# The Heapsort Algorithm

Sebastian Streich,
Enric Meinhardt

# Overview

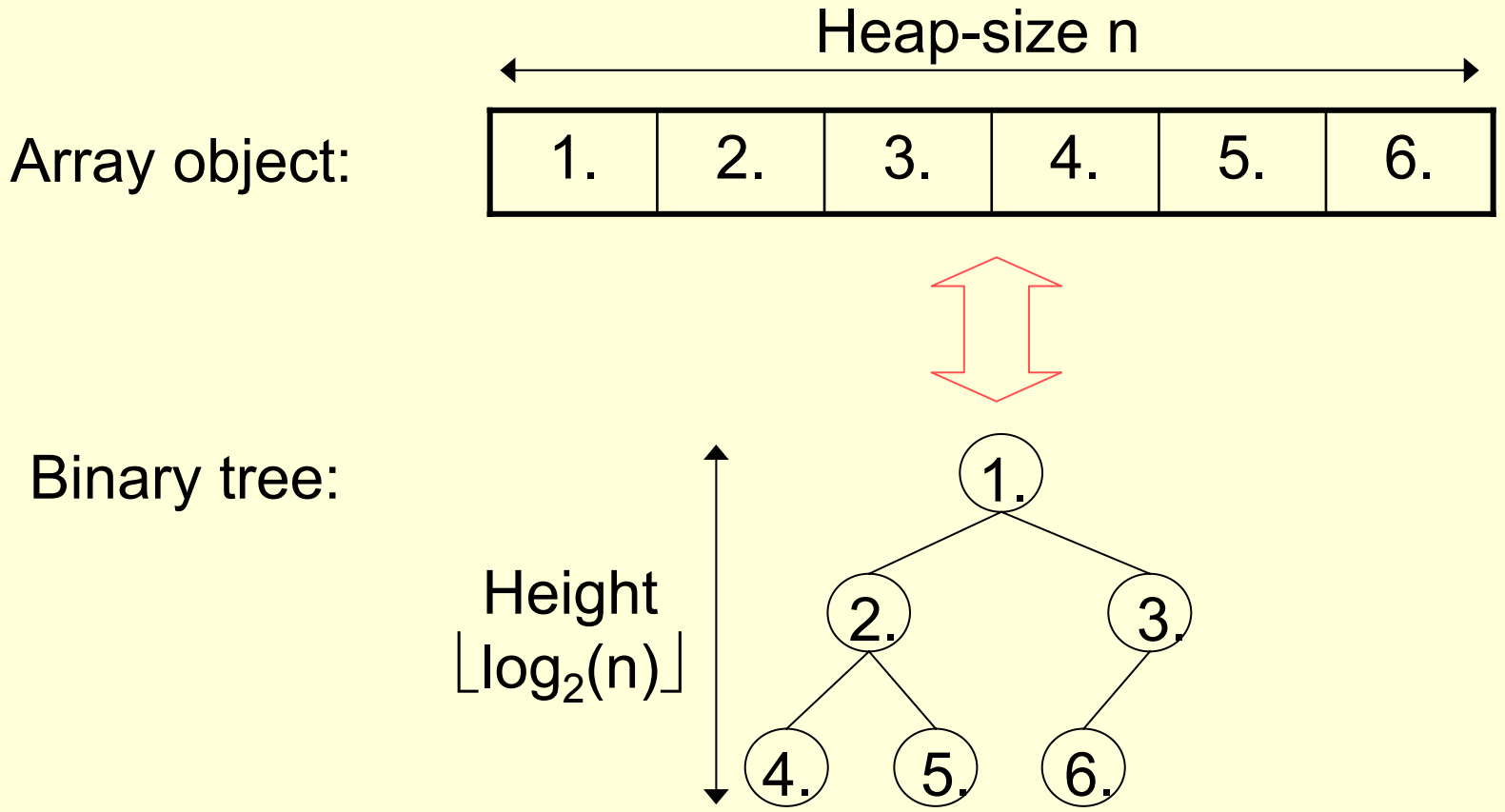
1. Introduction
2. What is a Heap?
3. Subroutines for Heaps
4. The Heapsort Function
5. Bottom-up Heapsort
6. Priority Queues

# Introduction

Important properties of sorting algorithms:

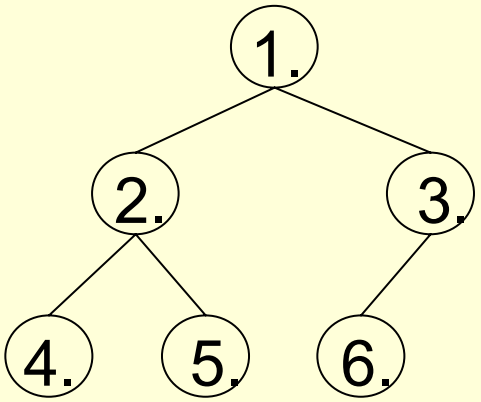
| | Processing time | Memory consumption |
|---|---|---|
| Merge sort | $O(n \log_2(n))$ | O(n) extra elements |
| Insertion sort | $O(n^2)$ | In place |
| Heapsort | $O(n \log_2(n))$ | In place |

# What is a Heap?



# What is a Heap?

Calculating the Indices:



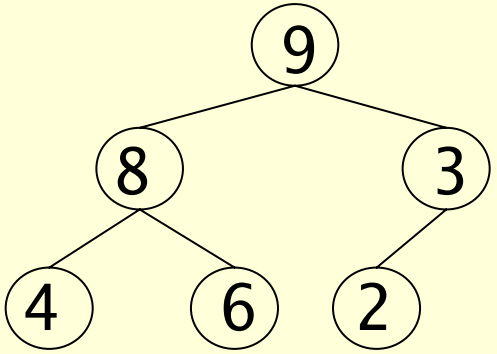

$$Parent(i) = \lfloor i/2 \rfloor$$

$$left\_Child(i) = 2 \cdot i$$

$$right\_Child(i) = 2 \cdot i+1$$

# What is a Heap?

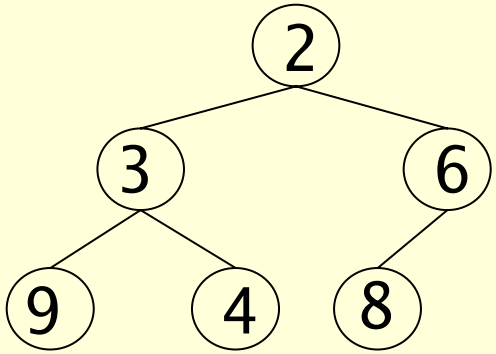
Heap property:




$A[Parent(i)] \geq A[i]$

→ maximum element in the root (max-heap)

or




$A[Parent(i)] \leq A[i]$

→ minimum element in the root (min-heap)

# Subroutines for Heaps

`Heapify(A,i)`

| | |
|---|---|
| Purpose: | make the subtree of `A` starting in node `i` fulfil the heap property |
| Pre-condition: | subtrees starting in `left_Child(i)` and `right_Child(i)` must be heaps already |

# Subroutines for Heaps

```
Heapify(A,i)
l := largest node of i and its children
if ( i ≠ l )
    exchange A[i] with A[l]
    Heapify(A, l)
```

A real implementation should not be recursive!
(overhead when passing the function's arguments)

# Subroutines for Heaps

`Heapify(A,i)`

Computational cost: $T(n) \leq T(2n/3) + \Theta(1)$

$\rightarrow$ $T(n) = O(\log_2(n))$
or $T(n) = O(h)$

with h being the height of node `i`

# Subroutines for Heaps

`Build_heap(A)`

Purpose: build a heap out of array `A`

Pre-condition: any array `A`

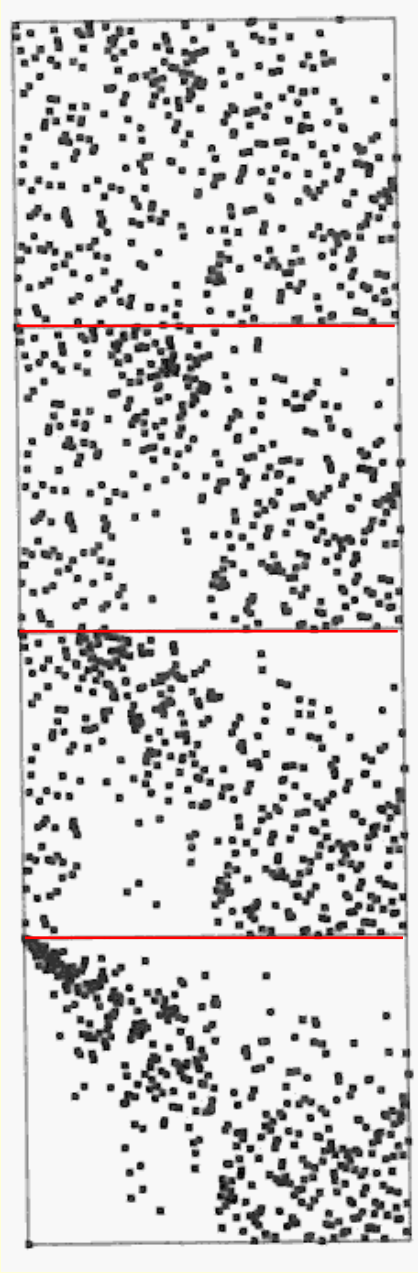
Idea: not many elements need to be exchanged

