# Recursion: Example

```
factorial(n)
  if (n == 1)
    return 1;        Base case
  else
    return n * factorial(n - 1);
```

Recurrence Relation

# Another Example: Fibonacci number

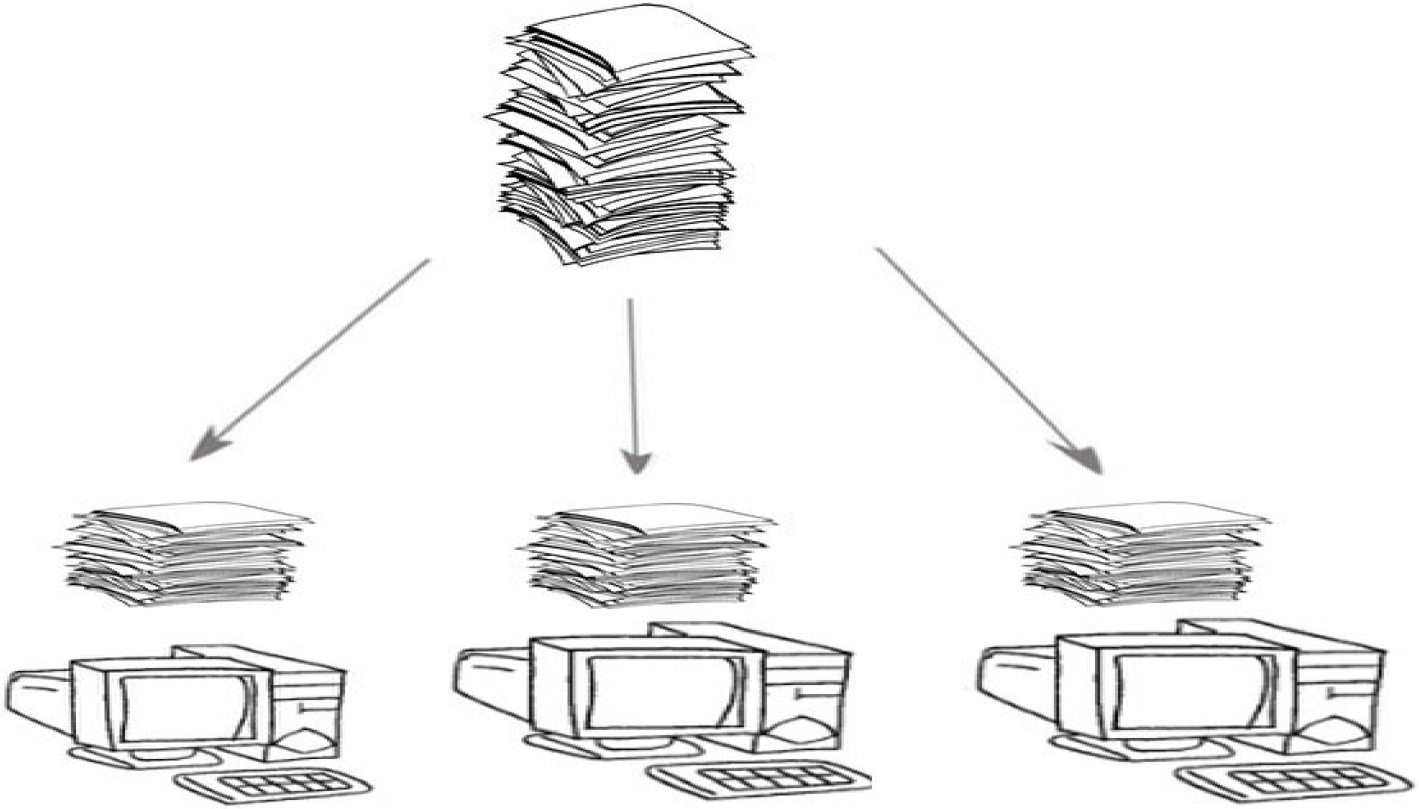
F0=F1=1

$F_n = F_{n-1} + F_{n-2}$ for $n > 1$

1, 1, 2, 3, 5, 8, 13, …..

```
fib(n)
{
if (n <= 1)   return 1;                         Base cases
else  return fib(n-1) + fib(n-2);               Recurrence relation
}
```

# Divide and Conquer



- Basic Principle
- Merge Sort
- Multiplication

# Divide and Conquer: Basic Principle

***At each level of recursion***

- ***Divide:*** ***split*** the problem into sub-problems that are similar to the original but smaller in size

- Recur: solve each sub-problem recursively.

- ***Conquer:*** ***Combine*** the solutions to create a solution to the original problem.

# 1. Merge Sort

# Sorting

Sorting. Given n elements, rearrange in ascending order.

Obvious sorting applications.

- List files in a directory.
- Organize an MP3 library.
- List names in a phone book.
- Display Google PageRank results.

Problems become easier once sorted.

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

Non-obvious sorting applications.

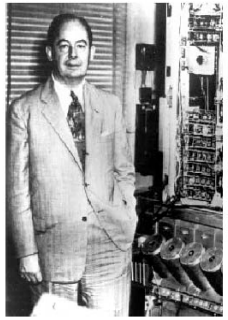
- Data compression.
- Computer graphics.
- Interval scheduling.
- Computational biology.
- Minimum spanning tree.
- Supply chain management.
- Simulate a system of particles.
- Book recommendations on Amazon.
- Load balancing on a parallel computer.

. . .

# Mergesort

## Mergesort.

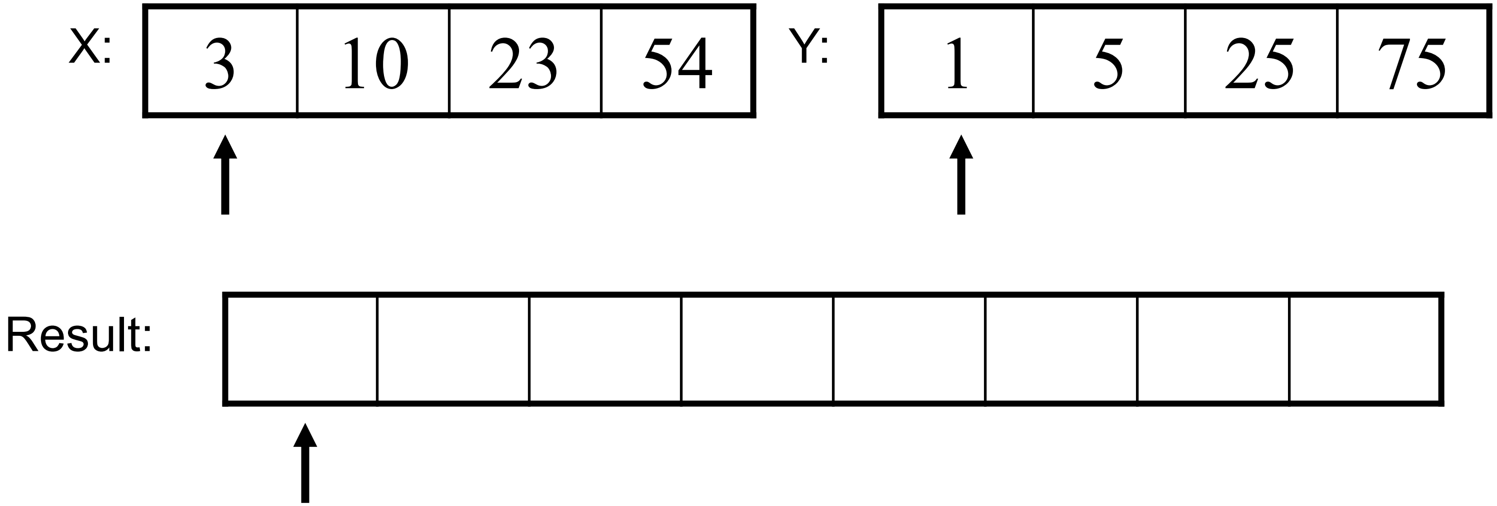
- Divide array into two halves.
- Recursively sort each half.
- Merge two halves to make sorted whole.



