# Sorting Algorithms

## Lecture 03

Ritu Kundu

King's College London

Tue, Oct 10, 2017

# Outline

## Problem

**input:** A sequence of $n$ numbers $< a_1, a_2, \ldots, a_n >$.
**output:** A permutation (reordering) $< a'_1, a'_2, \ldots, a'_n >$ of the input sequence such that $a'_1 \leq a'_2 \leq \ldots \leq a'_n$.

## Example

$< 31, 41, 59, 26, 41, 58 > \Rightarrow$ SORTING $\Rightarrow < 26, 31, 41, 41, 58, 59 >$

## Applications

- Easy search
- Database operations
- Basis of many algos (string processing, partitioning, intersection)

# Factors

- Running-time (worst, best, average): Asymptotic time-complexity
- Additional space requirement: In-place ($\theta(1)$ memory) or not
- Initial conditions(input order, key distribution, size) of input data: already-sorted, nearly-sorted, reverse-sorted, few unique keys, small/large size
- Stability: $=>$ two elements which have the same value will retain their relative order after sorting. Example:

$$< 31, 41^*, 41^+, 26, 41^-, 58 > \Rightarrow \boxed{\text{SORTING}} \Rightarrow < 26, 31, 41^*, 41^+, 41^-, 58 >$$

# An ideal algorithm

- $O(nlogn)$ worst time comparisons ($\Omega(n)$ lower bound with certain assumption about the data. In general can not be less than this [a].
- In-place
- Stable
- Adaptive

[a] A more detailed proof is given in Donald E. Knuth, The Art of Computer Programming, Volume 3: Sorting and Searching, 2nd Ed., Addison Wesley, 1998, Â§5.3.1, p.180.)

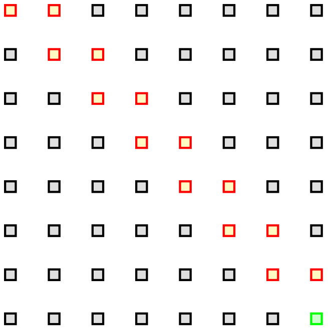
No algorithm is an absolute ideal.

# Types

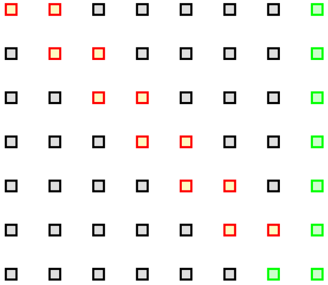
- External
- Internal
  - Bubble
  - Insertion
  - Selection
  - Merge

# Outline

## Pass 1

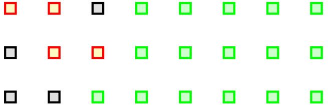


## Pass 2

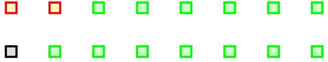


. . .

## Pass 6



## Pass 7



## Input

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 5 | 1 | 2 | 0 | 4 | 6 | 0 | 9 |

## Input

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 5 | 1 | 2 | 0 | 4 | 6 | 0 | 9 |

## Pass 1

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 5 | 1 | 2 | 0 | 4 | 6 | 0 | 9 |

## Input

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 5 | 1 | 2 | 0 | 4 | 6 | 0 | 9 |

## Pass 1

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | 5 | 2 | 0 | 4 | 6 | 0 | 9 |

## Input

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 5 | 1 | 2 | 0 | 4 | 6 | 0 | 9 |

## Pass 1

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | 2 | 5 | 0 | 4 | 6 | 0 | 9 |

## Input

5 1 2 0 4 6 0 9

## Pass 1

1 2 0 **5** **4** 6 0 9

## Input

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 5 | 1 | 2 | 0 | 4 | 6 | 0 | 9 |

## Pass 1

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | 2 | 0 | 4 | 5 | 6 | 0 | 9 |

