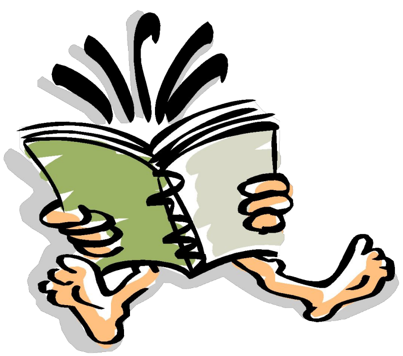
Merge Sort

Week 4 Lesson 2



Sorting

- Insertion sort
  - Design approach: incremental
  - Sorts in place: Yes
  - Best case: $\Theta(n)$
  - Worst case: $\Theta(n^2)$

- Bubble Sort
  - Design approach: incremental
  - Sorts in place: Yes
  - Running time: $\Theta(n^2)$

Sorting

- Selection sort
  - Design approach: incremental
  - Sorts in place: Yes
  - Running time: $\Theta(n^2)$
- Merge Sort
  - Design approach: divide and conquer
  - Sorts in place: No
  - Running time: *Let's see!!*

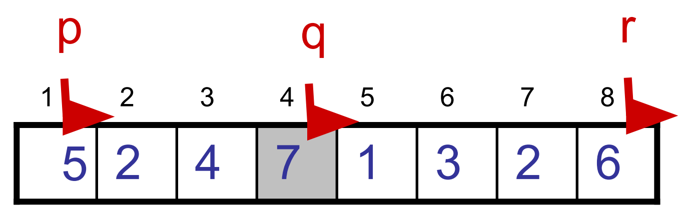
Divide-and-Conquer

- **Divide** the problem into a number of sub-problems
  - Similar sub-problems of smaller size
- **Conquer** the sub-problems
  - Solve the sub-problems recursively
  - Sub-problem size small enough $\Rightarrow$ solve the problems in straightforward manner
- **Combine** the solutions of the sub-problems
  - Obtain the solution for the original problem

Merge Sort Approach

- To sort an array $A[p \,..\, r]$:
- **Divide**
  - Divide the n-element sequence to be sorted into two subsequences of $n/2$ elements each
- **Conquer**
  - Sort the subsequences recursively using merge sort
  - When the size of the sequences is 1 there is nothing more to do
- **Combine**
  - Merge the two sorted subsequences

Merge Sort



Alg.: MERGE-SORT(*A*, p, r)

if p < r ▷ Check for base case

then *q* ← ⌊(p + r)/2⌋ ▷ Divide

MERGE-SORT(*A*, p, *q*) ▷ Conquer

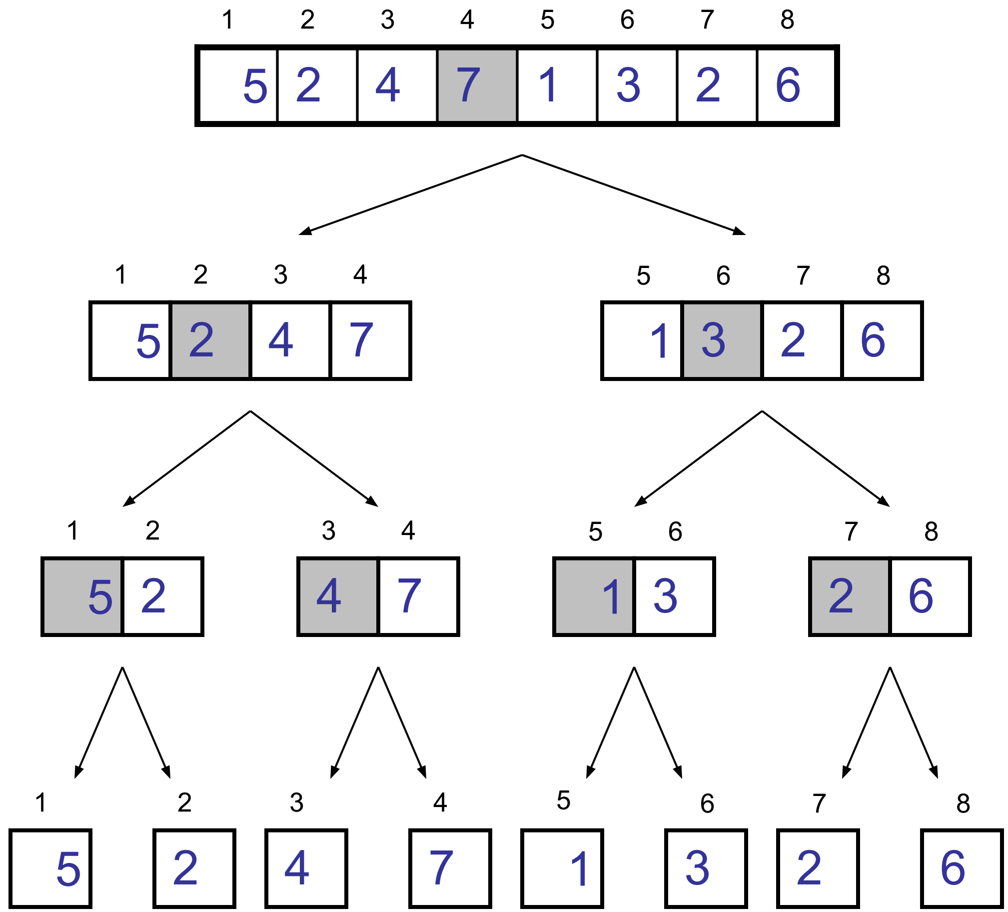
MERGE-SORT(*A*, *q* + 1, r) ▷ Conquer

MERGE(*A*, p, *q*, r) ▷ Combine

- Initial call: MERGE-SORT(*A*, 1, *n*)