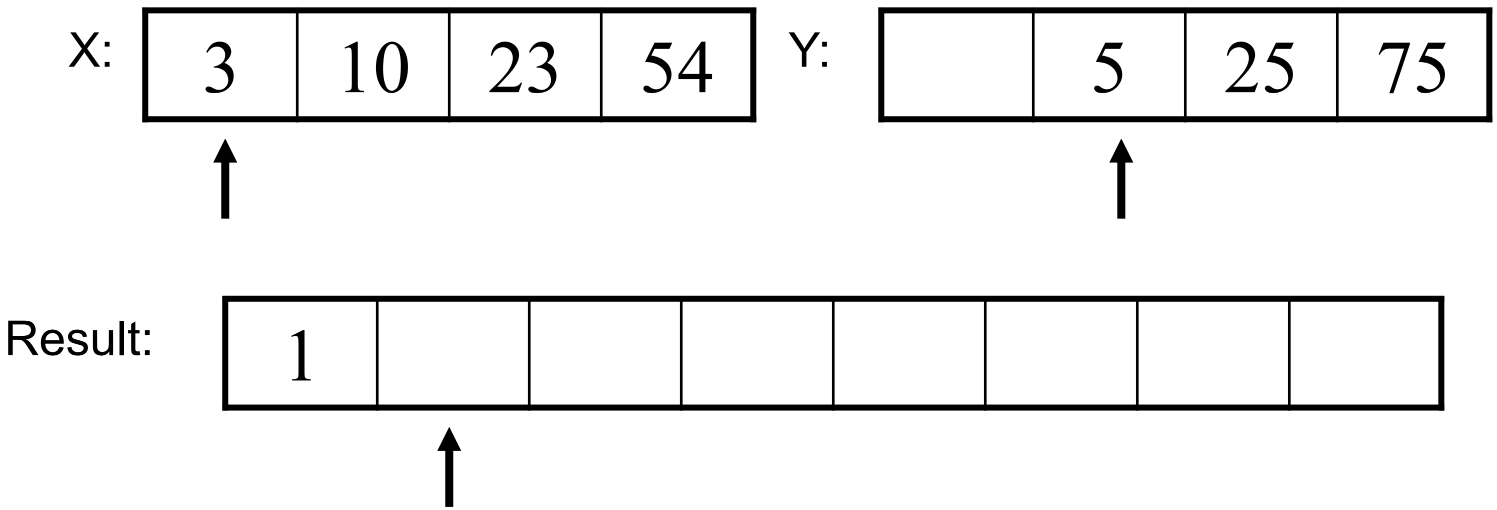
Jon von Neumann (1945)

| | | | |
|---|---|---|---|
| A L G O R I T H M S | | | |
| A L G O R | I T H M S | divide | $O(1)$ |
| A G L O R | H I M S T | sort | $2T(n/2)$ |
| A G H I L M O R S T | | merge | $O(n)$ |

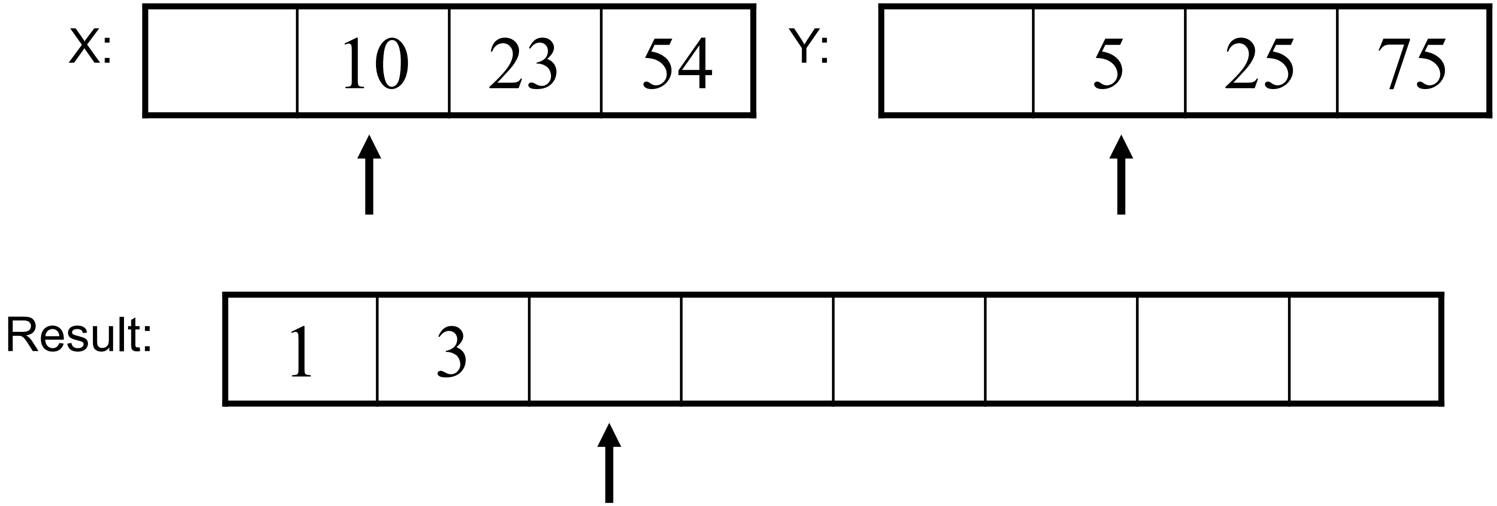
# Merging (cont.)



