

# Sorting: Merge Sort



Computer Science Department  
University of Central Florida

*COP 3502 – Computer Science I*



# Sorting: Merge Sort

---

- Problem with Bubble/Insertion/Selection Sorts:
  - All of these sorts make a large number of comparisons and swaps between elements
  - As mentioned last class (while covering  $n^2$  sorts):
    - Any algorithm that swaps adjacent elements can only run so fast
  - So one might ask is there a more clever way to sort numbers
    - A way that does not require looking at all these pairs
  - Indeed, there are several ways to do this
  - And one of them is Merge Sort



# Sorting: Merge Sort

---

## ■ Merge Sort

### ■ Conceptually, Merge Sort works as follows:

- If the “list” is of length 0 or 1, then it is already sorted!
- Otherwise:
  1. Divide the unsorted list into two sub-lists of about half the size
    - So if your list has  $n$  elements, you will divide that list into two sub-lists, each having approximately  $n/2$  elements:
  2. Recursively sort each sub-list by calling recursively calling Merge Sort on the two smaller lists
  3. **Merge** the two sub-lists back into one sorted list
    - This Merge is a function that we look at on its own



# Sorting: Merge Sort

---

## ■ Merge Sort

### ■ Basically, given a list:

- You will split this list into two lists of about half the size
- Then you recursively call Merge Sort on each list
- What does that do?
  - Each of these new lists will, individually, be split into two lists of about half the size.
  - So now we have four lists, each about  $\frac{1}{4}$  the size of the original list
- This keeps happening...the lists keep getting split into smaller and smaller lists
  - Until you get to a list of size 1 or size 0
- Then we Merge them into a larger, sorted list



# Sorting: Merge Sort

---

- Merge Sort

- Incorporates two main ideas to improve its runtime:

- 1) A small list will take fewer steps to sort than a large list

- 2) Fewer steps are required to construct a sorted list from two sorted lists than two unsorted lists

- For example:

- You only have to traverse each list once if they're already sorted



# Sorting: Merge Sort

---

## ■ Merge function

- The key to Merge Sort: the Merge function
- Given two sorted lists, Merge them into one sorted list
- Problem:
  - You are given two arrays, each of which is already sorted
  - Your job is to efficiently combine the two arrays into one larger array
  - The larger array should contain all the values of the two smaller arrays
  - Finally, the larger array should be in sorted order



# Sorting: Merge Sort

## ■ Merge function

- The key to Merge Sort: the Merge function
- Given two sorted lists, Merge them into one sorted list
- If you have two lists:
  - $X (x_1 < x_2 < \dots < x_m)$  and  $Y (y_1 < y_2 < \dots < y_n)$
  - Merge these into one list:  $Z (z_1 < z_2 < \dots < z_{m+n})$
- Example:
  - List 1 = {3, 8, 9} and List 2 = {1, 5, 7}
  - Merge(List 1, List 2) = {1, 3, 5, 7, 8, 9}



# Sorting: Merge Sort

---

## ■ Merge function

### ■ Solution:

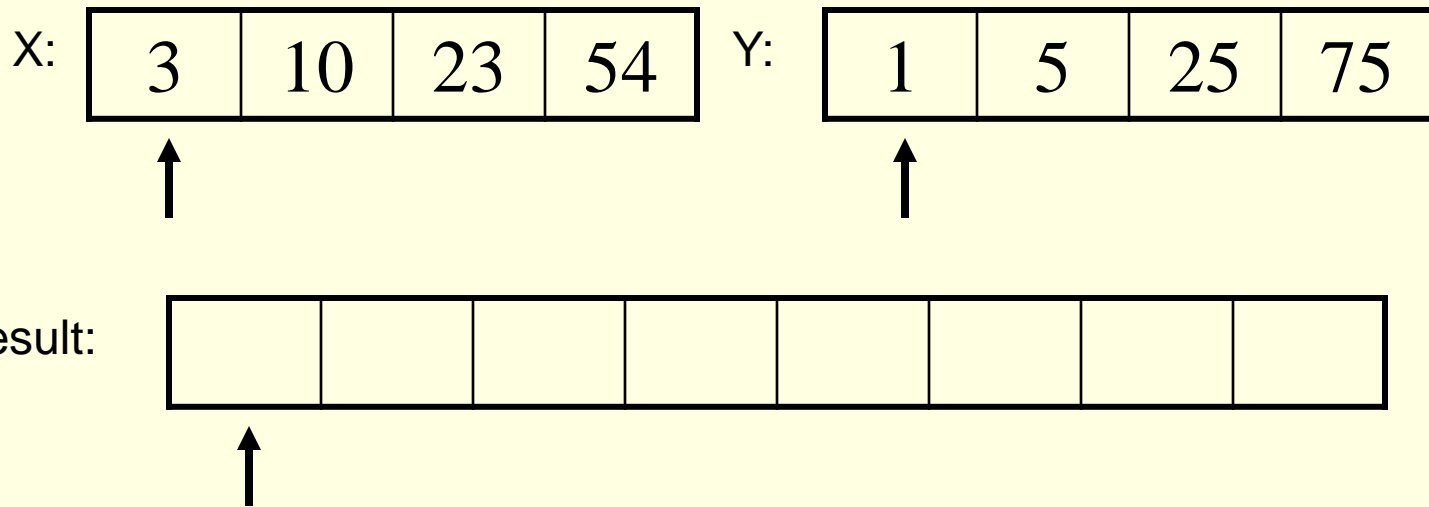
- Keep track of the smallest value in each array that hasn't been placed, in order, in the larger array yet
- Compare these two smallest values from each array
  - One of these **MUST** be the smallest of all the values in both arrays that are left
  - Place the smallest of the two values in the next location in the larger array
- Adjust the smallest value for the appropriate array
- Continue this process until all values have been placed in the large array





# Sorting: Merge Sort

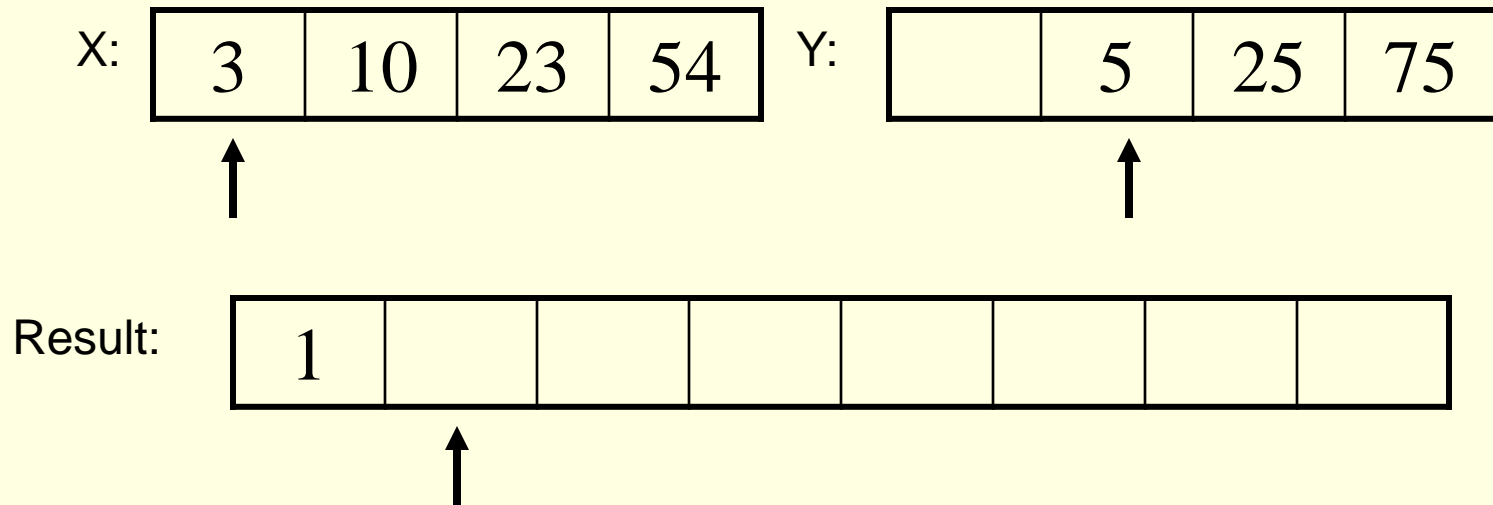
- Example of **Merge** function:





# Sorting: Merge Sort

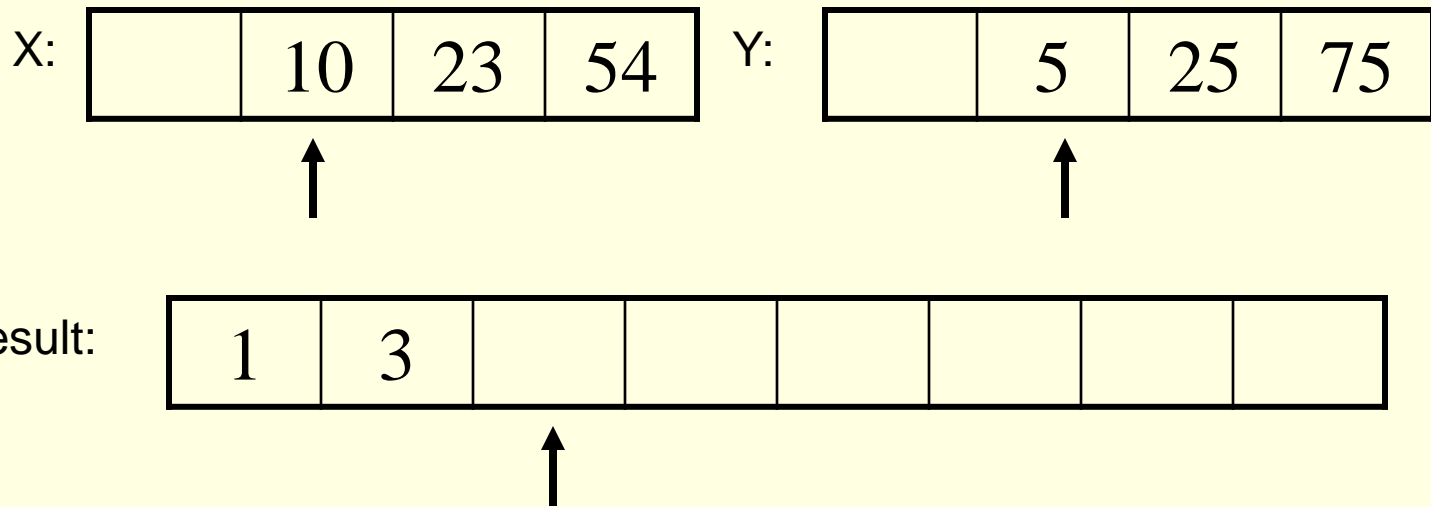
- Example of Merge function:





# Sorting: Merge Sort

- Example of Merge function:







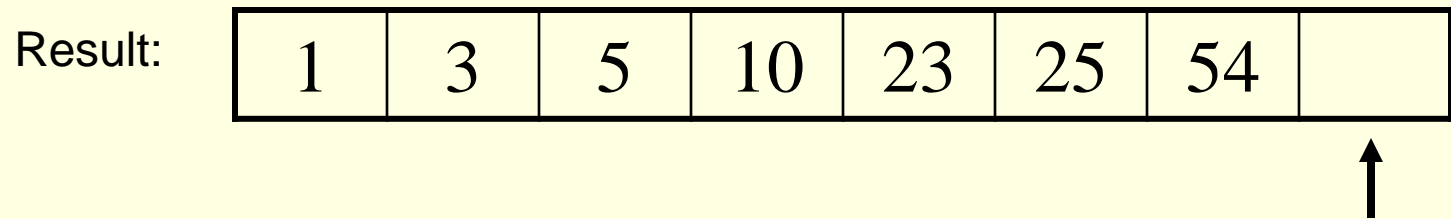
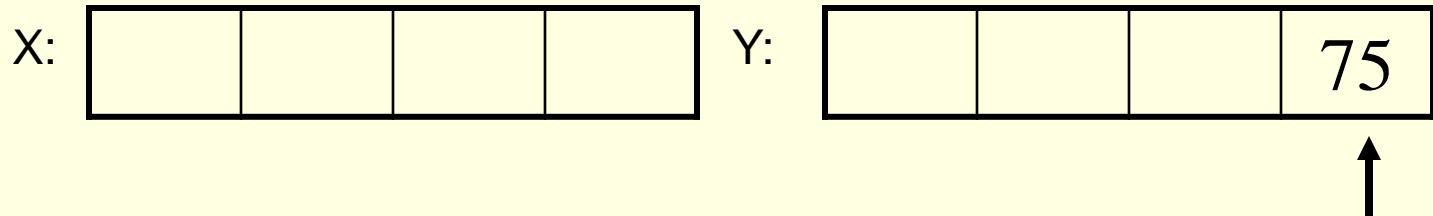






# Sorting: Merge Sort

- Example of Merge function:

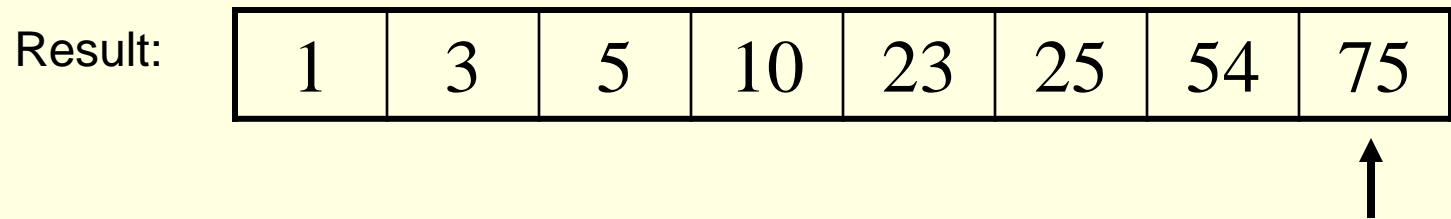
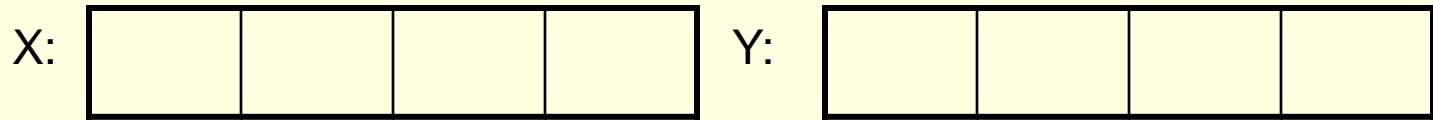






# Sorting: Merge Sort

- Example of Merge function:





# Sorting: Merge Sort

---

- **Merge** function

- The big question:

- How can we use this Merge function to sort an entire, unsorted array?
- This function only “sorts” a specific scenario:
  - You have to have two, **already sorted**, arrays
- **Merge** can then “sort” (merge) them into one larger array
- So can we use this Merge function to somehow sort a large, unsorted array???

- This brings us back to Merge Sort



# Sorting: Merge Sort

---

## ■ Merge Sort

- Again, here is the main idea for Merge Sort:
  - 1) Sort the first half of the array, using Merge Sort
  - 2) Sort the second half of the array, using Merge Sort
    - Now, we do indeed have a situation where we can use the Merge function!
    - Each half is already sorted!
  - 3) So simply merge the first half of the array with the second half.
- And this points to a recursive solution...



# Sorting: Merge Sort

---

## ■ Merge Sort

### ■ Conceptually, Merge Sort works as follows:

- If the “list” is of length 0 or 1, then it is already sorted!
- Otherwise:
  1. Divide the unsorted list into two sub-lists of about half the size
    - So if your list has  $n$  elements, you will divide that list into two sub-lists, each having approximately  $n/2$  elements:
  2. Recursively sort each sub-list by calling recursively calling Merge Sort on the two smaller lists
  3. **Merge** the two sub-lists back into one sorted list



# Sorting: Merge Sort

---

## ■ Merge Sort

### ■ Basically, given a list:

- You will split this list into two lists of about half the size
- Then you recursively call Merge Sort on each list
- What does that do?
  - Each of these new lists will, individually, be split into two lists of about half the size.
  - So now we have four lists, each about  $\frac{1}{4}$  the size of the original list
- This keeps happening...the lists keep getting split into smaller and smaller lists
  - Until you get to a list of size 1 or size 0
- Then we Merge them into a larger, sorted list



# Sorting: Merge Sort

- Merge sort idea:
  - Divide the array into two halves.
  - Recursively sort the two halves (using merge sort).
  - Use **Merge** to combine the two arrays.



mergeSort(0, n/2-1)



sort

mergeSort(n/2, n-1)



sort

merge(0, n/2, n-1)





98	23	45	14	6	67	33	42
----	----	----	----	---	----	----	----



98	23	45	14	6	67	33	42
----	----	----	----	---	----	----	----

98	23	45	14
----	----	----	----

6	67	33	42
---	----	----	----





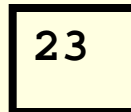
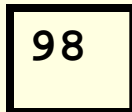
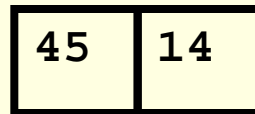
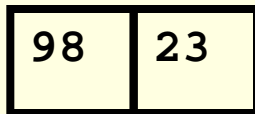
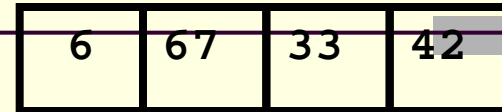
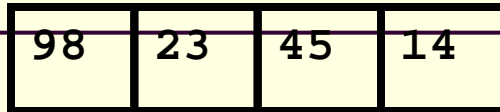
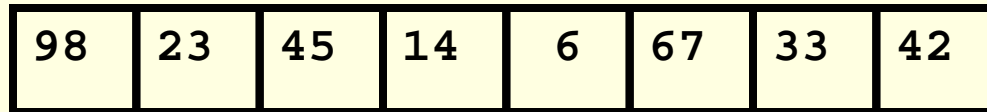
98	23	45	14	6	67	33	42
----	----	----	----	---	----	----	----

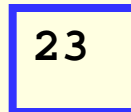
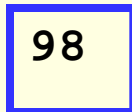
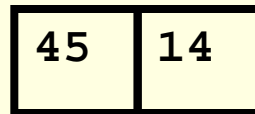
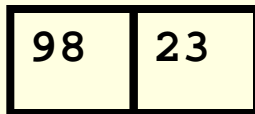
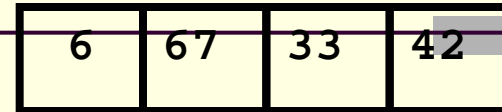
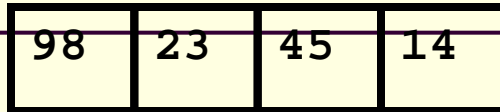
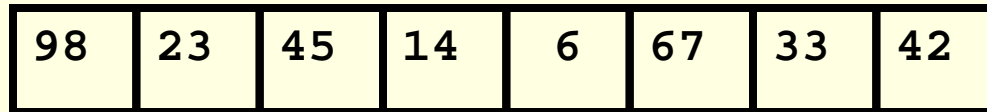
98	23	45	14
----	----	----	----

6	67	33	42
---	----	----	----

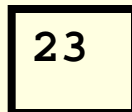
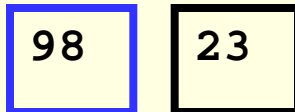
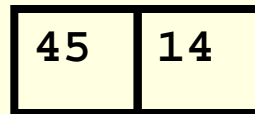
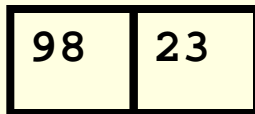
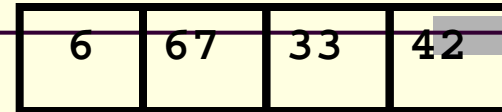
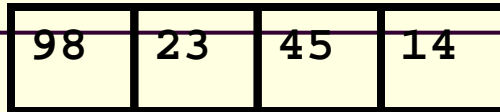
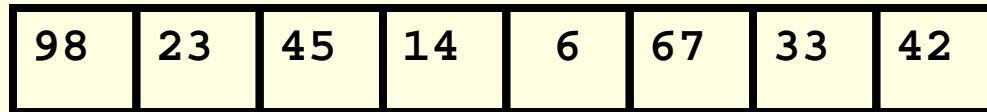
98	23
----	----

