Merge Sort

Problem with Bubble/Insertion/Selection

- They make a large number of comparisons and swaps between elements
- $O(n^2)$
- There are other clever way to sort numbers
- One of them is Merge Sort

Merge Sort

- Concepts: Divide and conquer
- 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 recursively calling Merge Sort on the two smaller lists
3. **Merge** the two sub-lists back into one sorted list

- Given a list:
  - Split this list into two lists of about half the size
  - Then, 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 ¼ 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

Ideas behind Merge Sort efficiency

- So, merge sort incorporates two main ideas to improve the runtime:
- A small list will take fewer steps to sort than a large list
- 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

Pseudo code steps

```
MergeSort(arr[], l,  r)
If r > l
    1. Find the middle point to divide the array into two halves:
          middle m = (l+r)/2

    2. Call mergeSort for first half:
          Call mergeSort(arr, l, m)

    3. Call mergeSort for second half:
          Call mergeSort(arr, m+1, r)

    4. Merge the two halves sorted in step 2 and 3:
          Call merge(arr, l, m, r)
```

Merge sort simulation

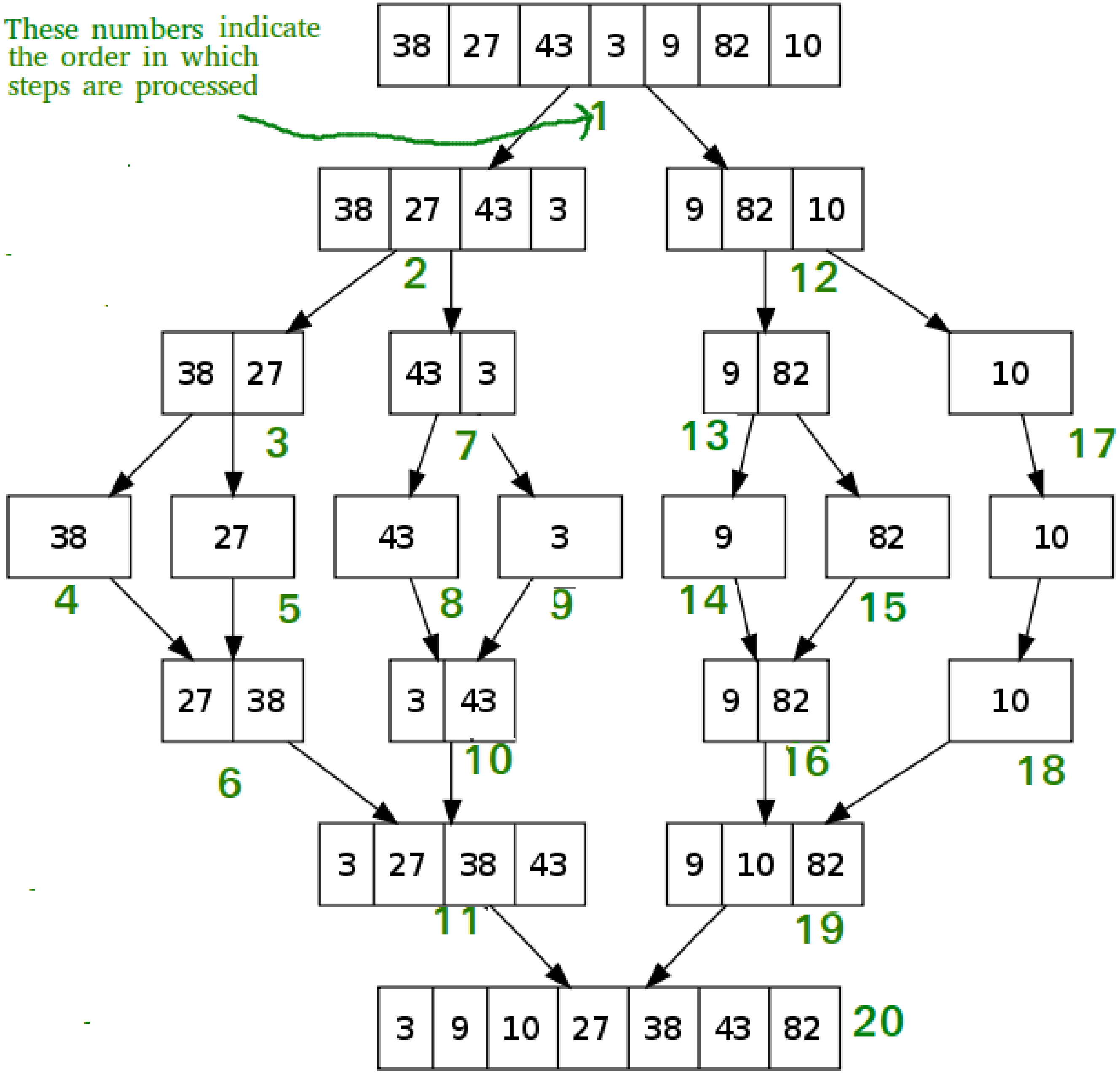


Image source: https://www.geeksforgeeks.org/merge-sort/

Merge function

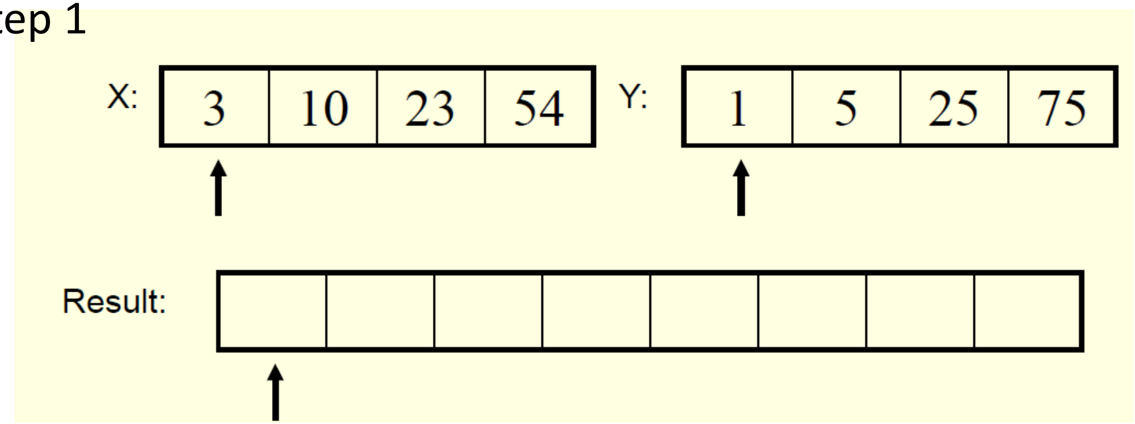
- The key to Merge Sort: the **Merge** function
- Given two sorted lists, **Merge** them into one sorted list
- Problem:
  - You are given two arrays, each of which is already sorted
  - Your job is to efficiently combine the two arrays into one larger array
- The larger array should contain all the values of the two smaller arrays
- Finally, the larger array should be in sorted order
- Example:
  - List1 = {3,8,9} and List2 = {1,5,7}
  - Merge(List1, List2) = {1,3,5,7,8,9}

Merge Function

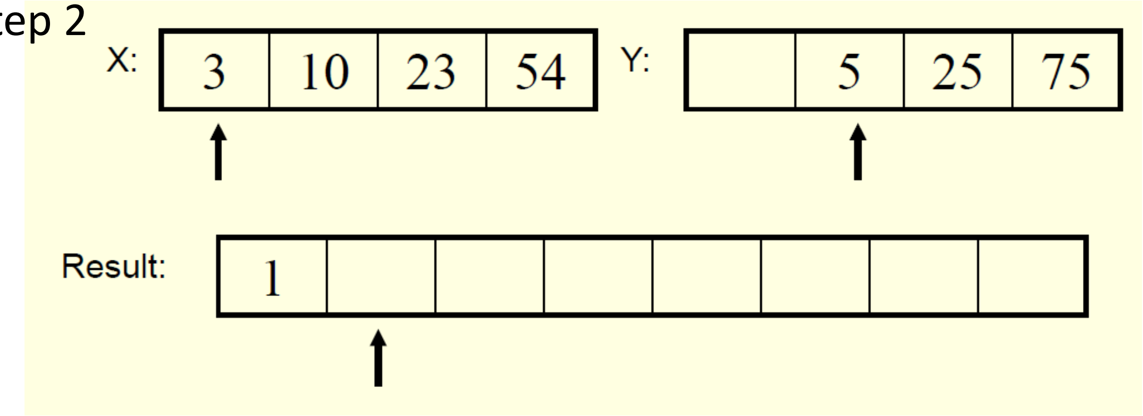
- Solution:
- The merge function fillip a larger array in sorted order from the data in smaller array.
- We keep track of the smallest value in each array that hasn’t been placed, in order, in the larger array yet
- Compare these two smallest values from each array
  - One of these MUST be the smallest 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