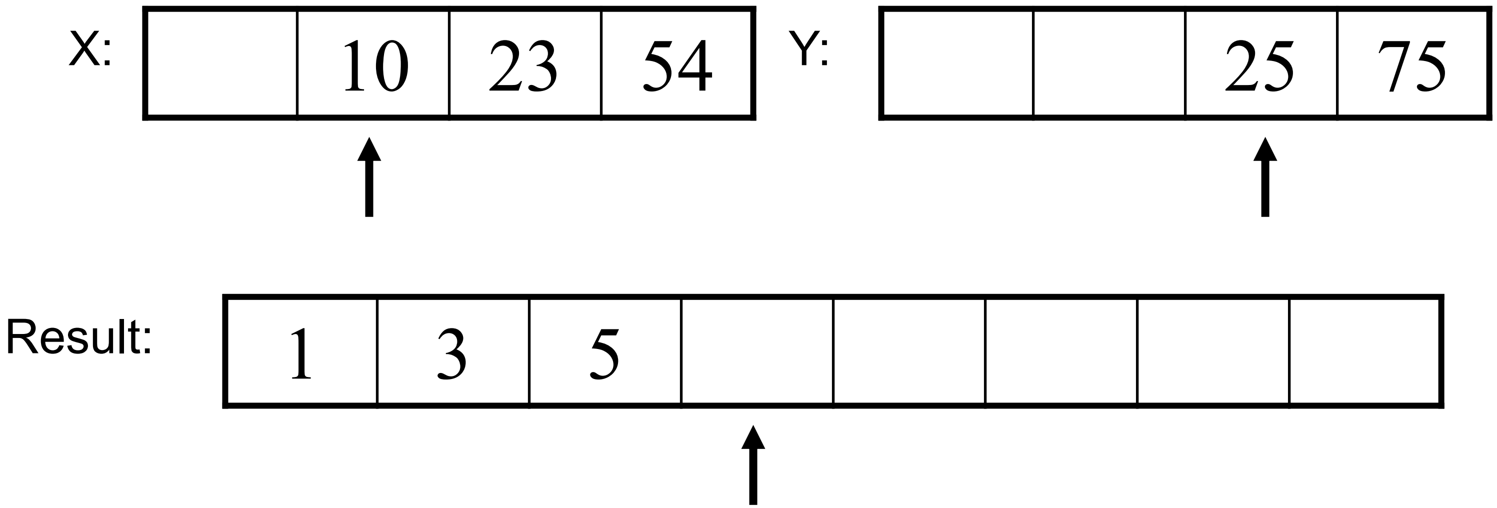
# Merging (cont.)



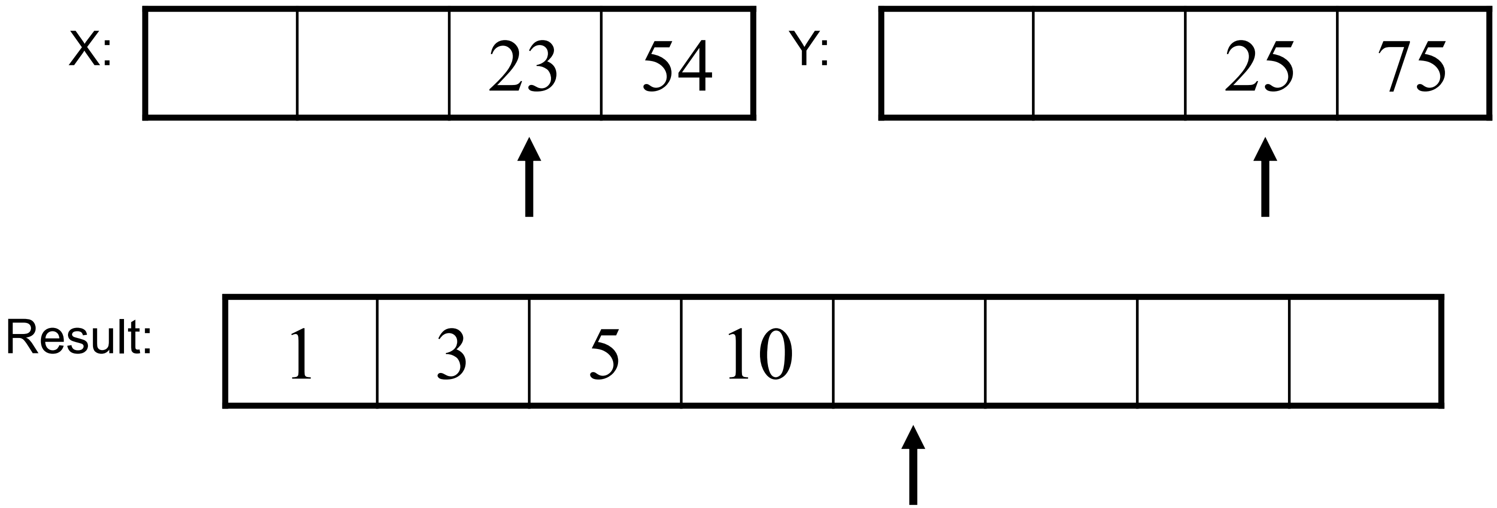
# Merging (cont.)



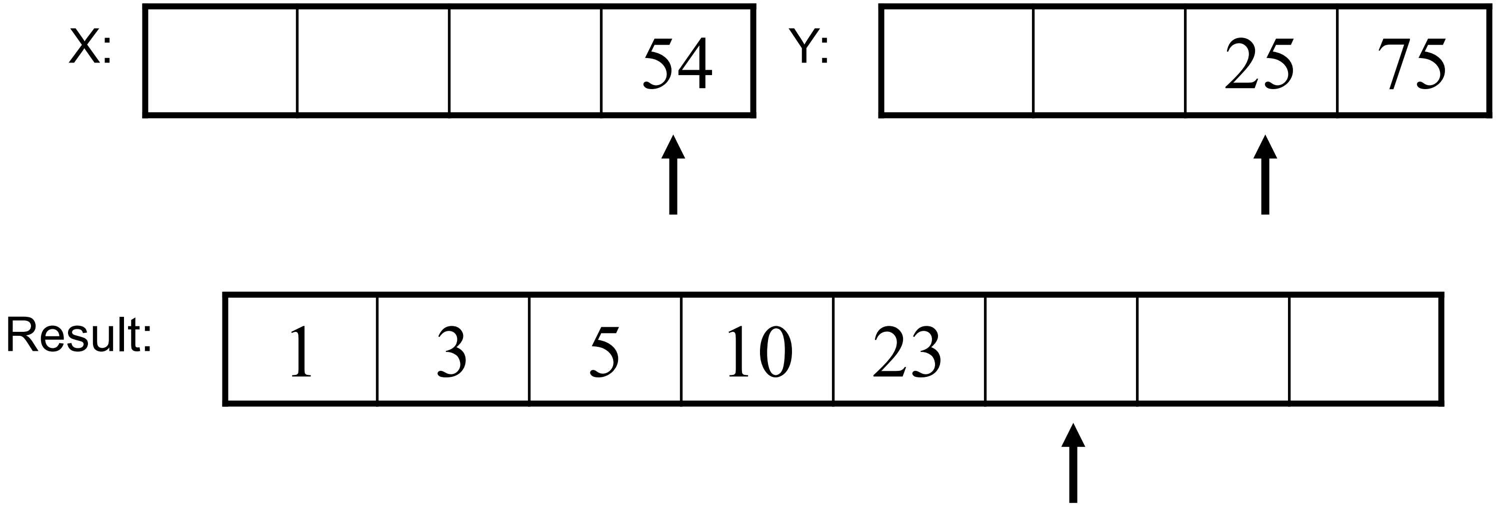
# Merging (cont.)