45	14
----	----

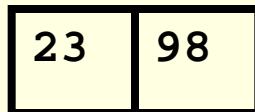
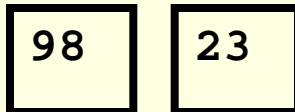
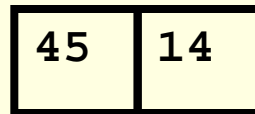
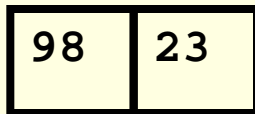
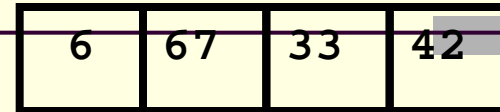
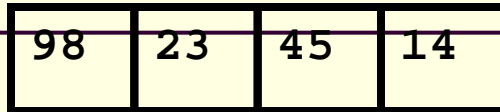
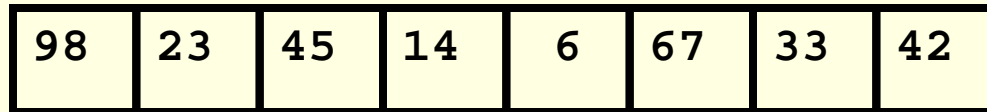




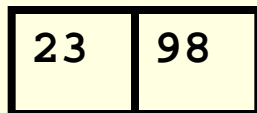
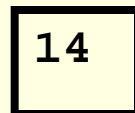
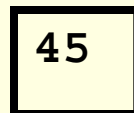
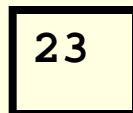
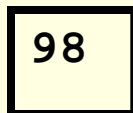
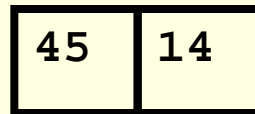
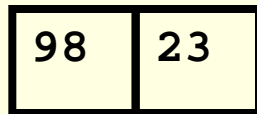
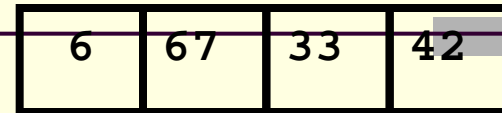
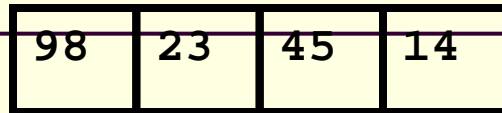
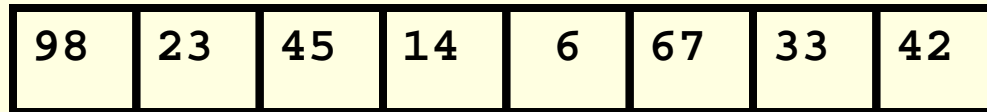
Merge

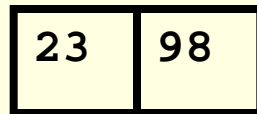
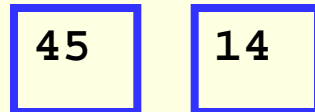
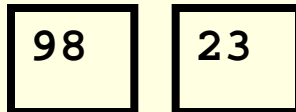
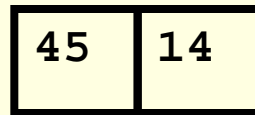
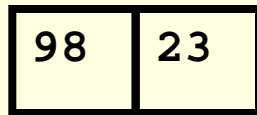
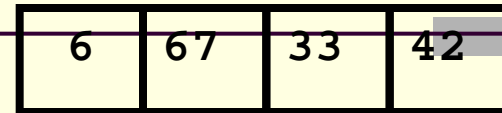
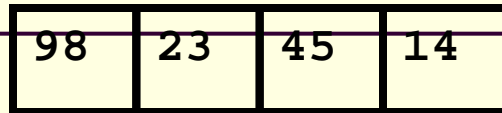
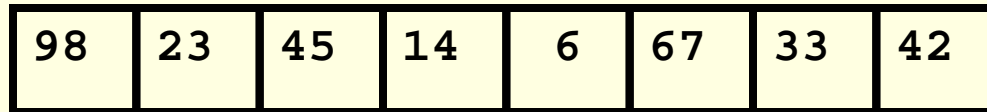


Merge

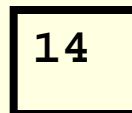
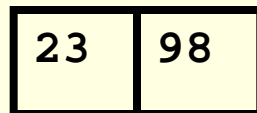
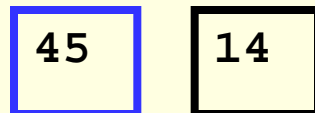
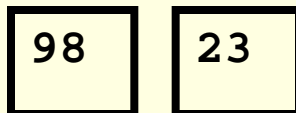
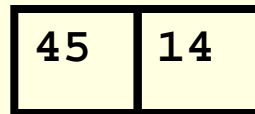
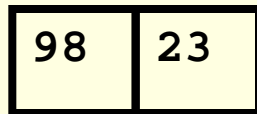
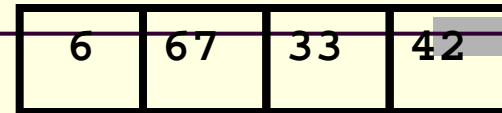
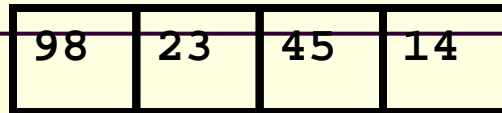
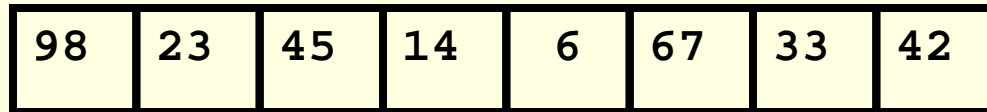


Merge



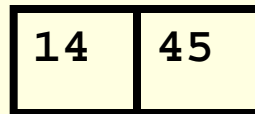
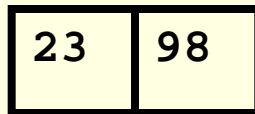
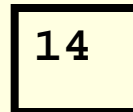
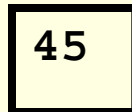
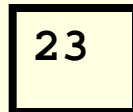
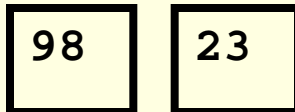
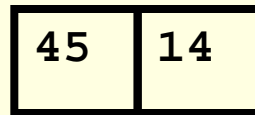
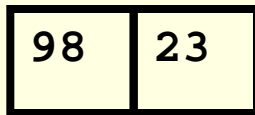
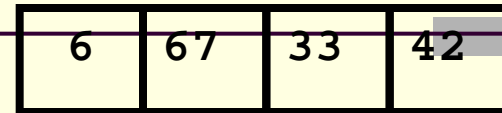
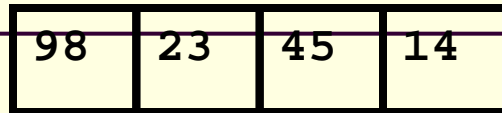
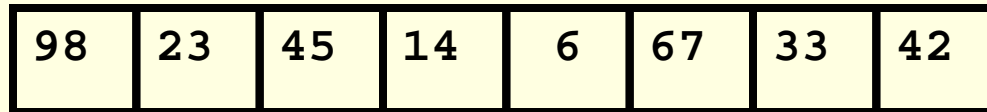


Merge

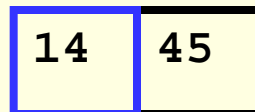
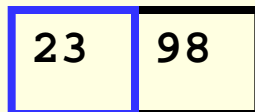
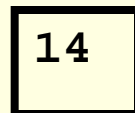
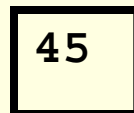
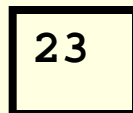
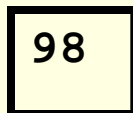
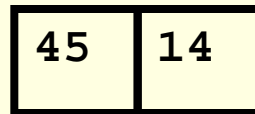
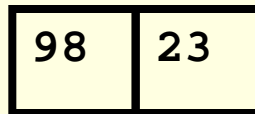
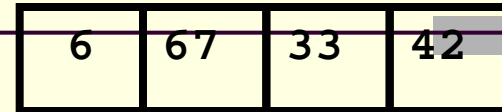
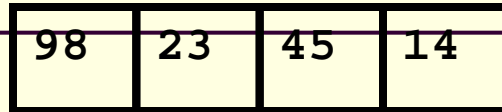
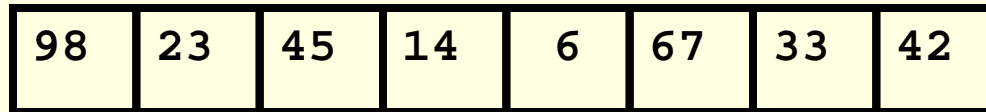


Merge

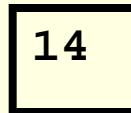
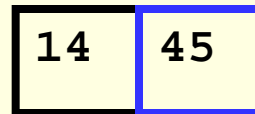
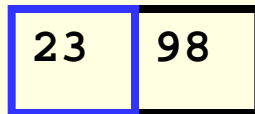
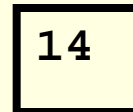
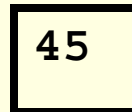
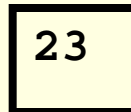
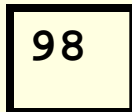
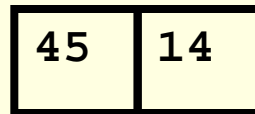
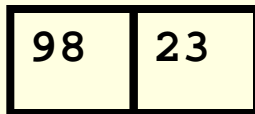
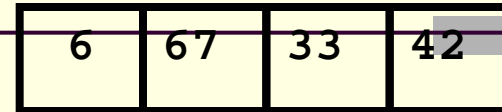
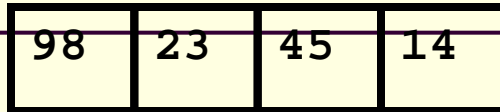
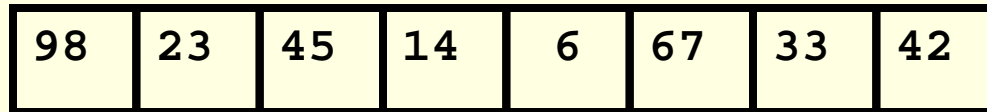


