COSC 121: Computer Programming II

Dr. Bowen Hui

University of British Columbia Okanagan

Improved Sorting

- Insertion and selection sort algorithms:
  - Relatively easy to understand
  - Order N^2, given N items to sort
  - Not efficient when N is large

- Idea: Take advantage of the "halfing" technique used in binary search

- Faster algorithms that run in worst case of order N log N
  - Merge Sort
  - Quick Sort

General Idea

- Based on a recursive process: divide and conquer
- Merge sort: Given an array to sort:
  - Split in half
  - Each half being sorted separately
  - Results are merged together
- When to stop? (base case)
- To sort each half, the algorithm repeats the same process

Recursion vs. Iteration

- Iteration:
  - Loop takes one item at a time
  - Applies common set of commands

- Recursion:
  - Subdivides problem into smaller problems
  - Applies common set of commands to each subproblem
  - More in Ch. 12 of text

Recursive Thinking

- Consider the following list of numbers:

```
24, 88, 40, 37
```

- Such a list can be defined as follows:

```
A List is a:  number
        or a:  number  comma  List
```

- The concept of a *List* is used to define itself
  - Requires a stopping criteria called base case

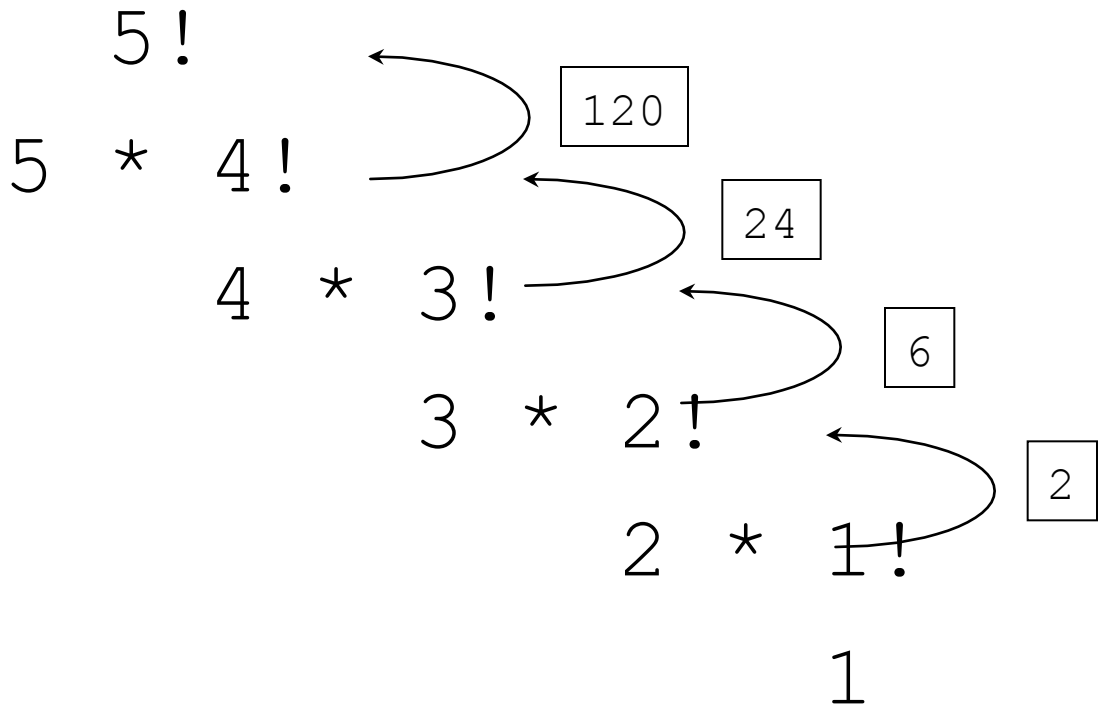
Recall: Factorial

- N! means for any positive integer N, this expression is defined to be the product of all integers between 1 and N inclusive
- Recursive definition:

```
1!   =   1
N!   =   N * (N-1)!
```

- A factorial is defined in terms of another factorial
- Eventually, get to base case of 1!

Recursive Factorial



Example

- Write a recursive definition of 5 * n, where n > 0.
  - 5 * 1 = 5
  - if n=2:
    5 * 2 = 5 + (5 * 1)
  - if n=3:
    5 * 3 = 5 + (5 * 2)
  - if n=4:
    5 * 4 = 5 + (5 * 3)
  - generally:
    5 * n = 5 + (5 * (n-1))

Recursion Exercise

- A recursive method is one that calls itself
- Example recursive method:

```
public int question( int x, int y )
{
   if( x == y )
     return 0;
   else
     return question(x-1, y);
 }
```

- What would `question(5, 3)` return?

Exercise 2

- Another example:

```
public int question( int x, int y )
{
   if( x == y )
     return 0;
   else
     return question(x-1, y) + 1;
 }
```

- What would `question(8, 3)` return?