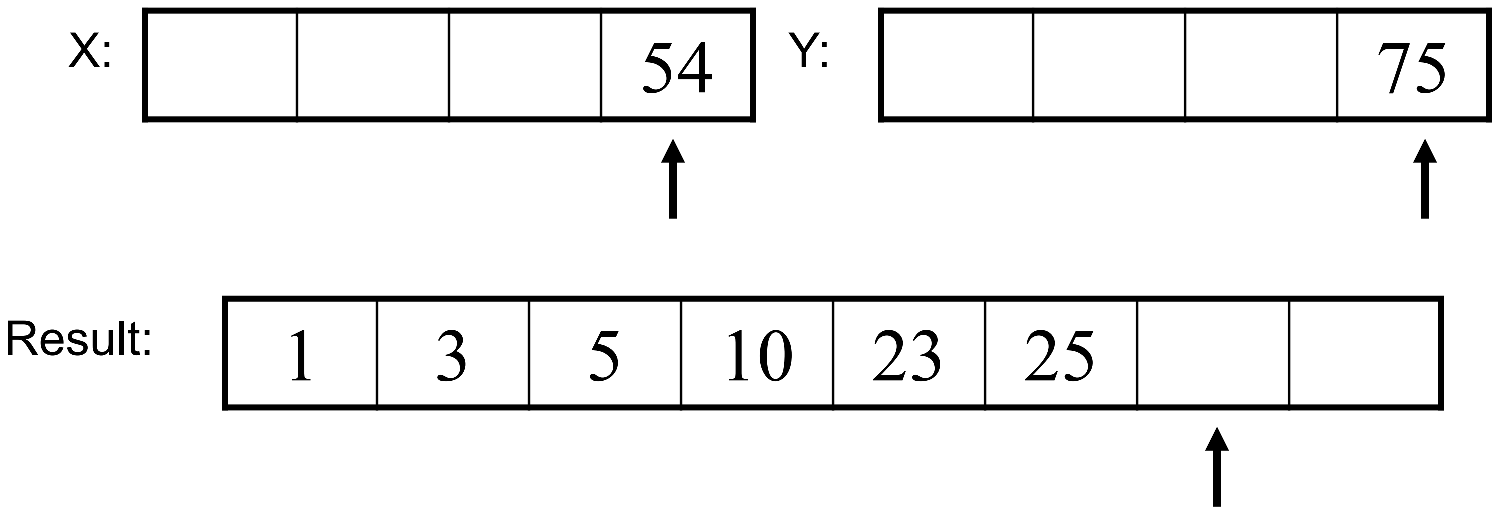
# Merging (cont.)



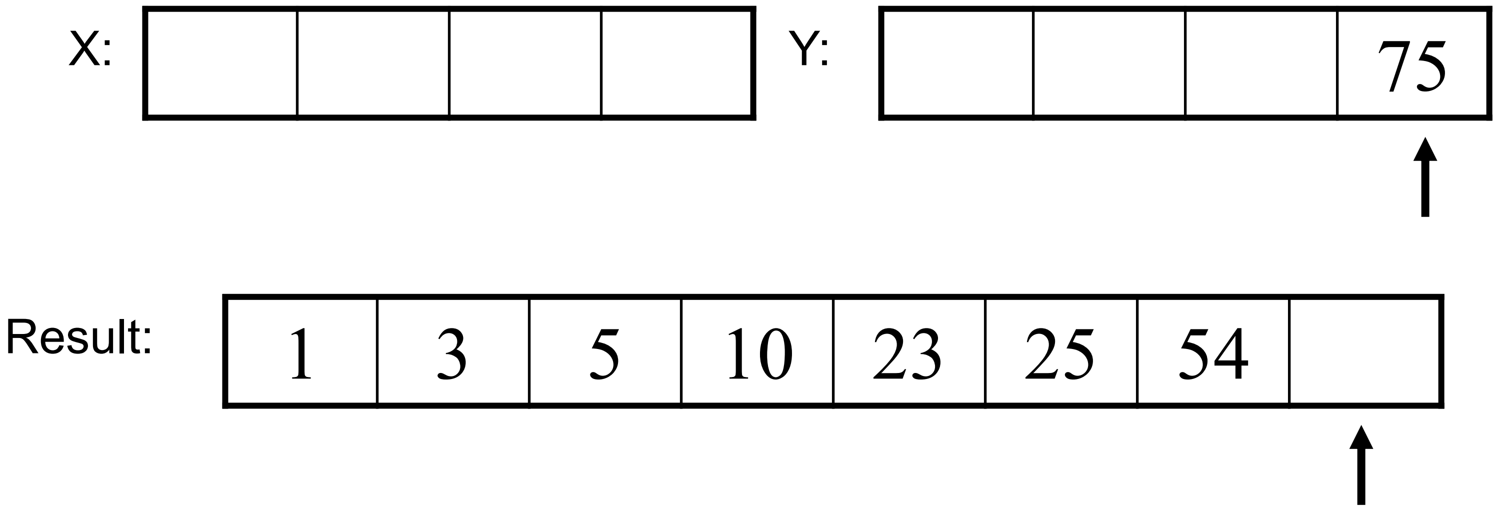
# Merging (cont.)



# Merging (cont.)



# Merging (cont.)



# Merging (cont.)

X:

| | | | |
|---|---|---|---|
| | | | |

Y:

| | | | |
|---|---|---|---|
| | | | |

Result:

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | 3 | 5 | 10 | 23 | 25 | 54 | 75 |

# Merge Sort Example

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 99 | 6 | 86 | 15 | 58 | 35 | 86 | 4 | 0 |

# Merge Sort Example

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 99 | 6 | 86 | 15 | 58 | 35 | 86 | 4 | 0 |

| | | | |
|---|---|---|---|
| 99 | 6 | 86 | 15 |

| | | | | |
|---|---|---|---|---|
| 58 | 35 | 86 | 4 | 0 |

# Merge Sort Example

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 99 | 6 | 86 | 15 | 58 | 35 | 86 | 4 | 0 |

| | | | |
|---|---|---|---|
| 99 | 6 | 86 | 15 |

| | | | | |
|---|---|---|---|---|
| 58 | 35 | 86 | 4 | 0 |

| | |
|---|---|
| 99 | 6 |

| | |
|---|---|
| 86 | 15 |

| | |
|---|---|
| 58 | 35 |

| | | |
|---|---|---|
| 86 | 4 | 0 |

# Merge Sort Example

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 99 | 6 | 86 | 15 | 58 | 35 | 86 | 4 | 0 |