## Input

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 5 | 1 | 2 | 0 | 4 | 6 | 0 | 9 |

## Pass 1

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | 2 | 0 | 4 | 5 | 6 | 0 | 9 |

### Input

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 5 | 1 | 2 | 0 | 4 | 6 | 0 | 9 |

### Pass 1

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | 2 | 0 | 4 | 5 | 0 | 6 | 9 |

### Input

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 5 | 1 | 2 | 0 | 4 | 6 | 0 | 9 |

### Pass 1

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | 2 | 0 | 4 | 5 | 0 | 6 | 9 |

## Input

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 5 | 1 | 2 | 0 | 4 | 6 | 0 | 9 |

## Pass 1

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | 2 | 0 | 4 | 5 | 0 | 6 | 9 |

## Pass 2

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | 2 | 0 | 4 | 5 | 0 | 6 | 9 |

## Input

5 1 2 0 4 6 0 9

## Pass 1

1 2 0 4 5 0 6 9

## Pass 2

1 2 0 4 5 0 6 9

## Input

5 1 2 0 4 6 0 9

## Pass 1

1 2 0 4 5 0 6 9

## Pass 2

1 0 2 4 5 0 6 9

## Input

5 1 2 0 4 6 0 9

## Pass 1

1 2 0 4 5 0 6 9

## Pass 2

1 0 2 4 5 0 6 9

## Input

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 5 | 1 | 2 | 0 | 4 | 6 | 0 | 9 |

## Pass 1

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | 2 | 0 | 4 | 5 | 0 | 6 | 9 |

## Pass 2

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | 0 | 2 | 4 | 5 | 0 | 6 | 9 |

### Input

5 1 2 0 4 6 0 9

### Pass 1

1 2 0 4 5 0 6 9

### Pass 2

1 0 2 4 0 5 6 9

## Input

5 1 2 0 4 6 0 9

## Pass 1

1 2 0 4 5 0 6 **9**

## Pass 2

1 0 2 4 0 5 **6** **9**

## Input

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 5 | 1 | 2 | 0 | 4 | 6 | 0 | 9 |

## Pass 1

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | 2 | 0 | 4 | 5 | 0 | 6 | 9 |

## Pass 2

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | 0 | 2 | 4 | 0 | 5 | 6 | 9 |

## Pass 3

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | 0 | 2 | 4 | 0 | 5 | 6 | 9 |

## Input

5 1 2 0 4 6 0 9

## Pass 1

1 2 0 4 5 0 6 9

## Pass 2

1 0 2 4 0 5 6 9

## Pass 3

0 1 2 4 0 5 6 9

## Input

5 1 2 0 4 6 0 9

## Pass 1

1 2 0 4 5 0 6 9

## Pass 2

1 0 2 4 0 5 6 9

## Pass 3

0 1 2 4 0 5 6 9

## Input

5 1 2 0 4 6 0 9

## Pass 1

1 2 0 4 5 0 6 9

## Pass 2

1 0 2 4 0 5 6 9

## Pass 3

0 1 2 4 0 5 6 9

## Input

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 5 | 1 | 2 | 0 | 4 | 6 | 0 | 9 |

## Pass 1

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | 2 | 0 | 4 | 5 | 0 | 6 | 9 |

## Pass 2

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | 0 | 2 | 4 | 0 | 5 | 6 | 9 |