# Subroutines for Heaps

```
Build_heap(A)

for i = n/2 downto 1
     Heapify(A, i)
```

- leaves of the tree are elements with $i \geq n/2$
- leaves are already heapified subtrees
- Build_heap runs in time O(n)



Initial random array

.

.

.

n/2 calls of `Heapify`

.

.

.

Array with the heap property

Figure taken from R. Sedgewick, Algorithms in C, Third edition, 1998 Addison-Wesley

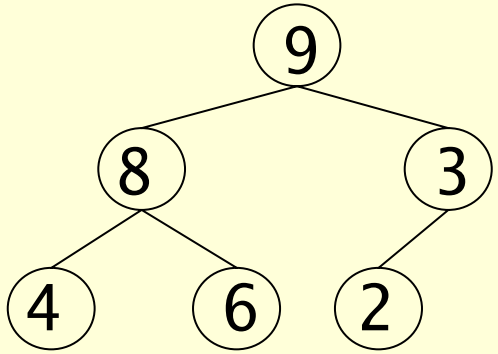
# The Heapsort Function

`Heapsort(A)`

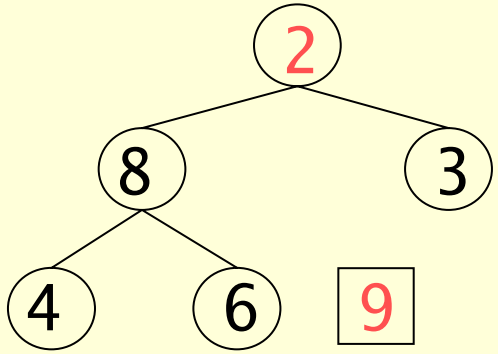
| | |
|---|---|
| Purpose: | sort array `A` (in place) |
| Pre-condition: | any array `A` |
| Idea: | make `A` a heap, then take out the root; repeat until the array is sorted |

# The Heapsort Function

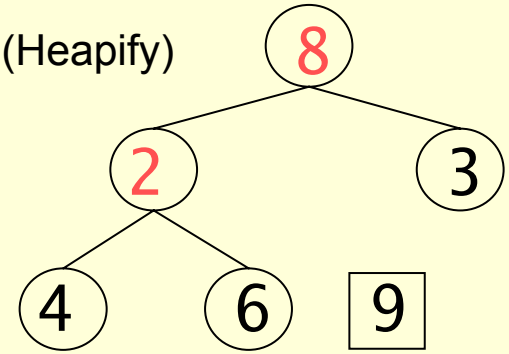
1.



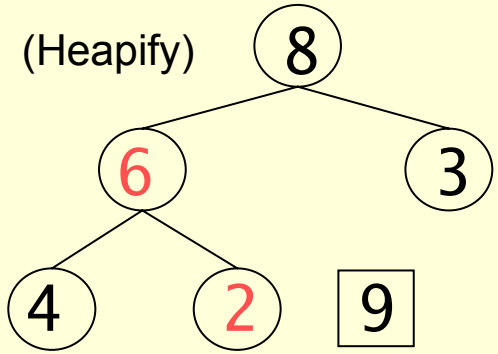

2.




3. (Heapify)




4. (Heapify)




…

# The Heapsort Function

```
Heapsort(A)
Build_heap(A)
for i = n downto 2
    exchange A[1] with A[i]
    A.size--
    Heapify(A, 1)
```