| | | | |
|---|---|---|---|
| 99 | 6 | 86 | 15 |

| | | | | |
|---|---|---|---|---|
| 58 | 35 | 86 | 4 | 0 |

| | |
|---|---|
| 99 | 6 |

| | |
|---|---|
| 86 | 15 |

| | |
|---|---|
| 58 | 35 |

| | | |
|---|---|---|
| 86 | 4 | 0 |

| |
|---|
| 99 |

| |
|---|
| 6 |

| |
|---|
| 86 |

| |
|---|
| 15 |

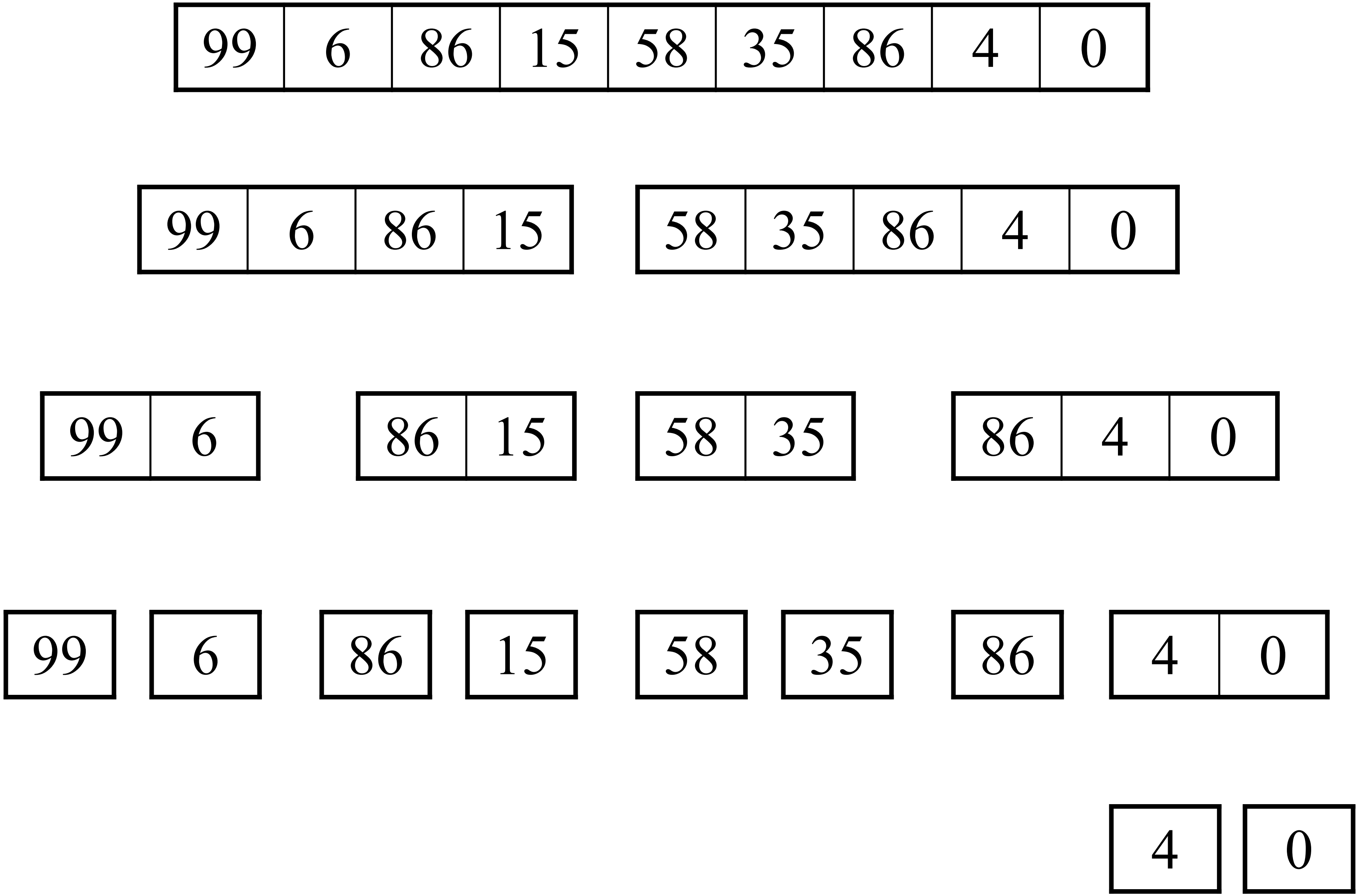
| |
|---|
| 58 |

| |
|---|
| 35 |

