SCC120 Fundamentals of Computer Science

Searching and Sorting

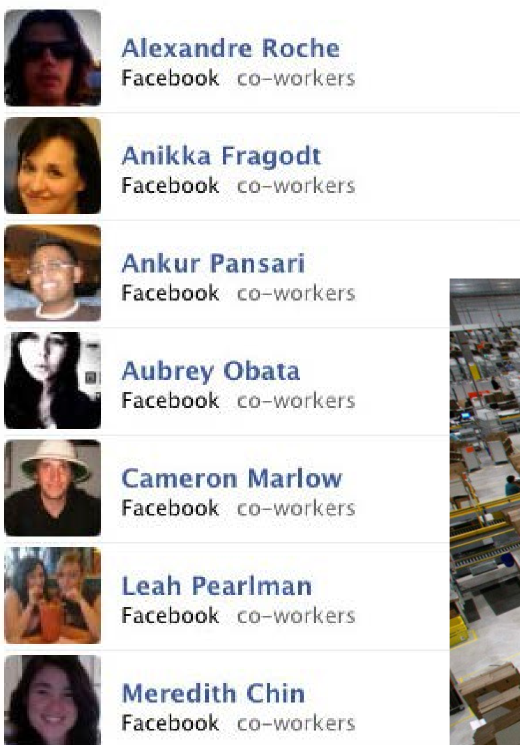
Unit 2: Sorting Techniques

Jidong Yuan

yuanjd@bjtu.edu.cn

Sorting is …

- Taking a set of values, or objects with keys, and arranging them into order of the values/keys
- Order may be ascending or descending



Reminder of previous Units

- Insertion sort (Taught @ Algorithms)
- searching – linear $O(N)$ and binary $O(\log_2 N)$

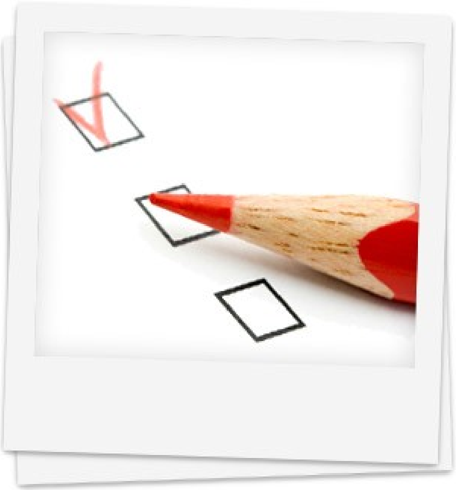
  for binary search, data must be sorted

This Unit

Four comparison-based sorting algorithms:

- Selection sort
- Insertion sort (recap)
- Quick sort
- Merge sort

Selection Sort



Selection Sort

- **Very Simple Idea:**
  - Find the smallest element in the array
  - Exchange it with the element in the first position
  - Find the second smallest element and exchange it with the element in the second position
  - Continue until the array is sorted

Selection Sort Example

8	4	6	9	2	3	1

1	4	6	9	2	3	8

1	2	6	9	4	3	8

1	2	3	9	4	6	8

1	2	3	4	9	6	8

1	2	3	4	6	9	8

1	2	3	4	6	8	9

1	2	3	4	6	8	9

Selection Sort Code

```
void selectionSort (int a[], int N)
{
    for (int i = 0; i < N; i++)
    {
        int min_idx = i;
        for (int j = i+1; j < N; j++)
        {
                if (a[j] < a[min_idx])
                    min_idx = j;
        }
        exch(a, i, min_idx);
    }

}
```

```
tmp = a[i];
a[i] = a[min_idx];
a[min_idx] = tmp;
```

Selection sort efficiency

```
for (int i = 0; i < 4; i++)
   ...
     for (int j = i+1; j < 4; j++)
          if (a[j] < a[min])
```

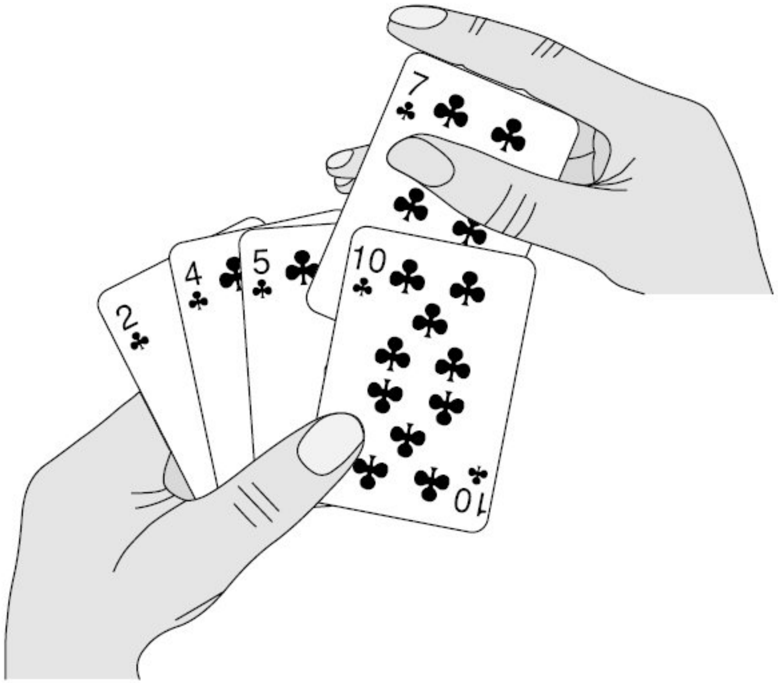
for N=4:

- 3 comparisons when i=0
- 2 comparisons when i=1
- 1 comparison when i=2
- 0 comparison when i=3

$$T(4) = 3 + 2 + 1 + 0 = 6$$

$$T(N) = (N-1) + (N-2) + \ldots + 1 + 0 = \frac{N(N-1)}{2} = (½N^2 – ½N) \sim ½N^2$$

Selection sort belongs to the complexity class $O(N^2)$



INSERTION SORT

Quick revision of Week 8 of Introduction to Algorithms

Insertion sort

A straightforward algorithm for an array or a linked list

Approach

array/list split into two segments

$\rightarrow$ left segment is already sorted

$\rightarrow$ right segment is yet to be sorted

Note: initially, left segment will be empty

sorted elements	unsorted elements

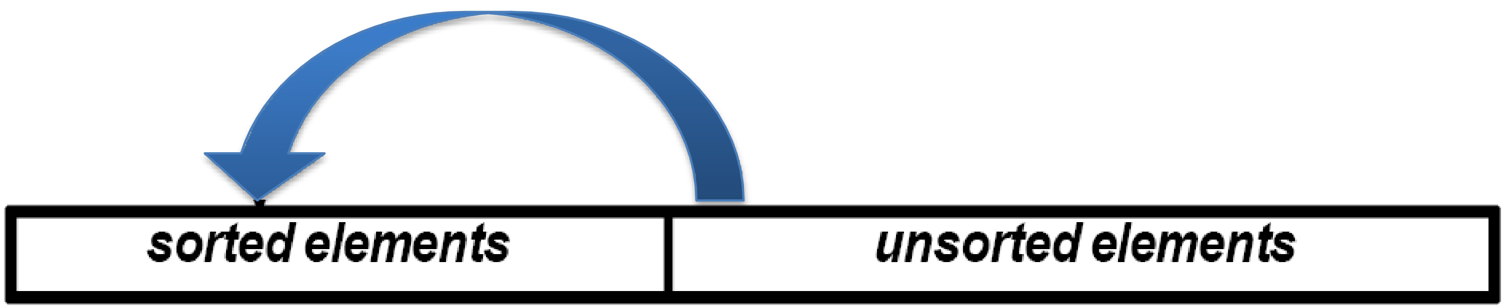
Insertion sort

Approach

take the first element of the right segment

insert at the correct position in the left segment

repeat until right segment is empty....