Example – n Power of 2

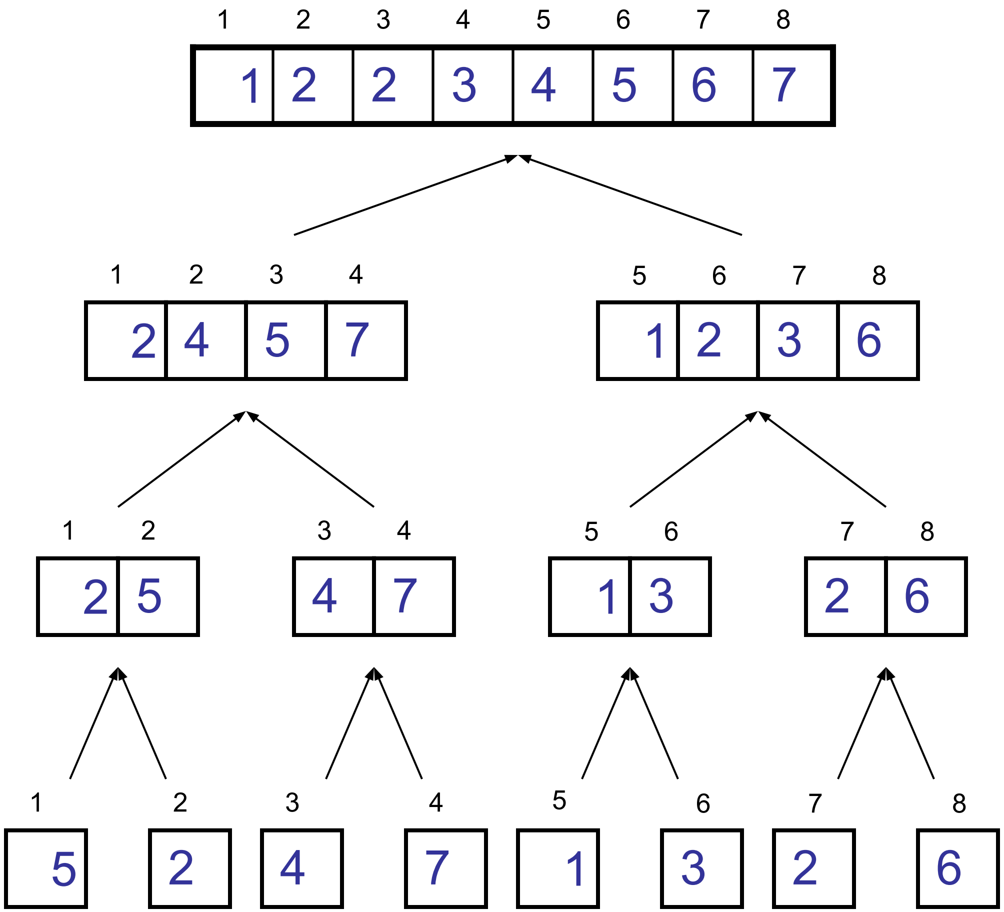
Divide



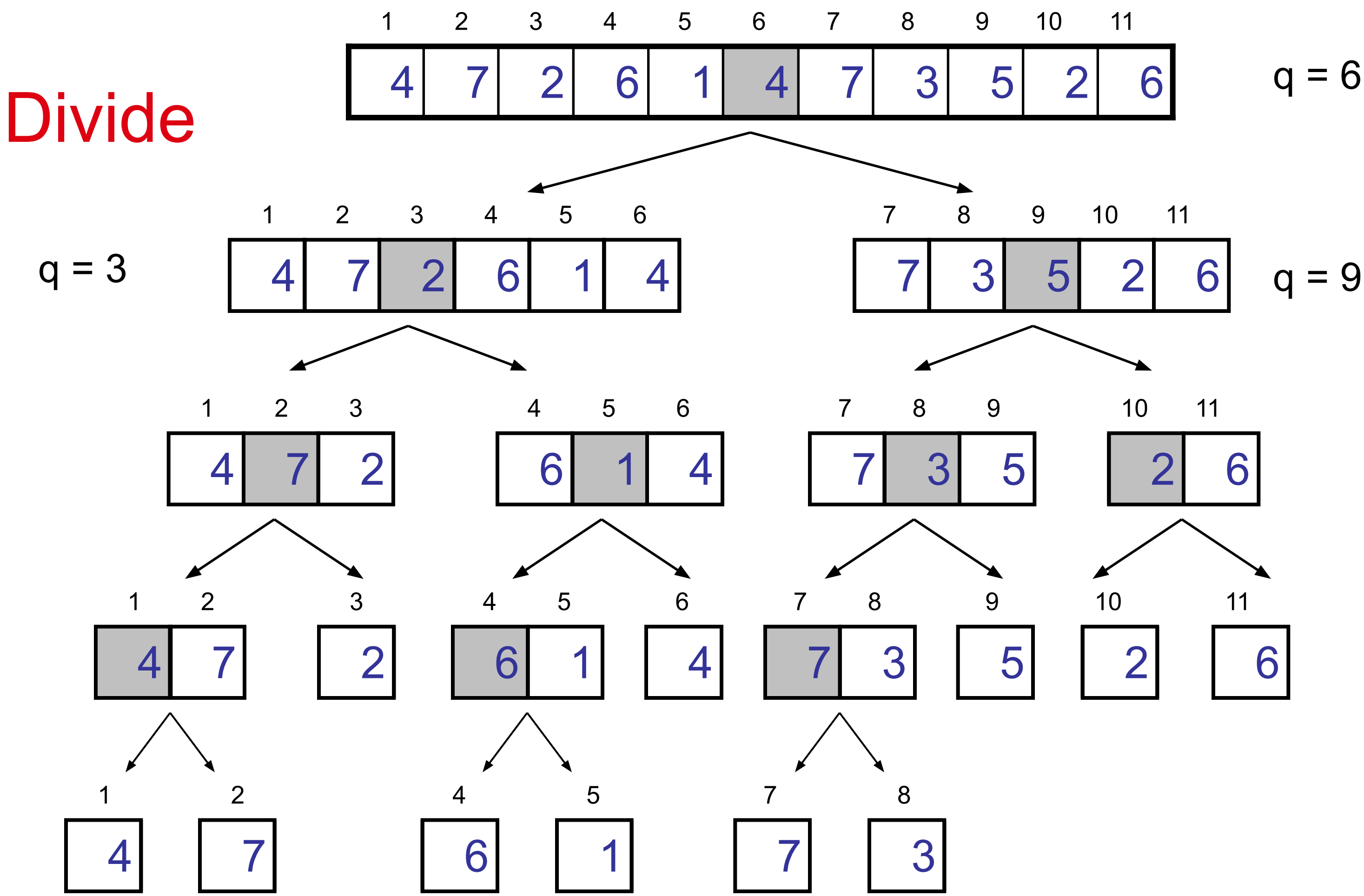
q = 4

Example – n Power of 2

Conquer and Merge

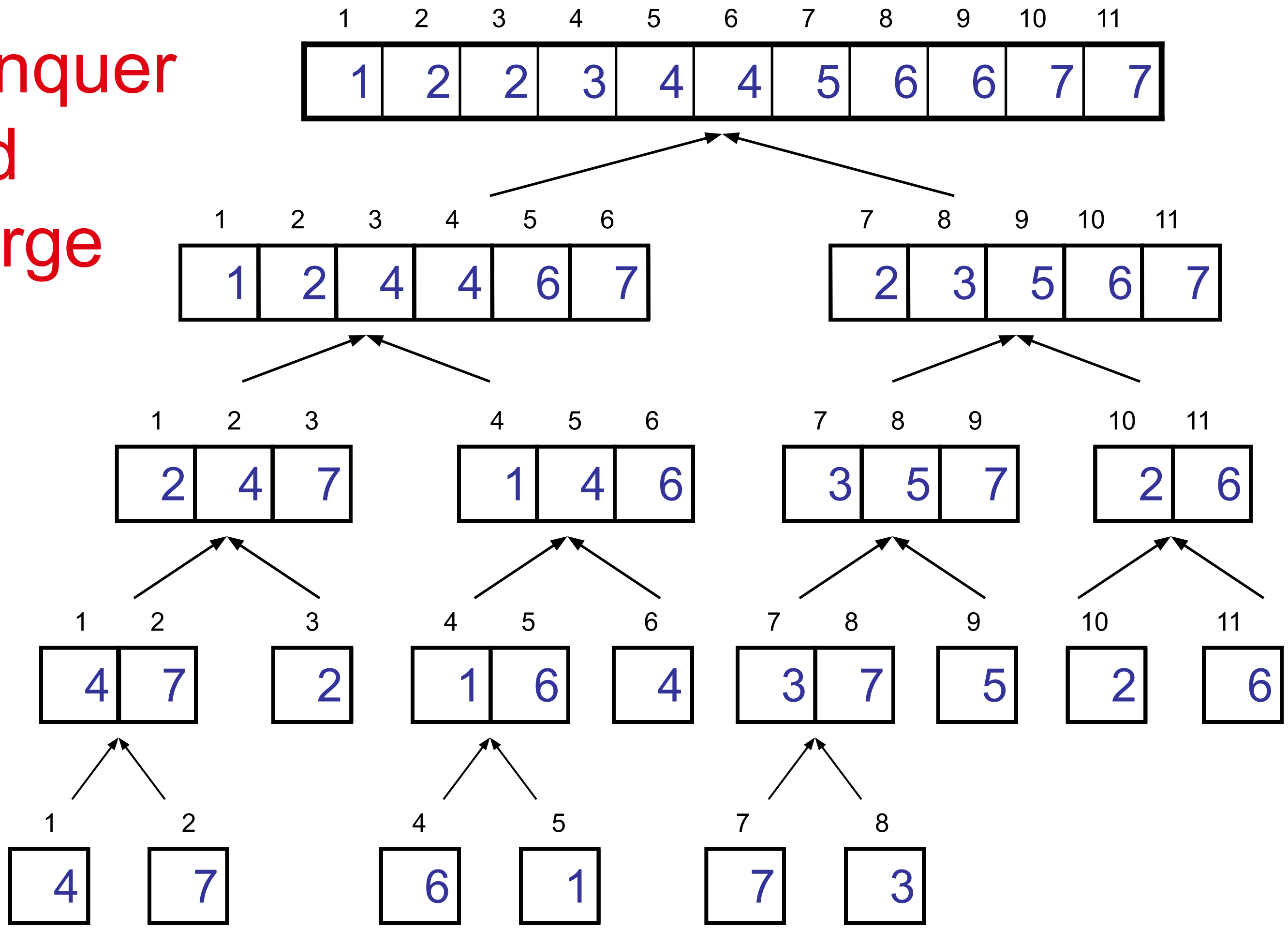


Example – n Not a Power of 2

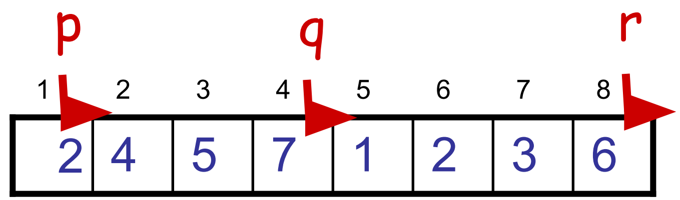


Example – *n* Not a Power of 2

Conquer and Merge



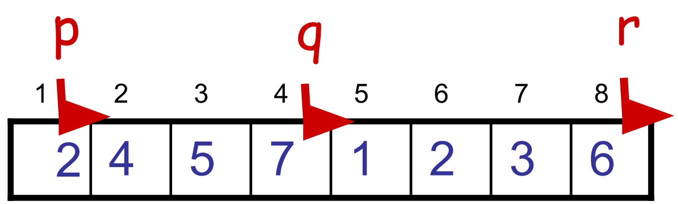
Merging



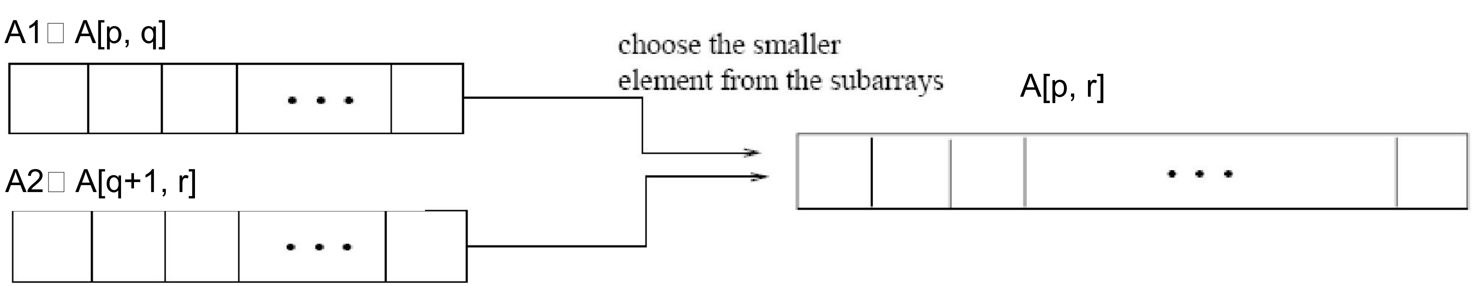
- **Input:** Array *A* and indices p, q, r such that p ≤ q < r
  - Subarrays *A*[p . . q] and *A*[q + 1 . . r] are sorted
- **Output:** One single sorted subarray *A*[p . . r]

Merging

- Idea for merging:



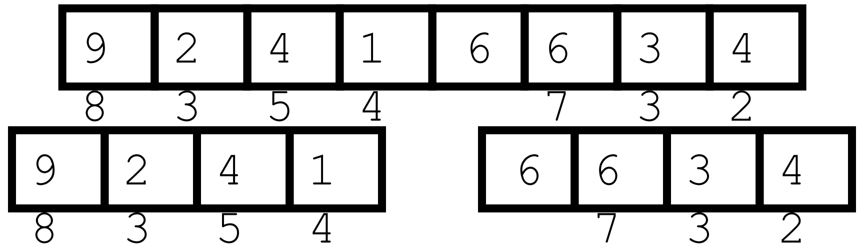
  – Two piles of sorted cards
    - Choose the smaller of the two top cards
    - Remove it and place it in the output pile
  – Repeat the process until one pile is empty
  – Take the remaining input pile and place it face-down onto the output pile



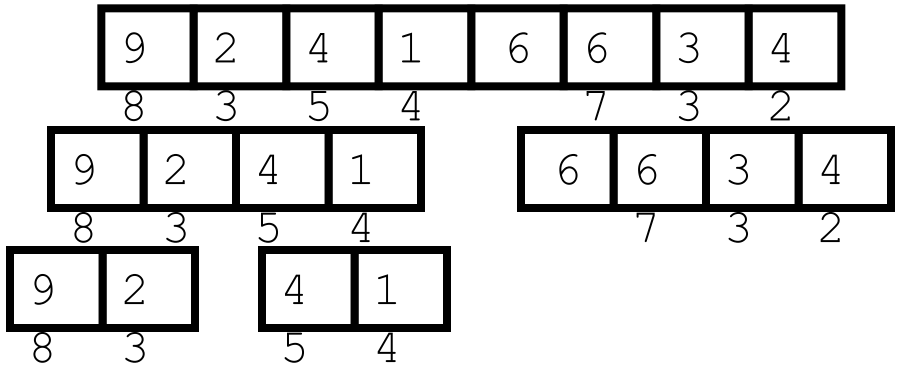
Merge Sort (recursive simulation)

98	23	45	14	6	67	33	42

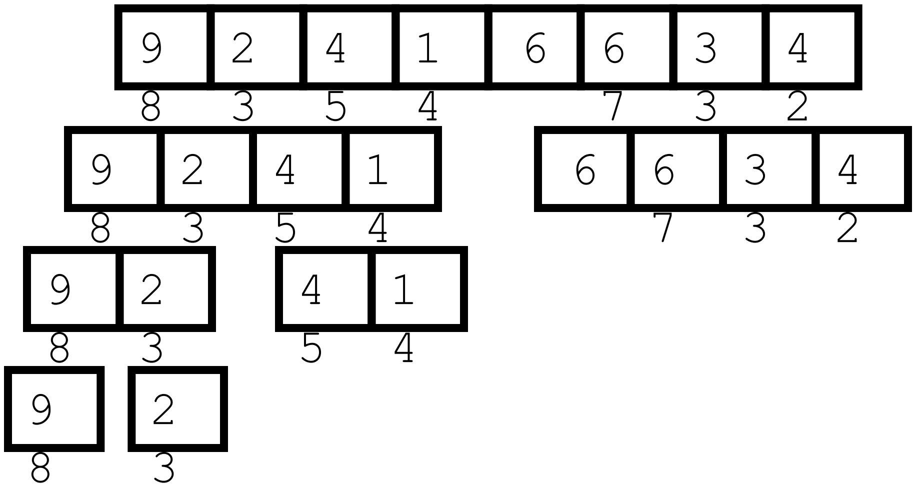
Merge Sort (recursive simulation)



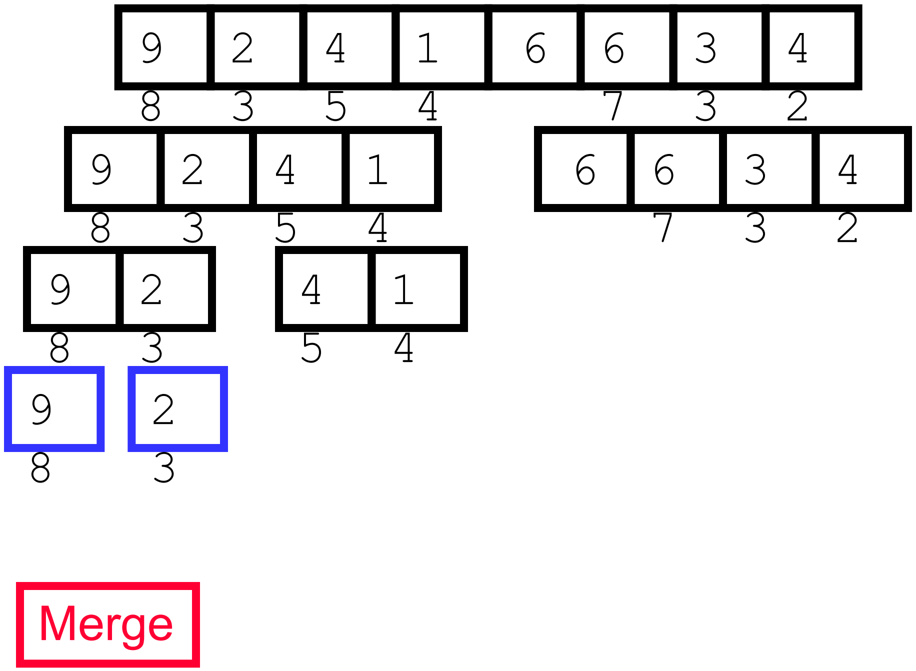
Merge Sort (recursive simulation)



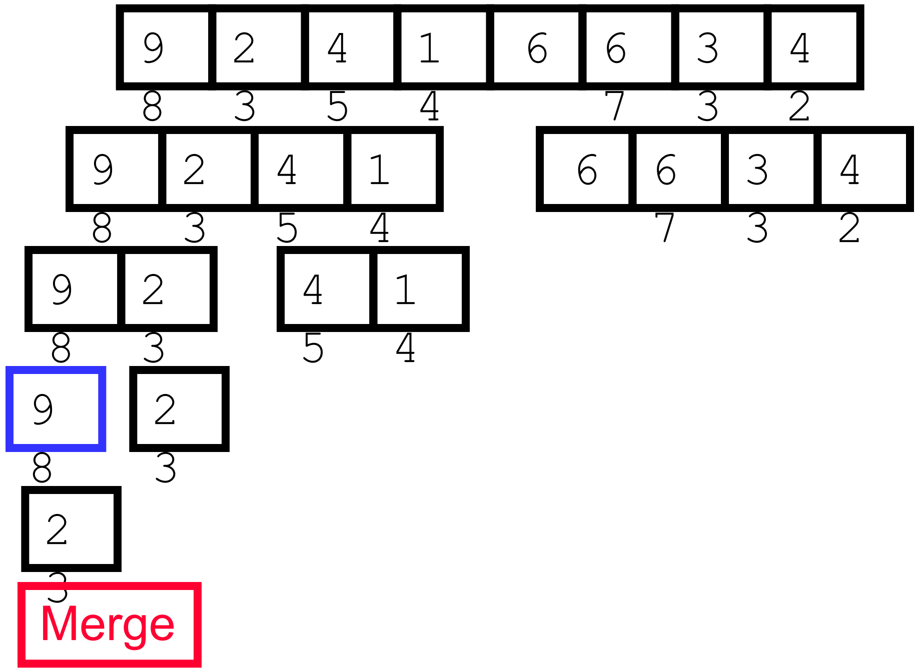
Merge Sort (recursive simulation)



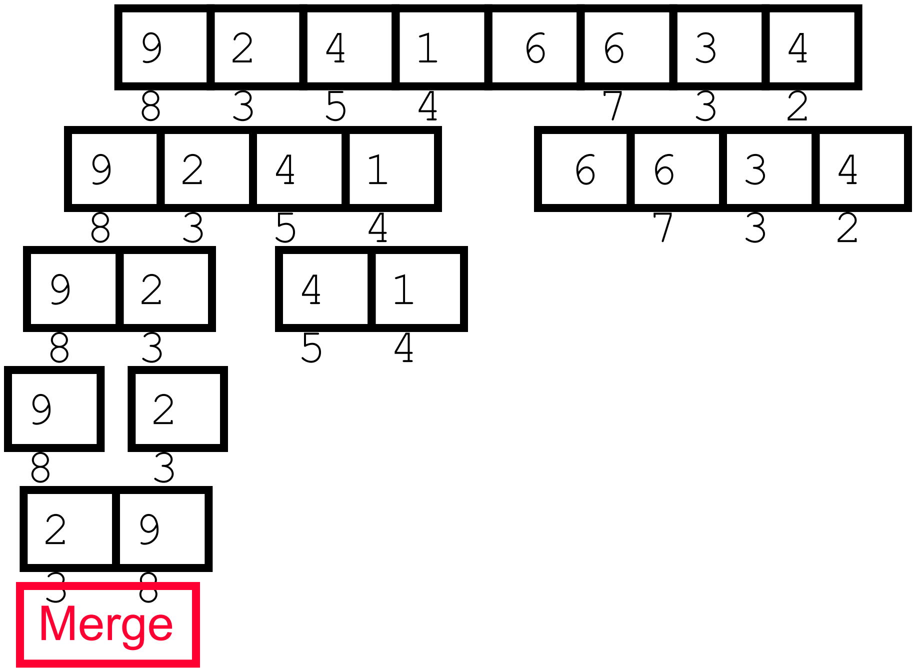
Merge Sort (recursive simulation)



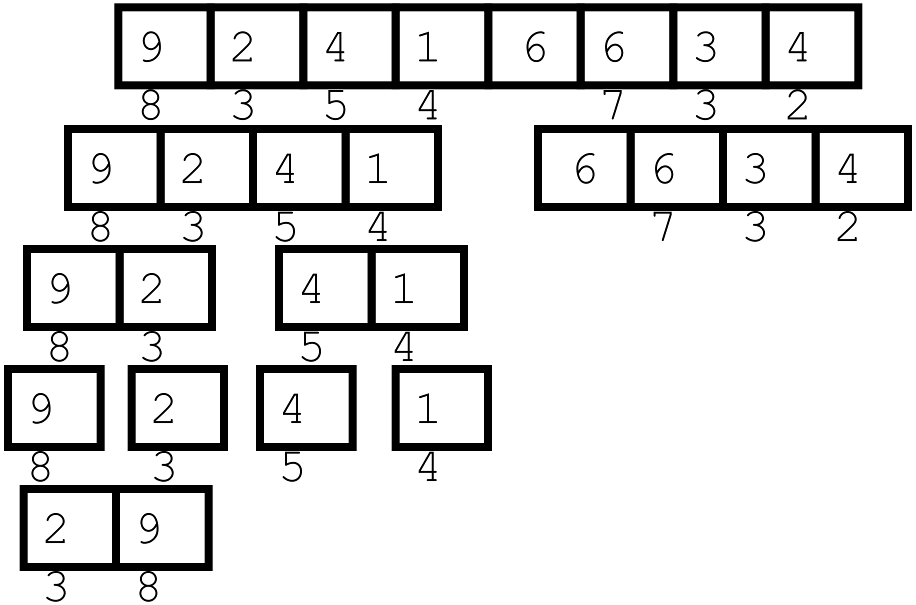
Merge Sort (recursive simulation)