Example of Merge function

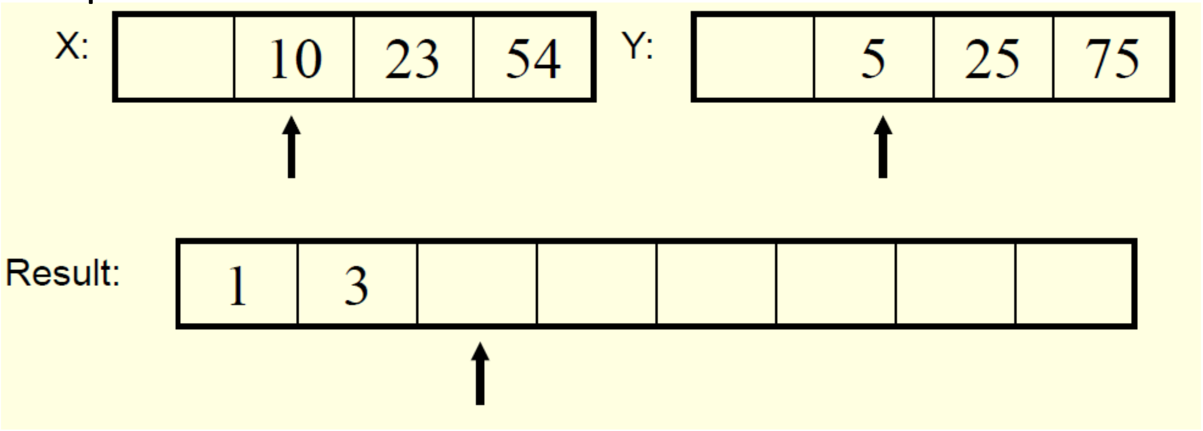
Step 1



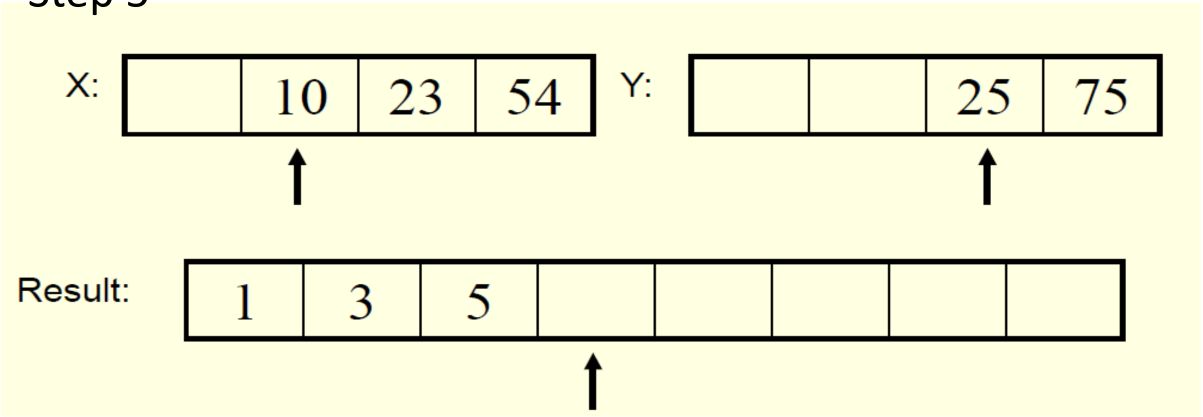
Step 2



Step 4

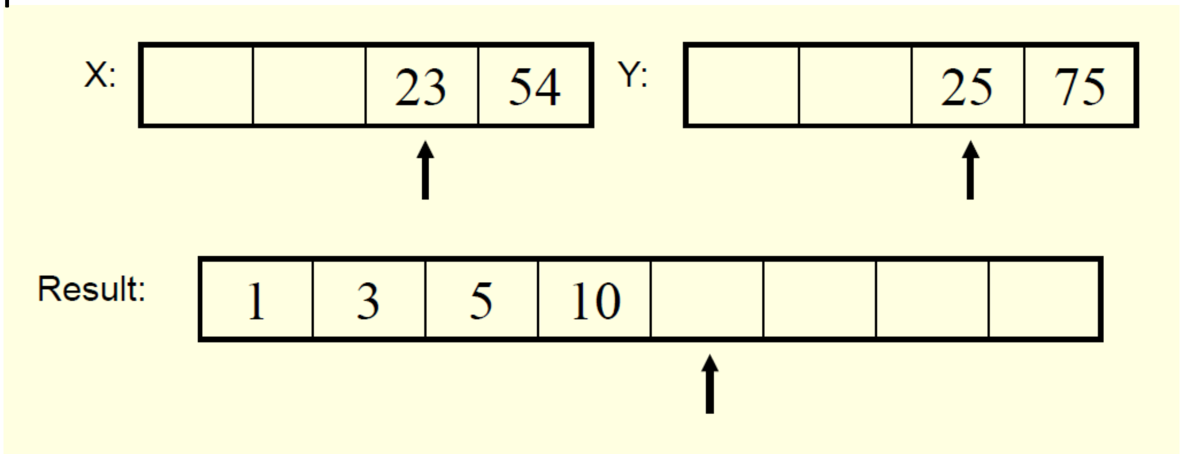