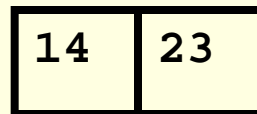
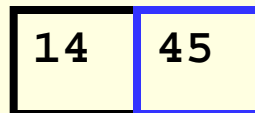
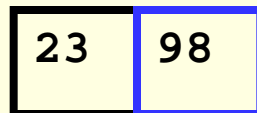
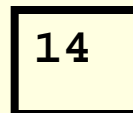
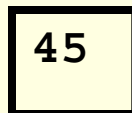
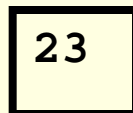
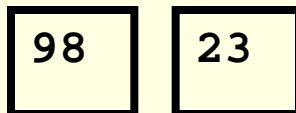
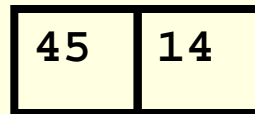
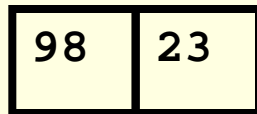
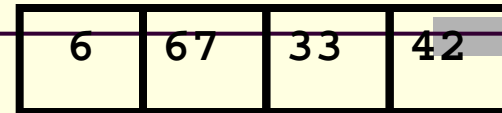
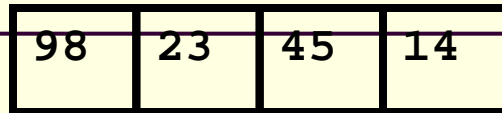
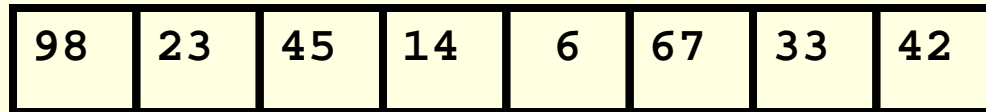


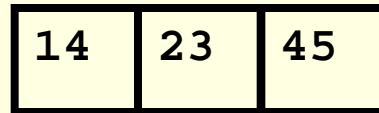
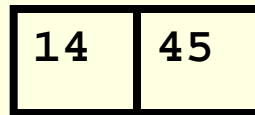
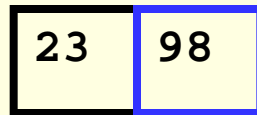
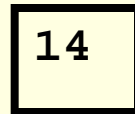
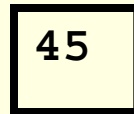
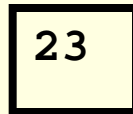
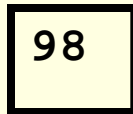
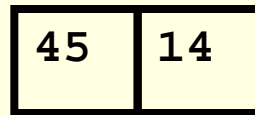
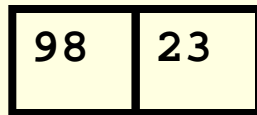
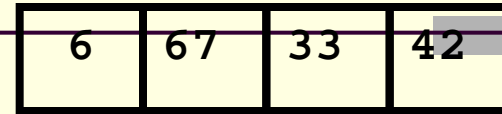
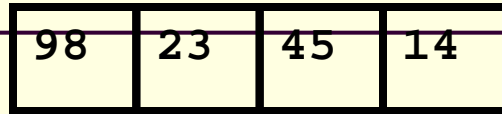
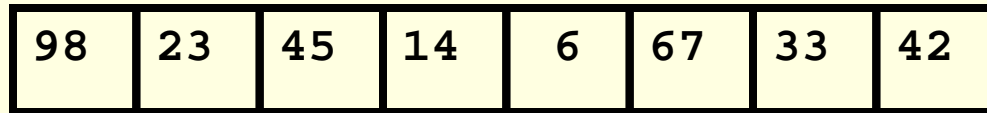
Merge

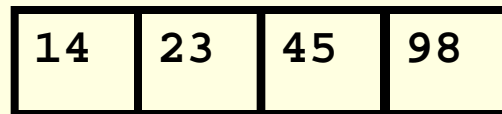
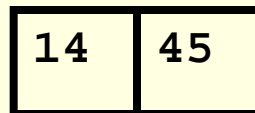
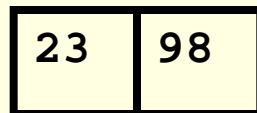
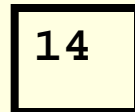
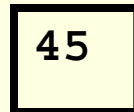
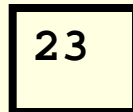
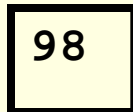
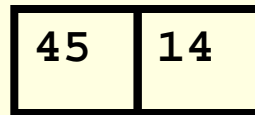
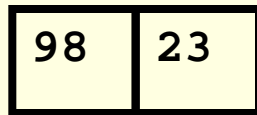
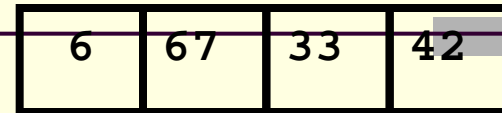
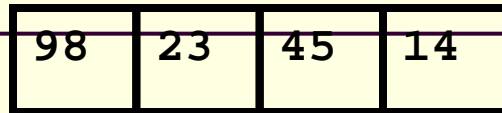
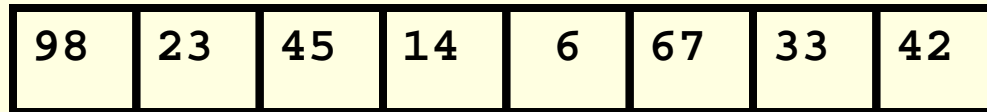


Merge











98	23	45	14	6	67	33	42
----	----	----	----	---	----	----	----

98	23	45	14
----	----	----	----

6	67	33	42
---	----	----	----

98	23
----	----

45	14
----	----

6	67
---	----

33	42
----	----

98
----

23
----

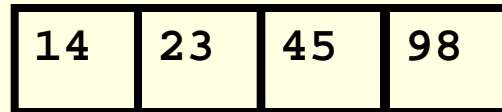
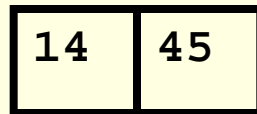
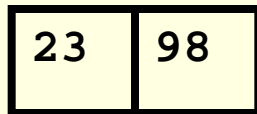
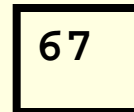
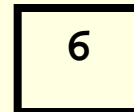
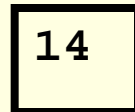
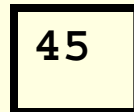
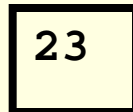
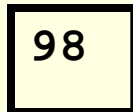
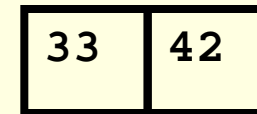
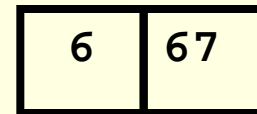
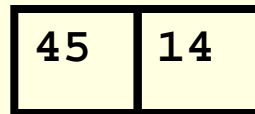
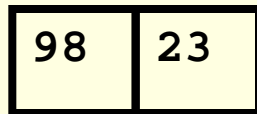
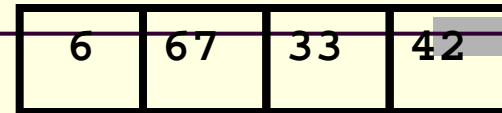
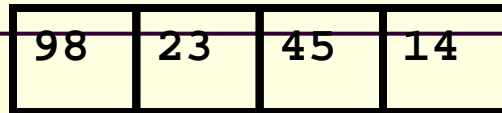
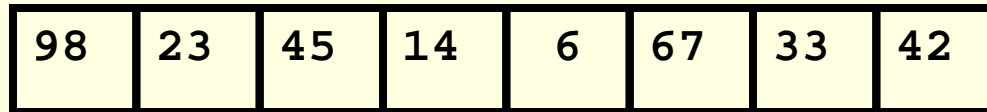
45
----

14
----

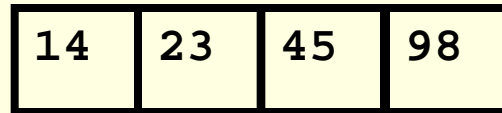
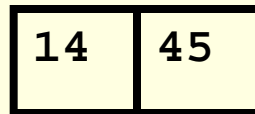
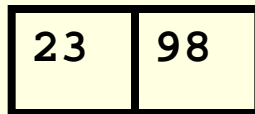
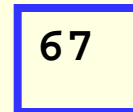
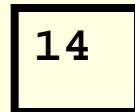
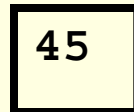
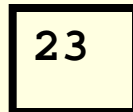
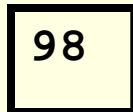
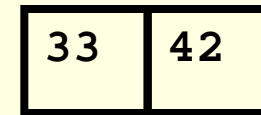
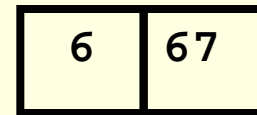
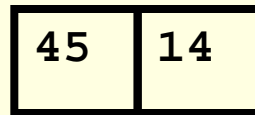
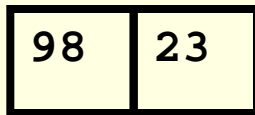
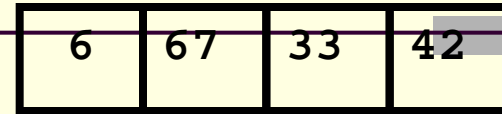
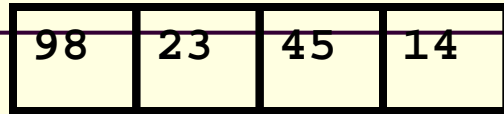
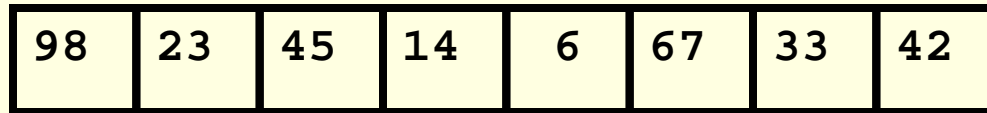
23	98
----	----