Writing Recursive Methods

- Consider sum of 1 to any positive integer N
- Recursive definition:

$$\sum_{i=1}^{N} i = N + \sum_{i=1}^{N-1} i = N + N-1 + \sum_{i=1}^{N-2} i$$

$$= N + N-1 + N-2 + \sum_{i=1}^{N-3} i$$

$$\vdots$$

$$= N + N-1 + N-2 + \cdots + 2 + 1$$

Implementing Sum of 1 to N

```
// computes the sum of 1 to num
public int sum( int num )
{
    int result;

    // base case
    if( num == 1 )
        result = 1;
    else
        // general case
        result = num + sum( num-1 );

    return result;
}
```

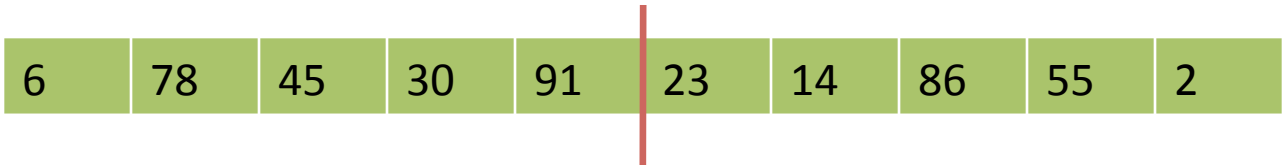
Recursion in Sorting

- Recursion is used to:
  - Take entire input (overall problem)
  - Subdivide input into smaller portions (subproblems)
  - Sort each smaller subproblem in the same way
  - Stop when base case is reached