Step 5

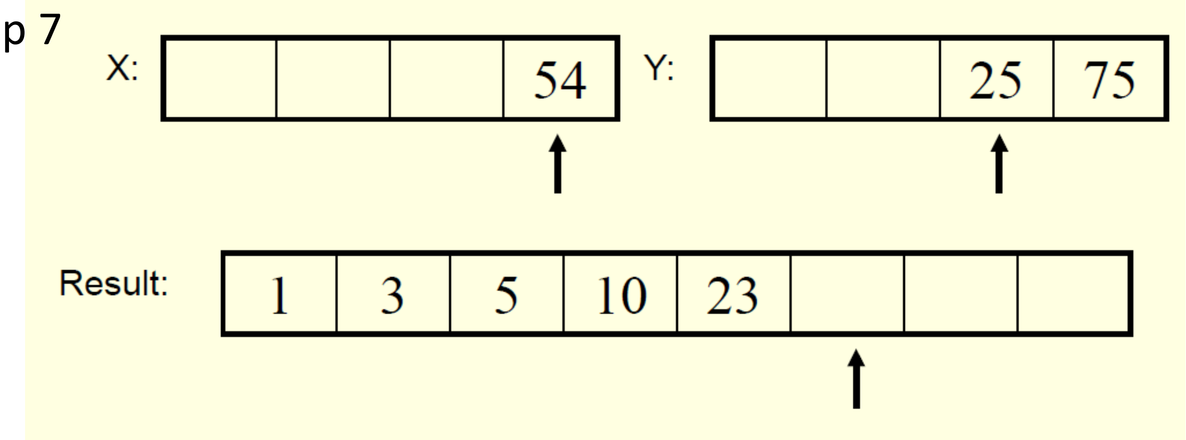


Example of Merge function Continue

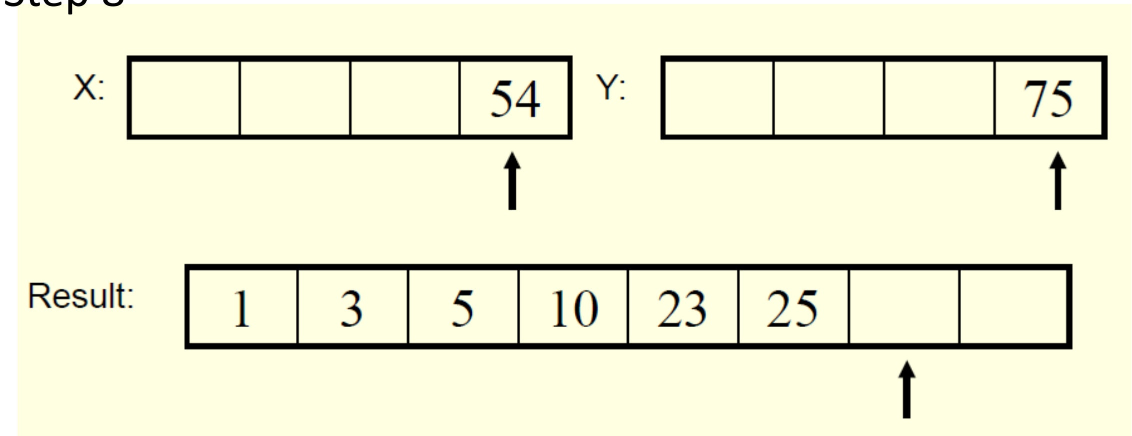
Step 6