14	45
----	----

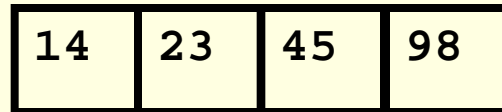
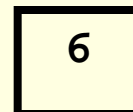
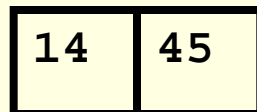
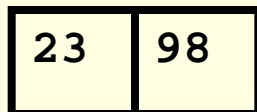
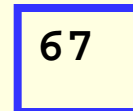
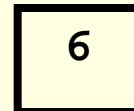
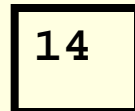
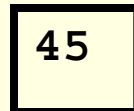
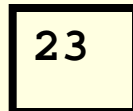
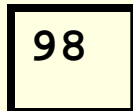
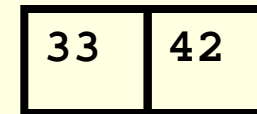
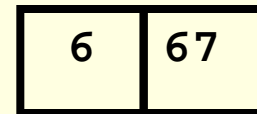
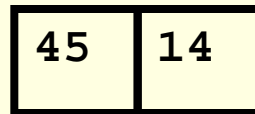
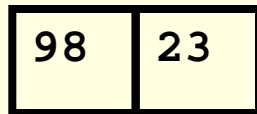
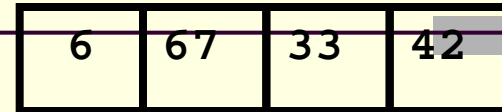
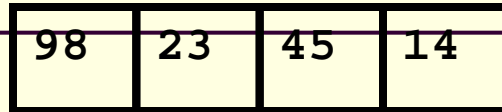
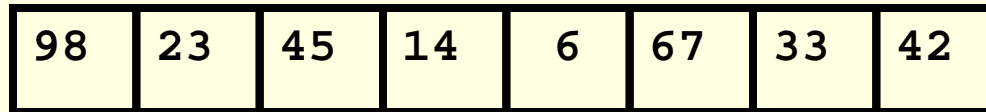
14	23	45	98
----	----	----	----

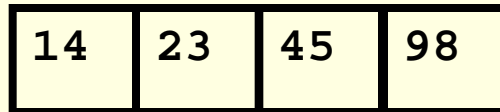
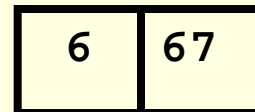
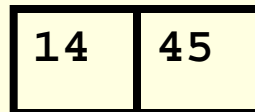
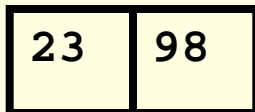
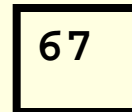
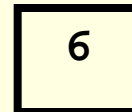
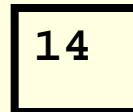
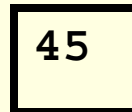
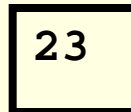
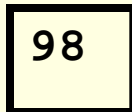
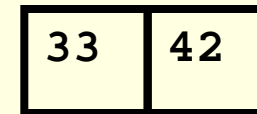
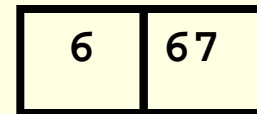
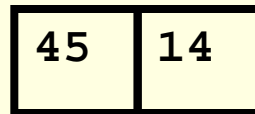
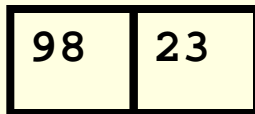
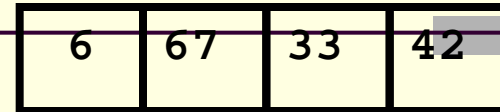
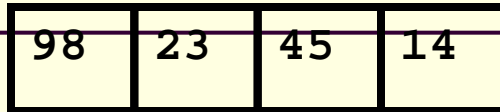
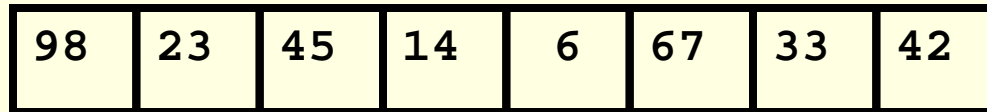






Merge







98	23	45	14	6	67	33	42
----	----	----	----	---	----	----	----

98	23	45	14
----	----	----	----

6	67	33	42
---	----	----	----

98	23
----	----

45	14
----	----

6	67
---	----

33	42
----	----

98
----

23
----

45
----

14
----

6
---

67
----

33
----

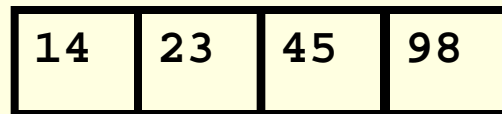
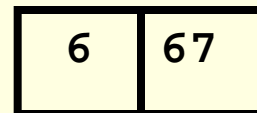
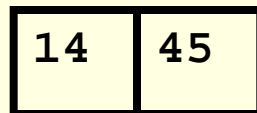
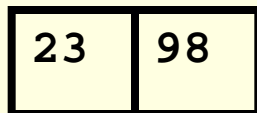
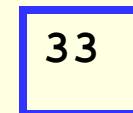
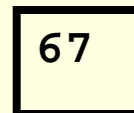
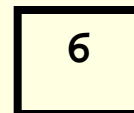
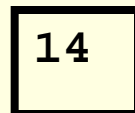
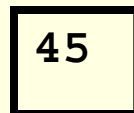
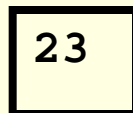
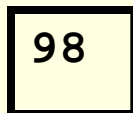
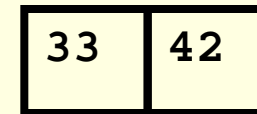
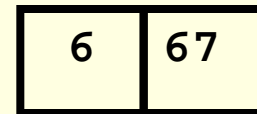
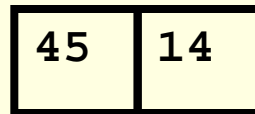
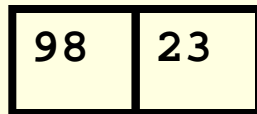
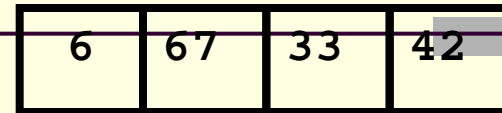
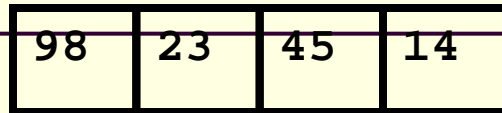
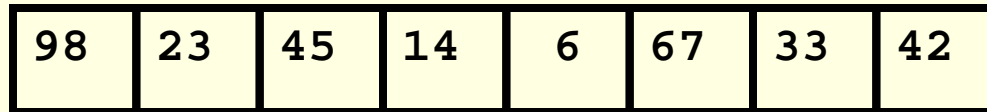
42
----

23	98
----	----

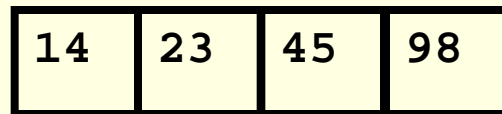
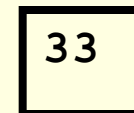
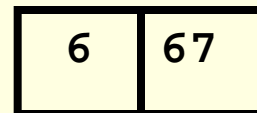
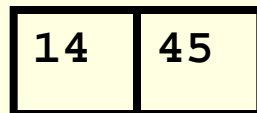
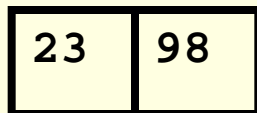
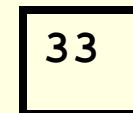
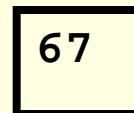
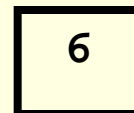
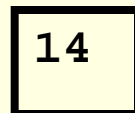
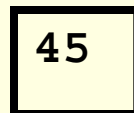
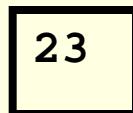
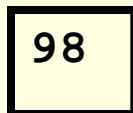
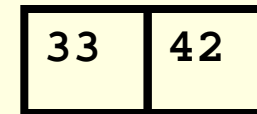
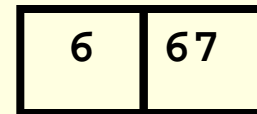
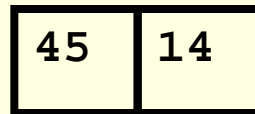
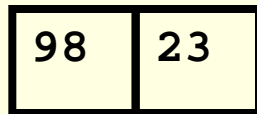
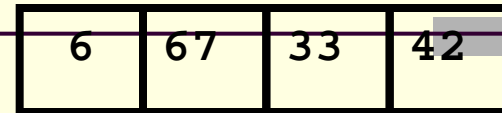
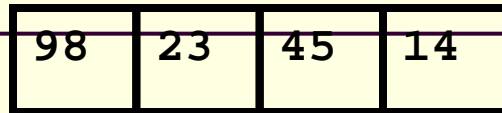
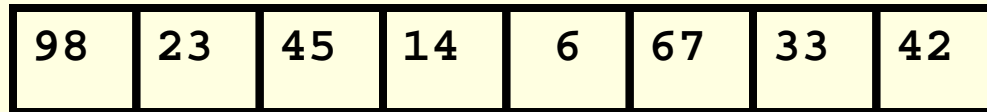
14	45
----	----