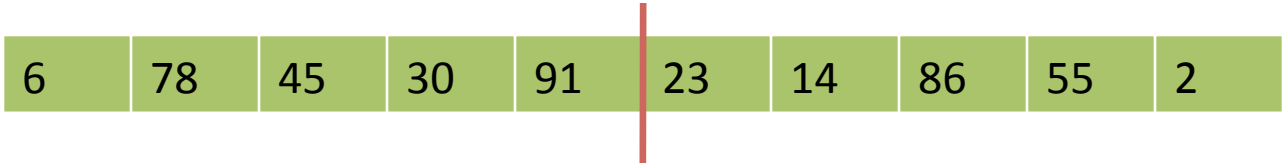
Merge Sort Illustration

6	78	45	30	91	23	14	86	55	2

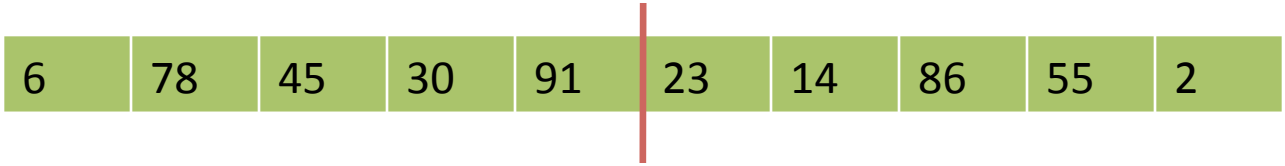
Merge Sort Illustration



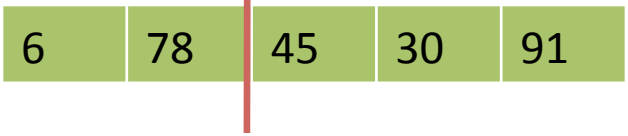
Merge Sort Illustration



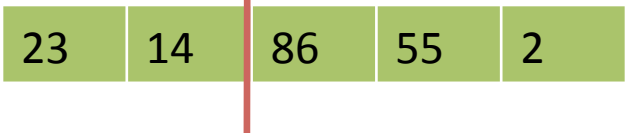
Merge Sort Illustration



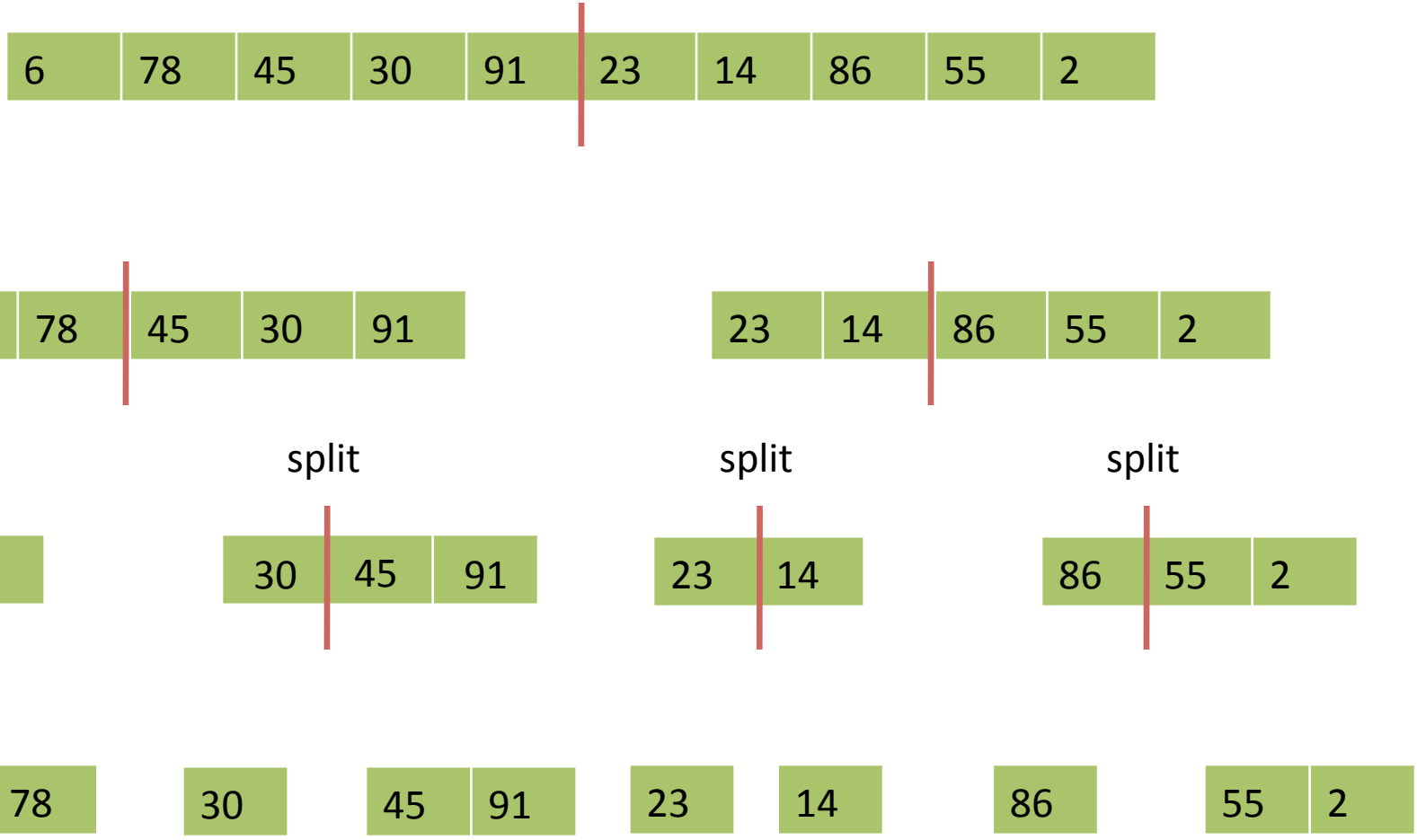
split



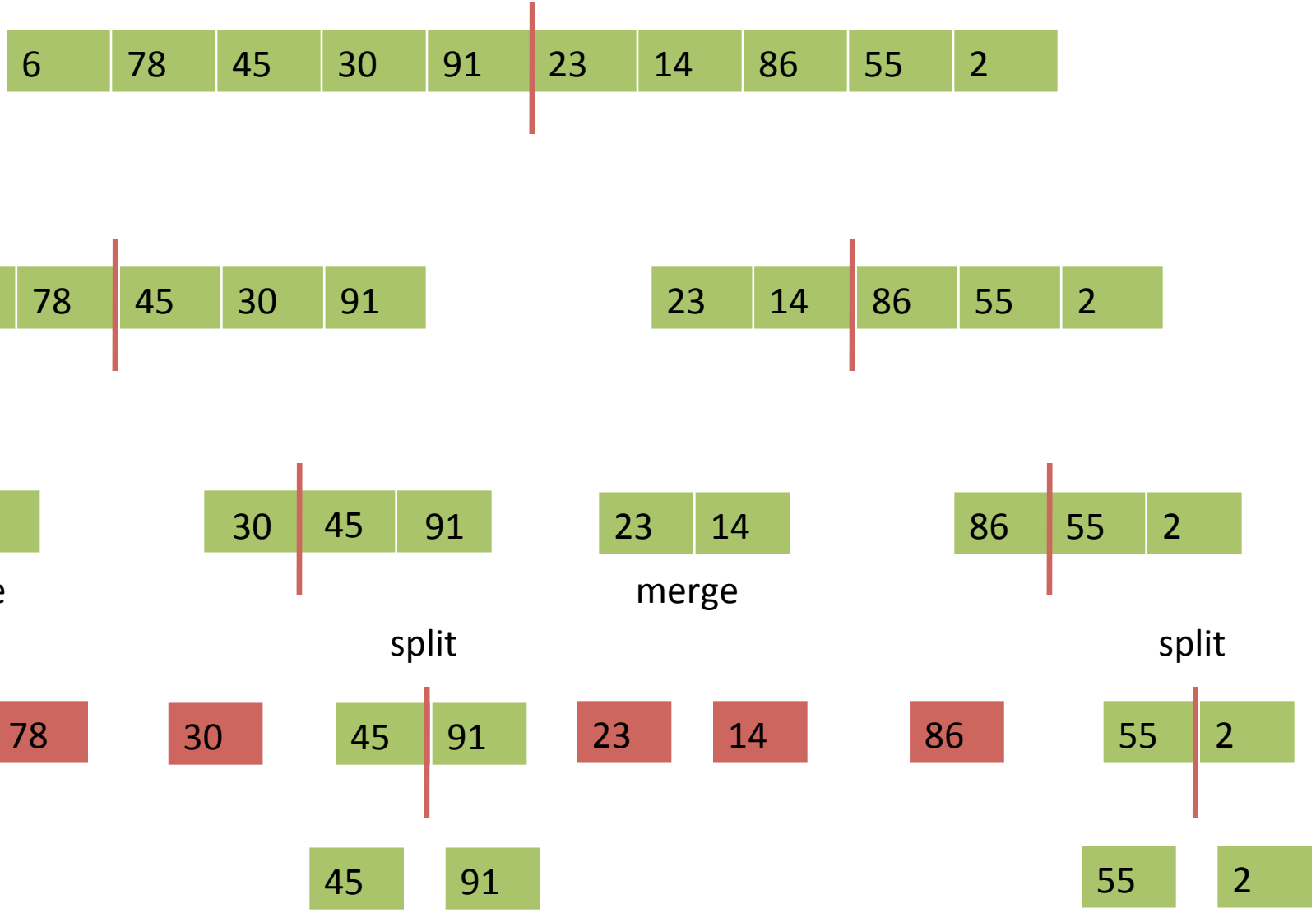
split



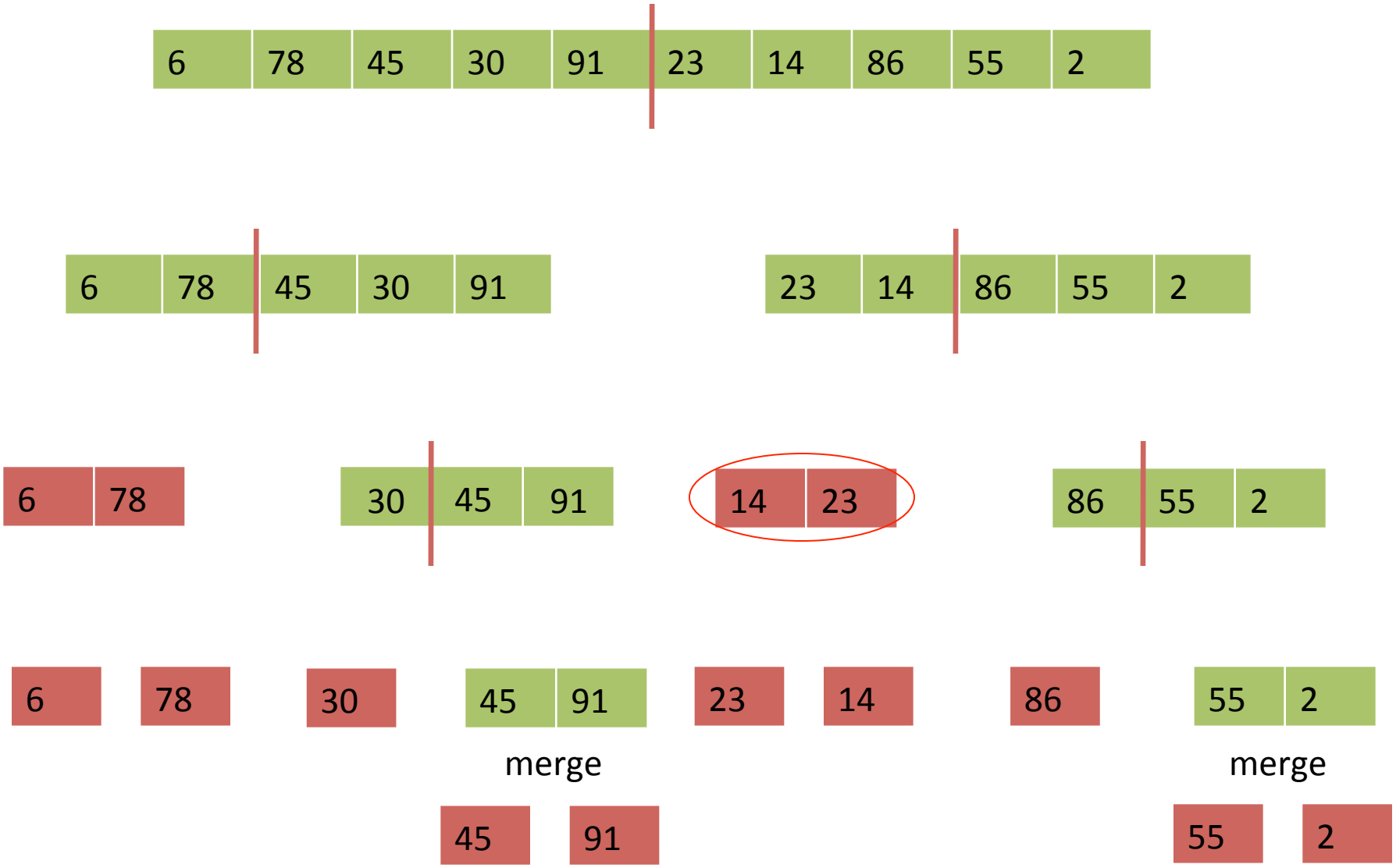