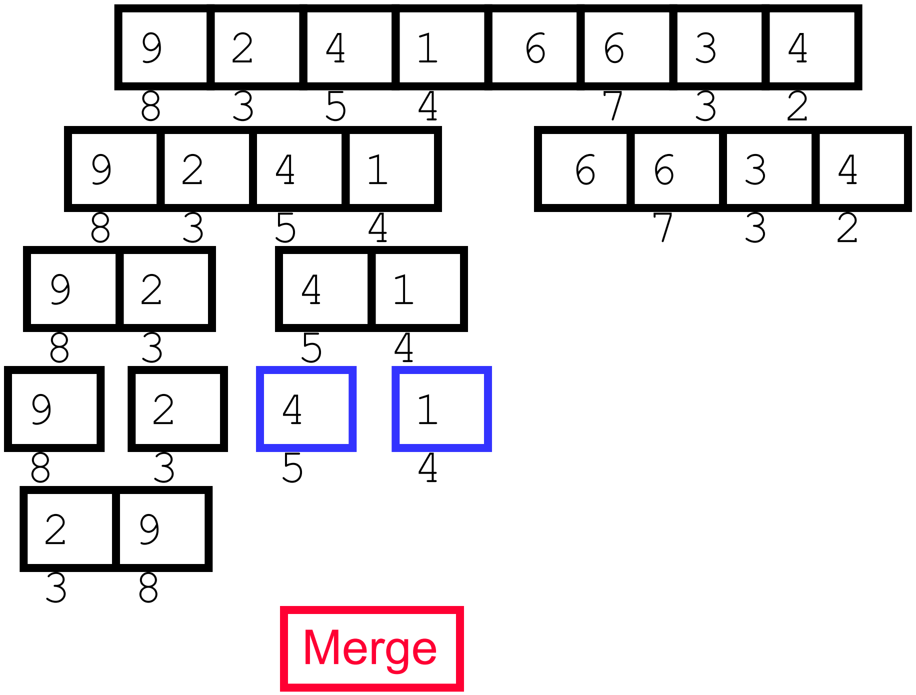
Merge Sort (recursive simulation)



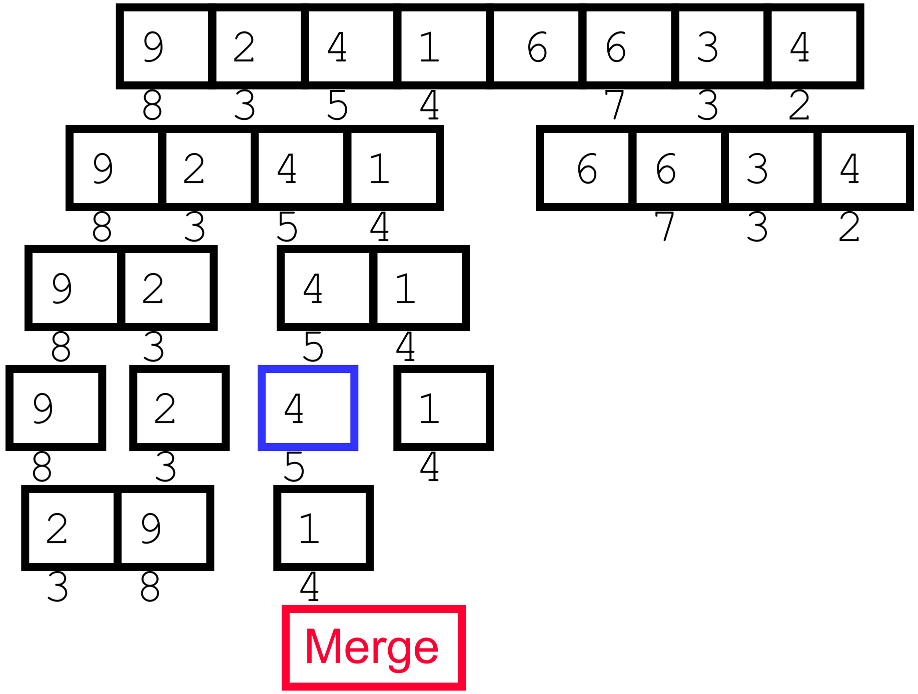
Merge Sort (recursive simulation)



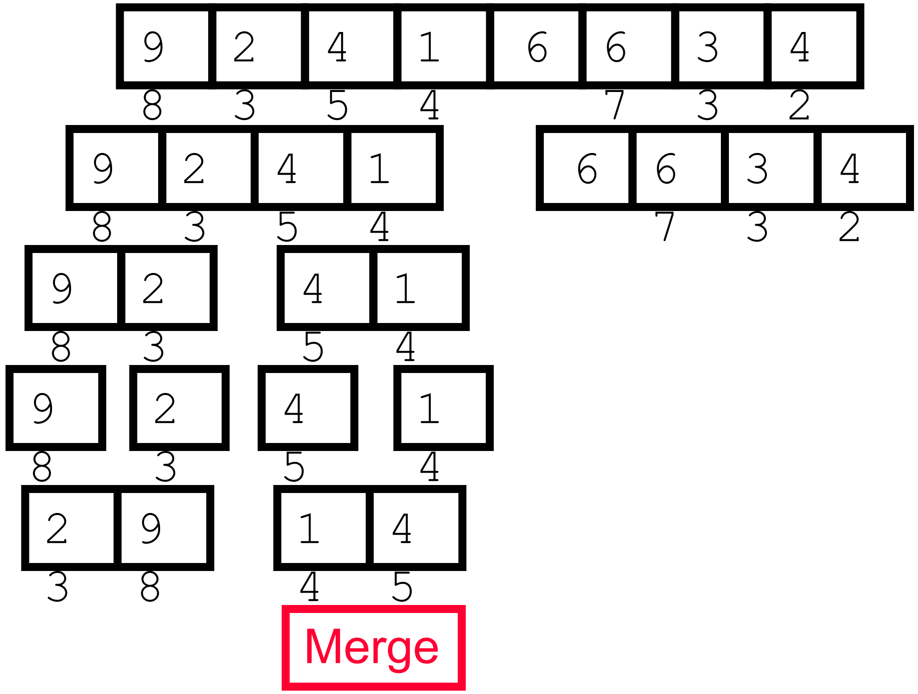
Merge Sort (recursive simulation)



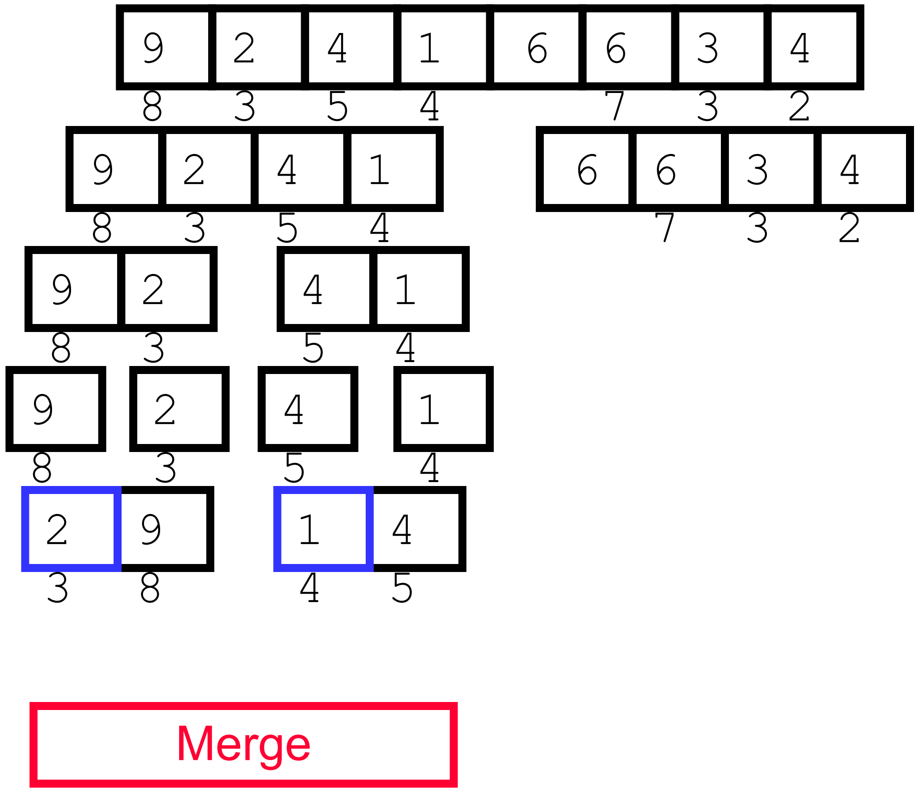
Merge Sort (recursive simulation)



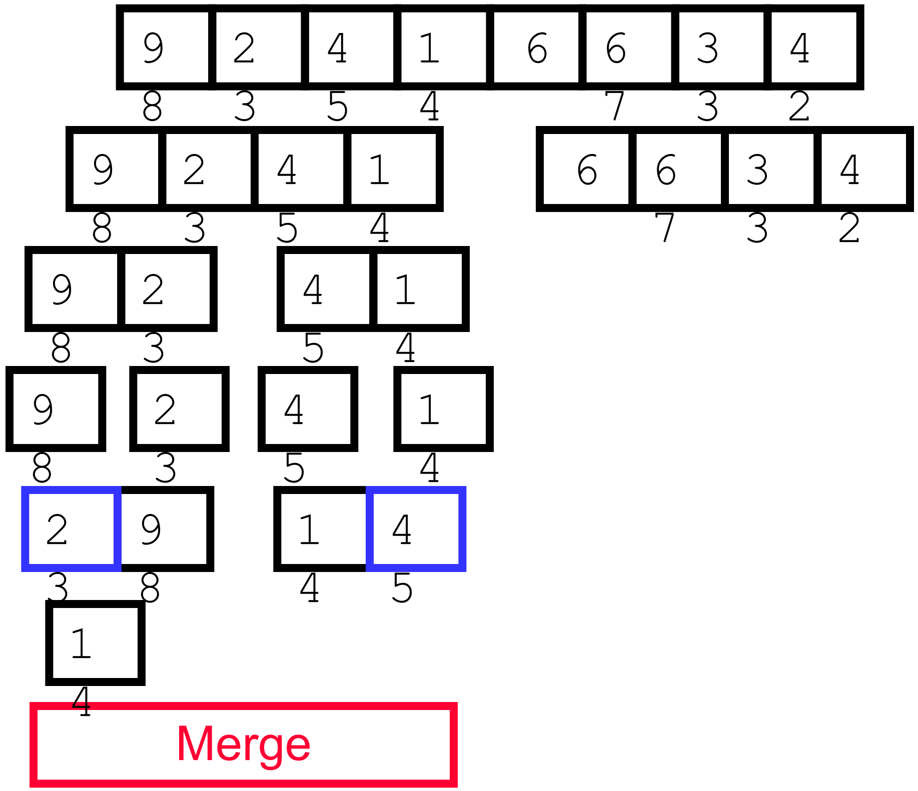
Merge Sort (recursive simulation)



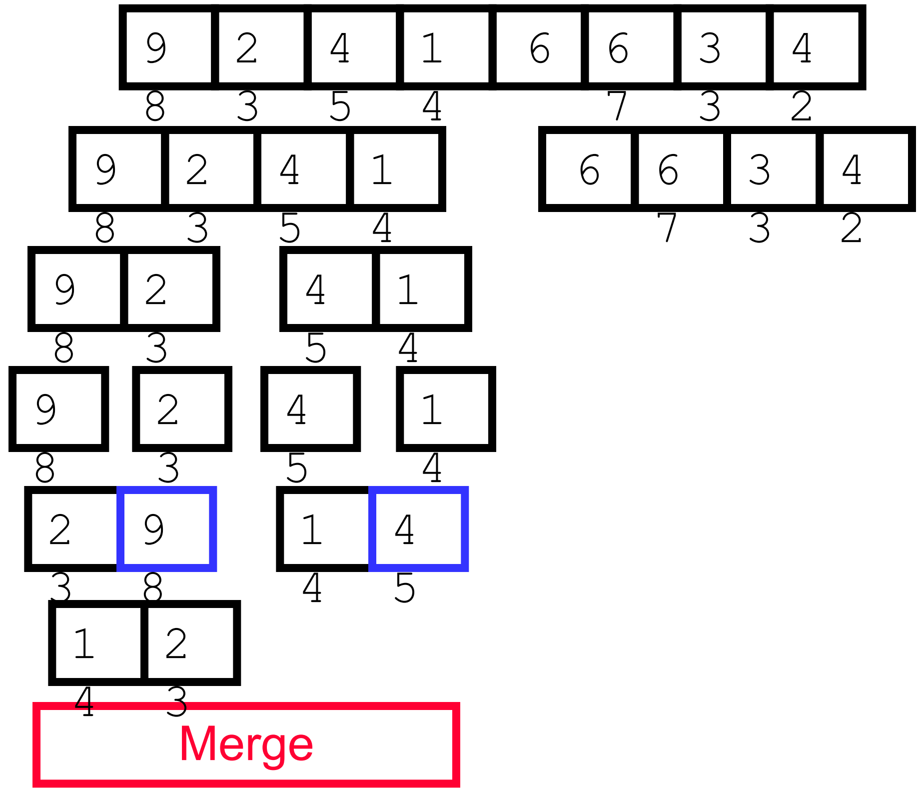
Merge Sort (recursive simulation)



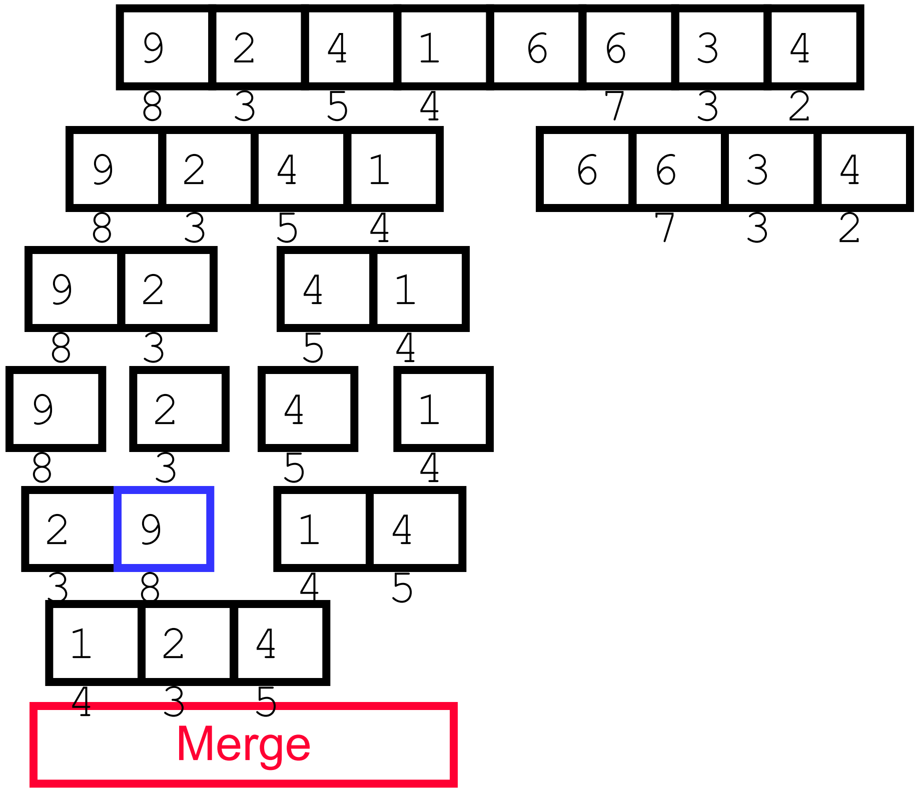
Merge Sort (recursive simulation)