6	67
---	----

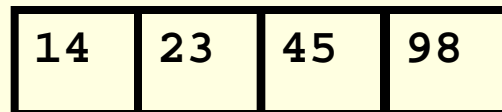
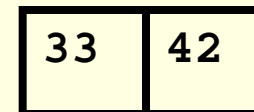
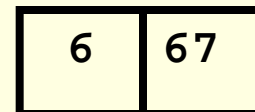
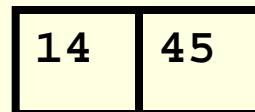
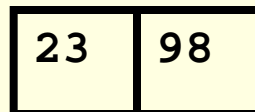
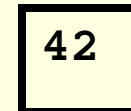
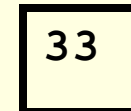
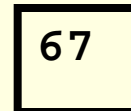
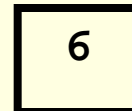
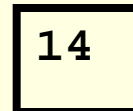
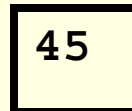
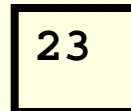
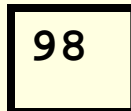
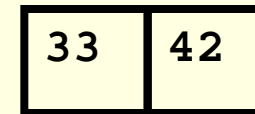
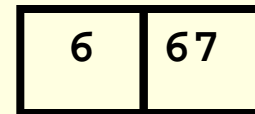
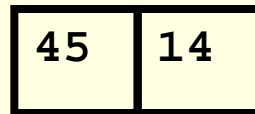
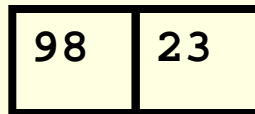
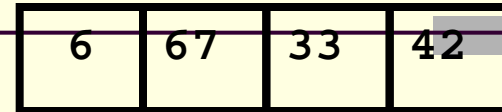
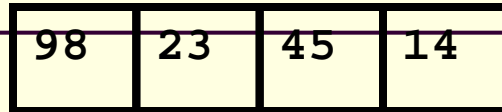
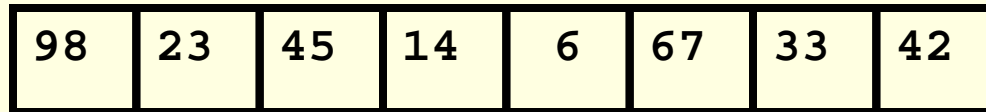
14	23	45	98
----	----	----	----



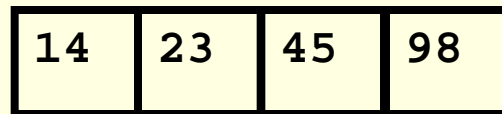
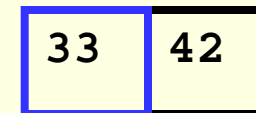
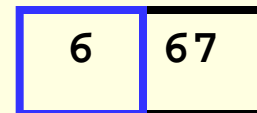
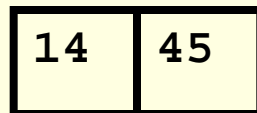
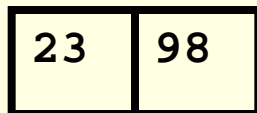
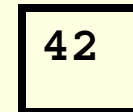
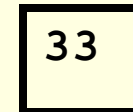
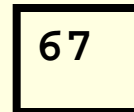
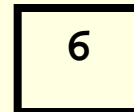
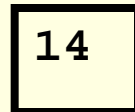
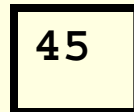
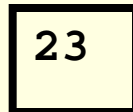
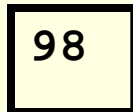
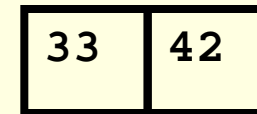
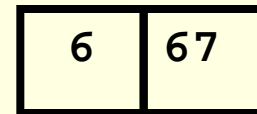
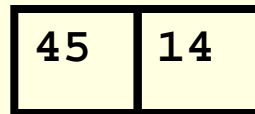
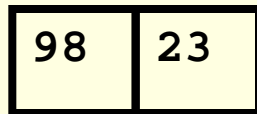
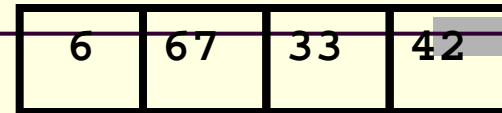
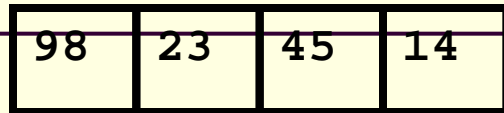
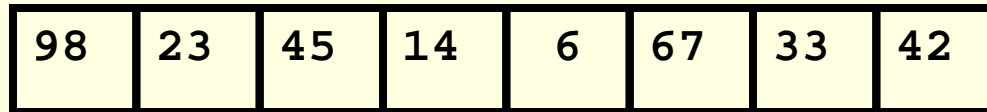
Merge



Merge

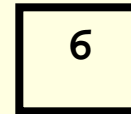
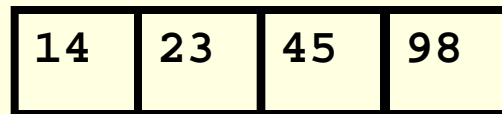
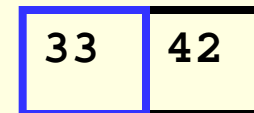
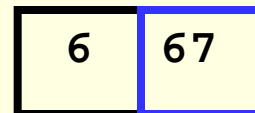
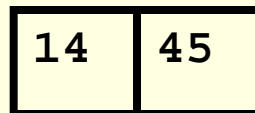
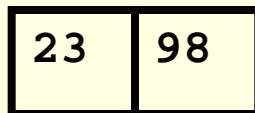
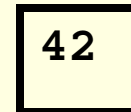
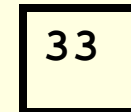
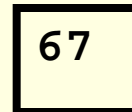
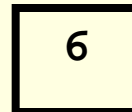
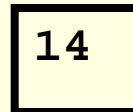
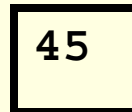
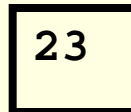
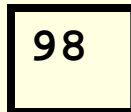
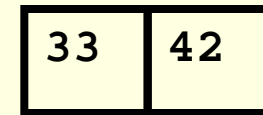
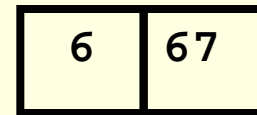
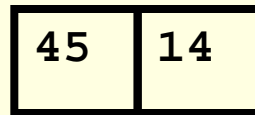
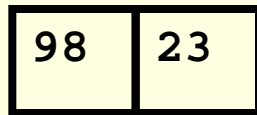
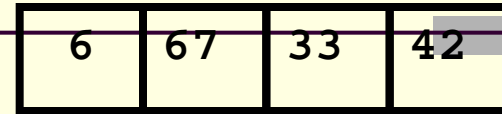
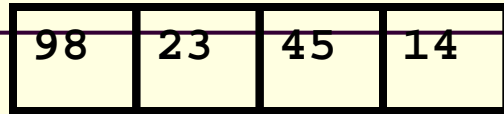
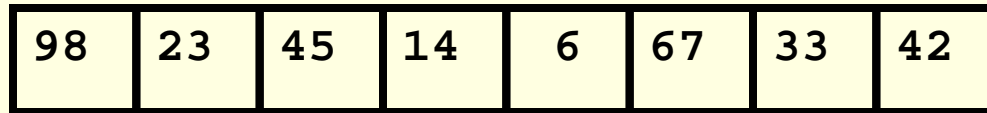


Merge

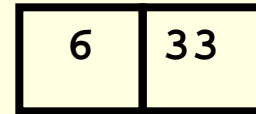
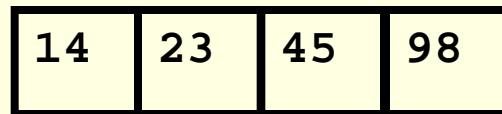
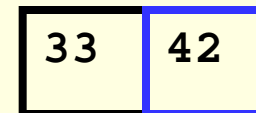
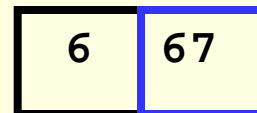
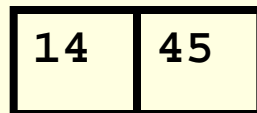
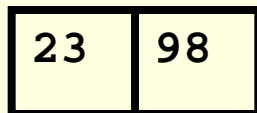
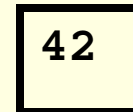
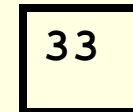
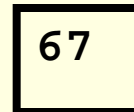
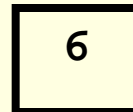
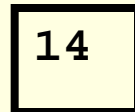
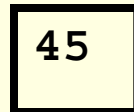
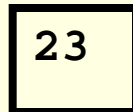
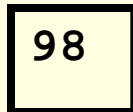
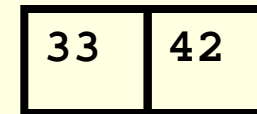
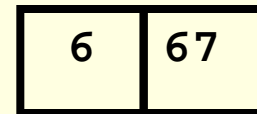
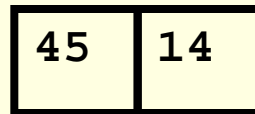
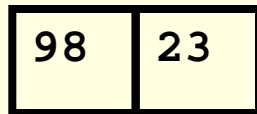
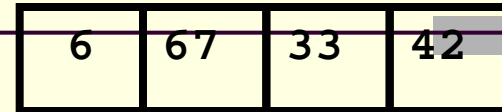
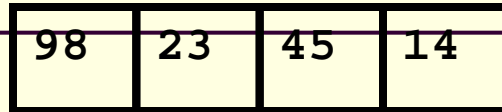
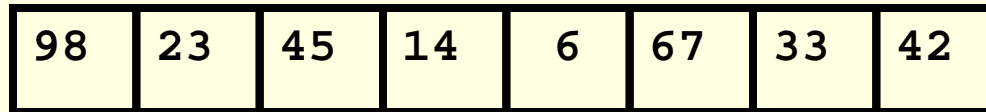