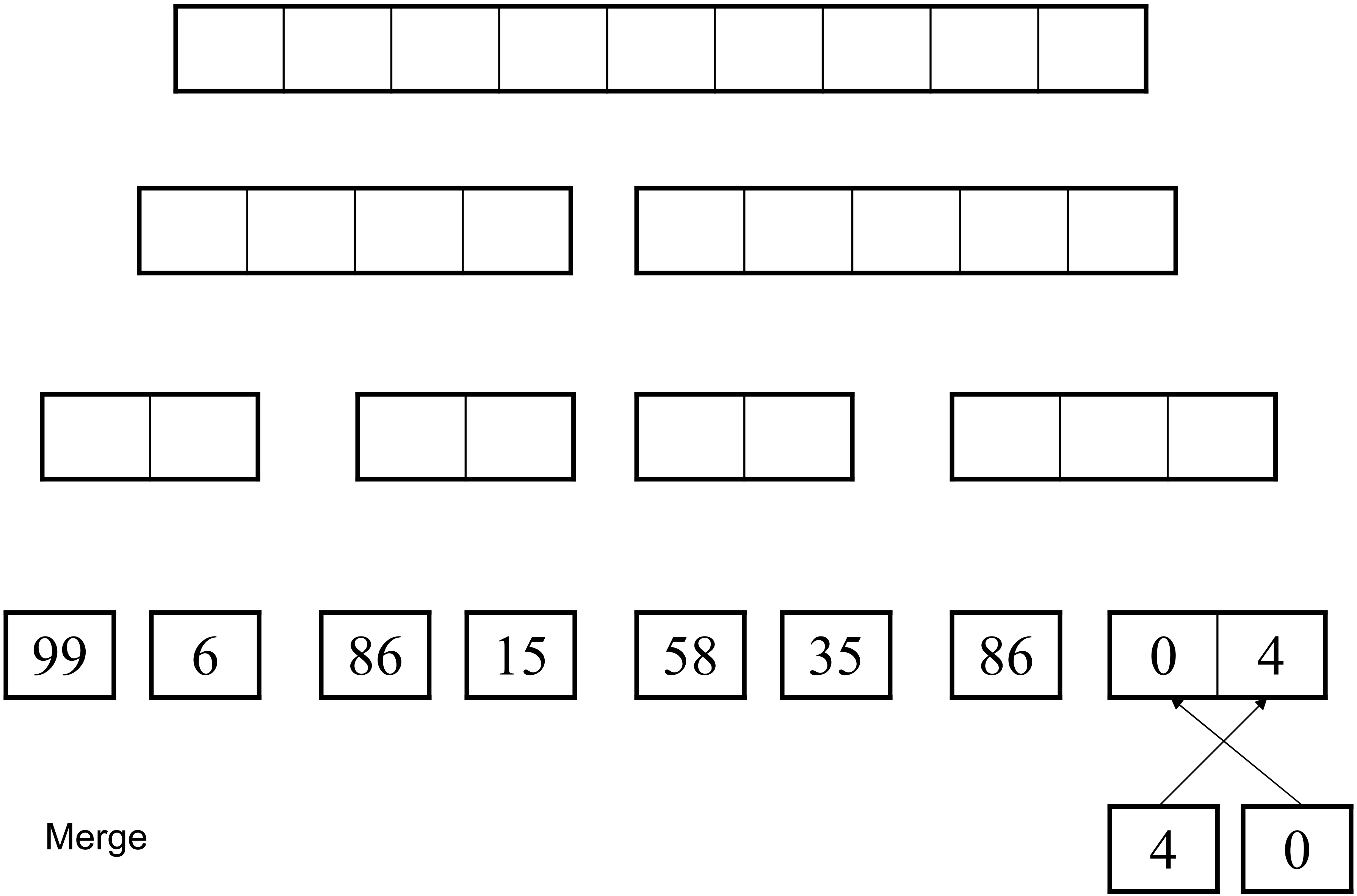
| |
|---|
| 86 |

| | |
|---|---|
| 4 | 0 |

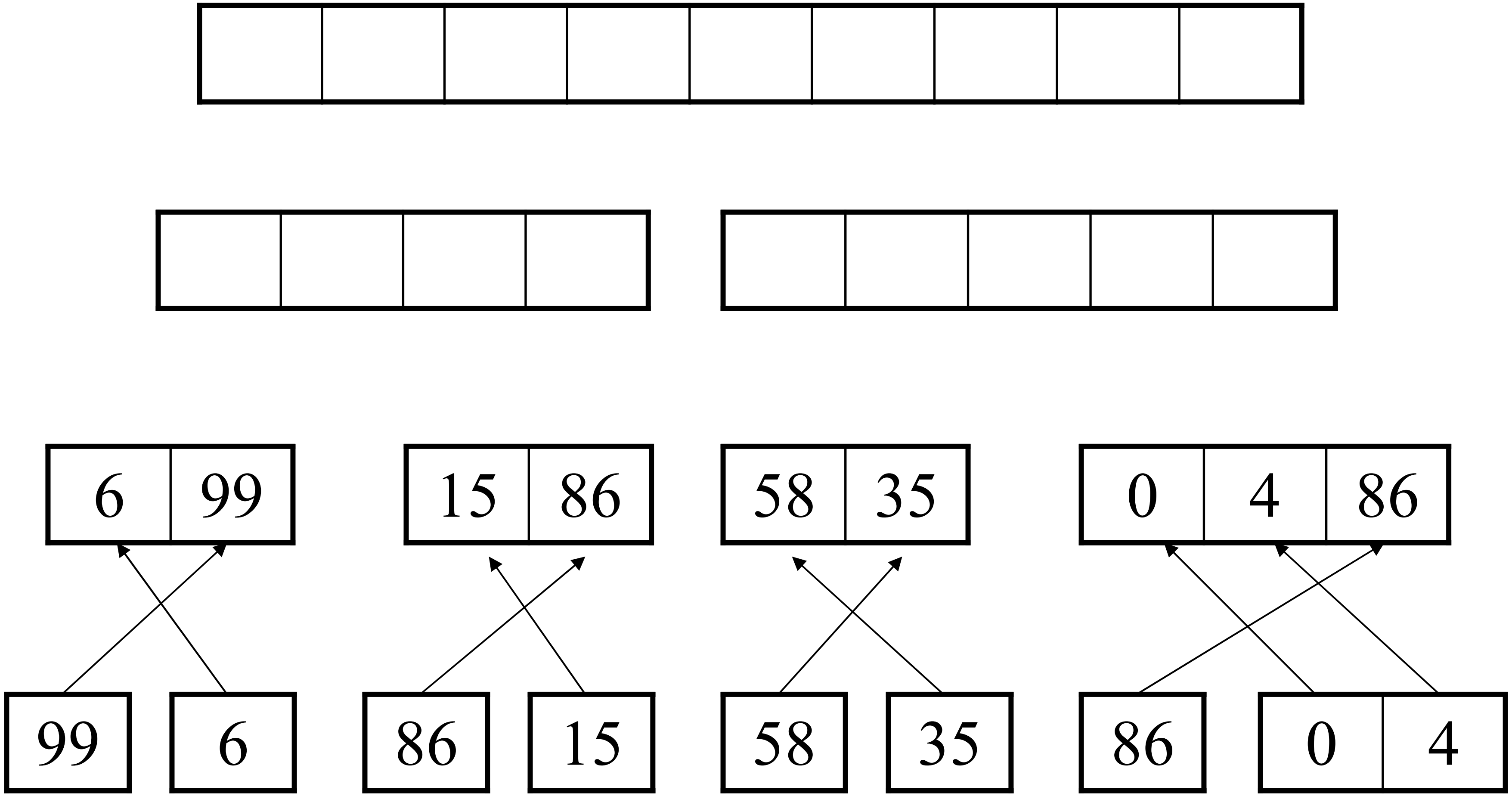
# Merge Sort Example



# Merge Sort Example

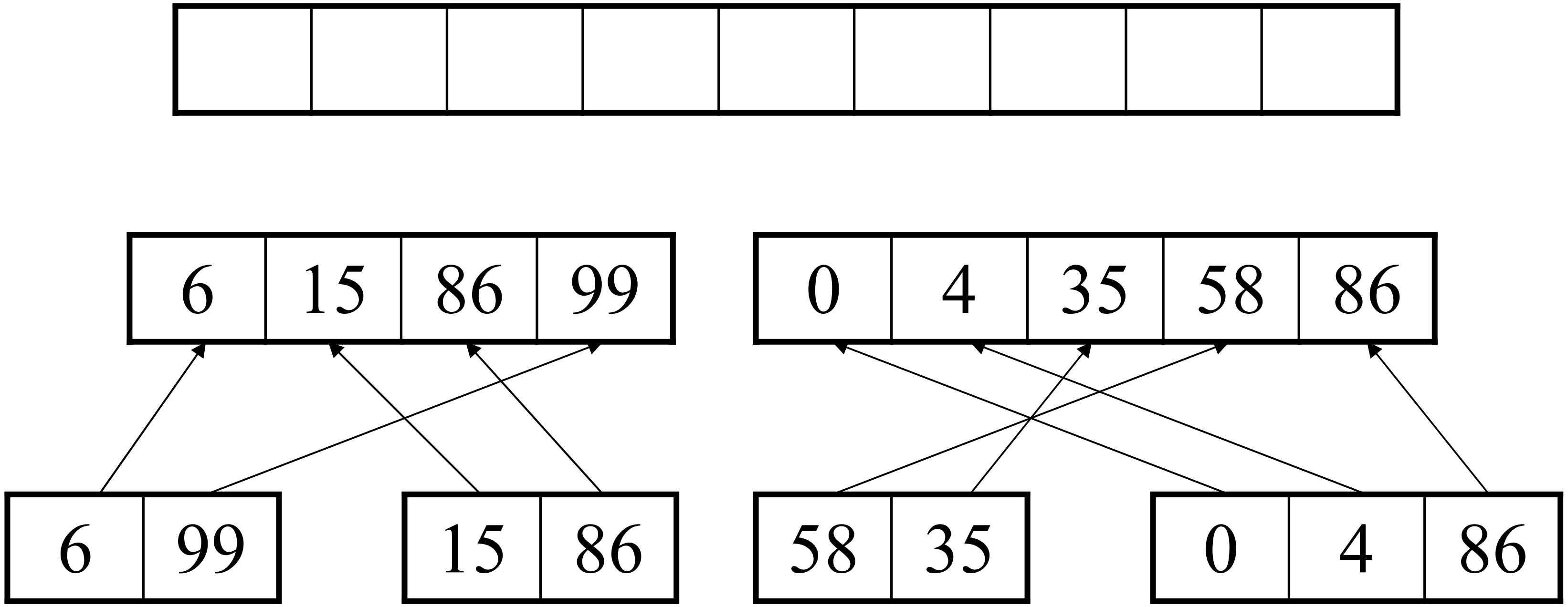


# Merge Sort Example




Merge

# Merge Sort Example




Merge

# Merge Sort Example

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 0 | 4 | 6 | 15 | 35 | 58 | 86 | 86 | 99 |