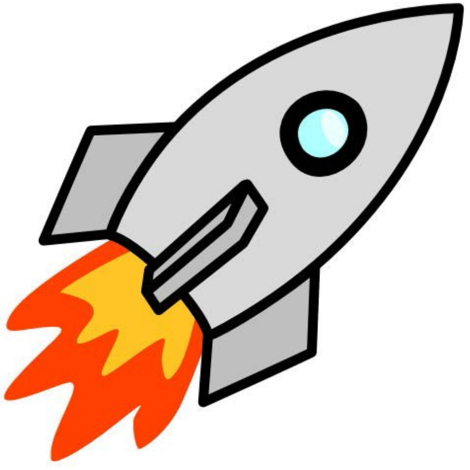


Insertion sort

```
void insertionSort (int A[]) {
      for (int i=1; i<A.length; i++) {
            int x = A[i];
            int j;
            for (j=i-1; j>=0 && A[j]>x; j--) {
                  A[j+1] = A[j];
            }
            A[j+1] = x;
      }
}
```

Time complexity:

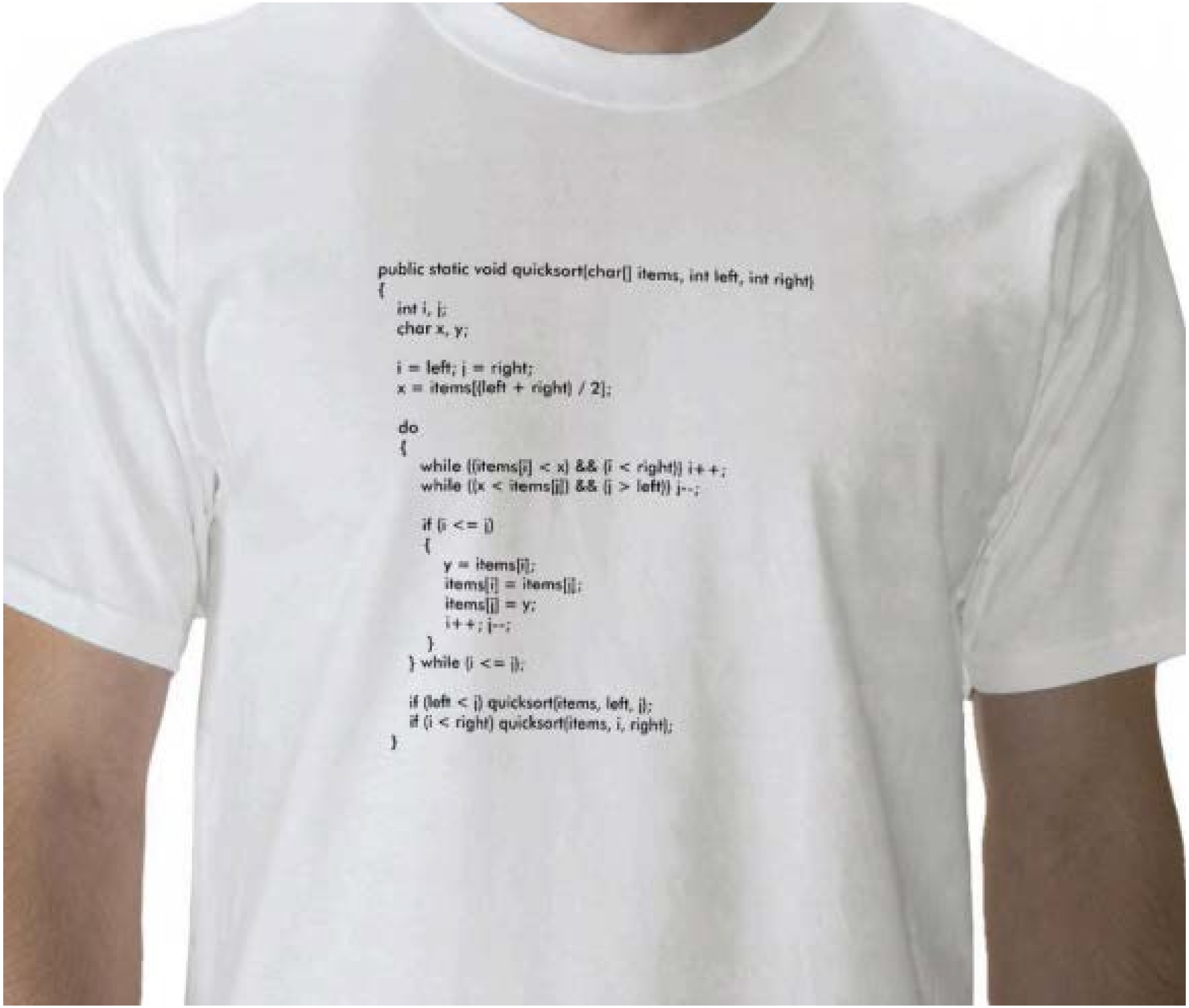
- Best case (sorted data): *O(N)*
- Worse case: $O(N^2)$



Quick Sort

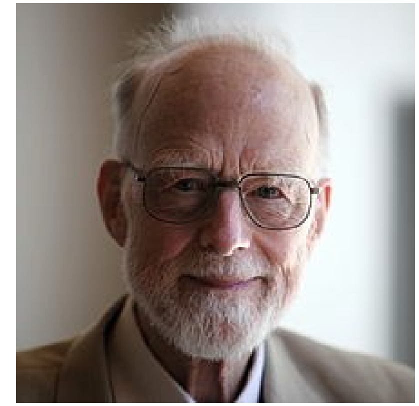
One of top 10 algorithms of 20th century in science and engineering.

Quicksort T-shirt

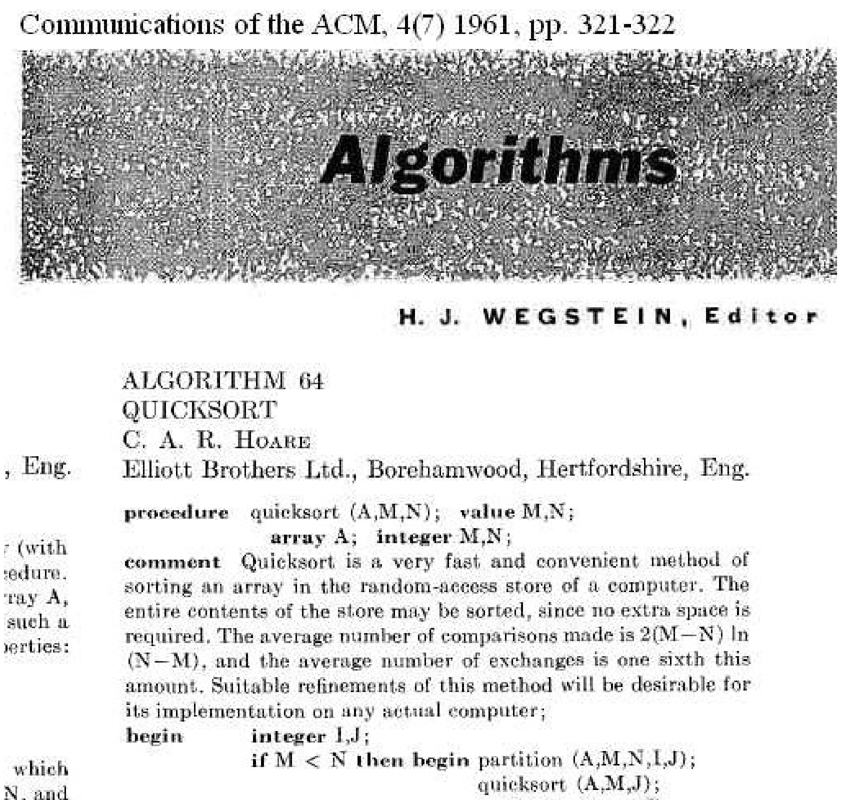


Tony Hoare

- Invented quicksort to translate Russian into English
- Learned Algol (and recursion).
- Implemented quick sort