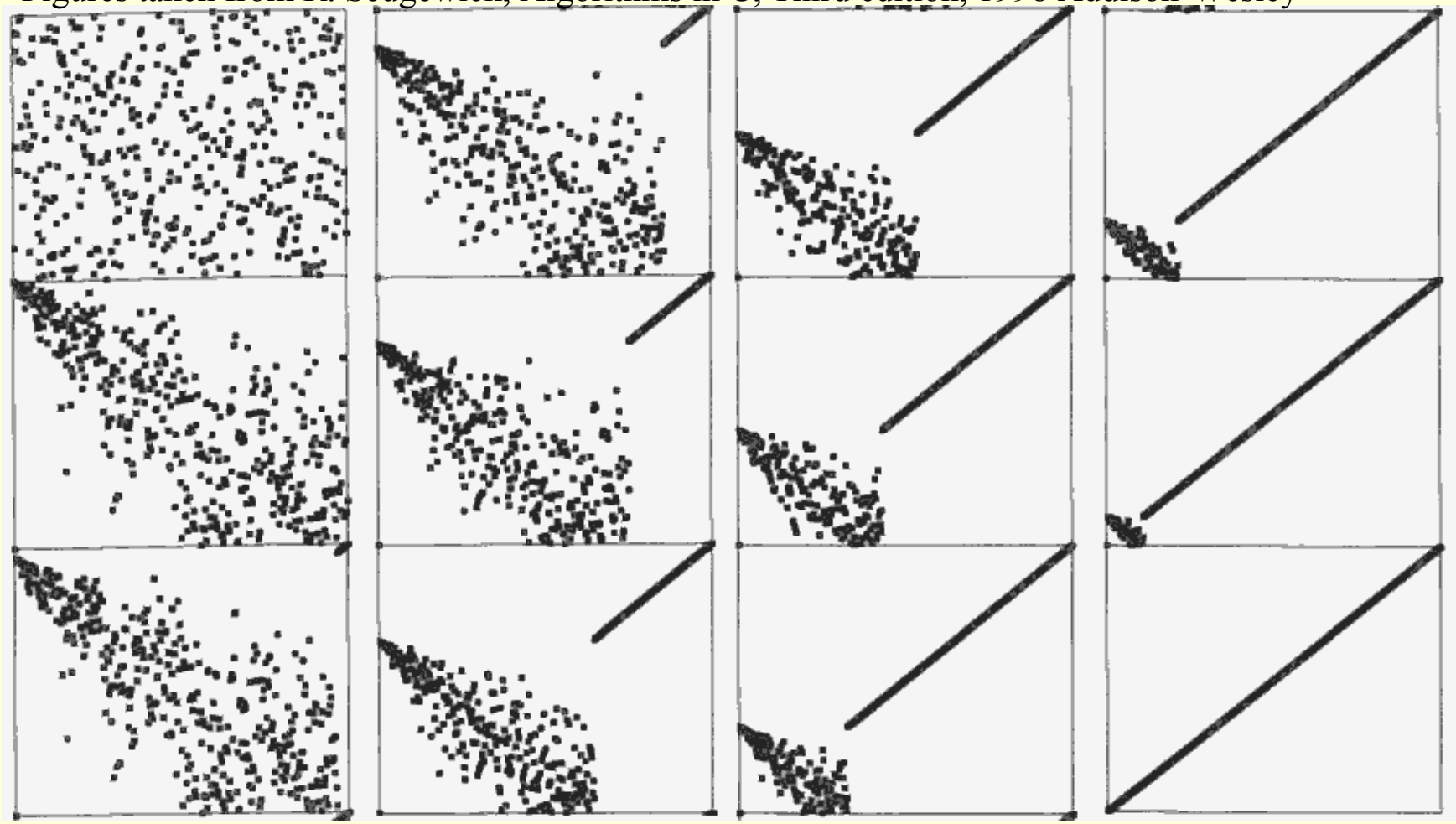
*root is swapped with last element*

→ *former root is outside* A