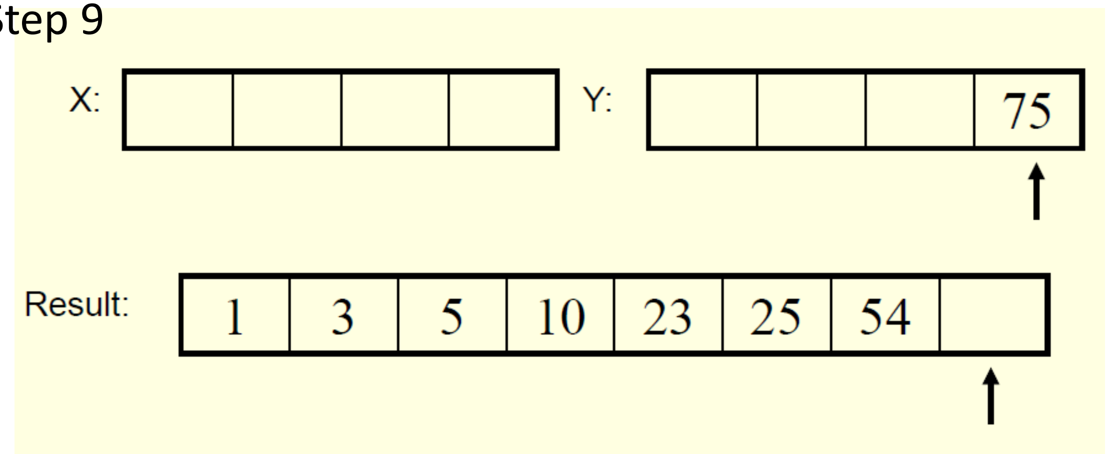
Step 7



Step 8

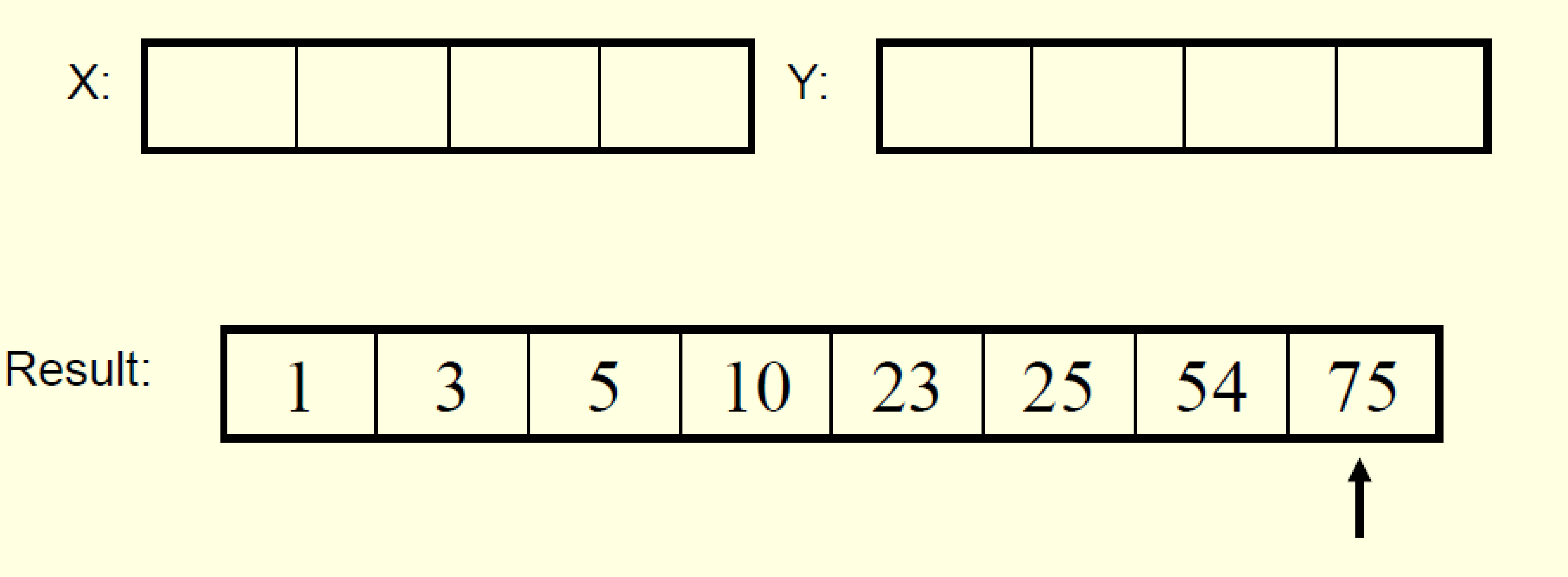


Step 9



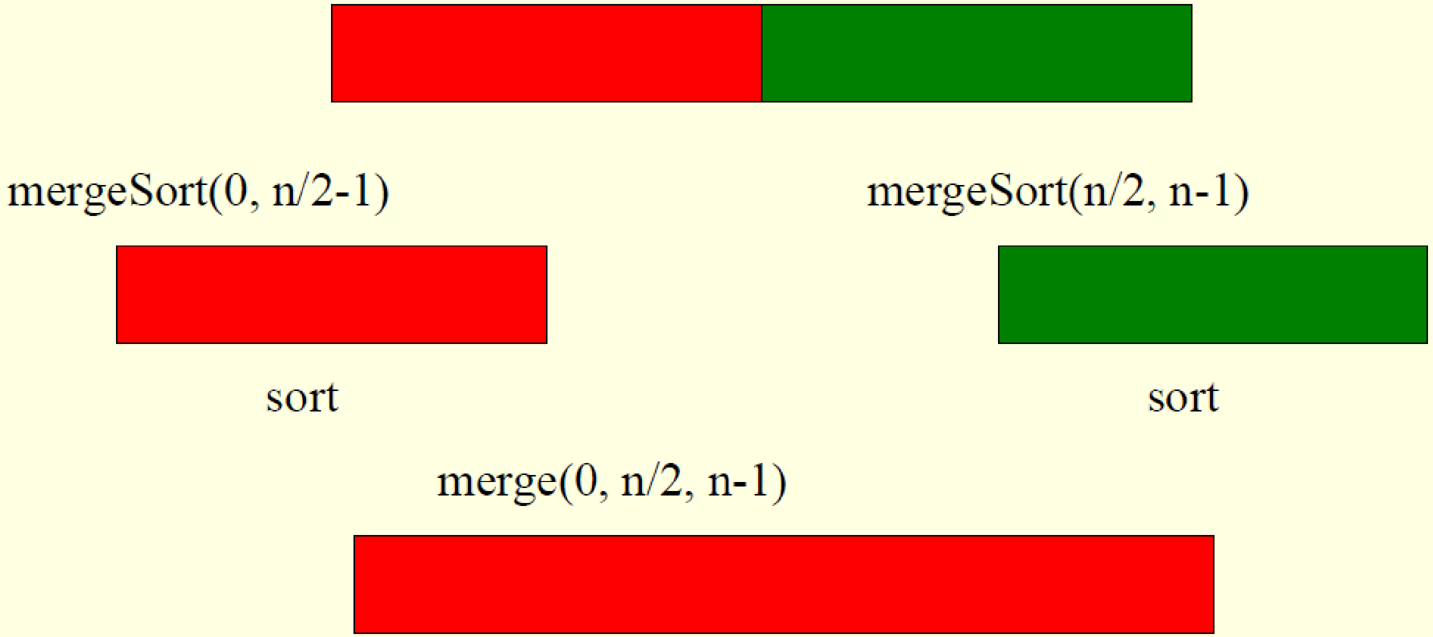
After Merging

Finally:



Going back to the Merge Sort

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



Remember about the Full Merge sort simulation

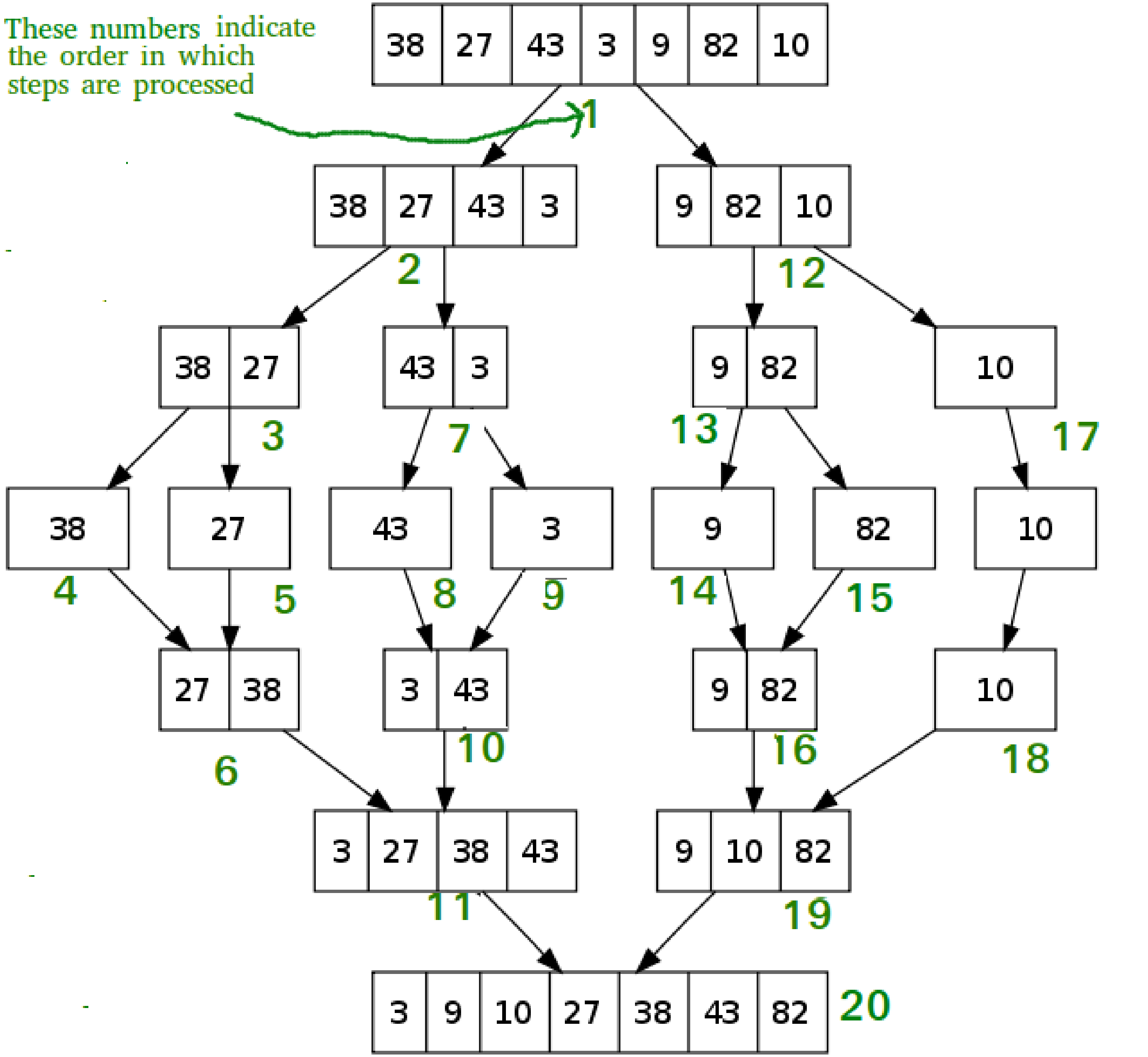
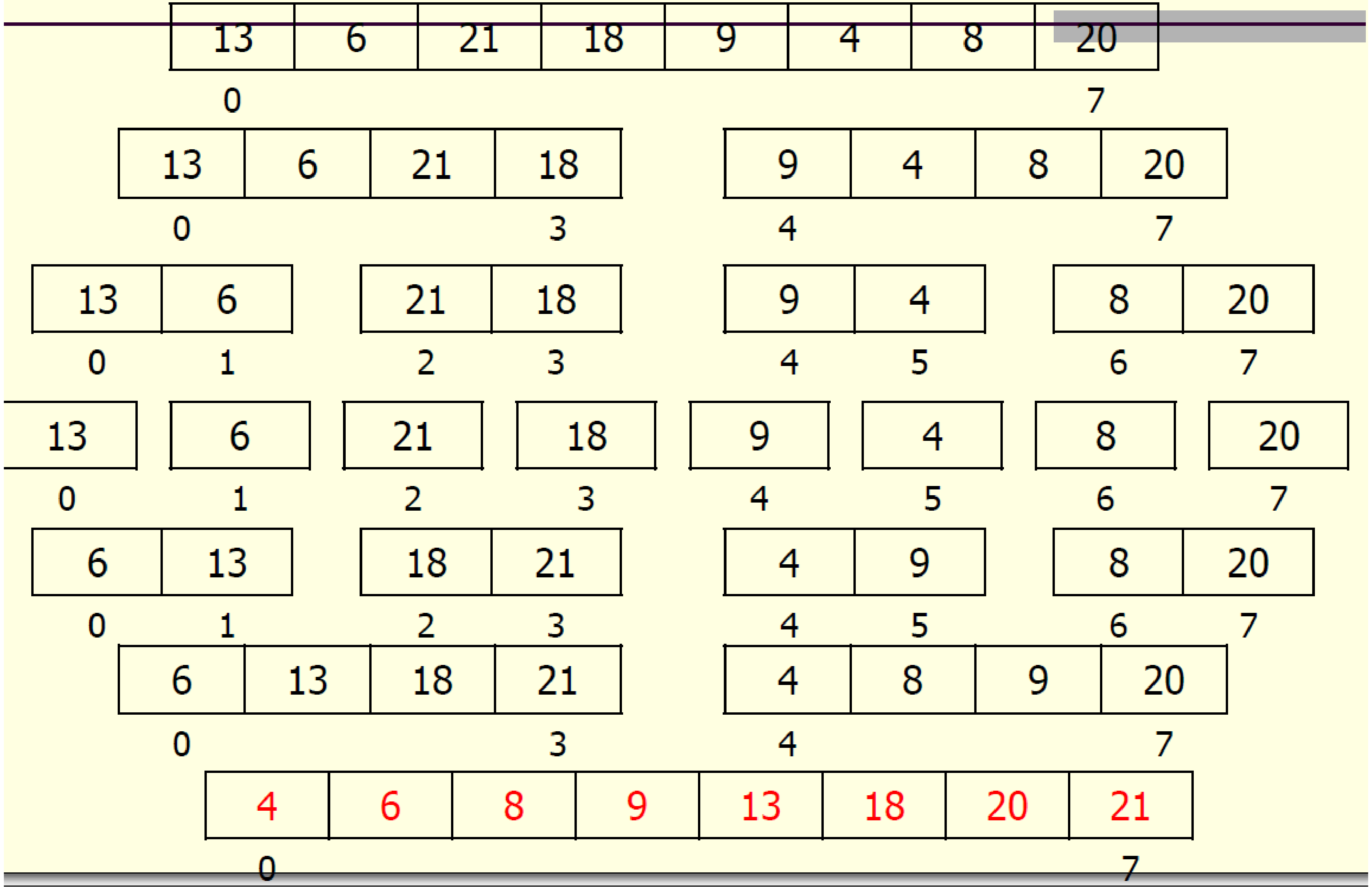