Tony Hoare
1980 Turing Award



Communications of the ACM, 4(7) 1961, pp. 321-322

Algorithms

H. J. WEGSTEIN, Editor

ALGORITHM 64
QUICKSORT
C. A. R. HOARE
Elliott Brothers Ltd., Borehamwood, Hertfordshire, Eng.

```
procedure  quicksort (A,M,N);  value M,N;
              array A;  integer M,N;
comment  Quicksort is a very fast and convenient method of
sorting an array in the random-access store of a computer. The
entire contents of the store may be sorted, since no extra space is
required. The average number of comparisons made is 2(M−N) ln
(N−M), and the average number of exchanges is one sixth this
amount. Suitable refinements of this method will be desirable for
its implementation on any actual computer;
begin        integer I,J;
             if M < N then begin partition (A,M,N,I,J);
                                  quicksort (A,M,J);
```

Quick sort

a *divide-and-conquer* algorithm

i.e. it divides up the data, processes each portion separately and recombines the results

Approach

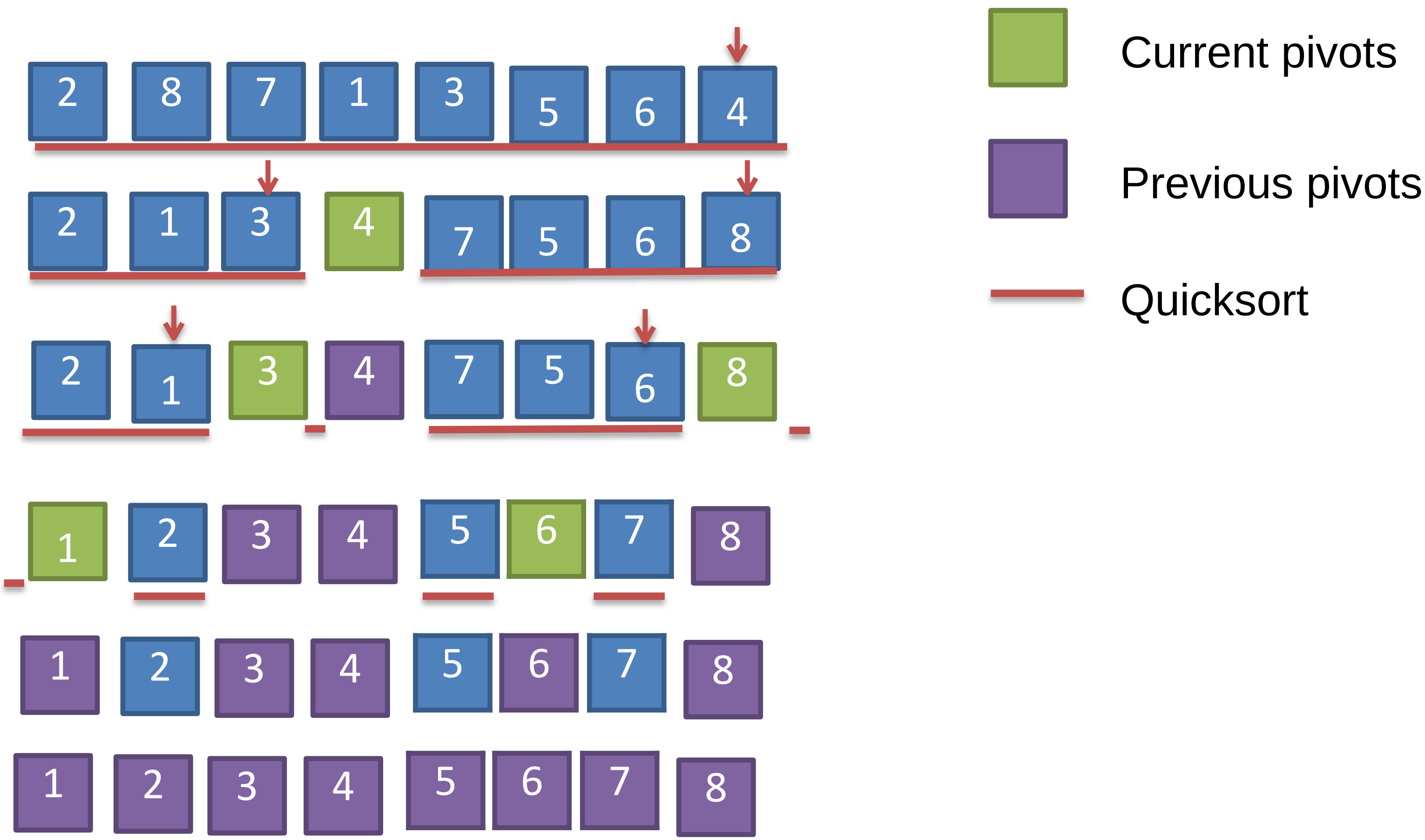
1 Pick an element X from the set of data

2 Put all elements < X on the left; put all elements >= X on the right; put X at the middle (**Partition**)

3 Quicksort (the array on the left of X)

4 Quicksort (the array on the right of X)

Quicksort Example

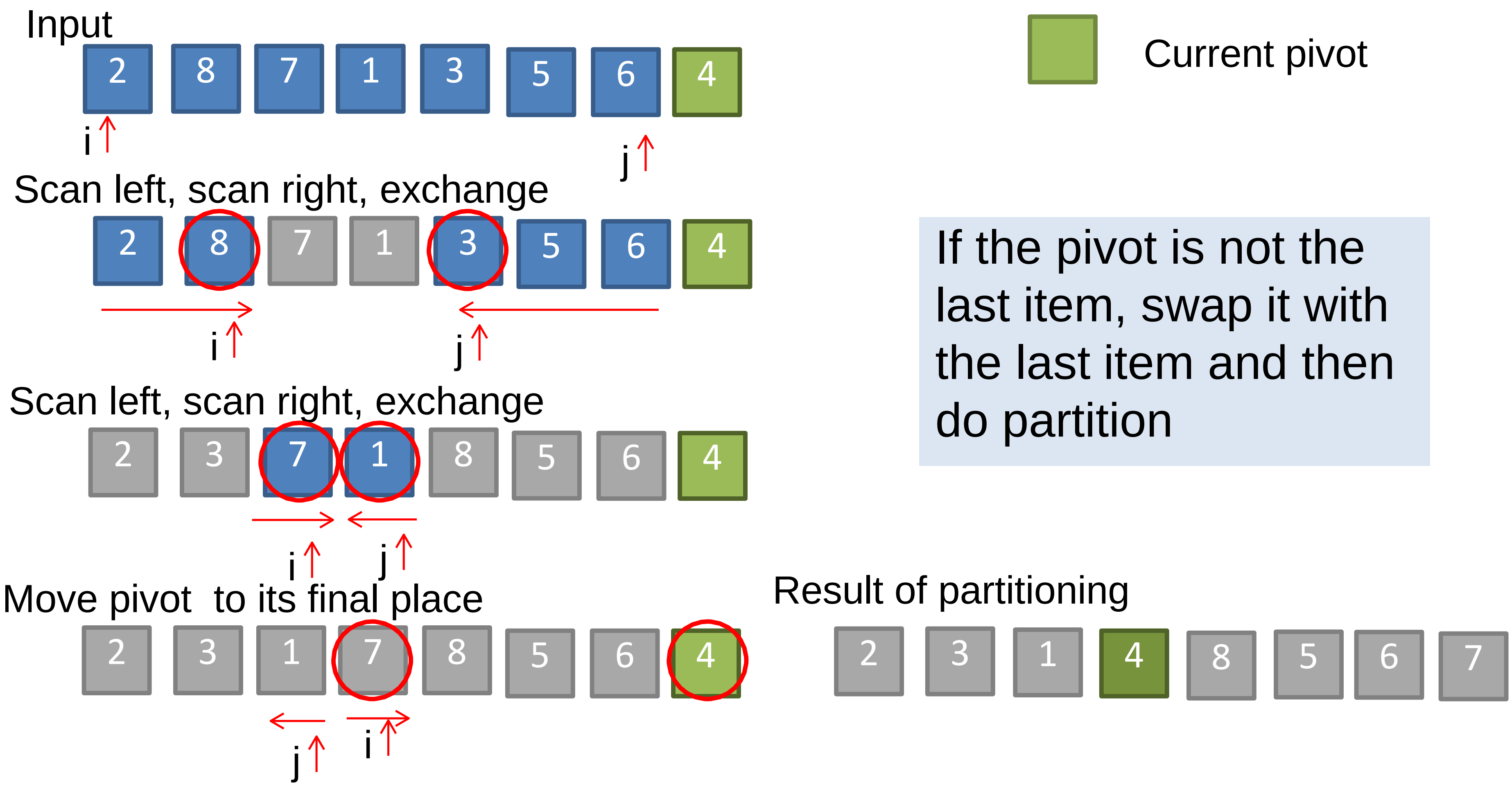


Quick sort

```
void QuickSort (int A[], int left, int right)
{
   int pivot_idx; // pivot marker
   if (left<right)
   {
      pivot_idx = partition(A, left, right);
      QuickSort(A, left, pivot_idx-1); // sort left segment
      QuickSort(A, pivot_idx+1, right);// sort right segment
   }
}
```