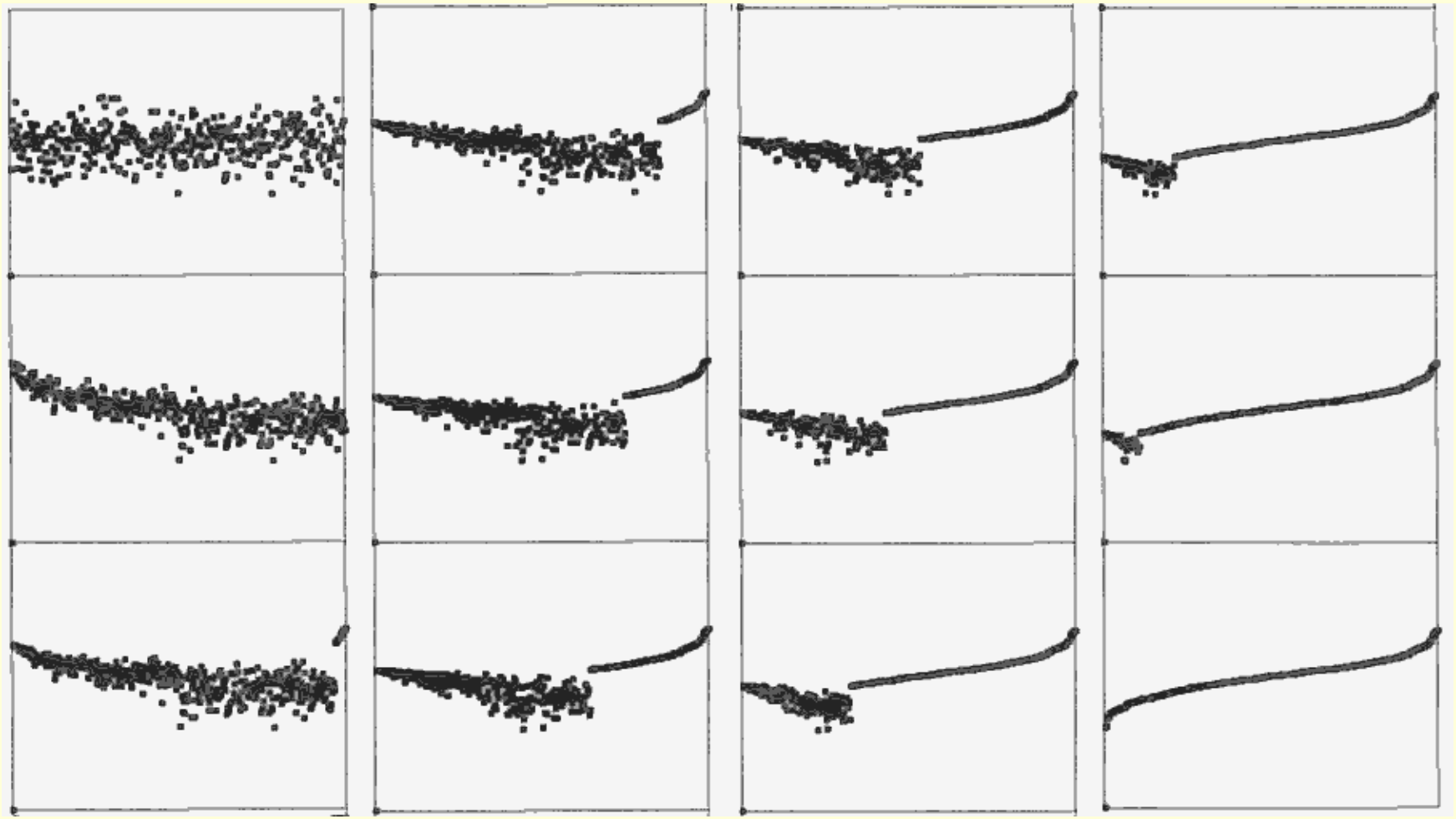
Heapsort runs in time $O(n \log_2(n))$

# The Heapsort Function