Image source: https://www.geeksforgeeks.org/merge-sort/

Sorting: Merge Sort Example #2



Going back to the Pseudo code steps

```
MergeSort(arr[], l,  r)
If r > l
    1. Find the middle point to divide the array into two halves:
         middle m = (l+r)/2

    2. Call mergeSort for first half:
         Call mergeSort(arr, l, m)

    3. Call mergeSort for second half:
         Call mergeSort(arr, m+1, r)

    4. Merge the two halves sorted in step 2 and 3:
         Call merge(arr, l, m, r)
```

The full source code is available in webcourses

Code of Merge sort

- The merge sort code is long to explain in slide.
- The example source code is available in webcourses.
- Now, we will go through the code and test it in the class.

Analysis of Merge Sort

- Remember the steps:

1. Merge Sort the first half of the array
2. Merge sort the second half of the array
3. Merge both halves together

- Let T(n) be the running time for an input size n
- So, T(n) = (time in step 1) + (time in step 2) + time in step 3)

Analysis of Merge Sort

- T(n) = (time in step 1) + (time in step 2) + time in step 3)
- Step 1 and Step 2 are sorting problem and they are of size n/2….the input size get halves.
- The merge function runs in O(n) time
- So, T(n) = T(n/2) + T(n/2) + O(n)
- T(n) = 2(T(n/2) + O(n)
- For the time being, let’s simplify O(n) to just n)
- So, T(n) = 2T(n/2) + n
- and we know that T(1) = 1
- So we now have a Recurrence Relation
- So, let’s solve it

Analysis of Merge Sort

- $T(n) = 2T(n/2) + n$ and $T(1) = 1$
- So we now have a Recurrence Relation
- Calculate $T(n/2)$ by replacing n by n/2: $T(n/2) = 2T(n/4) + n/2$
- So, $T(n) = 2T(n/2) + n = 2[2T(n/4) + n/2] + n$ [We substituted $T(n/2)$]
- $T(n) = 4T(n/4) + 2n$
- Calculate $T(n/4)$: $T(n/4) = 2T(n/8) + n/4$
- Now substitute $T(n/4)$:
- $T(n) = 4T(n/4) + 2n = 4[2T(n/8) + n/4] + 2n$
- Simplify: $T(n) = 8T(n/8) + 3n$

Analysis of Merge Sort

- Let’s find a pattern:
- $T(n) = 2T(n/2) + n$ // 1st step of recursion
- $T(n) = 4T(n/4) + 2n$ //2nd step of recursion
- $T(n) = 8T(n/8) + 3n$ //3rd step of recursion

- So on the kth step or stage of the recursion, we get a generalized recurrence relation:
- $T(n) = 2^kT(n/2^k) +kn$ //for kth step of recursion
- Are we done?

Analysis of Merge Sort

- We need to remove T(…) from: $2^kT(n/2^k) + kn$
- We know that $T(1) = 1$
- So make a substitution:
- $n/2^k = 1$
- So, $n = 2^k$
- Thus, $k = \log_2 n$
- So, $T(n) = 2^{\log_2 n}\, T(1) + (\log_2 n)\, n$
- So, $T(n) = n\,T(1) + n \log n = n + n \log n$
- So merge sort runs in : $O(n*\log n)$ time

Acknowledgement and more materials

- Some slides are taken from lecture notes of Prof Dr. Jonathan Cazalas:

More references:

Arup's note on merge sort (read on your own): http://www.cs.ucf.edu/~dmarino/ucf/transparency/cop3502/lec/MergeSort-20.doc

Another version of the code that does automatic testing:

http://www.cs.ucf.edu/~dmarino/ucf/transparency/cop3502/sampleprogs/mergesort.c