| | | | |
|---|---|---|---|
| 6 | 15 | 86 | 99 |

| | | | | |
|---|---|---|---|---|
| 0 | 4 | 35 | 58 | 86 |

Merge

# Merge Sort Example

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 0 | 4 | 6 | 15 | 35 | 58 | 86 | 86 | 99 |

# Merge-Sort

- Merge-sort on an input sequence $S$ with $n$ elements consists of three steps:
  - Divide: partition $S$ into two sequences $S_1$ and $S_2$ of about $n/2$ elements each
  - Recur: recursively sort $S_1$ and $S_2$
  - Conquer: merge $S_1$ and $S_2$ into a unique sorted sequence

**Algorithm** ***mergeSort***($S$)
    **Input** sequence $S$ with $n$ elements
    **Output** sequence $S$
        **if** $S.size() > 1$ {
            $(S_1, S_2) \leftarrow partition(S, n/2)$
            $mergeSort(S_1)$
            $mergeSort(S_2)$
            $S \leftarrow merge(S_1, S_2)$}

# A Useful Recurrence Relation

Def. T(n) = number of comparisons to mergesort an input of size n.

Mergesort recurrence.

$$T(n) \le \begin{cases} 0 & \text{if } n = 1 \\ \underbrace{T(\lceil n/2 \rceil)}_{\text{solve left half}} + \underbrace{T(\lfloor n/2 \rfloor)}_{\text{solve right half}} + \underbrace{n}_{\text{merging}} & \text{otherwise} \end{cases}$$

Solution. $T(n) = O(n \log_2 n)$.

Assorted proofs. We describe several ways to prove this recurrence. Initially we assume n is a power of 2 and replace $\le$ with $=$.

$Let\ n = 2^k.$

$$T(\mathrm{n}) = \mathrm{n} + 2 * \mathrm{T}\left(\frac{\mathrm{n}}{2}\right)$$

$$= \mathrm{n} + 2\left\{\frac{\mathrm{n}}{2} + 2 * \mathrm{T}\left(\frac{\mathrm{n}}{2^2}\right)\right\} = 2 * \mathrm{n} + 2^2\mathrm{T}\left(\frac{\mathrm{n}}{2^2}\right)$$

$$= 2 * \mathrm{n} + 2^2\left\{\frac{\mathrm{n}}{2^2} + 2 * \mathrm{T}\left(\frac{\mathrm{n}}{2^3}\right)\right\} = 3 * \mathrm{n} + 2^3\mathrm{T}\left(\frac{\mathrm{n}}{2^3}\right)$$

=....

$=k * n + 2^k T\left(\frac{n}{2^k}\right)$ Seting $\frac{n}{2^k} = 1$, we have k=log n.

$=\log(n) * n + n * T(1) = O(nlog(n)\ )$

$2^{\log n}$=n

# Proof by Telescoping

Claim. If T(n) satisfies this recurrence, then T(n) = n $\log_2$ n.

assumes n is a power of 2

$$T(n) = \begin{cases} 0 & \text{if } n = 1 \\ \underbrace{2T(n/2)}_{\text{sorting both halves}} + \underbrace{n}_{\text{merging}} & \text{otherwise} \end{cases}$$

Pf. For n > 1:

$$\begin{aligned} \frac{T(n)}{n} &= \frac{2T(n/2)}{n} + 1 \\ &= \frac{T(n/2)}{n/2} + 1 \\ &= \frac{T(n/4)}{n/4} + 1 + 1 \\ &\cdots \\ &= \frac{T(n/n)}{n/n} + \underbrace{1 + \cdots + 1}_{\log_2 n} \\ &= \log_2 n \end{aligned}$$

# 2. Counting Inversions

## Counting Inversions

*We can assume that the first rank is 1, 2, ,…n.*

example

Music site tries to match your song preferences with others.

- You rank n songs.
- Music site consults database to find people with similar tastes.

Similarity metric: number of inversions between two rankings.

- My rank: 1, 2, …, n.
- Your rank: $a_1, a_2, \ldots, a_n$.
- Songs i and j inverted if $i < j$, but $a_i > a_j$.

R1: g, q, p, r, m→1, 2, 3, 4, 5

R2: q, p, r, g, m→2, 3, 4, 1, 5

*Songs*

| | A | B | C | D | E |
|---|---|---|---|---|---|
| Me | 1 | 2 | 3 | 4 | 5 |
| You | 1 | 3 | 4 | 2 | 5 |

Inversions

3-2, 4-2

Brute force: check all $\Theta(n^2)$ pairs i and j.

# Applications

Applications.

- Voting theory.
- Collaborative filtering.
- Measuring the "sortedness" of an array.
- Sensitivity analysis of Google's ranking function.
- Rank aggregation for meta-searching on the Web.
- Nonparametric statistics (e.g., Kendall's Tau distance).

## Counting Inversions: Divide-and-Conquer

Divide-and-conquer.

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 5 | 4 | 8 | 10 | 2 | 6 | 9 | 12 | 11 | 3 | 7 |

# Counting Inversions: Divide-and-Conquer

Divide-and-conquer.

- Divide: separate list into two pieces.

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 5 | 4 | 8 | 10 | 2 | 6 | 9 | 12 | 11 | 3 | 7 |

Divide: $O(1)$.

| | | | | | |
|---|---|---|---|---|---|
| 1 | 5 | 4 | 8 | 10 | 2 |

| | | | | | |
|---|---|---|---|---|---|
| 6 | 9 | 12 | 11 | 3 | 7 |

# Counting Inversions: Divide-and-Conquer

Divide-and-conquer.

- Divide: separate list into two pieces.
- Conquer: recursively count inversions in each half.

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 5 | 4 | 8 | 10 | 2 | 6 | 9 | 12 | 11 | 3 | 7 |

Divide: $O(1)$.

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 5 | 4 | 8 | 10 | 2 | 6 | 9 | 12 | 11 | 3 | 7 |

Conquer: $2T(n / 2)$

5 blue-blue inversions

5-4, 5-2, 4-2, 8-2, 10-2

8 green-green inversions

6-3, 9-3, 9-7, 12-3, 12-7, 12-11, 11-3, 11-7

# Counting Inversions: Divide-and-Conquer

Divide-and-conquer.