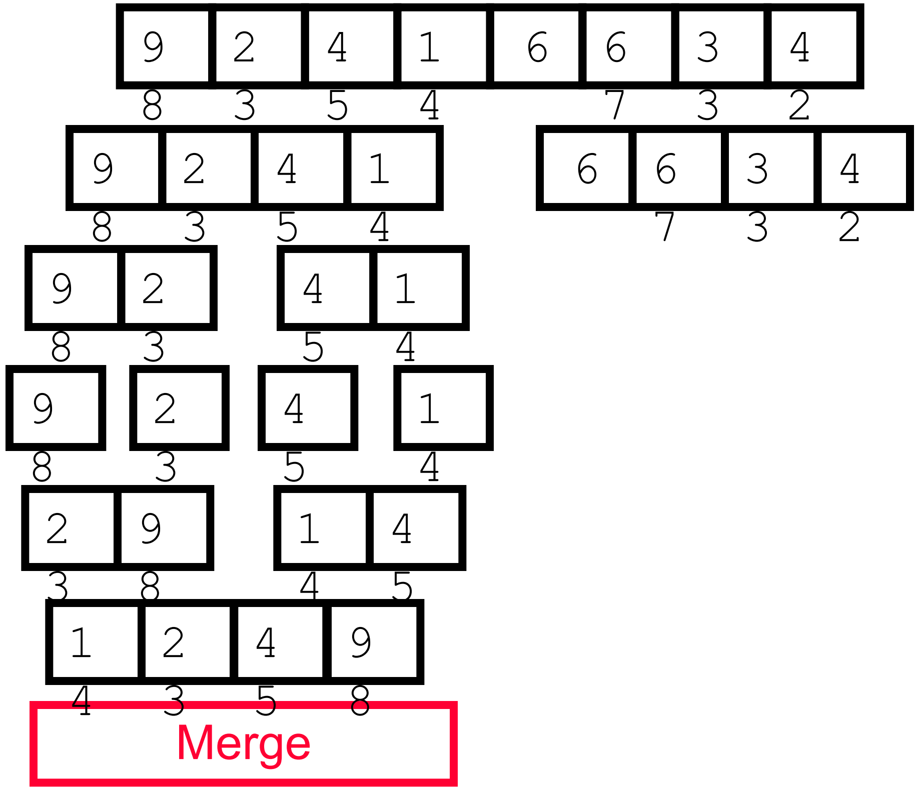
Merge Sort (recursive simulation)



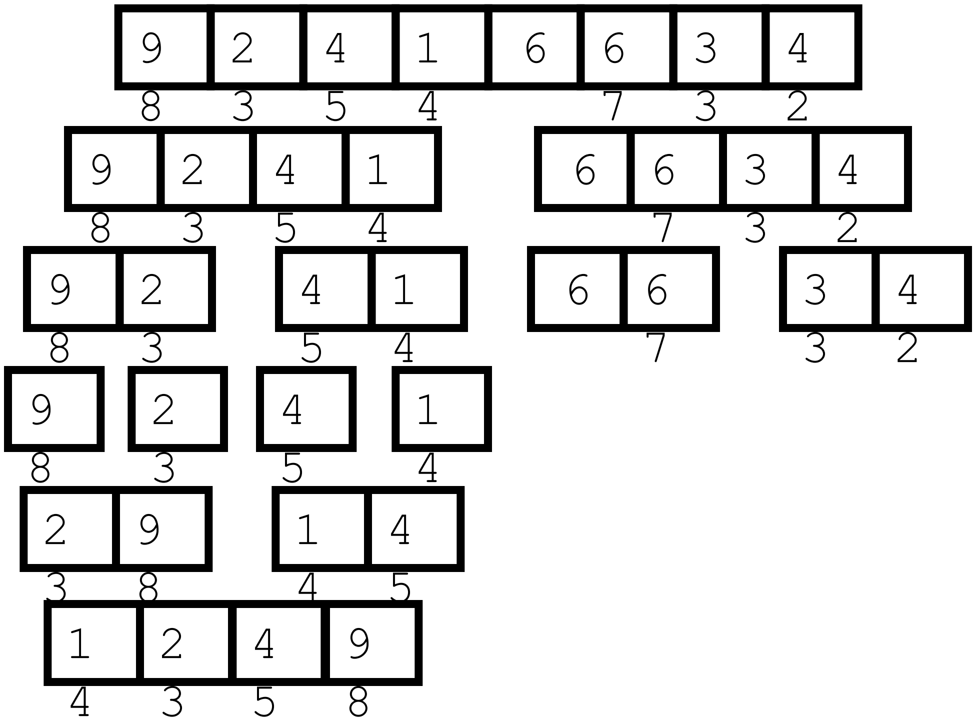
Merge Sort (recursive simulation)



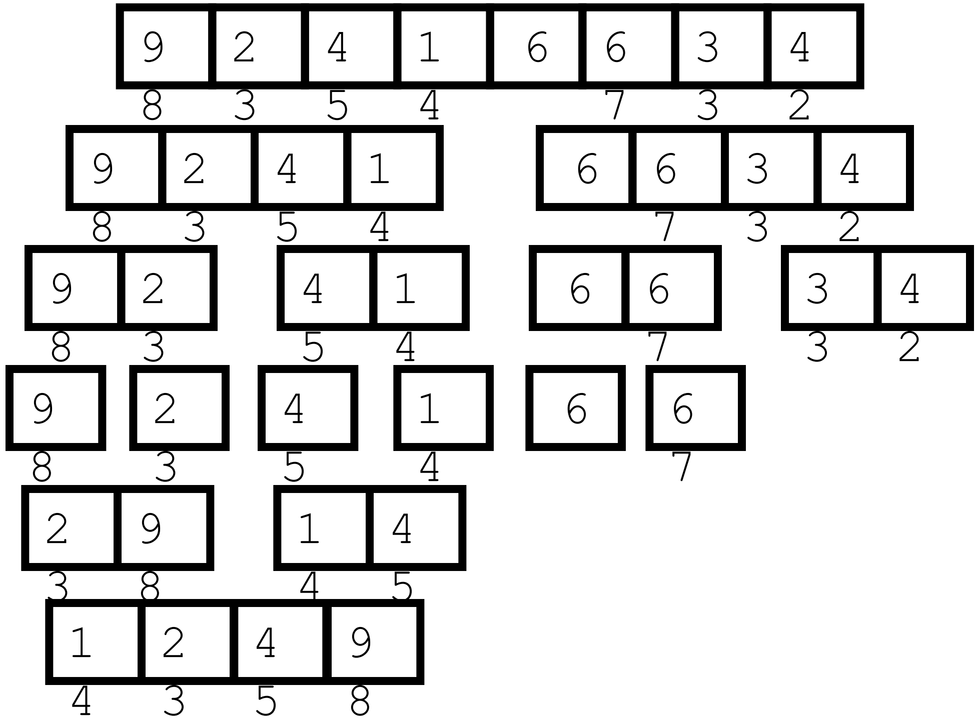
Merge Sort (recursive simulation)



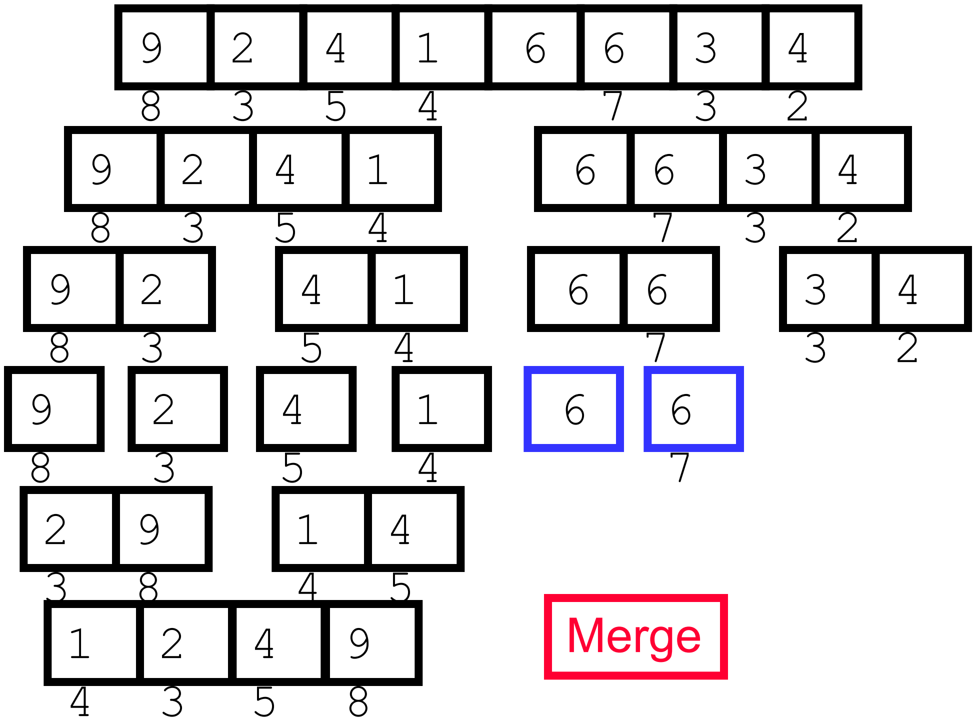
Merge Sort (recursive simulation)



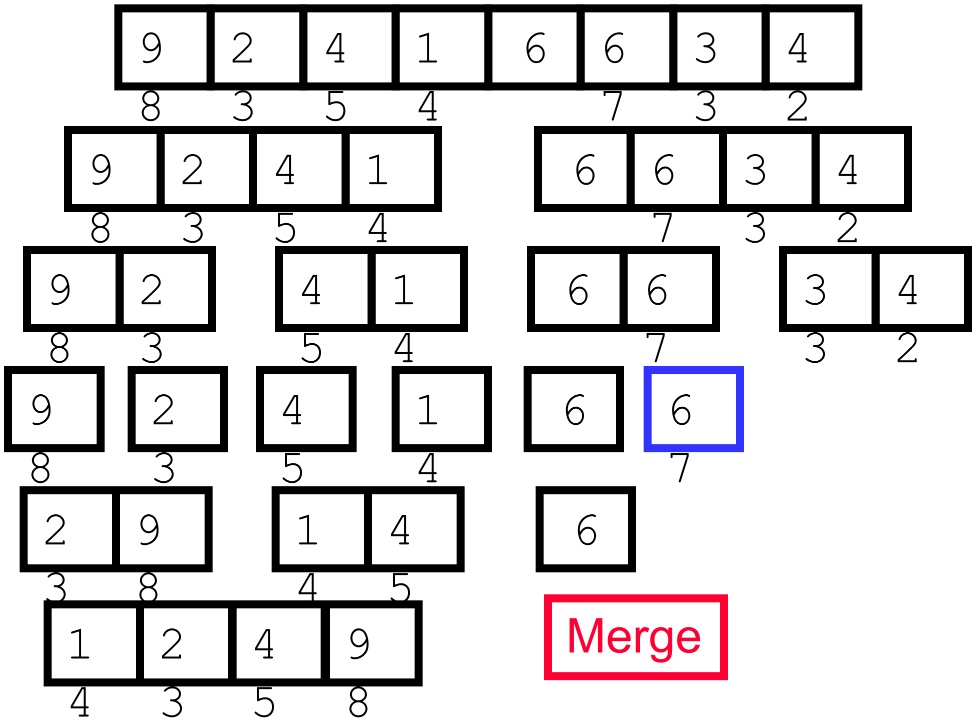
Merge Sort (recursive simulation)



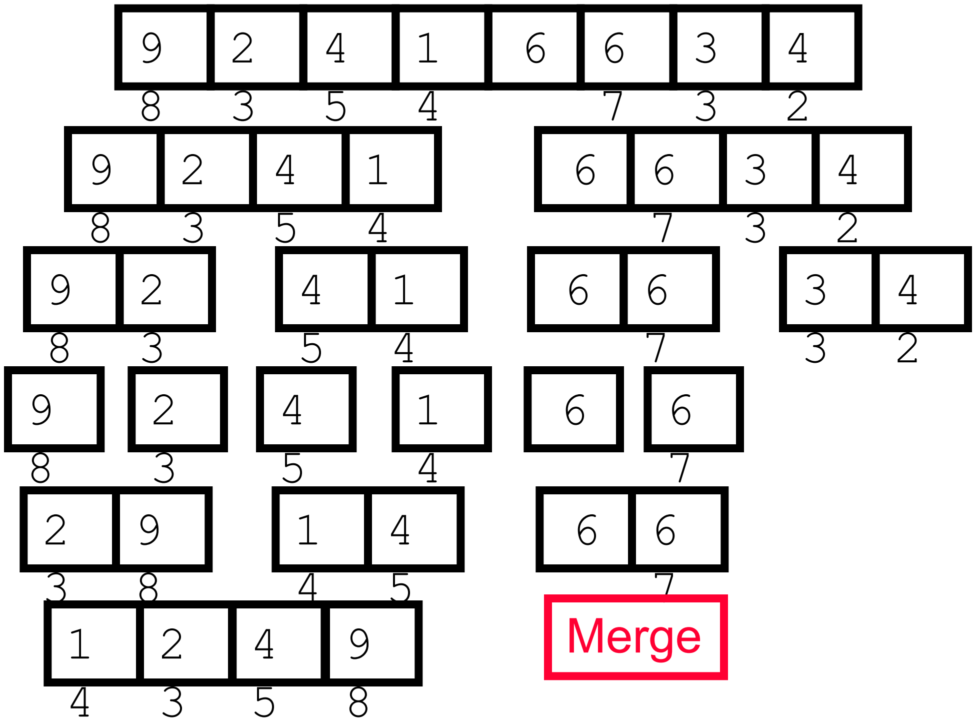
Merge Sort (recursive simulation)