Merge Sort Illustration



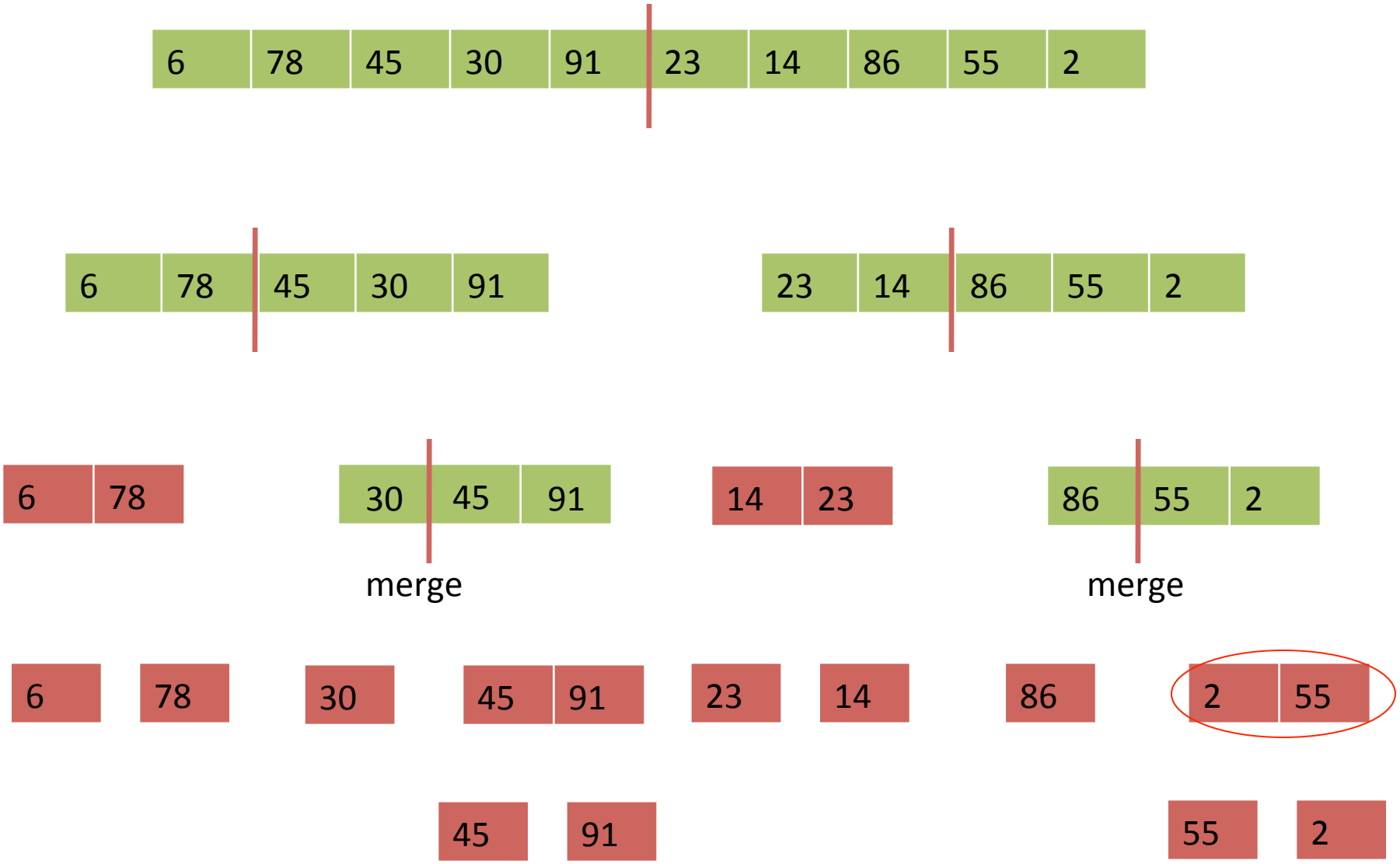
Merge Sort Illustration



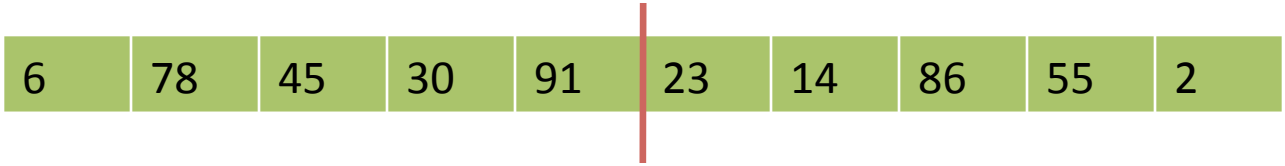
Merge Sort Illustration

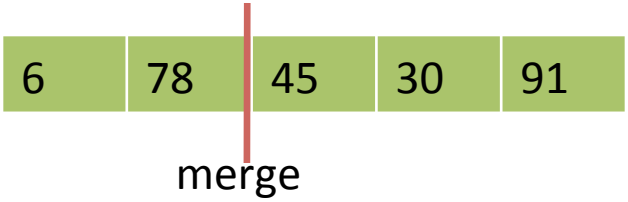


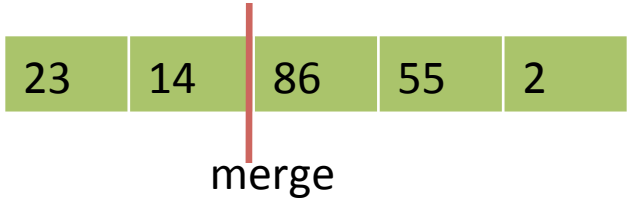
Merge Sort Illustration

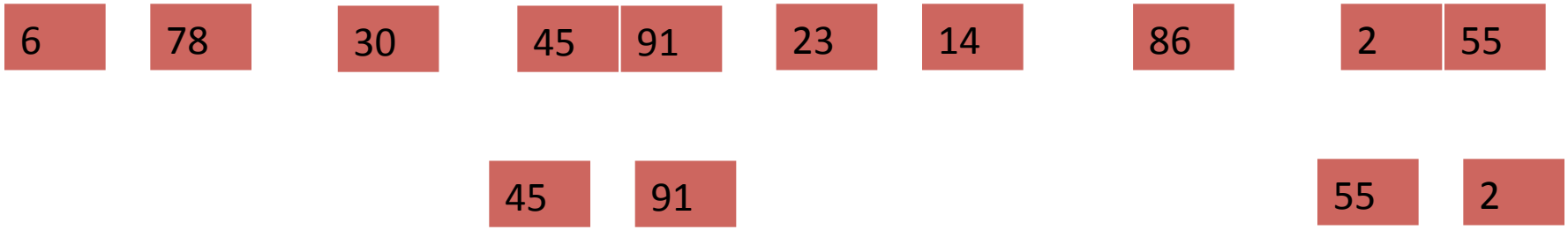


Merge Sort Illustration

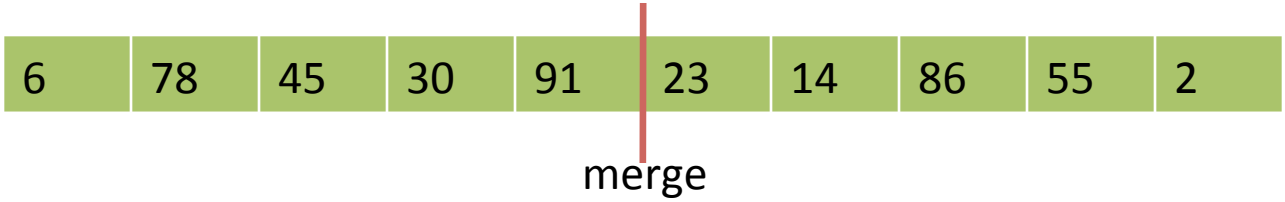


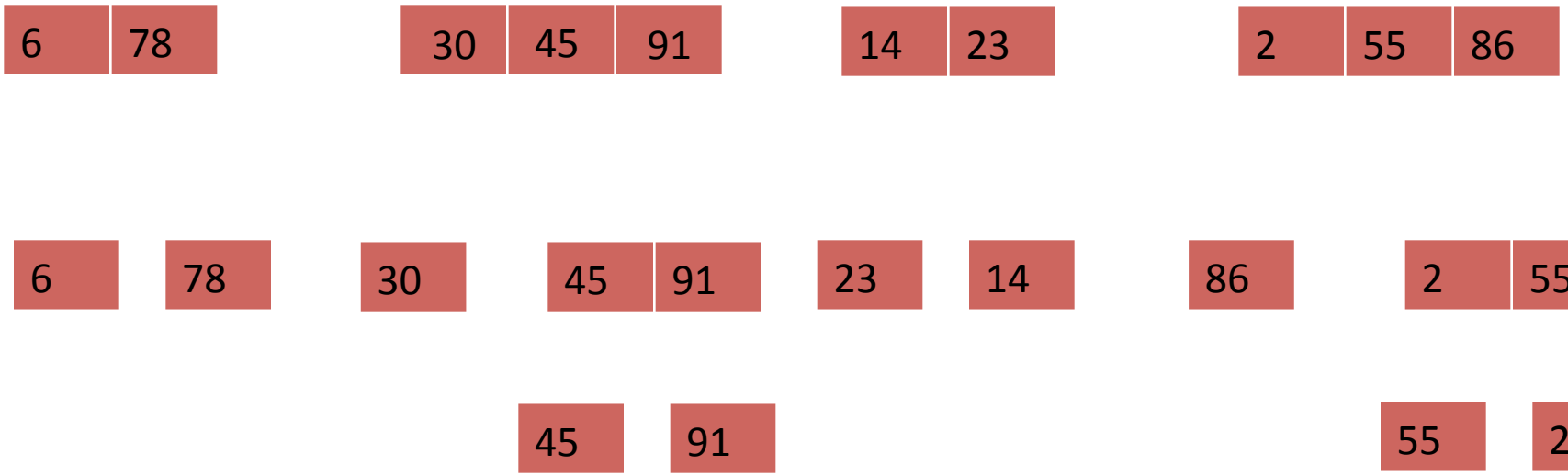




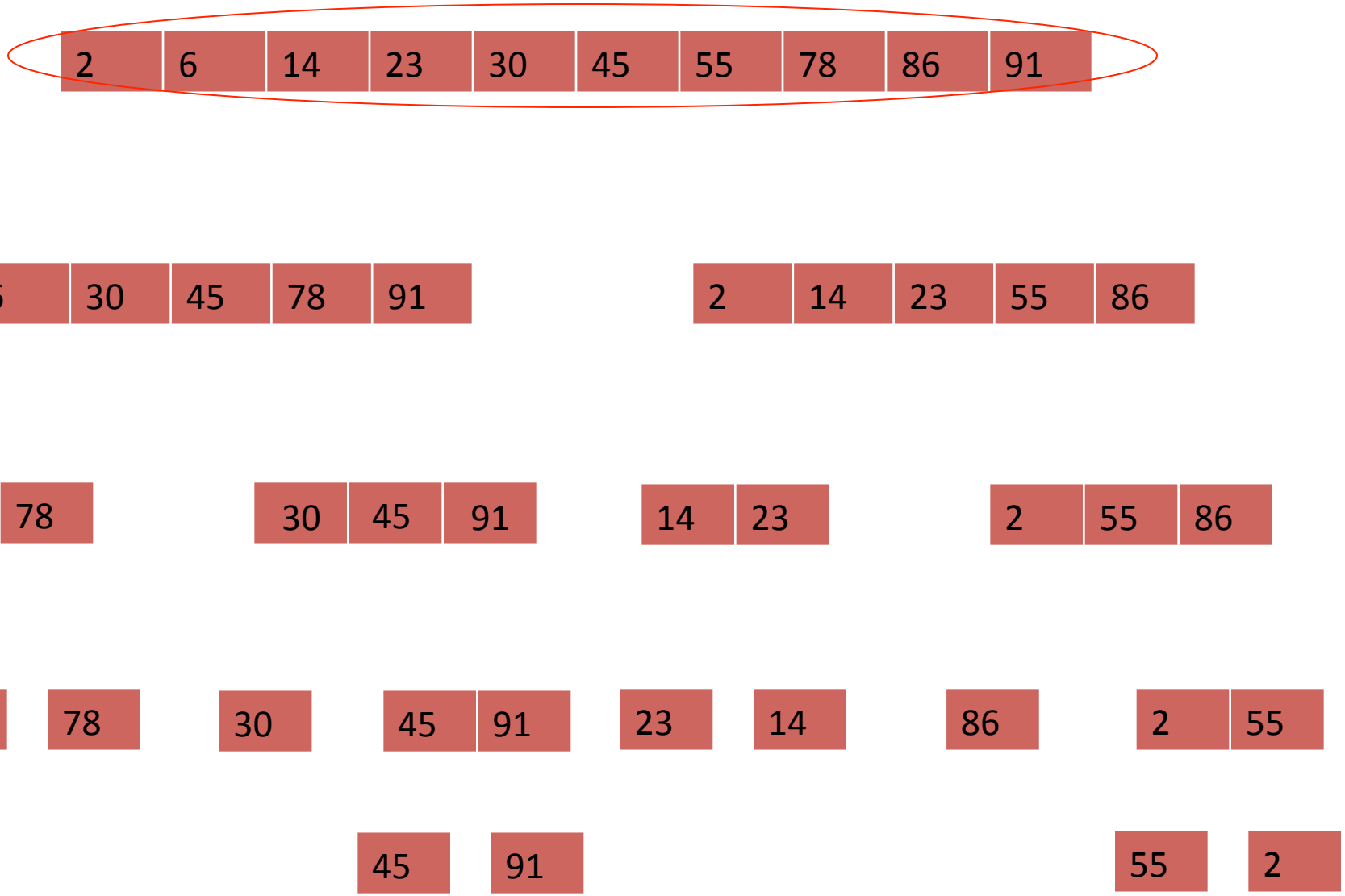