Quick sort Partition

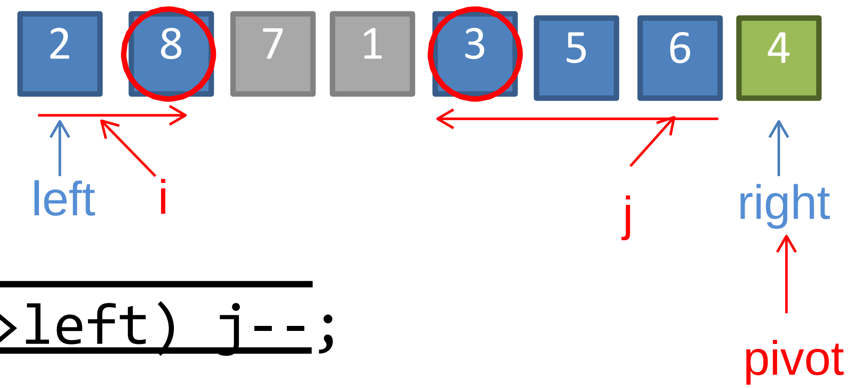
- Q: How to partition the data?
- A: Scan from the left, scan from the right, exchange



If the pivot is not the last item, swap it with the last item and then do partition

Quick sort partition

```
int partition(int A[], int left, int right)
{
   int pivot_idx = right; int X = A[pivot_idx];

   int i=left; int j=right-1;

   do {
         while(A[i] < X) i++;
         while(A[j] >= X && j>left) j--;

         if (i < j) exch(A, i, j);

   }while(i < j);

   exch(A, pivot_idx, i);
   return i;
}
```



Move pivot to its final place

Quick sort

```
void QuickSort (int A[], int left, int right)
{
   int pivot_idx;  // marker for boundary between segments
   if (left<right)
   {
      pivot_idx = partition(A, left, right);
      QuickSort(A, left, pivot_idx-1); // sort left segment
      QuickSort(A, pivot_idx+1, right);// sort right segment
   }
}
```

Quick sort Efficiency analysis

Worst case

Counting #comparisons gives:

$$W(N) = (N\text{-}1) + (N\text{-}2) + \ldots + 2 + 1$$

$$= 1 + 2 + 3 + \ldots. + (N-1)$$

$$= \tfrac{1}{2}N(N-1)$$

$$= \tfrac{1}{2}N^2 - \tfrac{1}{2}N$$

So, quick-sort is complexity class $O(N^2)$

Average case

For large N, average performance depends on $N\log_2 N$

Quick sort mini summary

- Insertion sort and Quick sort belong to the same *complexity class* $O(N^2)$, based on the worst case

- However, on average, Quick sort depends on $N\log_2 N$ but Insertion sort on N^2

- So, Quick sort is fast in general

- Insertion sort is faster if the problem size is small (because insertion sort doesn't have the extra overhead from recursive function calls)

MERGE SORT

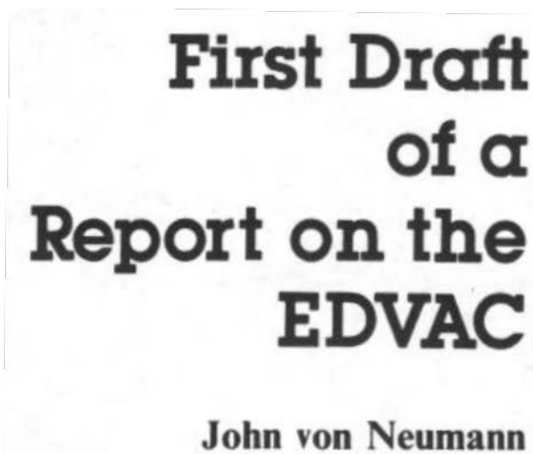
Merge sort

a *divide-and-conquer* algorithm

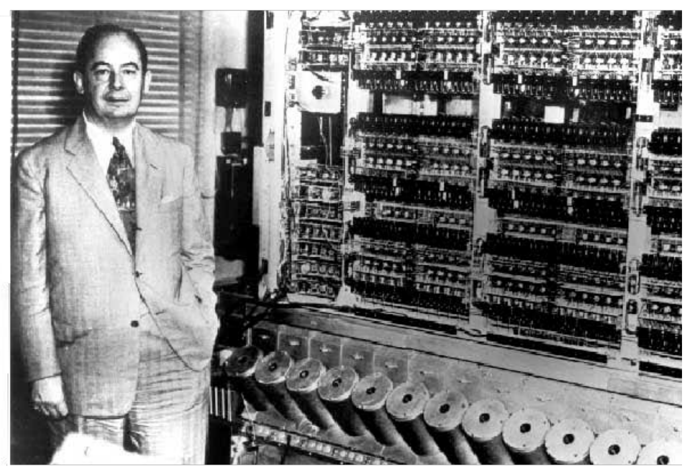
Method

split the list into equal halves
recursively sort each half
recombine by *merging* the halves

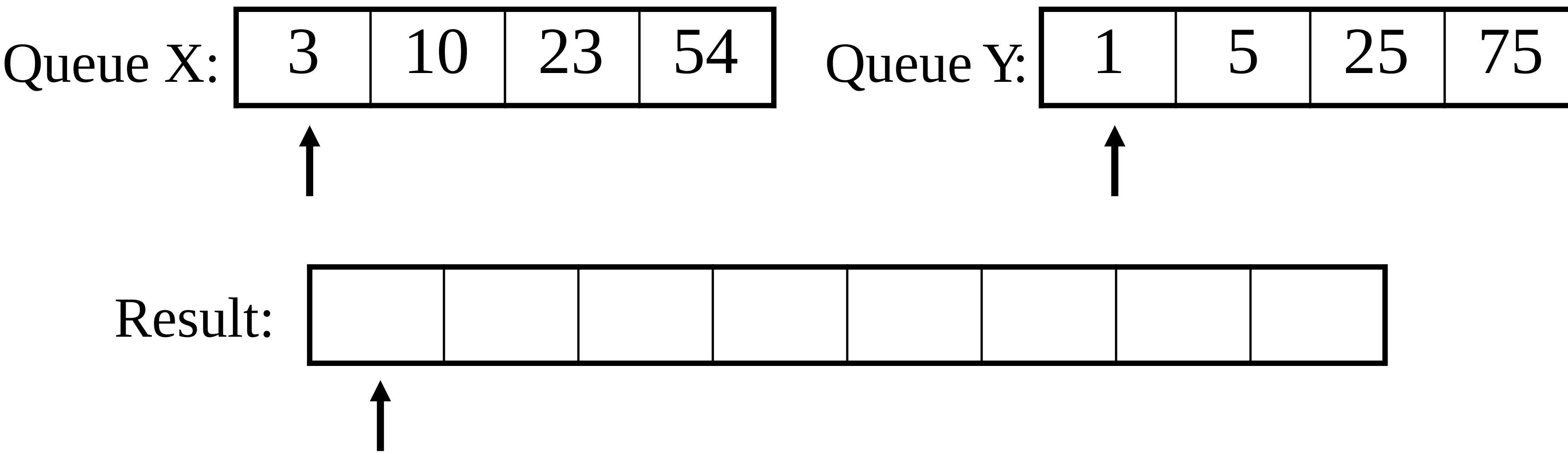
Merging – combines two sorted lists (fast, linear time)


