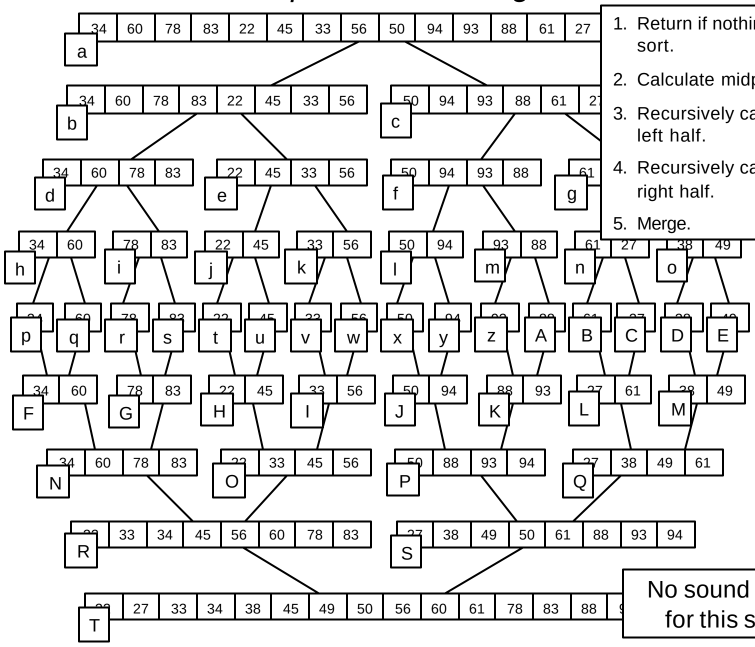
## Sequence: Complex



1. Return if nothing to sort.
2. Calculate midpoint.
3. Recursively call to sort left half.
4. Recursively call to sort right half.
5. Merge.

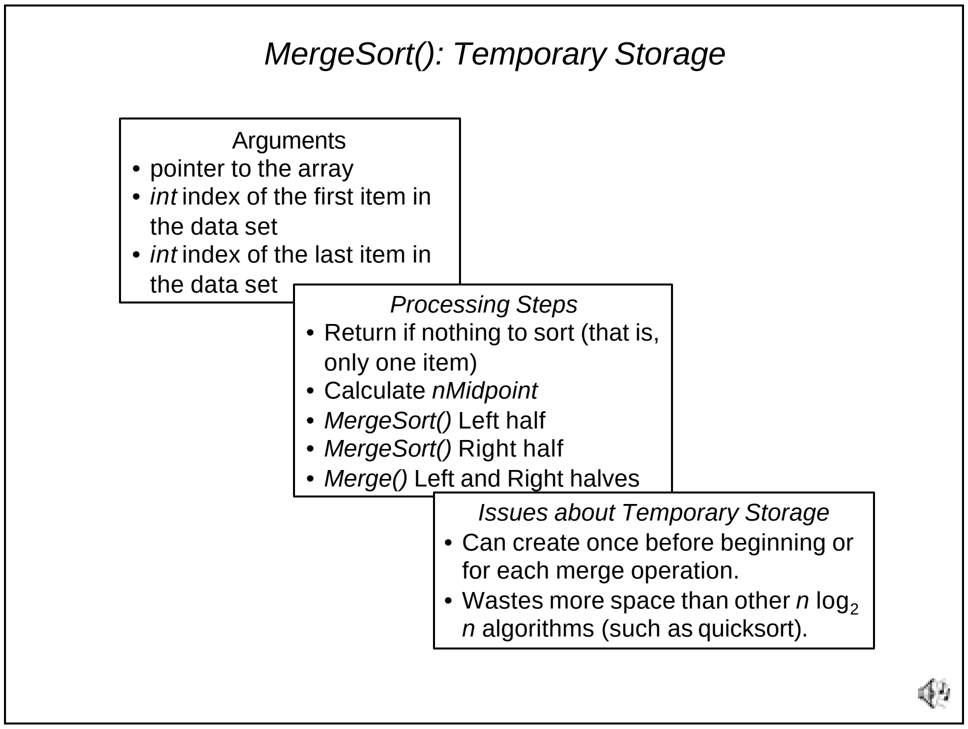
Identify order in which each lettered node is visited?

## Sequence: Challenge



1. Return if nothing to sort.
2. Calculate midpoint.
3. Recursively call to sort left half.
4. Recursively call to sort right half.
5. Merge.

No sound object for this slide.



## *MergeSort(): Temporary Storage*

Arguments

- pointer to the array
- *int* index of the first item in the data set
- *int* index of the last item in the data set

*Processing Steps*

- Return if nothing to sort (that is, only one item)
- Calculate *nMidpoint*
- *MergeSort()* Left half
- *MergeSort()* Right half
- *Merge()* Left and Right halves

*Issues about Temporary Storage*

- Can create once before beginning or for each merge operation.
- Wastes more space than other $n \log_2 n$ algorithms (such as quicksort).