Merge Sort Illustration





Merge Sort Illustration



Exercise

56	13	1	99	65	12	27	8	44	83	2

Sort the following array using:

1. Merge sort

MergeSort on Integer array

```
public class MergeDemo
{
   public static void main (String[] args)
   {
      int[] someNumbers = {75, 9, 39, 42, 61, 21, 56, 32};

      for( int x : someNumbers )
         System.out.println( x );

      MergeSort sortObject = new MergeSort( someNumbers );

      sortObject.sort();

      for( int x : sortObject.getNumbers() )
         System.out.println( x );
   }
}
```

MergeSort on Integer array

```
public class MergeDemo
{
   public static void main (String[] args)
   {
      int[] someNumbers = {75, 9, 39, 42, 61, 21, 56, 32};

      for( int x : someNumbers )
         System.out.println( x );

      MergeSort sortObject = new MergeSort( someNumbers );

      sortObject.sort();

      for( int x : sortObject.getNumbers() )
         System.out.println( x );
   }
}
```

MergeSort.java

```
public class MergeSort
{
  // attribute
  private int[] numbers;

  // methods
  public MergeSort( int[] list ) {...}
  public void sort() {...}
  // other methods as needed...
}
```

Recursive Structure

```
public void sort()
{
   // if one elem left, stop: sorted
   // else:
   // sort on left half
   // sort on right half
   // combine two halves
}
```