First Draft
of a
Report on the
EDVAC

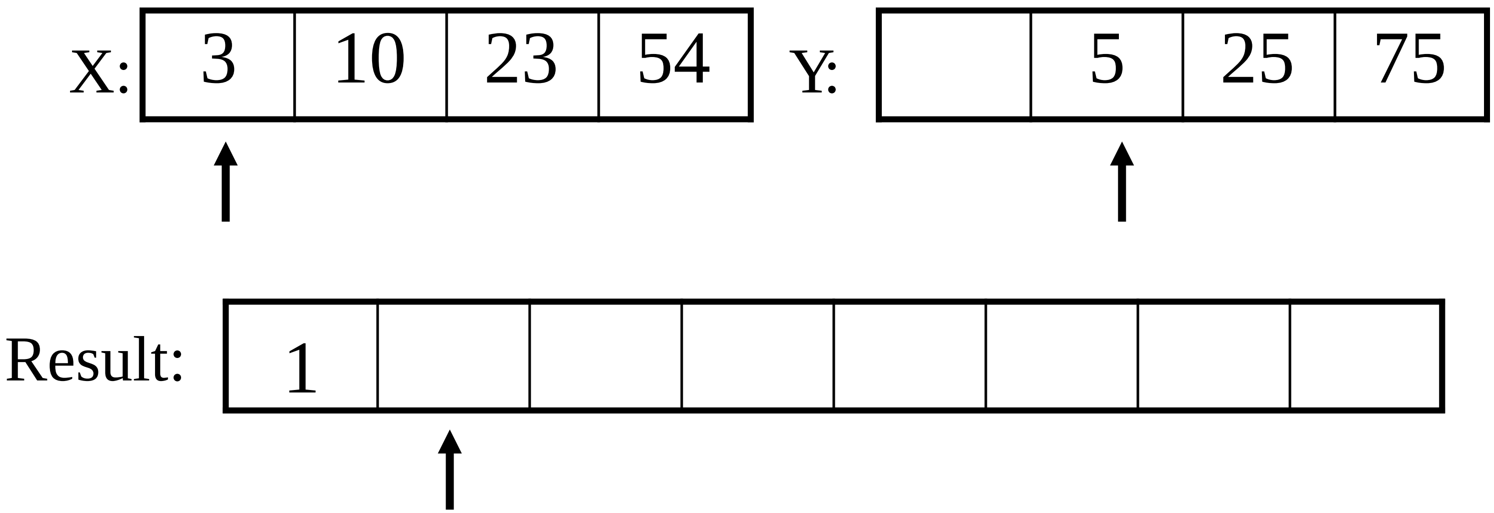
John von Neumann



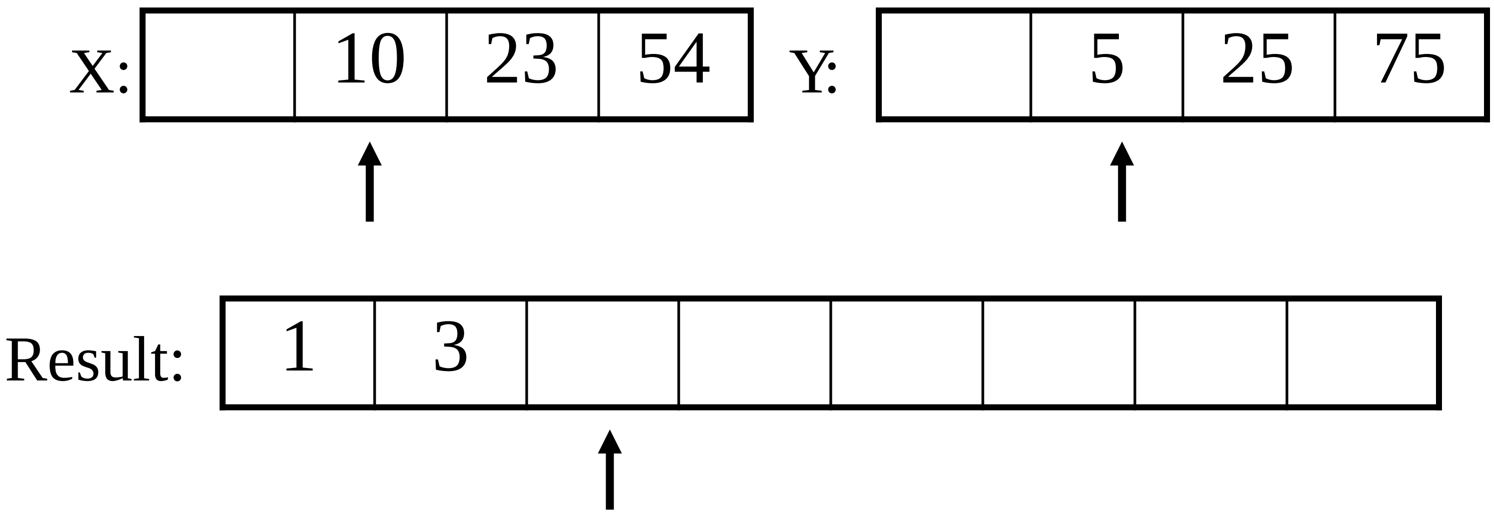
Merging Example (1)



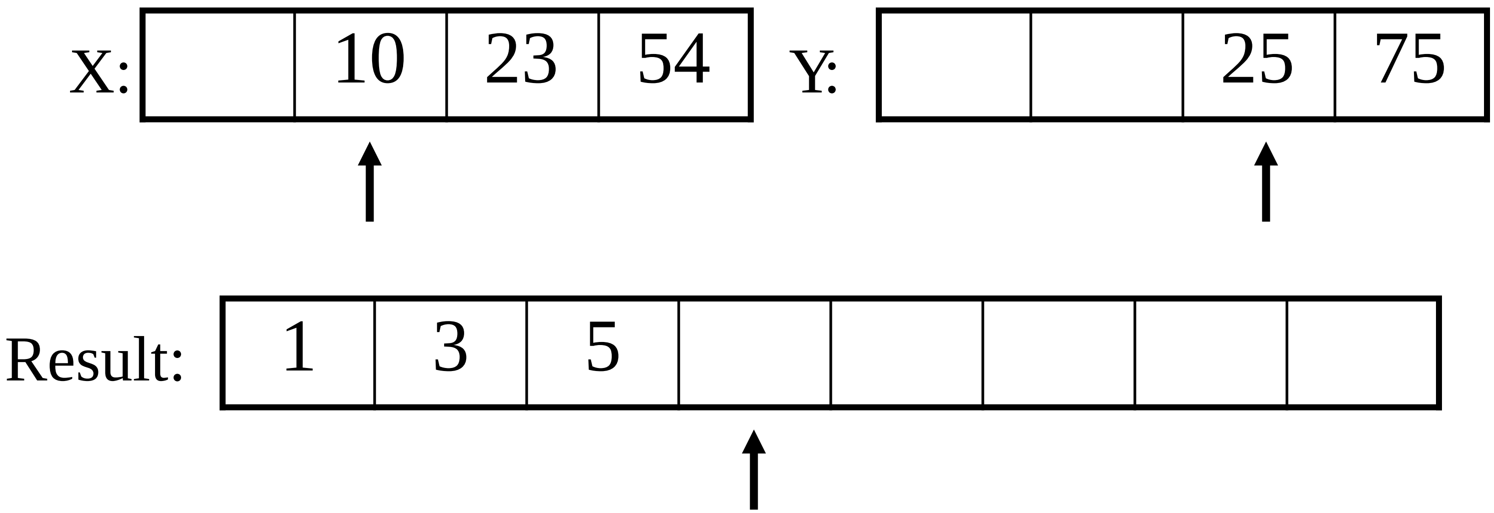
Merging Example (2)