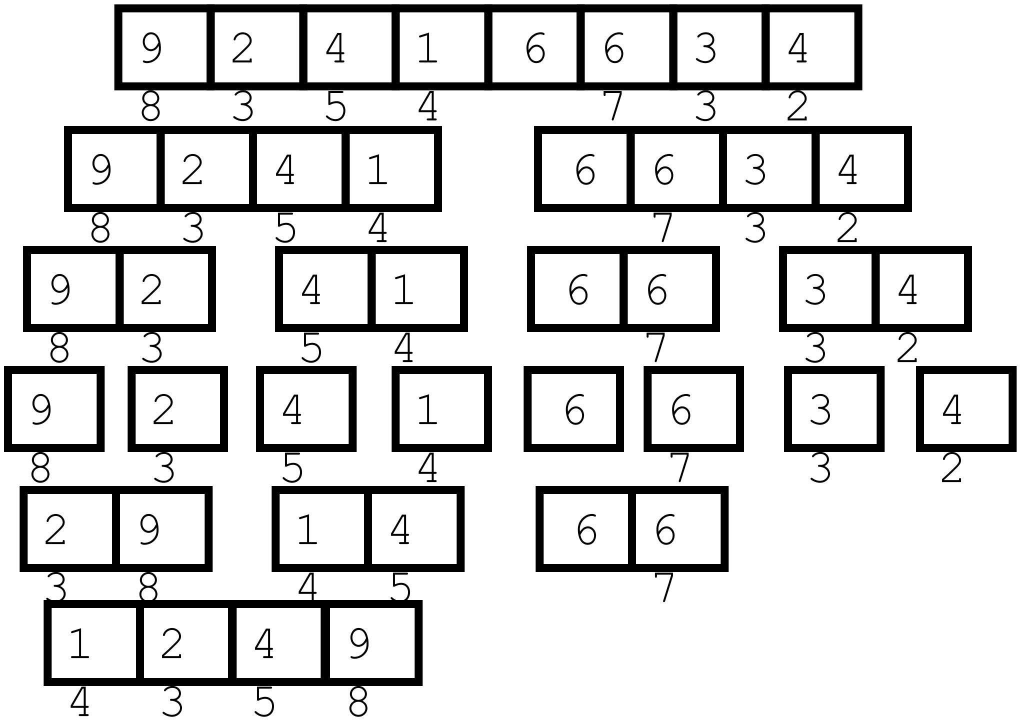
Merge Sort (recursive simulation)



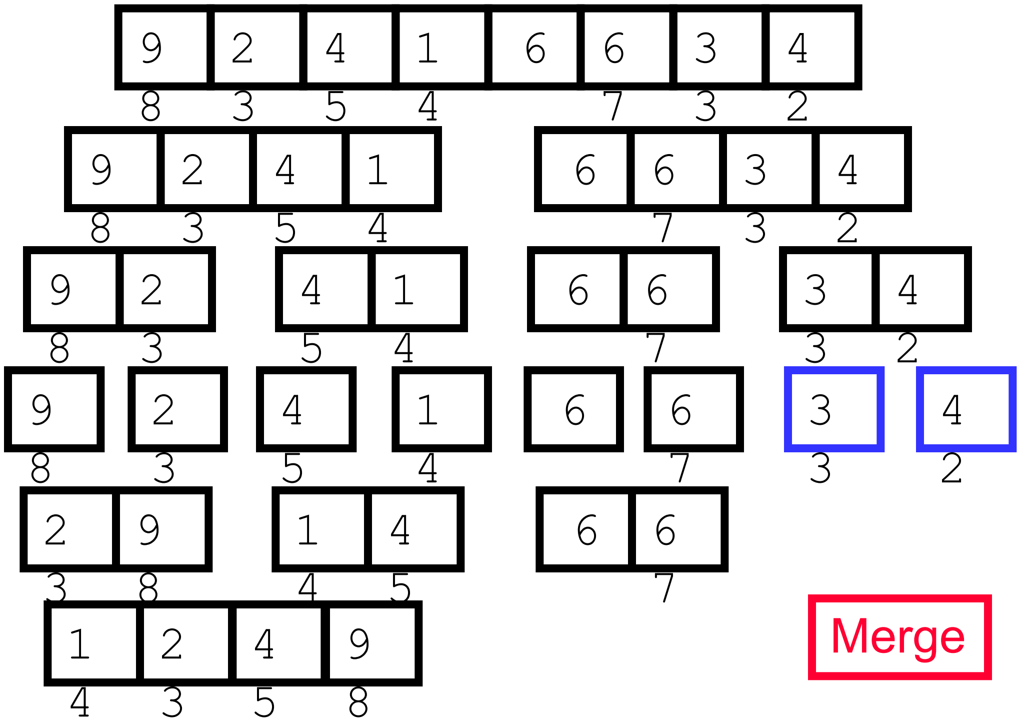
Merge Sort (recursive simulation)



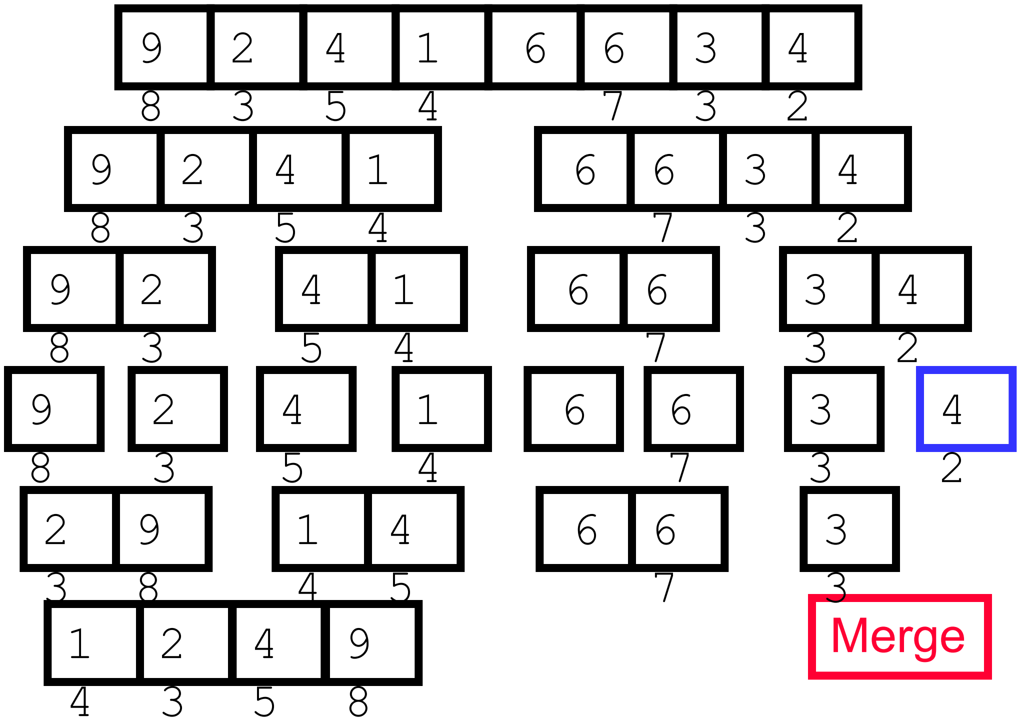
Merge Sort (recursive simulation)



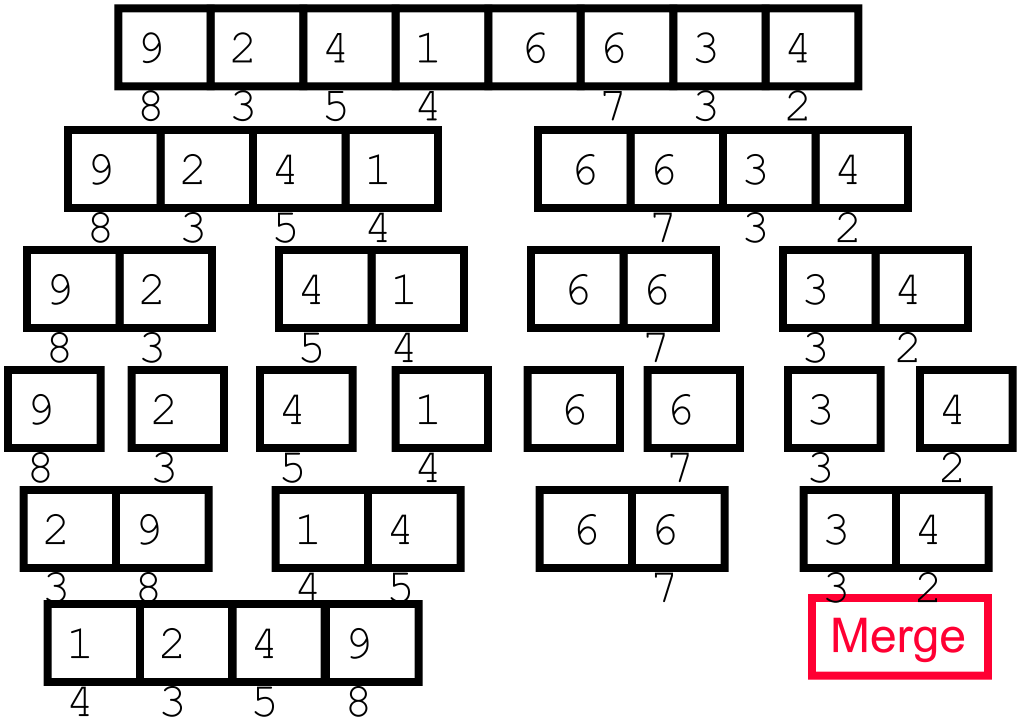
Merge Sort (recursive simulation)



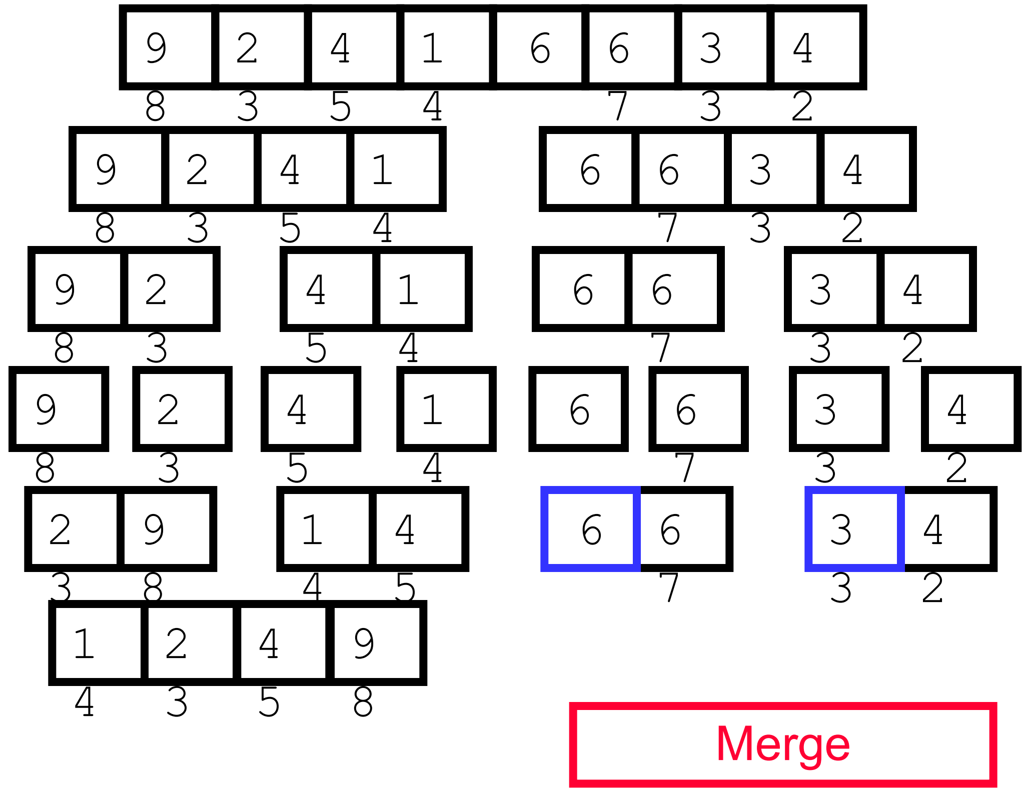
Merge Sort (recursive simulation)



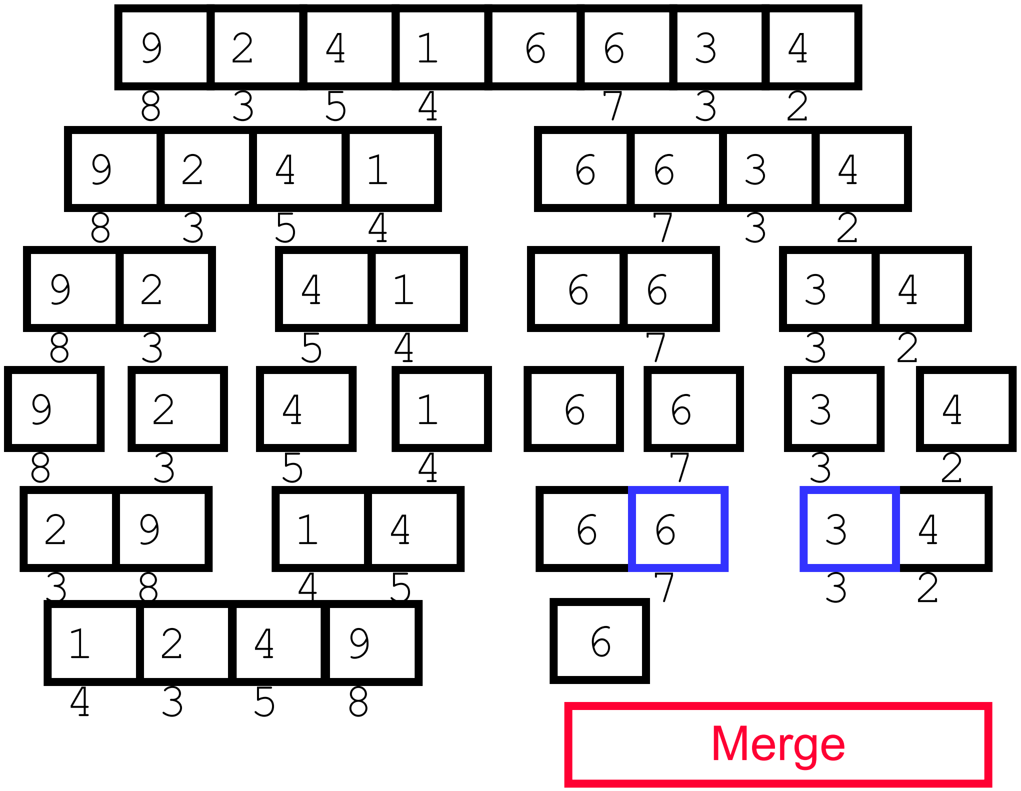
Merge Sort (recursive simulation)