- Divide: separate list into two pieces.
- Conquer: recursively count inversions in each half.
- Combine: count inversions where $a_i$ and $a_j$ are in different halves, and return sum of three quantities.

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 5 | 4 | 8 | 10 | 2 | 6 | 9 | 12 | 11 | 3 | 7 |

Divide: O(1).

| | | | | | |
|---|---|---|---|---|---|
| 1 | 5 | 4 | 8 | 10 | 2 |

| | | | | | |
|---|---|---|---|---|---|
| 6 | 9 | 12 | 11 | 3 | 7 |

Conquer: 2T(n / 2)

5 blue-blue inversions

8 green-green inversions

9 blue-green inversions

5-3, 4-3, 8-6, 8-3, 8-7, 10-6, 10-9, 10-3, 10-7

Combine: ???

Total = 5 + 8 + 9 = 22.

# Counting Inversions: Combine

Combine: count blue-green inversions

- Assume each half is sorted.
- Count inversions where $a_i$ and $a_j$ are in different halves.
- Merge two sorted halves into sorted whole.

to maintain sorted invariant

| 3 | 7 | 10 | 14 | 18 | 19 |
|---|---|---|---|---|---|

| 2 | 11 | 16 | 17 | 23 | 25 |
|---|---|---|---|---|---|
| 6 | 3 | 2 | 2 | 0 | 0 |

13 blue-green inversions: 6 + 3 + 2 + 2 + 0 + 0

Count: O(n)

| 2 | 3 | 7 | 10 | 11 | 14 | 16 | 17 | 18 | 19 | 23 | 25 |
|---|---|---|---|---|---|---|---|---|---|---|---|

Merge: O(n)

$$T(n) \le T\left(\lfloor n/2 \rfloor\right) + T\left(\lceil n/2 \rceil\right) + O(n) \quad \Rightarrow \quad T(n) = O(n \log n)$$

# Counting Inversions: Implementation

Pre-condition. [Merge-and-Count] A and B are sorted.
Post-condition. [Sort-and-Count] L is sorted.

```
Sort-and-Count(L) {
   if list L has one element
      return 0 and the list L

   Divide the list into two halves A and B
   (rA, A) ← Sort-and-Count(A)
   (rB, B) ← Sort-and-Count(B)
   (r , L) ← Merge-and-Count(A, B)

   return r = rA + rB + r and the sorted list L
}
```

# Merge and Count Process: Another example.

1, 2, 8, 10, 11, 12;     3, 4, 5, 6, 7, 9;

*4 4 4 4 4 3*

*# of inversions*

1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12.

Time for merge : *O(n).*

# 3. Number Multiplication

# Integer Arithmetic

Given two N-digit integers a and b, compute a + b.

- O(N) bit operations.

Multiplication: given two N-digit integers a and b, compute ab.

- Brute force solution: $\Theta$ ($N^2$) bit operations.

Application.

- Cryptography.

```
  1 1 1 1 1 1 0 1
    1 1 0 1 0 1 0 1
+   0 1 1 1 1 1 0 1
  1 0 1 0 1 0 0 1 0
```

```
                  1 1 0 1 0 1 0 1
                * 0 1 1 1 1 1 0 1
                1 1 0 1 0 1 0 1 0
              0 0 0 0 0 0 0 0 0
            1 1 0 1 0 1 0 1 0
          1 1 0 1 0 1 0 1 0
        1 1 0 1 0 1 0 1 0
      1 1 0 1 0 1 0 1 0
    1 1 0 1 0 1 0 1 0
  0 0 0 0 0 0 0 0 0
0 1 1 0 1 0 0 0 0 0 0 0 0 0 1 0
```

30

# Divide-and-Conquer Multiplication:  First Attempt

To multiply two N-digit integers:

- Multiply four N/2-digit integers.
- Add two N/2-digit integers, and shift to obtain result.

$$
\begin{aligned}
123{,}456 \times 987{,}654 &= (10^3 w + x) \times (10^3 y + z) \\
&= 10^6 (wy) + 10^3 (wz + xy) + 10^0 (xz) \\
&= 10^6 (121{,}401) + 10^3 (80{,}442 + 450{,}072) + 10^0 (298{,}224) \\
&= 121{,}401{,}299{,}224
\end{aligned}
$$

$$
\begin{aligned}
w &= 123 \\
x &= 456 \\
y &= 987 \\
z &= 654
\end{aligned}
$$

N is a power of 2 ⇒

$$ab = (10^{N/2} w + x)(10^{N/2} y + z)$$

$$T(N) = \underbrace{4T(N/2)}_{\text{recursive calls}} + \underbrace{\Theta(N)}_{\text{add, shift}} \Rightarrow T(N) = \Theta(N^2)$$

# Karatsuba Multiplication

To multiply two N-digit integers:

- Add two N/2 digit integers.
- Multiply three N/2-digit integers.
- Subtract two N/2-digit integers, and shift to obtain result.

$$
\begin{aligned}
123{,}456 \times 987{,}654 &= (10^3 w + x)\times(10^3 y + z) \\
&= 10^6 (wy) + 10^3 (wz + xy) + 10^0 (xz) \\
&= 10^6 (p) + 10^3 (r - p - q) + 10^0 (q) \\
&= 10^6 (121{,}401) + 10^3 (950{,}139 - 121{,}401 - 298{,}224) + 10^0 (298{,}224) \\
& \phantom{=} 121{,}401{,}299{,}224
\end{aligned}
$$

$$
\begin{aligned}
w &= 123 \\
x &= 456 \\
y &= 987 \\
z &= 654
\end{aligned}
$$

$$
\begin{aligned}
p &= wy \\
q &= xz \\
r &= (w + x)(y + z)
\end{aligned}
$$

$$(wz + xy) = r - p - q$$

# Karatsuba Multiplication: Analysis

*Solving recurrence equation is not required.*

To multiply two N-digit integers:

- Add two N/2 digit integers.
- Multiply three N/2-digit integers.
- Subtract two N/2-digit integers, and shift to obtain result.

Karatsuba-Ofman (1962).

- $O(N^{1.585})$ bit operations.

$$
\begin{aligned}
p &= wy \\
q &= xz \\
r &= (w+x)(y+z)
\end{aligned}
$$

$$(wz + xy) = r -$$

$$ab = (10^{N/2} w + x)(10^{N/2} y + z)$$

$$T(N) \le \underbrace{T(\lfloor N/2 \rfloor) + T(\lceil N/2 \rceil) + T(1 + \lceil N/2 \rceil)}_{\text{recursive calls}} + \underbrace{\Theta(N)}_{\text{add, subtract, shift}}$$

$$\Rightarrow T(N) = O(N^{\log_2 3})$$

# Summary

- Basic Principle
- Merge Sort
- Multiplication (Fun part, will not be tested

We use this example to show that

  - Recurrence equations play important role to estimate the running time
  - Solving recurrence equations is very hard and there are postgraduate courses for solving recurrence equations.

- Solving recurrence equation of slide 33 is not required.