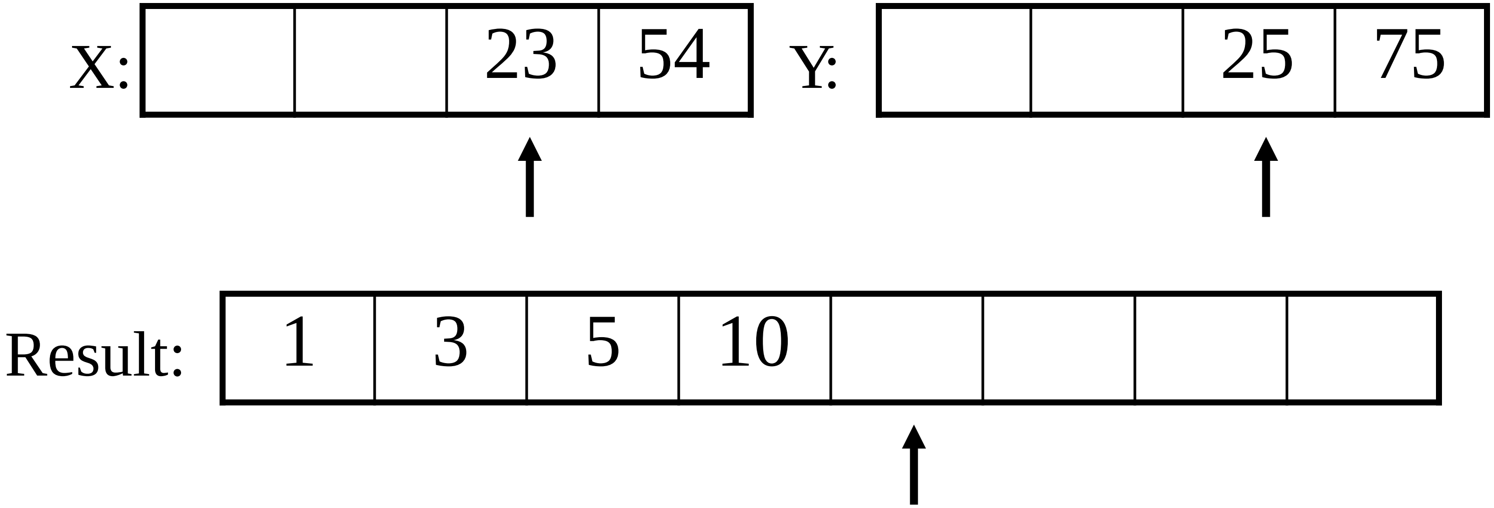
Merging (cont)



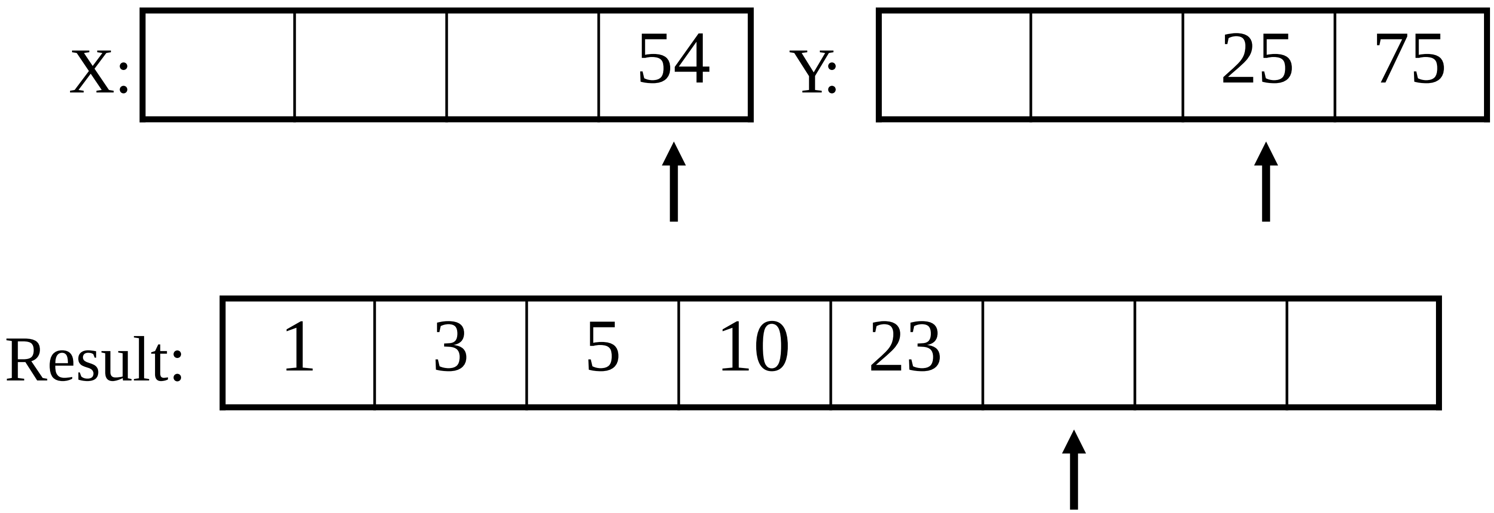
Merging Example (3)



Merging Example (4)



Merging Example (5)



Merging Example (6)

X:

			54

Y:

			75

Result:

1	3	5	10	23	25		

Merging Example (7)

X:

Y:

			75

Result:

1	3	5	10	23	25	54	

Merging Example (8)

X:

Y:

Result:

1	3	5	10	23	25	54	75

Merge Sort Algorithm

Given a list L with a length k:

- If k == 1 → the list is sorted
- Else:
  - Merge Sort the left side (1 to k/2)
  - Merge Sort the right side (k/2+1 to k)
  - Merge the right side with the left side