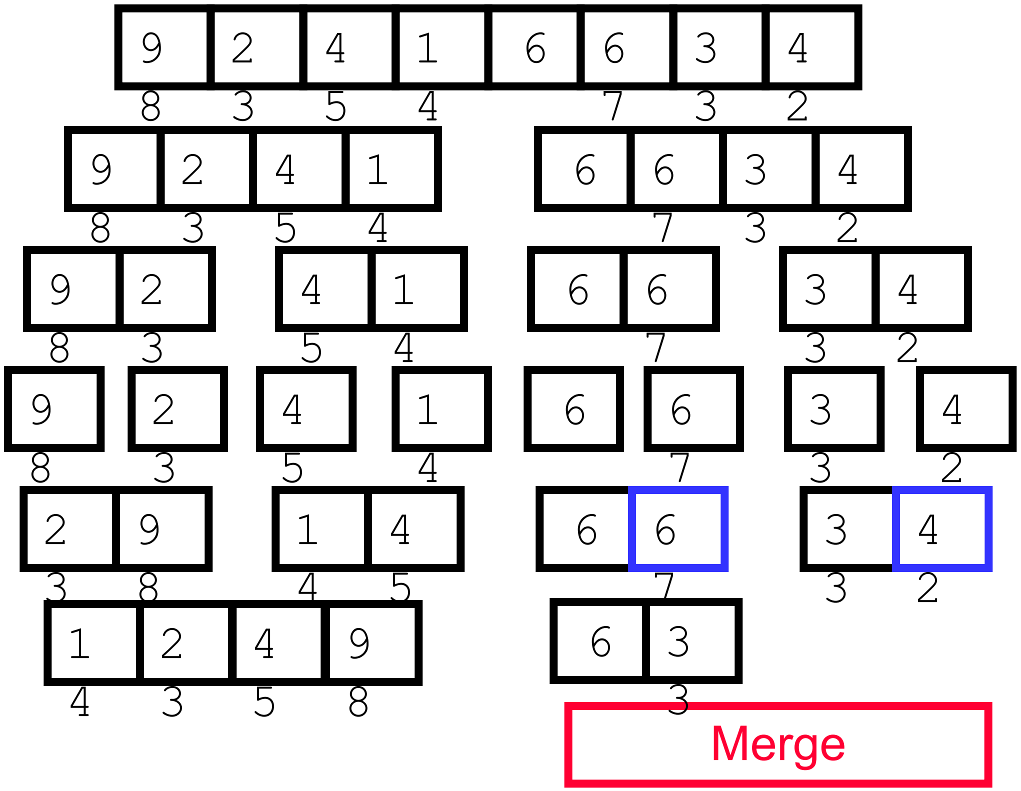
Merge Sort (recursive simulation)



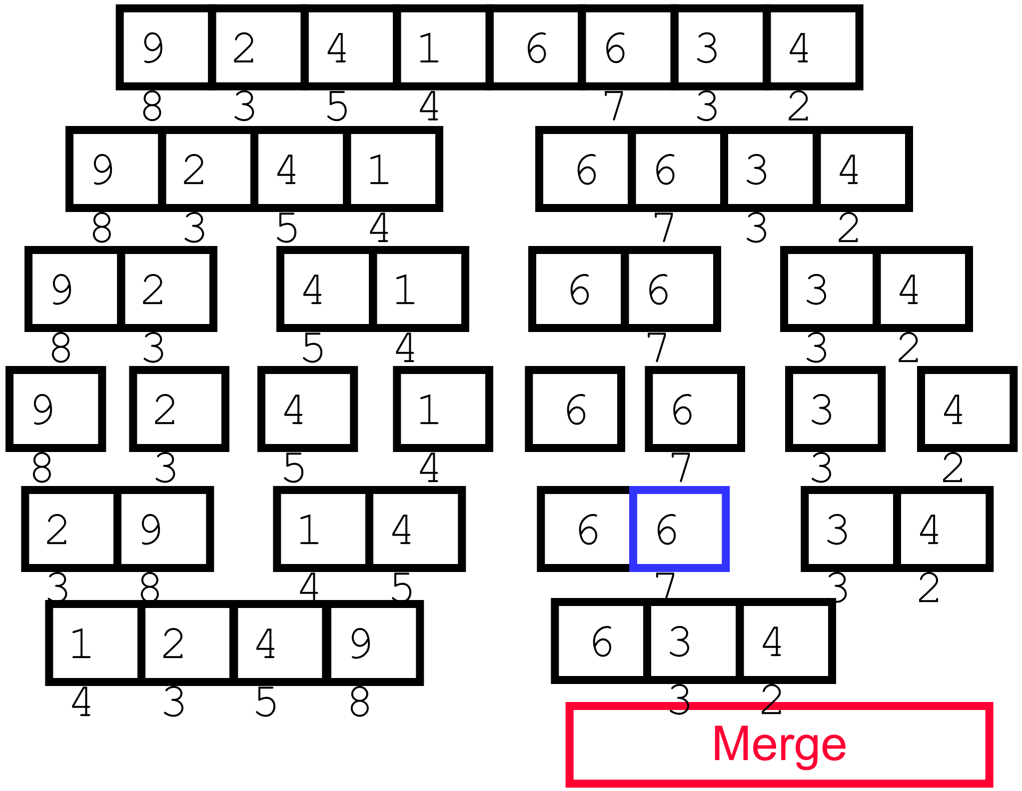
Merge Sort (recursive simulation)



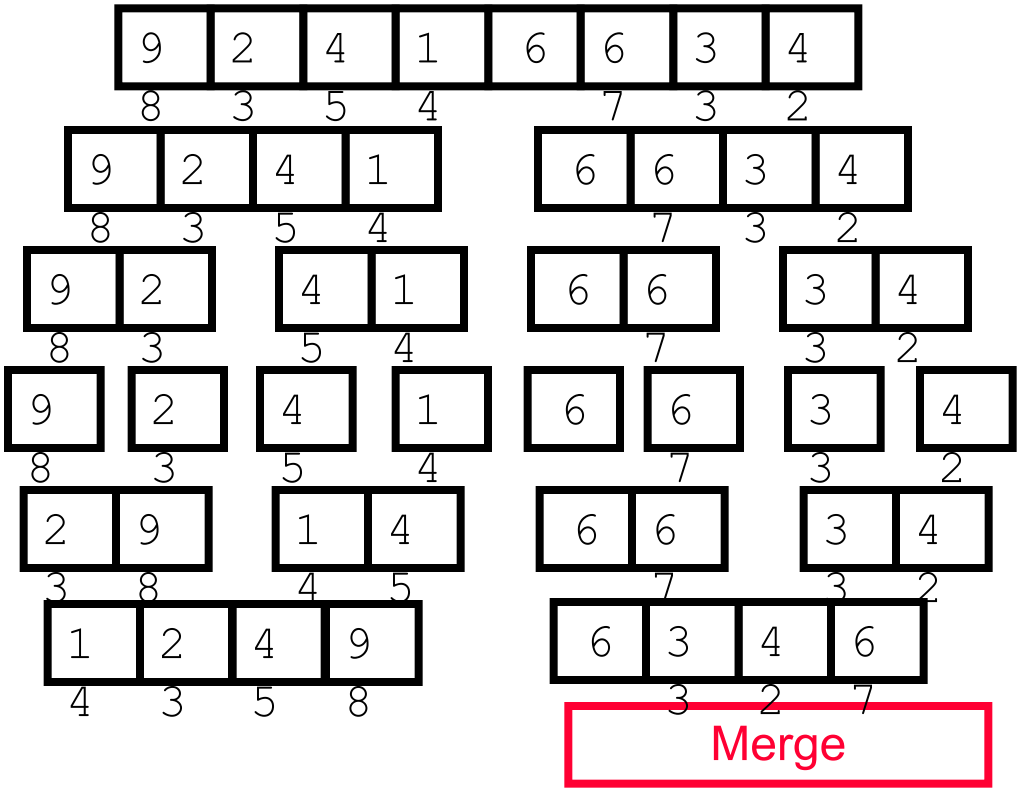
Merge Sort (recursive simulation)



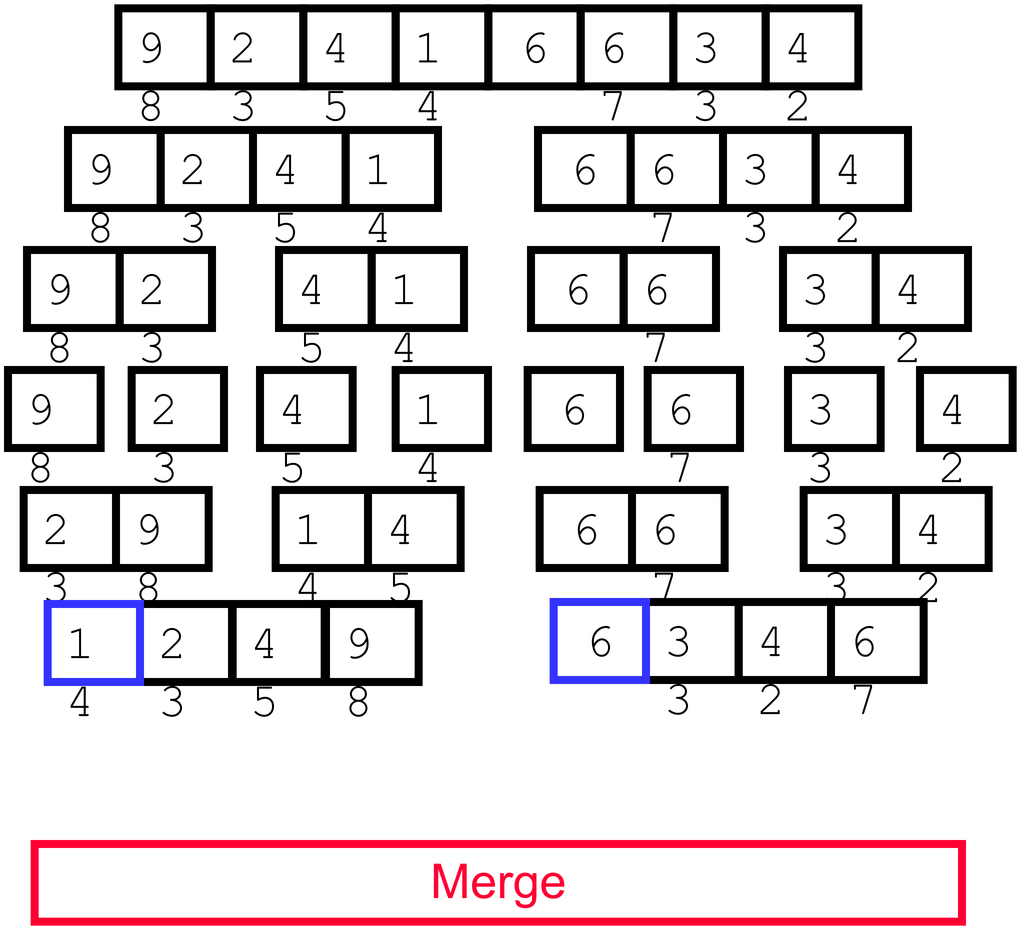
Merge Sort (recursive simulation)



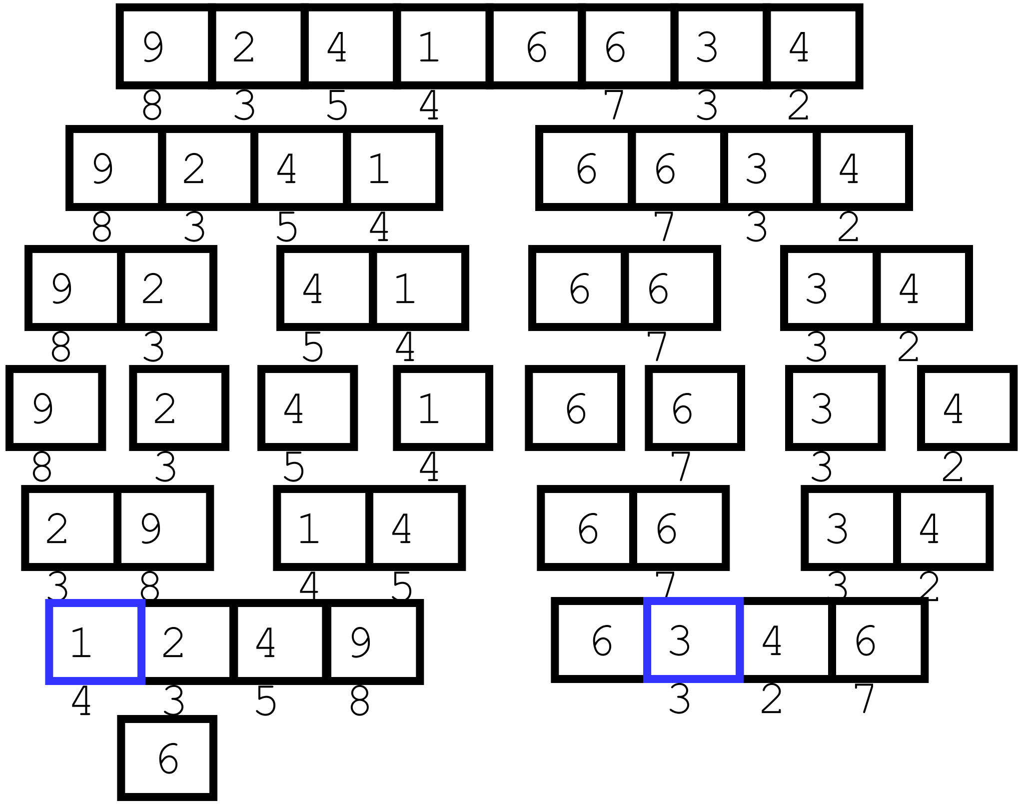
Merge Sort (recursive simulation)



Merge Sort (recursive simulation)