Merge





Merge



Merge



98	23	45	14	6	67	33	42
----	----	----	----	---	----	----	----

98	23	45	14
----	----	----	----

6	67	33	42
---	----	----	----

98	23
----	----

45	14
----	----

6	67
---	----

33	42
----	----

98
----

23
----

45
----

14
----

6
---

67
----

33
----

42
----

23	98
----	----

14	45
----	----

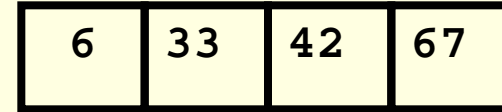
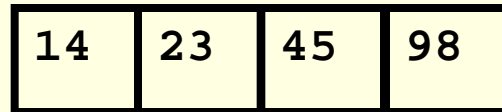
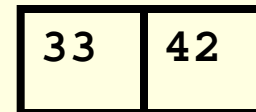
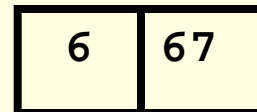
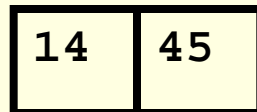
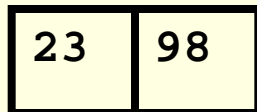
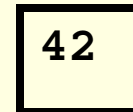
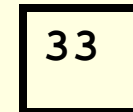
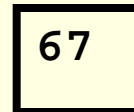
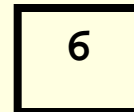
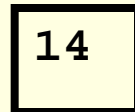
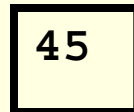
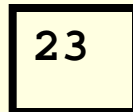
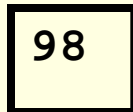
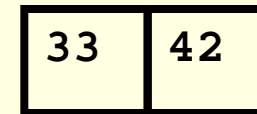
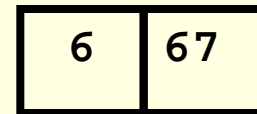
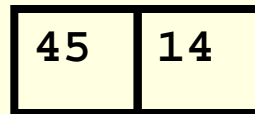
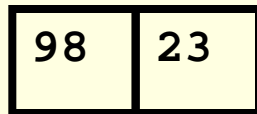
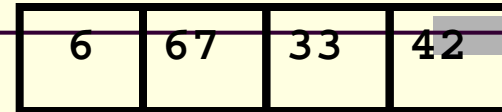
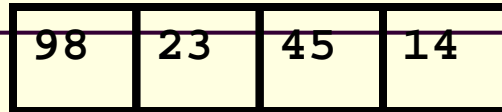
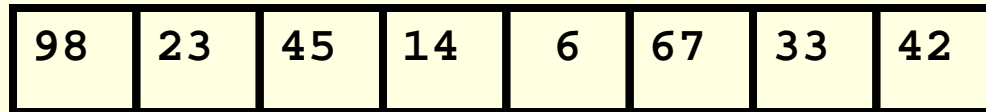
6	67
---	----