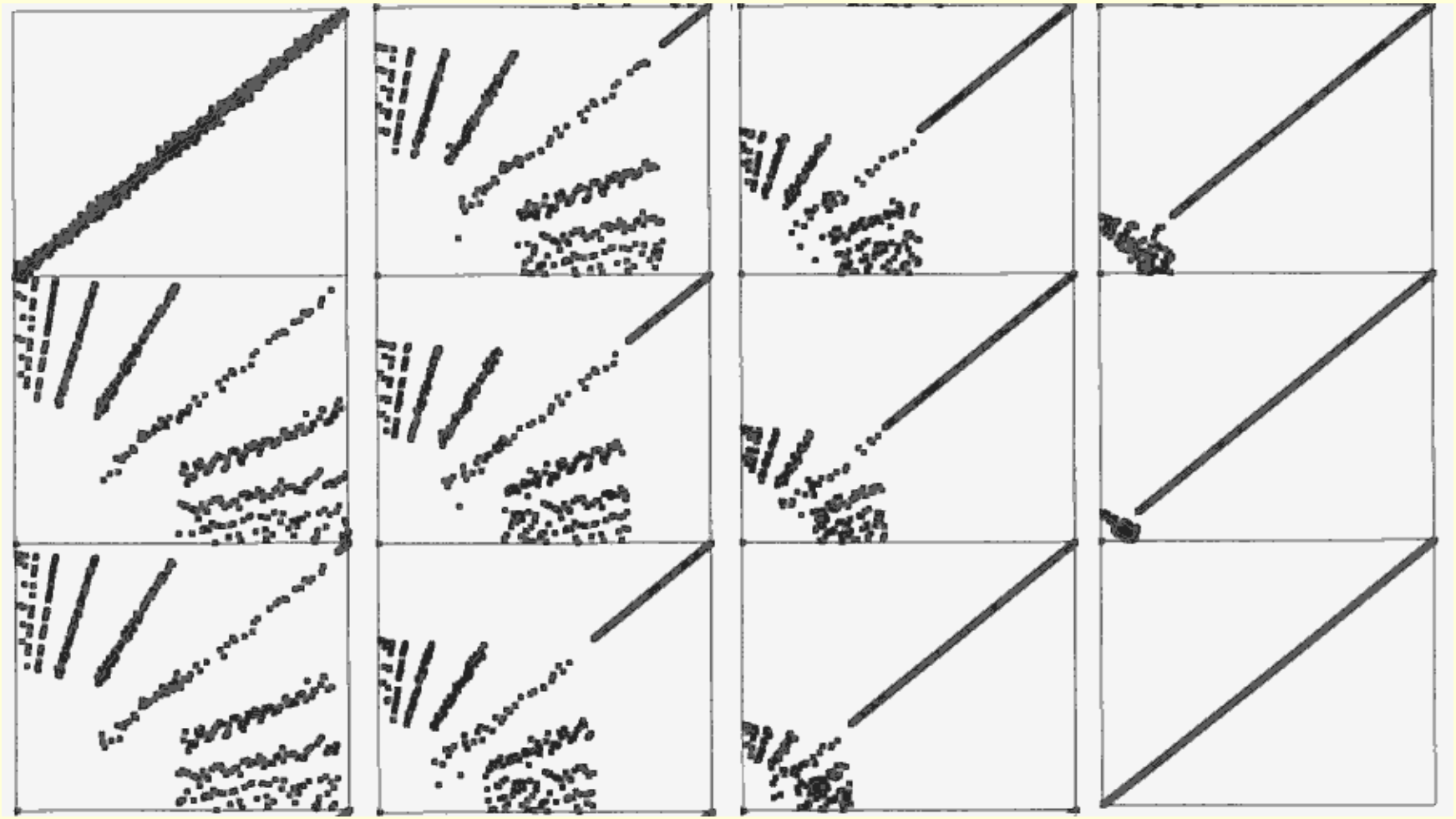
Figures taken from R. Sedgewick, Algorithms in C, Third edition, 1998 Addison-Wesley