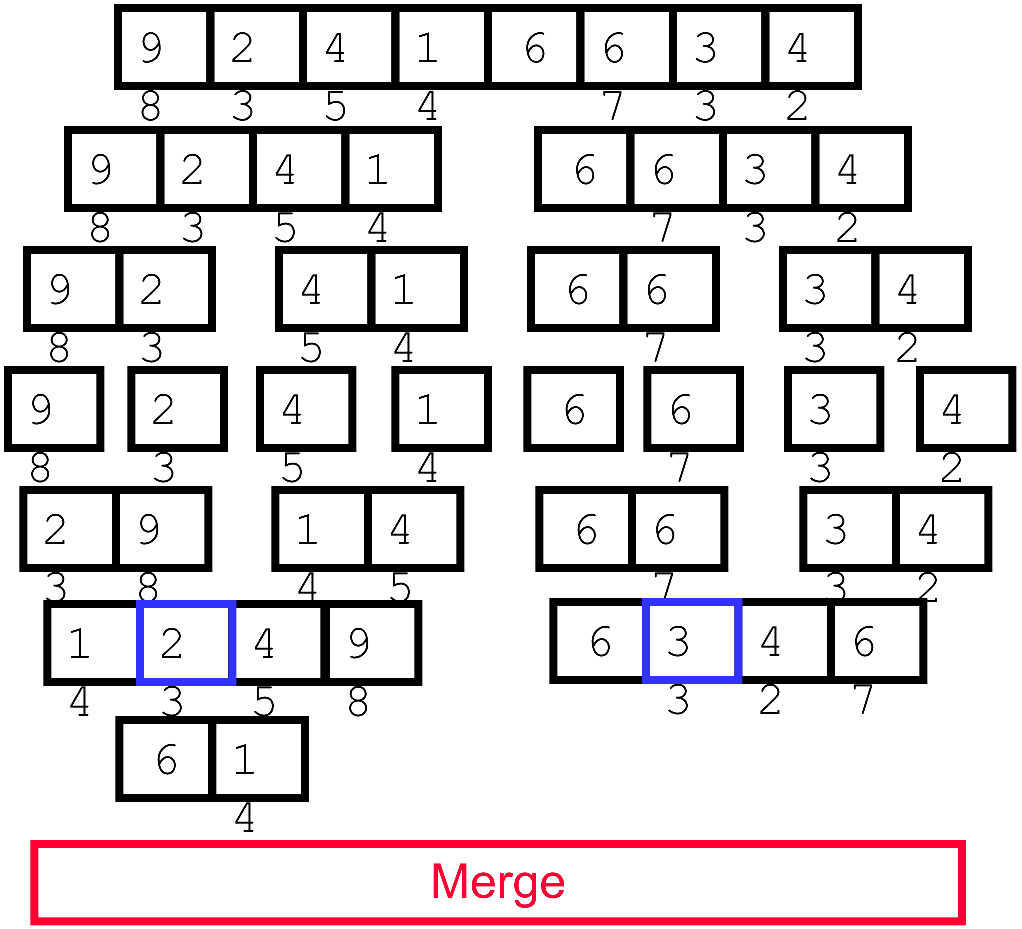


Merge Sort (recursive simulation)

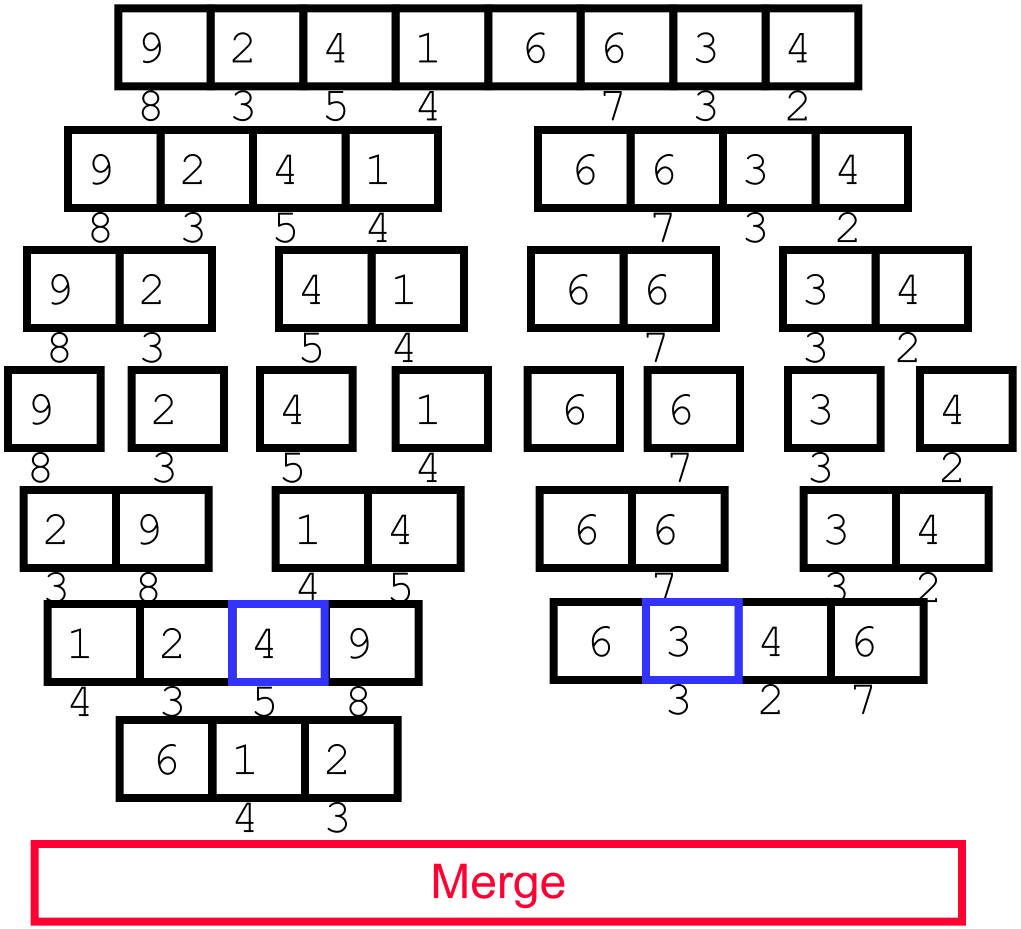


Merge

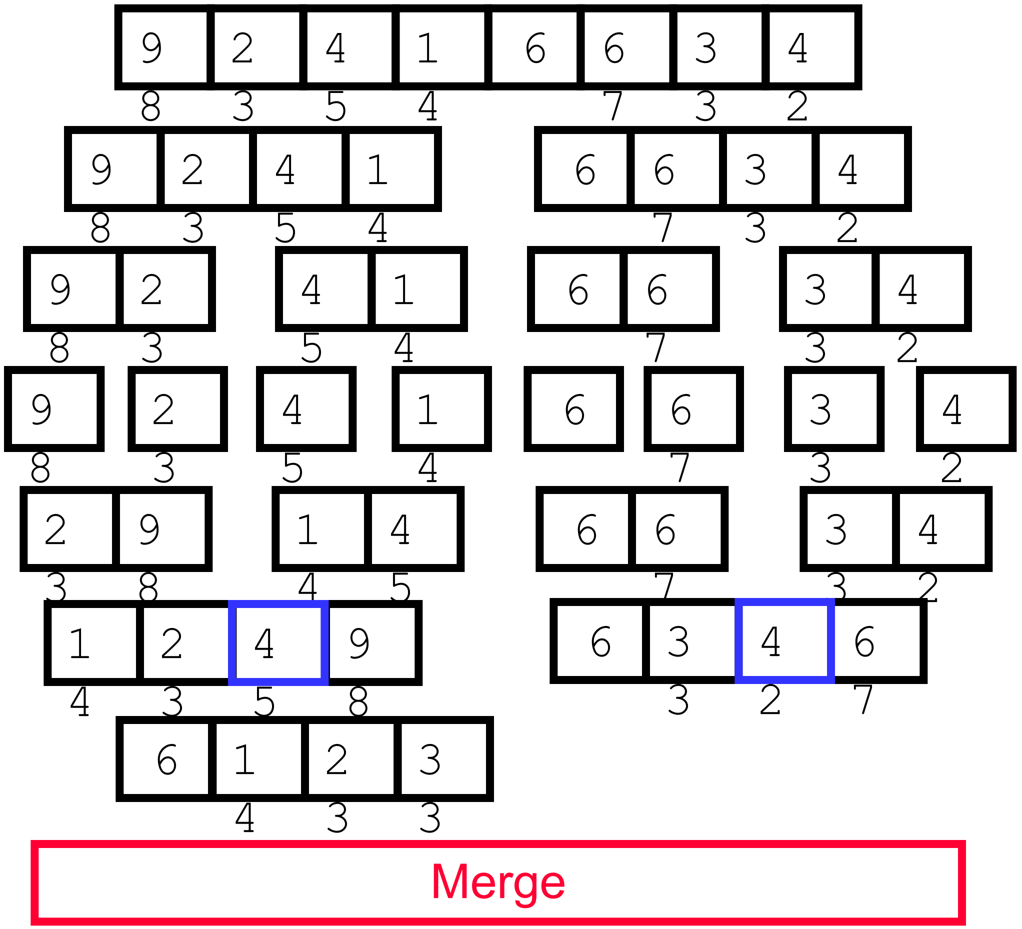
Merge Sort (recursive simulation)



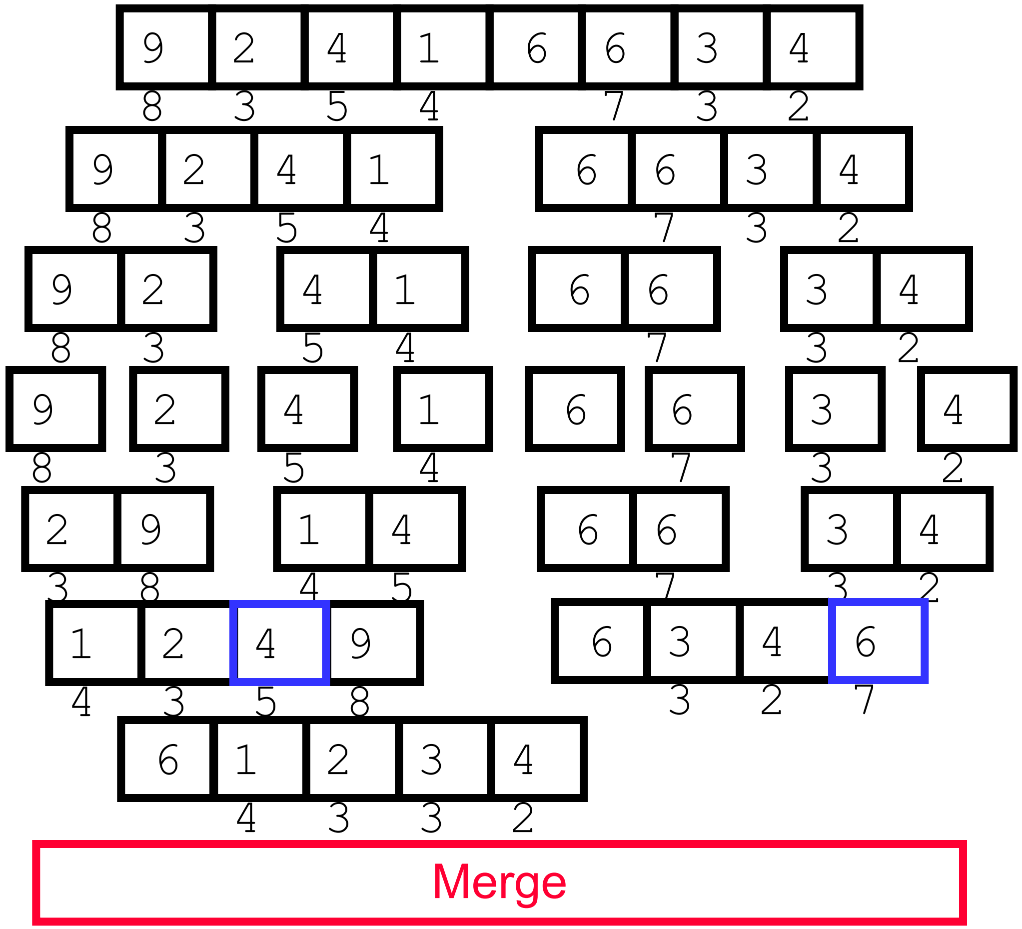
Merge Sort (recursive simulation)



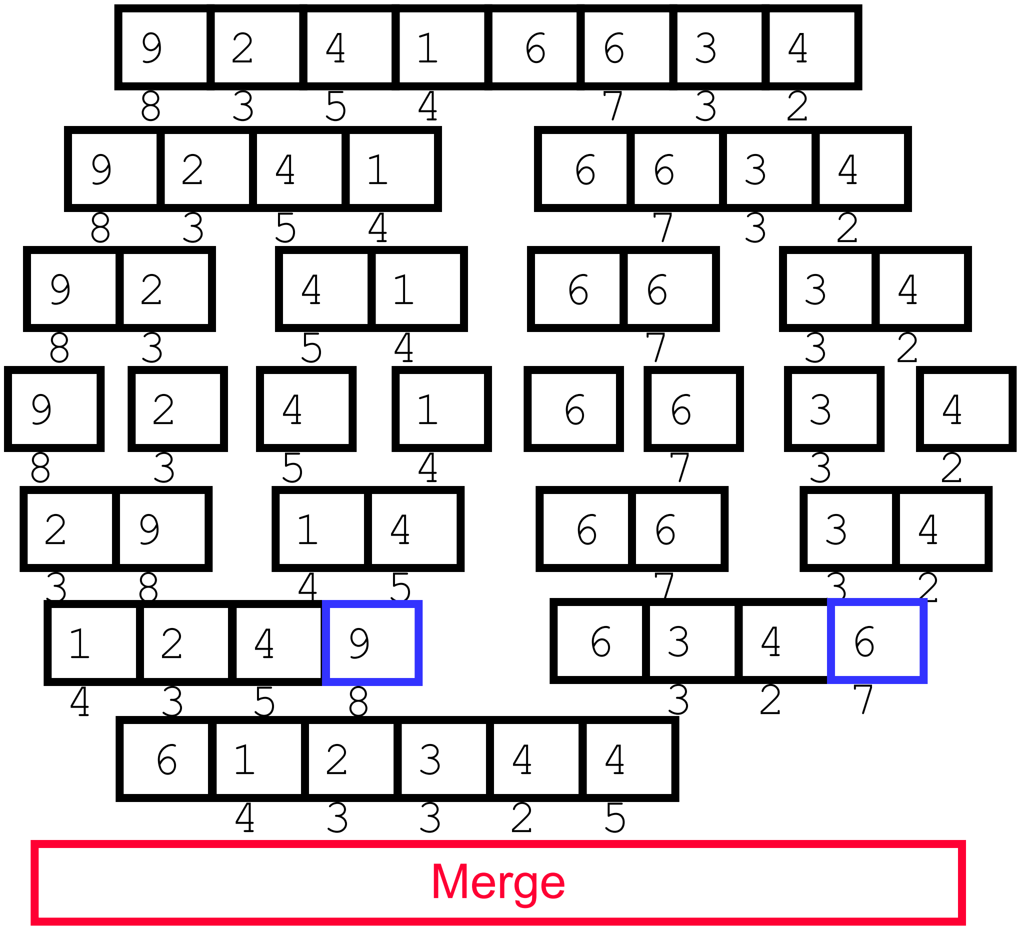
Merge Sort (recursive simulation)