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

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

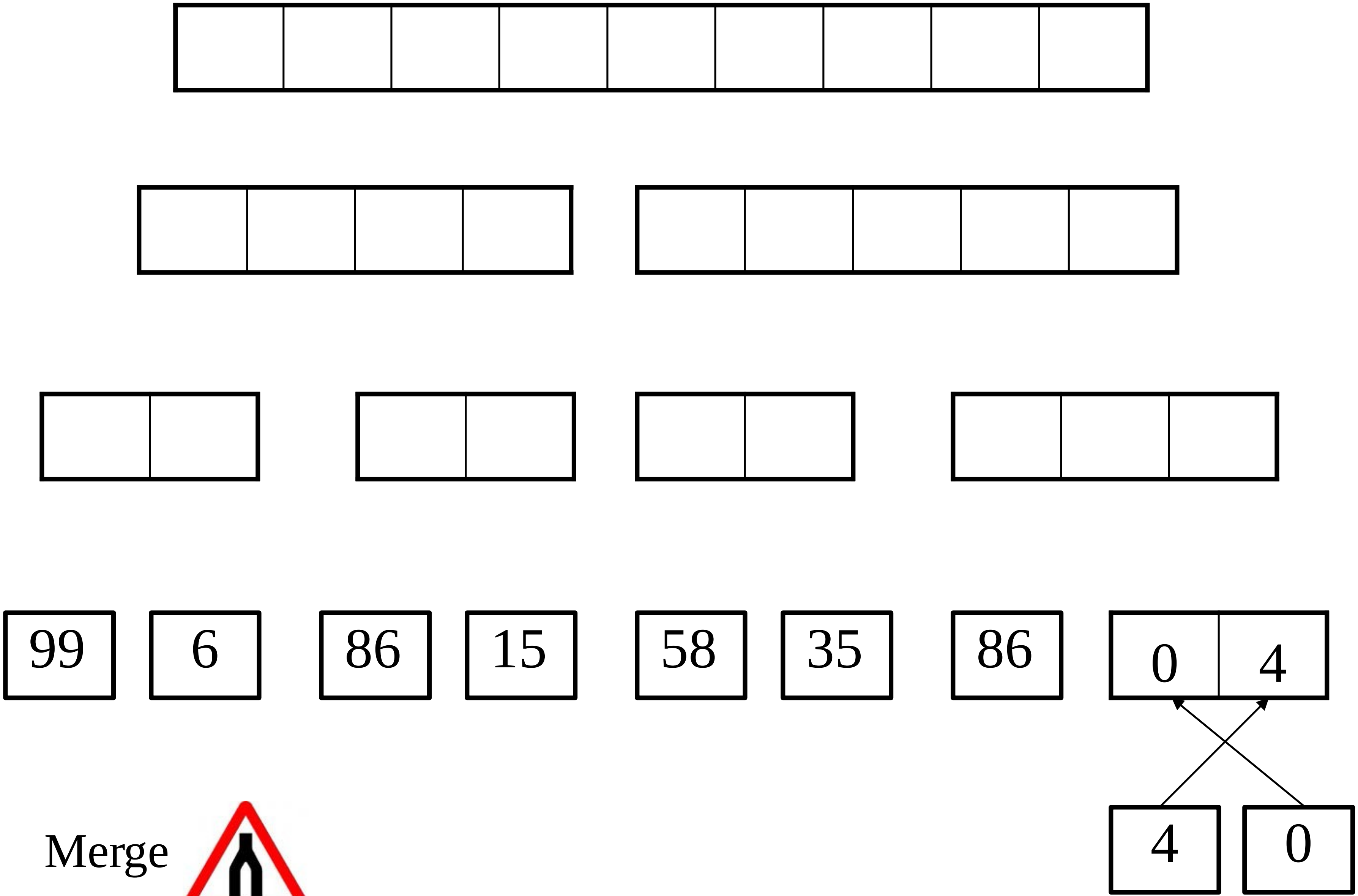
86

4	0

4

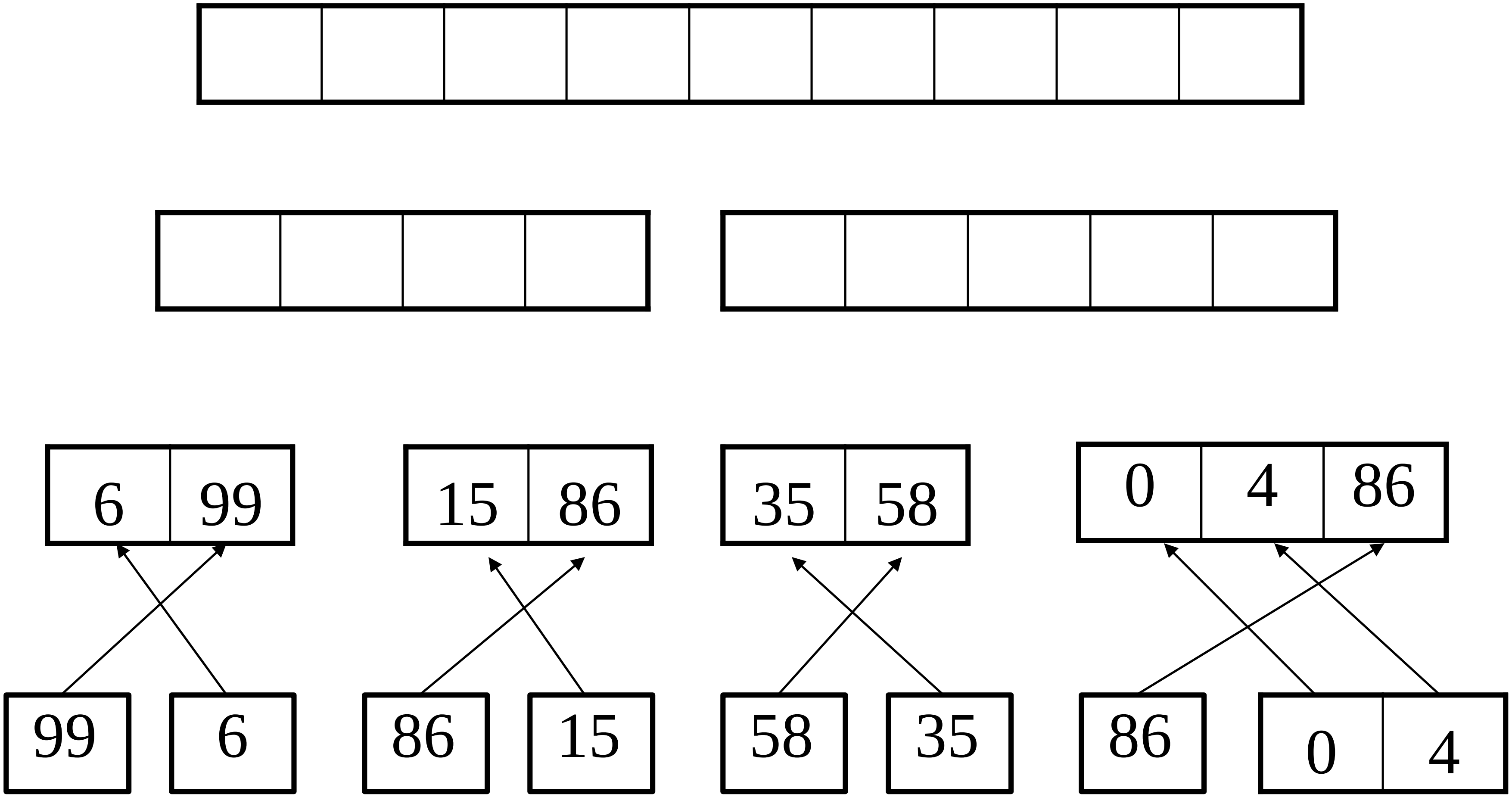
0

Merge Sort Example



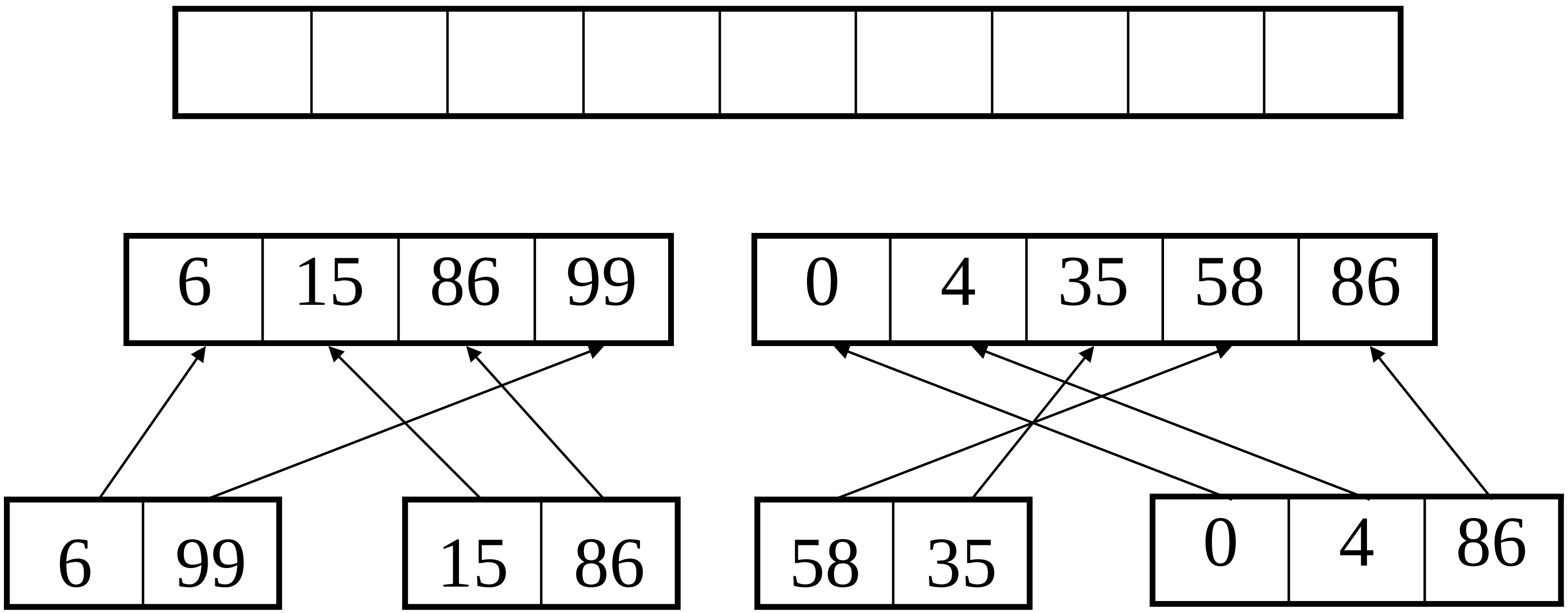
Merge

Merge Sort Example



Merge

Merge Sort Example



Merge

Merge Sort Example

0	4	6	15	35	58	86	86	99

6	15	86	99

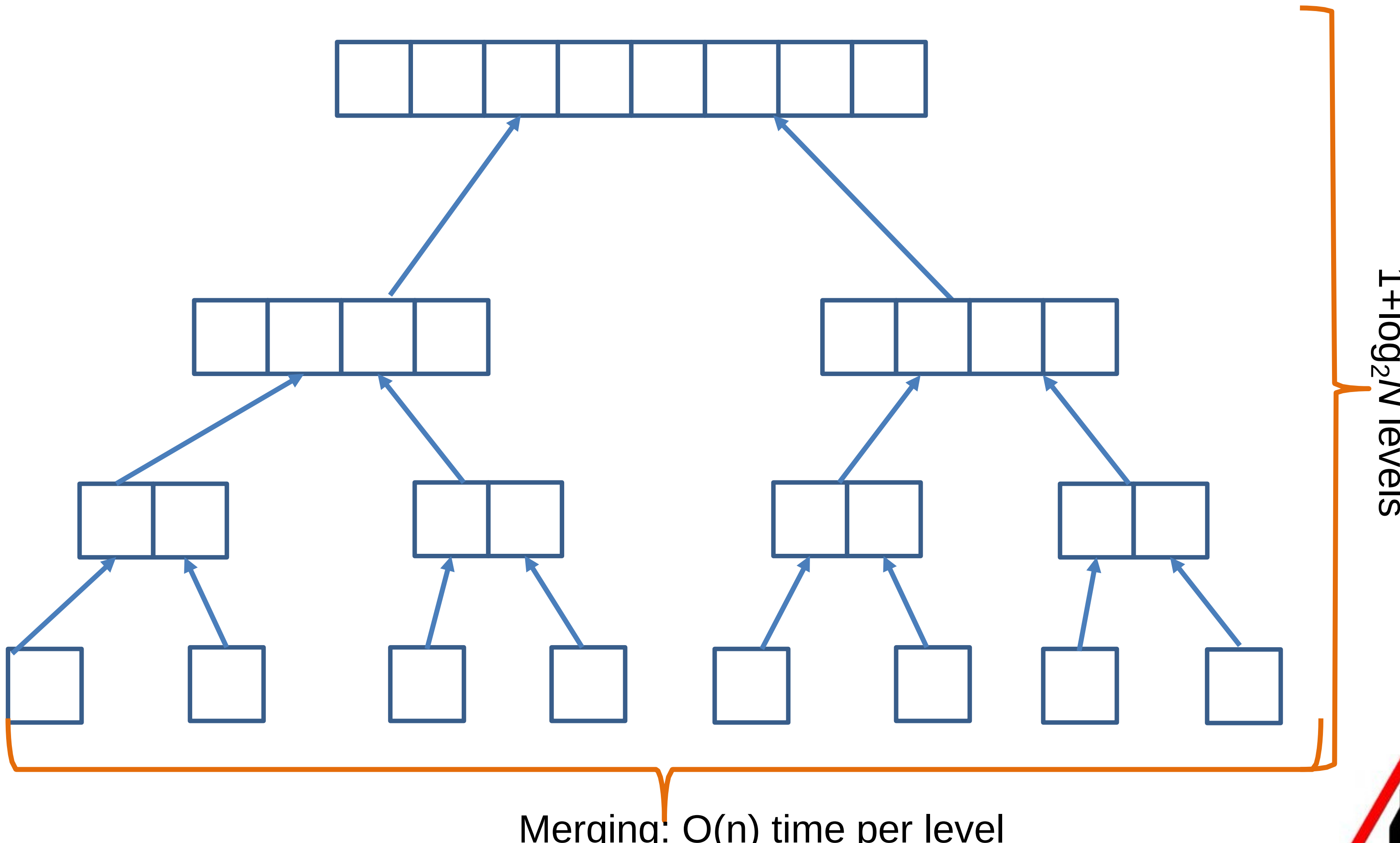
0	4	35	58	86

Merge

Merge Sort Efficiency

Approximate estimation: (N) * (1+log$_2$*N) ~ O(N*log$_2$*N) time*

Merge sort is natural for linked list, but it can't be done in place for arrays



Comparisons of sorting algorithms

Running time estimates:

- PC executes 10^8 compares/second.
- Supercomputer executes 10^{12} compares/second.

	insertion sort (N^2)			mergesort (N log N)			quicksort (N log N)		
computer	thousand	million	billion	thousand	million	billion	thousand	million	billion
PC	instant	2.8 hours	317 years	instant	1 second	18 min	instant	0.6 sec	12 min
super	instant	1 second	1 week	instant	instant	instant	instant	instant	instant

Lesson 1. Good algorithms are better than supercomputers.

Lesson 2. Great algorithms are better than good ones.

Source: http://algs4.cs.princeton.edu/

Sorting Applications

- Sort a list of names.
- Organize an MP3 library.
- Display Google searching results.

obvious applications

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

problems become easy once items are in sorted order

- Data compression.
- Computational biology
- Load balancing on a parallel computer.

Non-obvious applications

Summary of Unit 2

Selection sort $O(N^2)$

Insertion sort
Worse case $O(N^2)$
Average case $O(N^2)$

Quick sort