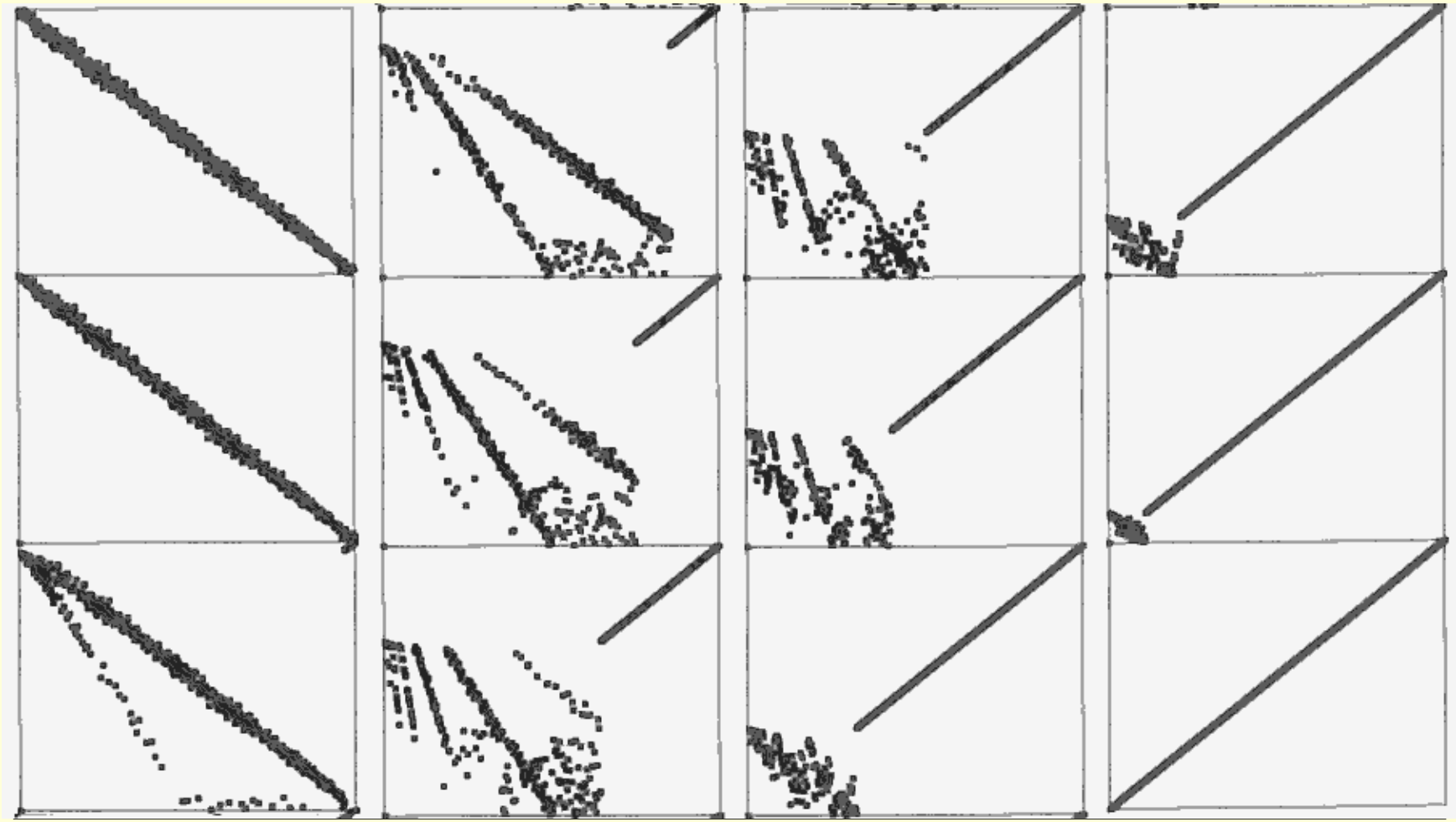
# The Heapsort Function



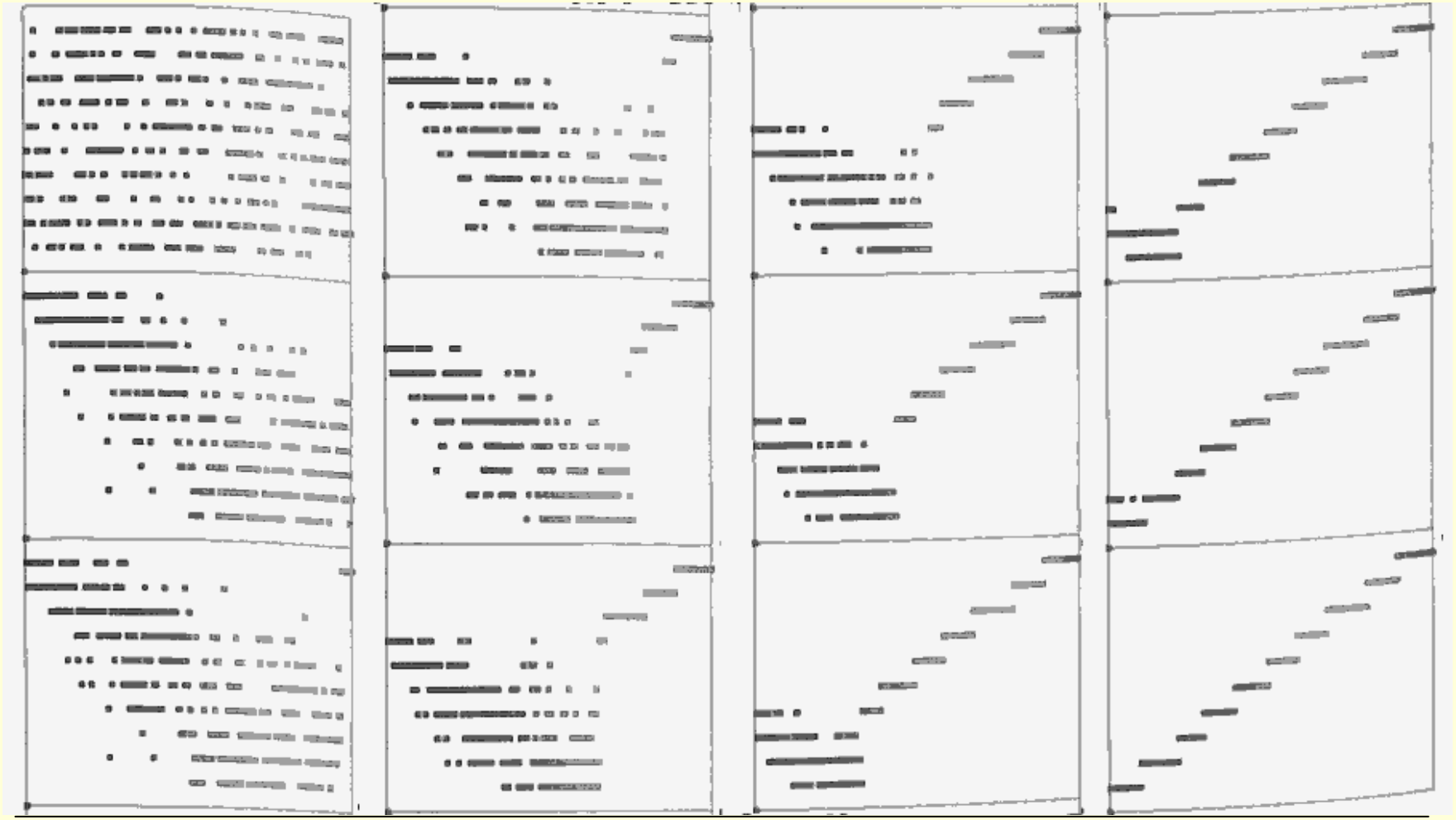
# The Heapsort Function



# The Heapsort Function



# The Heapsort Function



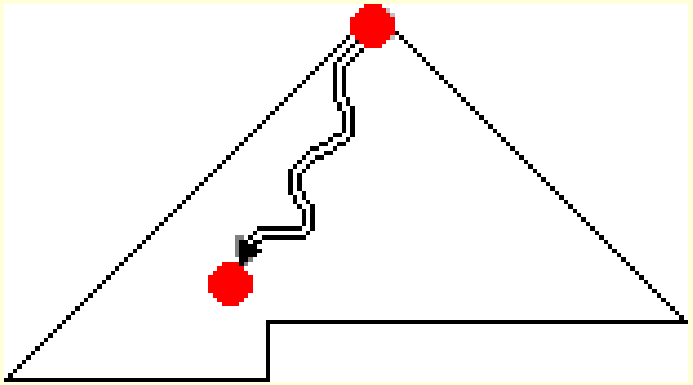
# Bottom-up Heapsort

Variant of heapsort with better performance (average)

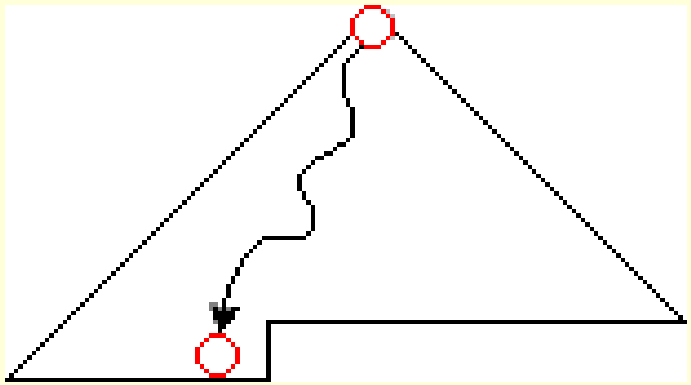
- last element of the heap is supposed to be very small
- pass it all the way down after swapping with the root
- then move it up to its proper position

# Bottom-up Heapsort

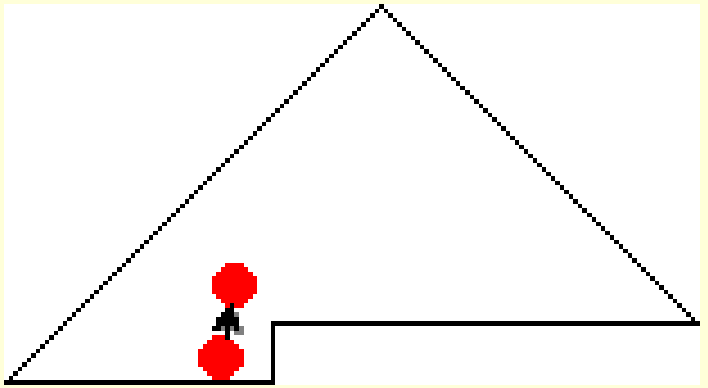
Instead of



do



+



Figures taken from http://www.iti.fh-flensburg.de/lang/algorithmen/sortieren/heap/heap.htm

# Priority Queues

A priority queue is an ADT with the following operations:

- Insert a new element into the queue
- Find the element with the largest key
- Delete the element with the largest key

Other common operations:

- Increase the key of an element

# Priority Queues

Heaps provide an efficient implementation of priority queues:

| | | |
|---|---|---|
| get the maximum | → | take the root |
| delete the maximum | → | move the last element to the root and heapify |
| insert a new element | → | put it at the end and raise it until it's in place |

Link to Build_heap applet:

http://www.iti.fh-flensburg.de/lang/algorithmen/sortieren/heap/heapen.htm