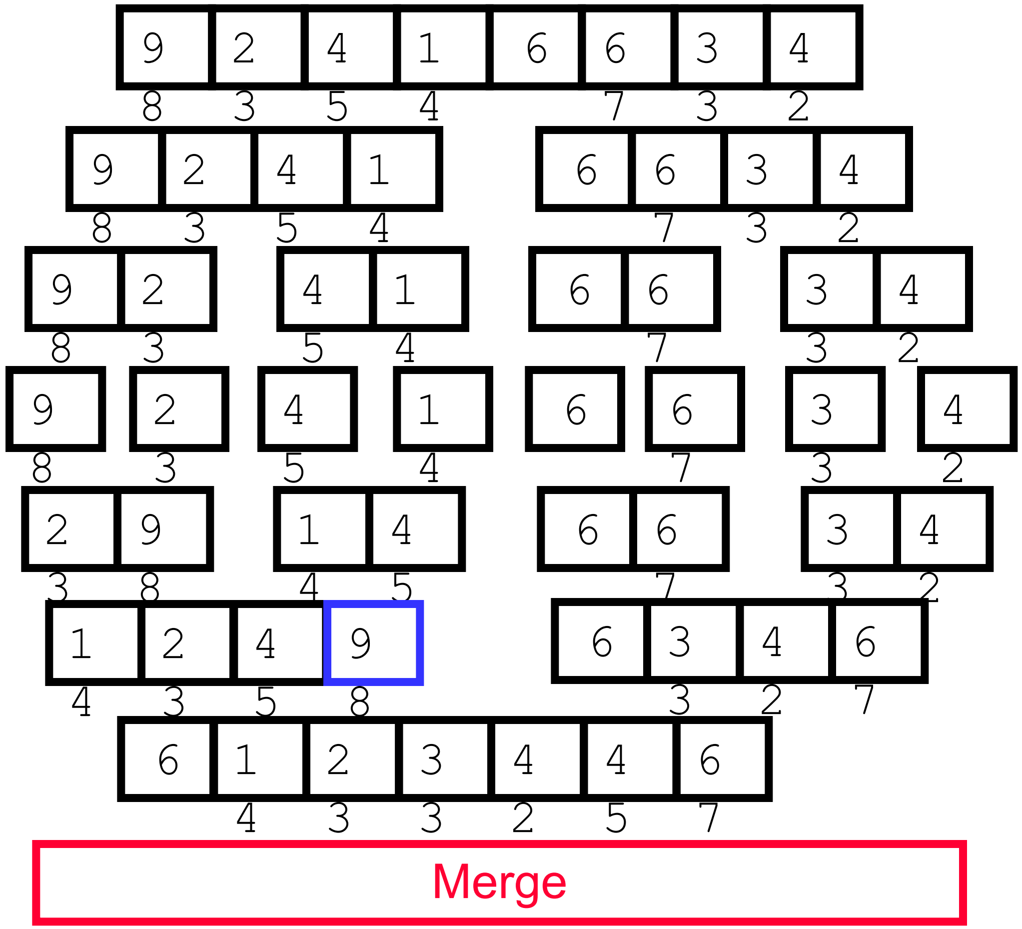
Merge Sort (recursive simulation)



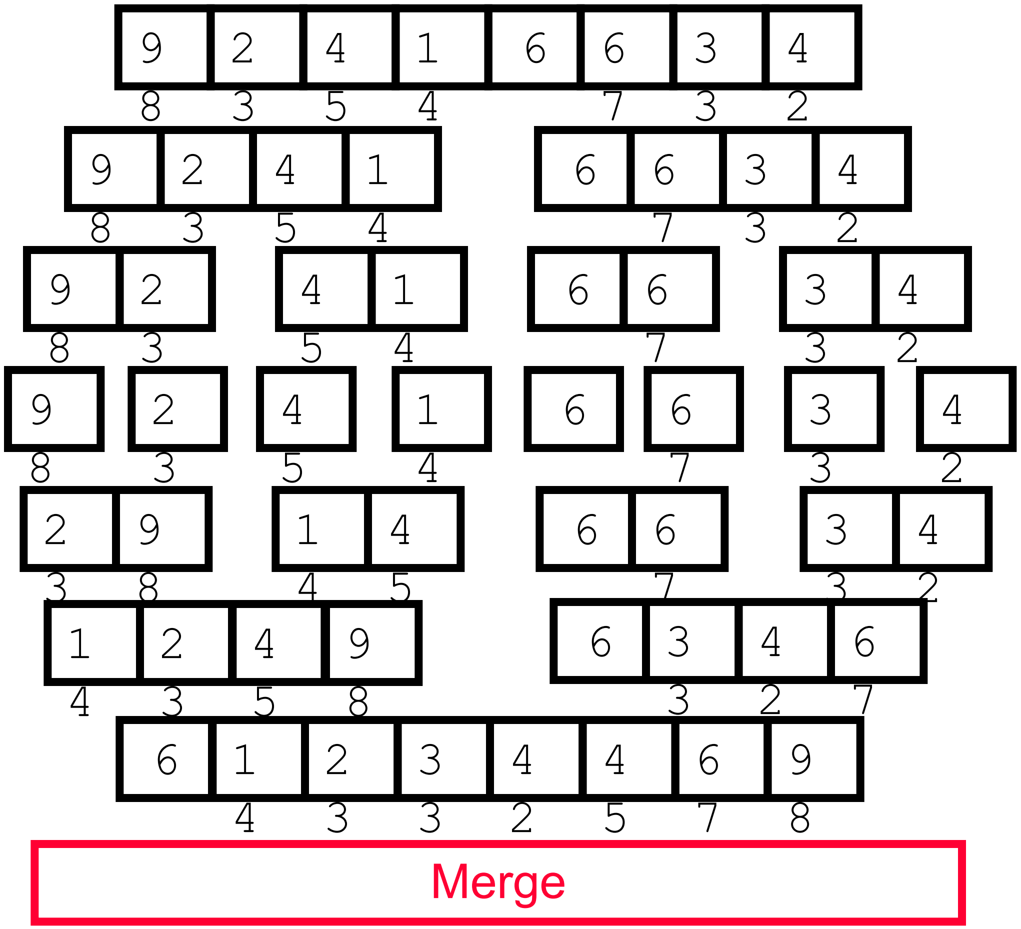
Merge Sort (recursive simulation)



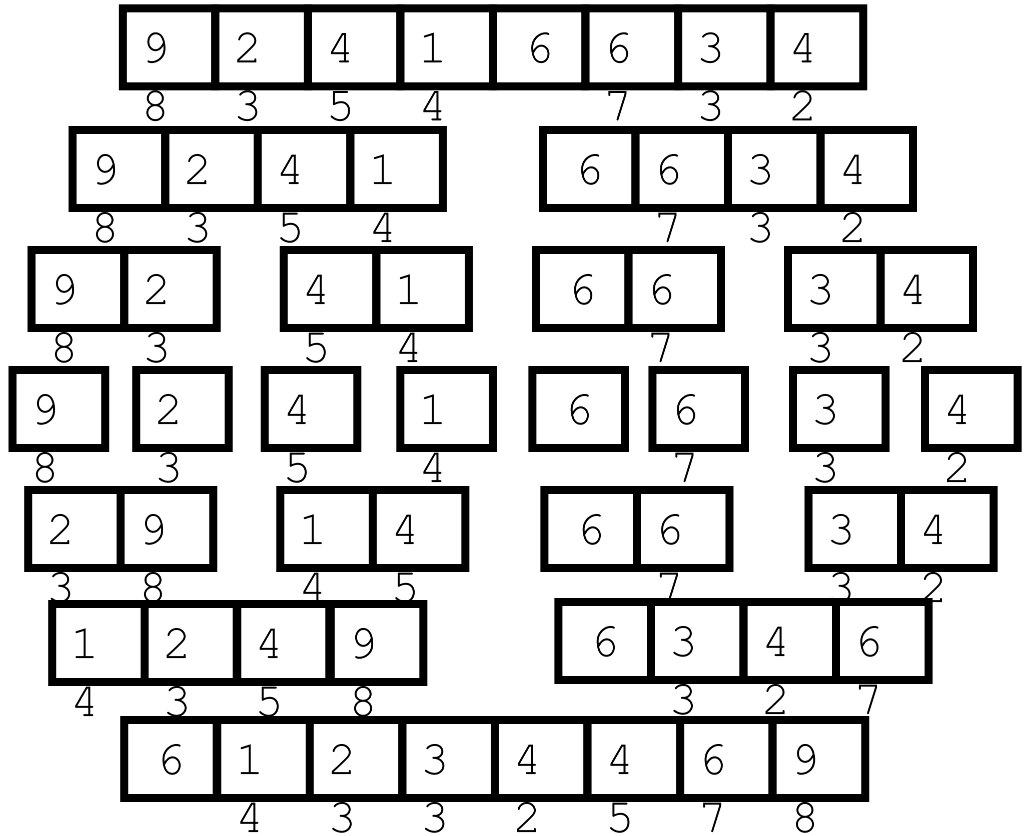
Merge Sort (recursive simulation)

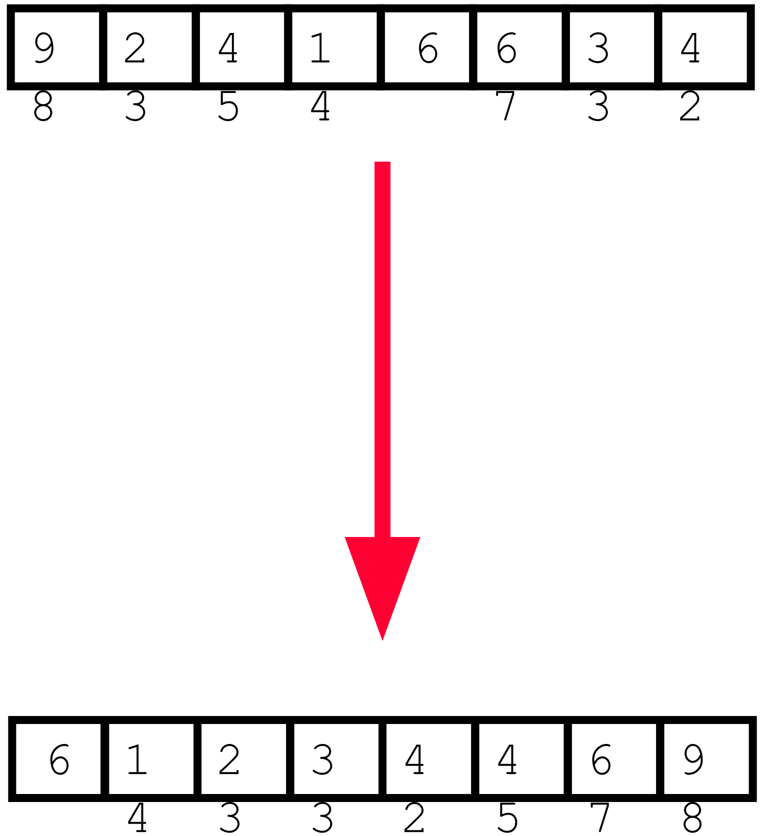


Merge Sort (recursive simulation)

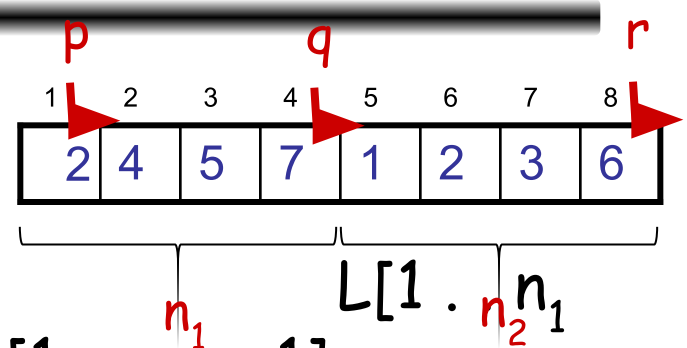


Merge Sort (recursive simulation)



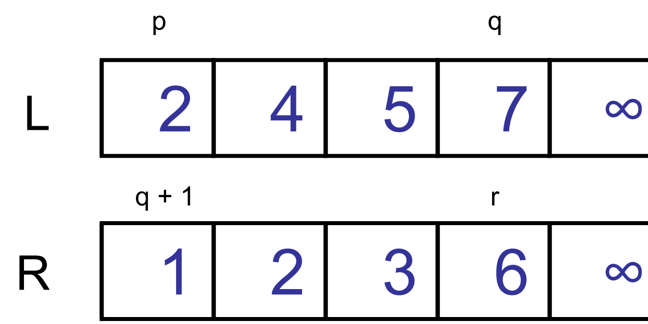

9 2 4 1 6 6 3 4
8 3 5 4 7 3 2
6 1 2 3 4 4 6 9
4 3 3 2 5 7 8

Merge - Pseudocode



Alg.: MERGE(A, p, q, r)

1. Compute n_1 and n_2
2. Copy the first n_1 elements into $L[1 \,.\,.\, n_1 + 1]$ and the next n_2 elements into $R[1 \,.\,.\, n_2 + 1]$
3. $L[n_1 + 1] \leftarrow \infty$; $R[n_2 + 1] \leftarrow \infty$
4. $i \leftarrow 1$; $j \leftarrow 1$
5. **for** $k \leftarrow p$ **to** r
6. **do if** $L[\,i\,] \leq R[\,j\,]$
7. **then** $A[k] \leftarrow L[\,i\,]$
8. $i \leftarrow i + 1$
9. **else** $A[k] \leftarrow R[\,j\,]$
10. $j \leftarrow j + 1$



Merge Sort - Discussion

- Running time insensitive of the input

- Advantages:
  – Guaranteed to run in $\Theta(n \lg n)$

- Disadvantage
  – Requires extra space $\approx N$

Sorting Challenge 1

Problem: Sort a file of huge records with tiny keys

Example application: Reorganize your MP-3 files

Which method to use?

A. merge sort, guaranteed to run in time ~NlgN
B. selection sort
C. bubble sort
D. a custom algorithm for huge records/tiny keys
E. insertion sort

Sorting Files with Huge Records and Small Keys

- Insertion sort or bubble sort?
  - NO, too many exchanges
- Selection sort?
  - YES, it takes linear time for exchanges
- Merge sort or custom method?
  - Probably not: selection sort simpler, does less swaps

Sorting Challenge 2

Problem: Sort a huge randomly-ordered file of small records

Application: Process transaction record for a phone company

Which sorting method to use?

A. Bubble sort
B. Selection sort
C. Mergesort guaranteed to run in time ~N*lg*N
D. Insertion sort

Sorting Huge, Randomly - Ordered Files

- Selection sort?
  - NO, always takes quadratic time
- Bubble sort?
  - NO, quadratic time for randomly-ordered keys
- Insertion sort?
  - NO, quadratic time for randomly-ordered keys
- Mergesort?
  - YES, it is designed for this problem

Sorting Challenge 3

Problem: sort a file that is already almost in order

Applications:

- Re-sort a huge database after a few changes
- Doublecheck that someone else sorted a file

Which sorting method to use?

A. Mergesort, guaranteed to run in time ~NlgN
B. Selection sort
C. Bubble sort
D. A custom algorithm for almost in-order files
E. Insertion sort

Sorting Files That are Almost in Order

- Selection sort?
  - NO, always takes quadratic time
- Bubble sort?
  - NO, bad for some definitions of "almost in order"
  - Ex: B C D E F G H I J K L M N O P Q R S T U V W X Y Z A
- Insertion sort?
  - YES, takes linear time for most definitions of "almost in order"
- Mergesort or custom method?
  - Probably not: insertion sort simpler and faster

Sorting Applications

Sorting algorithms are essential in a broad variety of applications

- Sort a list of names.
- Organize an MP3 library.
- Display Google PageRank results.
- List RSS news items in reverse chronological order.

- Find the median.
- Find the closest pair.
- Binary search in a database.
- Identify statistical outliers.
- Find duplicates in a mailing list

Now I know merge sort!