Define Parameters for Recursion

```
public void sort()
{
  helper( 0, numbers.length-1 );
}

private void helper( int min, int max )
{
  // if one elem left, stop: sorted
  // else:
  // sort on left half
  // sort on right half
  // combine two halves
}
```

```
// main method to call for sorting
public void sort()
{
  helper( 0, numbers.length-1 );
}

// recursive method for partitioning the array into two halves
private void helper( int min, int max )
{
  System.out.println( "inside helper:" );
  System.out.println( "\tmin=" + min + ", max=" + max );
  if( min < max )
  {
    int mid = min + (max-min)/2;

    // sort each half independently
    helper( min,   mid );
    helper( mid+1, max );

    // when done, combine two sorted arrays
    combine( min, mid, max );
  }
  // otherwise, array is sorted
}
```

recursive method

method to do all the hard work

Main Steps in `combine()`

```
private void combine( int low, int mid, int high)
{
  // 1. create extra array called temp
  int[] temp = . . .

  // 2. setup indices
  int left  = . . .    // left side starts at low
  int right = . . .    // right side starts at mid+1
  int pos   = . . .    // pos (to merge) starts at low

  // 3. compare each value in left + right
  // copy smaller value into numbers[pos]
  while( left <= mid && right <= high ) { ... }

  // 4. copy rest of left side into numbers
  while( left <= mid ) { ... }
}
```

```
private void combine( int low, int mid, int high)
{
  // 1. create extra array called temp
  int[] temp = new int[numbers.length];
  for( int i=low; i<=high; i++ )
    temp[i] = numbers[i];

  // 2. setup indices
  int left  = low;      // left side starts at low
  int right = mid + 1;  // right side starts at mid+1
  int pos   = low;      // pos (to merge) starts at low

  // 3. compare each value in left + right
  // copy smaller value into numbers[pos]
  while( left <= mid && right <= high ) { ... }

  // 4. copy rest of left side into numbers
  while( left <= mid ) { ... }
}
```

```
private void combine( int low, int mid, int high)
{
  // 1. create extra array called temp
  // 2. setup indices
  // 3. compare each value in left + right
  // copy smaller value into numbers[pos]
  while( left <= mid && right <= high ) {
    if( temp[left] <= temp[right] ) {
      numbers[pos] = temp[left];
      left++;
    }
    else {
      numbers[pos] = temp[right];
      right++;
    }
    pos++;
  }
  // 4. copy rest of left side into numbers
 while( left <= mid ) { ... }
}
```

when do we exit this loop?

what was in numbers all this time?

```
private void combine( int low, int mid, int high)
{
  // 1. create extra array called temp

  // 2. setup indices

  // 3. compare each value in left + right
  // copy smaller value into numbers[pos]
  while( left <= mid && right <= high ) { ... }

  // 4. copy rest of left side into numbers
  while( left <= mid )
  {
    numbers[pos] = temp[left];
    left++;
    pos++;
  }
}
```

Tracing through MergeSort

- numbers = [75 9 39 42 61 21 56 32]

  0 1 2 3 4 5 6 7

- Method calls:

helper(0,7)

Tracing through MergeSort

- numbers = [75 9 39 42 61 21 56 32]

  0 1 2 3 4 5 6 7

- Method calls:

helper(0,7)

  helper(0,3)

Tracing through MergeSort

- numbers = [75 9 39 42 61 21 56 32]

  0 1 2 3 4 5 6 7

- Method calls:

helper(0,7)

  helper(0,3)

    helper(0,1)

Tracing through MergeSort

- numbers = [75 9 39 42 61 21 56 32]

  0 1 2 3 4 5 6 7

- Method calls:

helper(0,7)

  helper(0,3)

    helper(0,1)

      helper(0,0) // 0 < 0 is false

Tracing through MergeSort

- numbers = [75 9 39 42 61 21 56 32]
  0 1 2 3 4 5 6 7
- Method calls:

```
helper(0,7)
   helper(0,3)
      helper(0,1)
         helper(0,0)          // 0 < 0 is false
         helper(1,1)          // 1 < 1 is false
```

Tracing through MergeSort

- numbers = [75 9 39 42 61 21 56 32]
  0 1 2 3 4 5 6 7
- Method calls:

helper(0,7)
  helper(0,3)
    helper(0,1)
      helper(0,0) // 0 < 0 is false
      helper(1,1) // 1 < 1 is false
      combine(0,0,1)
        temp = [75 9]
          0 1

Tracing through MergeSort

- numbers = [75 9 39 42 61 21 56 32]
  0 1 2 3 4 5 6 7
- Method calls:

helper(0,7)
  helper(0,3)
    helper(0,1)
      helper(0,0) // 0 < 0 is false
      helper(1,1) // 1 < 1 is false
      combine(0,0,1)
        temp = [75 9]
          0 1 left=0, right=1, pos=0
- numbers = [9 9 39 42 61 21 56 32] when exit comparison loop
  0 1 2 3 4 5 6 7

Tracing through MergeSort

- numbers = [75 9 39 42 61 21 56 32]
  0 1 2 3 4 5 6 7
- Method calls:

helper(0,7)
  helper(0,3)
    helper(0,1)
      helper(0,0) // 0 < 0 is false
      helper(1,1) // 1 < 1 is false
      combine(0,0,1)
        temp = [75 9]
          0 1 left=0, right=1, pos=0
- numbers = [9 9 39 42 61 21 56 32] when exit comparison loop
  0 1 2 3 4 5 6 7
- numbers = [9 75 39 42 61 21 56 32] when exit combine
  0 1 2 3 4 5 6 7

Merge Sort

- Very intuitive and easy to remember
- New technique: recursion
- Sort method:
  - Manages recursive structure
- Combine method:
  - Keeping track of indices to sort in-place
- Next:
  - Continue tracing implementation + quicksort