Worse case $O(N^2)$
Average case $O(N\log_2 N)$

Merge sort
Worse case $O(N\log_2 N)$
Average case $O(N\log_2 N)$

- Quicksort has 39% more comparisons than merge sort on the average case
- Faster than merge sort in practice because of less data movement

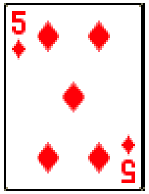
Homework

- Implement in C/Java, a quick sort algorithm to sort an array of integers.
- Implement merge sort to sort a list of words.

SCC120 SaS unit map

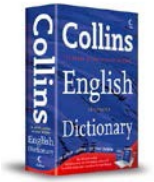
Unit 1 – Searching & Hashing





Unit 2 - Sorting techniques





Unit 3 - Sorting with trees



Resources

- Visualisation of sorting algorithms
  - http://sorting.at/
- Sorting Algorithm Animations by David Martin
  - http://www.sorting-algorithms.com/
- The Sound of Quicksort
  - https://www.youtube.com/watch?v=m1PS8IR6Td0

Engineering Quick Sort

- A beautiful bug report.
  - Allan Wilks and Rick Becker, 1991

> We found that qsort is unbearably slow on "organ-pipe" inputs like “123..nn..321":
>
> “A qsort run that should have taken a few minutes was chewing up hours of CPU time”

At the time, almost all `qsort()` implementations based on those in:

Version 7 Unix (1979): quadratic time to sort organ-pipe arrays.

BSD Unix (1983): quadratic time to sort random arrays of 0s and 1s.

Source: http://programmingisterrible.com/post/41512566174/engineering-quicksort

Engineering Quick Sort

- Bentley-McIlroy quicksort.
  - Cutoff to insertion sort for small subarrays.
  - Partitioning item (X): median of 3 or 9 items of the array.
  - Widely used in standard C, C++, Java 6 libraries…

- Dual Pivot Quicksort
  - Yaroslavskiy, September 2009
  - Is now used in **Java standard libraries**