33	42
----	----

14	23	45	98
----	----	----	----

6	33	42
---	----	----

Merge





98	23	45	14	6	67	33	42
----	----	----	----	---	----	----	----

98	23	45	14
----	----	----	----

6	67	33	42
---	----	----	----

98	23
----	----

45	14
----	----

6	67
---	----

33	42
----	----

98
----

23
----

45
----

14
----

6
---

67
----

33
----

42
----

23	98
----	----

14	45
----	----

6	67
---	----

33	42
----	----

14	23	45	98
----	----	----	----

6	33	42	67
---	----	----	----

Merge



98	23	45	14	6	67	33	42
----	----	----	----	---	----	----	----

98	23	45	14
----	----	----	----

6	67	33	42
---	----	----	----

98	23
----	----

45	14
----	----

6	67
---	----

33	42
----	----

98
----

23
----

45
----

14
----

6
---

67
----

33
----

42
----

23	98
----	----

14	45
----	----

6	67
---	----

33	42
----	----

14	23	45	98
----	----	----	----

6	33	42	67
---	----	----	----

6
---

Merge



98	23	45	14	6	67	33	42
----	----	----	----	---	----	----	----

98	23	45	14
----	----	----	----

6	67	33	42
---	----	----	----

98	23
----	----

45	14
----	----

6	67
---	----

33	42
----	----

98
----

23
----

45
----

14
----

6
---

67
----

33
----

42
----

23	98
----	----

14	45
----	----

6	67
---	----

33	42
----	----

14	23	45	98
----	----	----	----

6	33	42	67
---	----	----	----

6	14
---	----

Merge



98	23	45	14	6	67	33	42
----	----	----	----	---	----	----	----

98	23	45	14
----	----	----	----

6	67	33	42
---	----	----	----

98	23
----	----

45	14
----	----

6	67
---	----

33	42
----	----

98
----

23
----

45
----

14
----

6
---

67
----

33
----

42
----

23	98
----	----

14	45
----	----

6	67
---	----

33	42
----	----

14	23	45	98
----	----	----	----

6	33	42	67
---	----	----	----

6	14	23
---	----	----

Merge





98	23	45	14	6	67	33	42
----	----	----	----	---	----	----	----

98	23	45	14
----	----	----	----

6	67	33	42
---	----	----	----

98	23
----	----

45	14
----	----

6	67
---	----

33	42
----	----

98
----

23
----

45
----

14
----

6
---

67
----

33
----

42
----

23	98
----	----

14	45
----	----

6	67
---	----

33	42
----	----

14	23	45	98
----	----	----	----

6	33	42	67
---	----	----	----

6	14	23	33
---	----	----	----

Merge



98	23	45	14	6	67	33	42
----	----	----	----	---	----	----	----

98	23	45	14
----	----	----	----

6	67	33	42
---	----	----	----

98	23
----	----

45	14
----	----

6	67
---	----

33	42
----	----

98
----

23
----

45
----

14
----

6
---

67
----

33
----

42
----

23	98
----	----

14	45
----	----

6	67
---	----

33	42
----	----

14	23	45	98
----	----	----	----

6	33	42	67
---	----	----	----

6	14	23	33	42
---	----	----	----	----

Merge



98	23	45	14	6	67	33	42
----	----	----	----	---	----	----	----

98	23	45	14
----	----	----	----

6	67	33	42
---	----	----	----

98	23
----	----

45	14
----	----

6	67
---	----

33	42
----	----

98
----

23
----

45
----

14
----

6
---

67
----

33
----

42
----

23	98
----	----

14	45
----	----

6	67
---	----

33	42
----	----

14	23	45	98
----	----	----	----

6	33	42	67
---	----	----	----

6	14	23	33	42	45
---	----	----	----	----	----

Merge



98	23	45	14	6	67	33	42
----	----	----	----	---	----	----	----

98	23	45	14
----	----	----	----

6	67	33	42
---	----	----	----

98	23
----	----

45	14
----	----

6	67
---	----

33	42
----	----

98
----

23
----

45
----

14
----

6
---

67
----

33
----

42
----

23	98
----	----

14	45
----	----

6	67
---	----

33	42
----	----

14	23	45	98
----	----	----	----

6	33	42	67
---	----	----	----

6	14	23	33	42	45	67
---	----	----	----	----	----	----

Merge



98	23	45	14	6	67	33	42
----	----	----	----	---	----	----	----

98	23	45	14
----	----	----	----

6	67	33	42
---	----	----	----

98	23
----	----

45	14
----	----

6	67
---	----

33	42
----	----

98
----

23
----

45
----

14
----

6
---

67
----

33
----

42
----

23	98
----	----

14	45
----	----

6	67
---	----

33	42
----	----

14	23	45	98
----	----	----	----

6	33	42	67
---	----	----	----

6	14	23	33	42	45	67	98
---	----	----	----	----	----	----	----

Merge



98	23	45	14	6	67	33	42
----	----	----	----	---	----	----	----

98	23	45	14
----	----	----	----

6	67	33	42
---	----	----	----

98	23
----	----

45	14
----	----

6	67
---	----

33	42
----	----

98
----

23
----

45
----

14
----

6
---

67
----

33
----

42
----

23	98
----	----

14	45
----	----

6	67
---	----

33	42
----	----

14	23	45	98
----	----	----	----

6	33	42	67
---	----	----	----

6	14	23	33	42	45	67	98
---	----	----	----	----	----	----	----



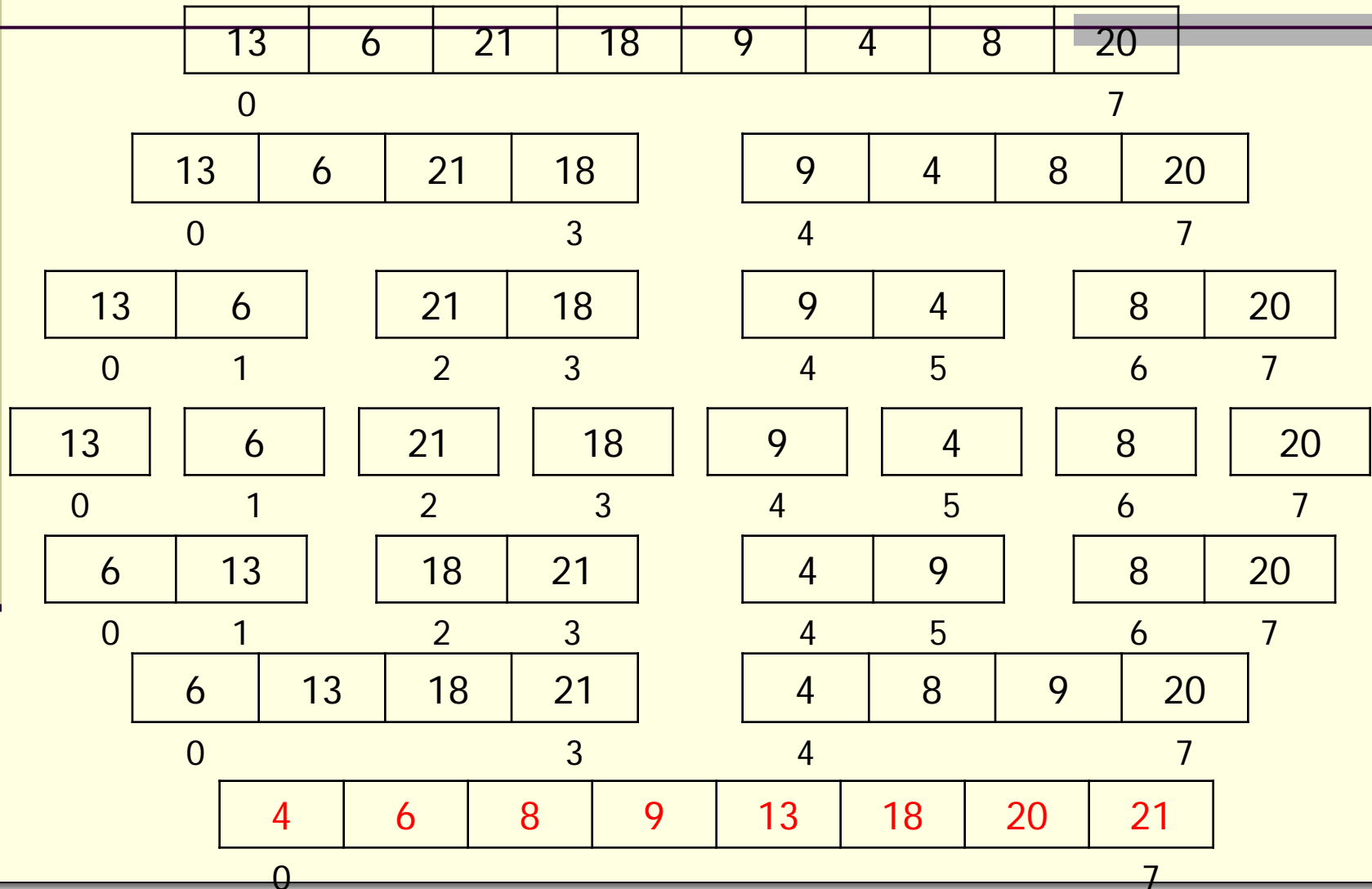
98	23	45	14	6	67	33	42
----	----	----	----	---	----	----	----



6	14	23	33	42	45	67	98
---	----	----	----	----	----	----	----



# Sorting: Merge Sort Example #2







# Brief Interlude: FAIL Picture





# UCF Weekly Bike Fail



Courtesy of  
Sean Lunceford





# UCF Weekly Bike Fail



Courtesy of  
Sean Lunceford



# Sorting: Merge Sort

## ■ Merge Sort Code

```
void MergeSort(int values[], int start, int end) {
    int mid;
    // Check if our sorting range is more than one element.
    if (start < end) {

        mid = (start+end)/2;

        // Sort the first half of the values.
        MergeSort(values, start, mid);

        // Sort the last half of the values.
        MergeSort(values, mid+1, end);

        // Put it all together.
        Merge(values, start, mid+1, end);
    }
}
```



# Sorting: Merge Sort

---

## ■ Merge Code

- This code is longer
- And a bit convoluted
  - But all it does it Merge the values from two arrays into one larger array
  - Of course, keeping the items in order
  - Just like the example shown earlier in the slides
- Code can be found here on the website:
  - <http://www.cs.ucf.edu/courses/cop3502/spr2011/programs/sorting/mergesort.c>
  - You need to fully understand how this code works
    - Including the Merge function!



# Sorting: Merge Sort

---

- Merge Sort Analysis
  - Again, here are the steps of Merge Sort:
    - 1) Merge Sort the first half of the list
    - 2) Merge Sort the second half of the list
    - 3) Merge both halves together
  
  - Let  $T(n)$  be the running time of Merge Sort on an input size  $n$
  - Then we have:
    - $T(n) = (\text{Time in step 1}) + (\text{Time in step 2}) + (\text{Time in step 3})$



# Sorting: Merge Sort

- Merge Sort Analysis
  - $T(n)$ : running time of Merge Sort on input size  $n$
  - Therefore, we have:
    - $T(n) = (\text{Time in step 1}) + (\text{Time in step 2}) + (\text{Time in step 3})$
  - Notice that Step 1 and Step 2 are sorting problems also
    - But they are of size  $n/2$ ...we are halving the input
  - And the Merge function runs in  $O(n)$  time
  - Thus, we get the following equation for  $T(n)$
  - $T(n) = T(n/2) + T(n/2) + O(n)$
  - $T(n) = 2T(n/2) + O(n)$



# Sorting: Merge Sort

---

- Merge Sort Analysis
  - $T(n) = 2T(n/2) + O(n)$
  - For the time being, let's simplify  $O(n)$  to just  $n$
  - $T(n) = 2T(n/2) + n$
  - and we know that  $T(1) = 1$
  - So we now have a Recurrence Relation
  - Is it solved?
    - NO!
  - Why?
    - Damn T's!





# Sorting: Merge Sort

---

## ■ Merge Sort Analysis

- $T(n) = 2T(n/2) + n$       and       $T(1) = 1$
- So we need to solve this, by removing the  $T(\dots)$ 's from the right hand side
- Then  $T(n)$  will be in its closed form
- And we can state its Big-O running time
- We do this in steps
  - We replace  $n$  with  $n/2$  on both sides of the equation
  - We plug the result back in
  - And then we do it again...till a “light goes off” and we see something



# Sorting: Merge Sort

## ■ Merge Sort Analysis

- $T(n) = 2T(n/2) + n$       and       $T(1) = 1$
- Do you know what  $T(n/2)$  equals
  - Does it equal 2,125 operations? We don't know!
- So we need to develop an equation for  $T(n/2)$
- How?
- Take the original equation shown above
- **Wherever you see an 'n', substitute with 'n/2'**
- $T(n/2) = 2T(n/4) + n/2$
- So now we have an equation for  $T(n/2)$



# Sorting: Merge Sort

## ■ Merge Sort Analysis

- $T(n) = 2T(n/2) + n$       and       $T(1) = 1$
- $T(n/2) = 2T(n/4) + n/2$
- So now we have an equation for  $T(n/2)$ 
  - We can take this equation and substitute it back into the original equation
- $T(n) = 2T(n/2) + n = 2[2T(n/4) + n/2] + n$ 
  - now simplify
- $T(n) = 4T(n/4) + 2n$ 
  - Same thing here: do you know what  $T(n/4)$  equals?
  - No we don't! So we need to develop an eqn for  $T(n/4)$



# Sorting: Merge Sort

---

## ■ Merge Sort Analysis

- $T(n) = 2T(n/2) + n$       and       $T(1) = 1$
- $T(n/2) = 2T(n/4) + n/2$
- $T(n) = 4T(n/4) + 2n$ 
  - Same thing here: do you know what  $T(n/4)$  equals?
  - No we don't! So we need to develop an eqn for  $T(n/4)$
  - Take the eqn above and again substitute 'n/2' for 'n'
- $T(n/4) = 2T(n/8) + n/4$
- So now we have an equation for  $T(n/4)$ 
  - We can take this equation and substitute it back the equation that we currently have in terms of  $T(n/4)$



# Sorting: Merge Sort

## ■ Merge Sort Analysis

- $T(n) = 2T(n/2) + n$       and       $T(1) = 1$
- $T(n/2) = 2T(n/4) + n/2$
- $T(n) = 4T(n/4) + 2n$
- $T(n/4) = 2T(n/8) + n/4$
- So now we have an equation for  $T(n/4)$ 
  - We can take this equation and substitute it back the equation that we currently have in terms of  $T(n/4)$
- $T(n) = 4T(n/4) + 2n = 4[2T(n/8) + n/4] + 2n$ 
  - Simplify a bit
- $T(n) = 8T(n/8) + 3n$



# Sorting: Merge Sort

## ■ Merge Sort Analysis

- So now we have three equations for  $T(n)$ :

- $T(n) = 2T(n/2) + n$        $\leftarrow$  1<sup>st</sup> step of recursion

- $T(n) = 4T(n/4) + 2n$        $\leftarrow$  2<sup>nd</sup> step of recursion

- $T(n) = 8T(n/8) + 3n$        $\leftarrow$  3<sup>rd</sup> step of recursion

- So on the  $k$ th step/stage of the recursion, we get a generalized recurrence relation:

- $T(n) = 2^k T(n/2^k) + kn$        $\leftarrow$   $k^{\text{th}}$  step of recursion

- Whew! So now we're done right? Wrong!



# Sorting: Merge Sort

---

- Merge Sort Analysis
  - So on the  $k$ th step/stage of the recursion, we get a generalized recurrence relation:
  - $T(n) = 2^k T(n/2^k) + kn$
  - We need to get rid of the  $T(\dots)$ 's on the right side
  - Remember, we know  $T(1) = 1$
  - So we make a substitution:
    - Let  $n = 2^k$
    - and also solve for  $k$
    - $k = \log_2 n$
  - Plug these back in...



# Sorting: Merge Sort

- Merge Sort Analysis
  - So on the  $k$ th step/stage of the recursion, we get a generalized recurrence relation:
  - $T(n) = 2^k T(n/2^k) + kn$ 
    - Let  $n = 2^k$
    - and also solve for  $k$
    - $k = \log_2 n$
  - Plug these back in...
  - $T(n) = 2^{\log_2 n} T(n/n) + (\log_2 n)n$
  - $T(n) = n * T(1) + n \log n = n + n * \log n$
  - So Merge Sort runs in  $O(n * \log n)$  time





# Sorting: Merge Sort

---

## ■ Merge Sort Summary

- Avoids all the unnecessary swaps of  $n^2$  sorts
- Uses recursion to split up a list until we get to “lists” of 1 or 0 elements
- Uses a Merge function to merge (“sort”) these smaller lists into larger lists
- Is MUCH faster than  $n^2$  sorts
- Merge Sort runs in  $O(n \log n)$  time



# Sorting: Merge Sort

---

**WASN'T  
THAT  
THE COOLEST!**



# Daily Demotivator

---



# Sorting: Merge Sort



Computer Science Department  
University of Central Florida

*COP 3502 – Computer Science I*