## Pass 3

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 0 | 4 | 5 | 6 | 9 |

## Input

5 1 2 0 4 6 0 9

## Pass 1

1 2 0 4 5 0 6 **9**

## Pass 2

1 0 2 4 0 5 **6** **9**

## Pass 3

0 1 2 0 4 **5** **6** **9**

## Input

5 1 2 0 4 6 0 9

## Pass 1

1 2 0 4 5 0 6 9

## Pass 2

1 0 2 4 0 5 6 9

## Pass 3

0 1 2 0 4 5 6 9

## Pass 4

0 1 2 0 4 5 6 9

## Input

5 1 2 0 4 6 0 9

## Pass 1

1 2 0 4 5 0 6 9

## Pass 2

1 0 2 4 0 5 6 9

## Pass 3

0 1 2 0 4 5 6 9

## Pass 4

0 1 2 0 4 5 6 9

## Input

5 1 2 0 4 6 0 9

## Pass 1

1 2 0 4 5 0 6 9

## Pass 2

1 0 2 4 0 5 6 9

## Pass 3

0 1 2 0 4 5 6 9

## Pass 4

0 1 2 0 4 5 6 9

## Input

5 1 2 0 4 6 0 9

## Pass 1

1 2 0 4 5 0 6 9

## Pass 2

1 0 2 4 0 5 6 9

## Pass 3

0 1 2 0 4 5 6 9

## Pass 4

0 1 0 2 4 5 6 9

## Input

5 1 2 0 4 6 0 9

## Pass 1

1 2 0 4 5 0 6 **9**

## Pass 2

1 0 2 4 0 5 **6** **9**

## Pass 3

0 1 2 0 4 **5** **6** **9**

## Pass 4

0 1 0 2 **4** **5** **6** **9**

## Input

5 1 2 0 4 6 0 9

## Pass 1

1 2 0 4 5 0 6 9

## Pass 2

1 0 2 4 0 5 6 9

## Pass 3

0 1 2 0 4 5 6 9

## Pass 4

0 1 0 2 4 5 6 9

## Pass 5

0 1 0 2 4 5 6 9

## Input

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 5 | 1 | 2 | 0 | 4 | 6 | 0 | 9 |

## Pass 1

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | 2 | 0 | 4 | 5 | 0 | 6 | 9 |

## Pass 2

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | 0 | 2 | 4 | 0 | 5 | 6 | 9 |

## Pass 3

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 0 | 4 | 5 | 6 | 9 |

## Pass 4

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 2 | 4 | 5 | 6 | 9 |

## Pass 5

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 2 | 4 | 5 | 6 | 9 |

## Input

5 1 2 0 4 6 0 9

## Pass 1

1 2 0 4 5 0 6 9

## Pass 2

1 0 2 4 0 5 6 9

## Pass 3

0 1 2 0 4 5 6 9

## Pass 4

0 1 0 2 4 5 6 9

## Pass 5

0 0 1 2 4 5 6 9

## Input

5 1 2 0 4 6 0 9

## Pass 1

1 2 0 4 5 0 6 **9**

## Pass 2

1 0 2 4 0 5 **6** **9**

## Pass 3

0 1 2 0 4 **5** **6** **9**

## Pass 4

0 1 0 2 **4** **5** **6** **9**

## Pass 5

0 0 1 **2** **4** **5** **6** **9**

## Input

5 1 2 0 4 6 0 9

## Pass 1

1 2 0 4 5 0 6 9

## Pass 2

1 0 2 4 0 5 6 9

## Pass 3

0 1 2 0 4 5 6 9

## Pass 4

0 1 0 2 4 5 6 9

## Pass 5

0 0 1 2 4 5 6 9

## Pass 6

0 0 1 2 4 5 6 9

### Input

5 1 2 0 4 6 0 9

### Pass 1

1 2 0 4 5 0 6 9

### Pass 2

1 0 2 4 0 5 6 9

### Pass 3

0 1 2 0 4 5 6 9

### Pass 4

0 1 0 2 4 5 6 9

### Pass 5

0 0 1 2 4 5 6 9

### Pass 6

0 0 1 2 4 5 6 9

## Input

5 1 2 0 4 6 0 9

## Pass 1

1 2 0 4 5 0 6 **9**

## Pass 2

1 0 2 4 0 5 **6** **9**

## Pass 3

0 1 2 0 4 **5** **6** **9**

## Pass 4

0 1 0 2 **4** **5** **6** **9**

## Pass 5

0 0 1 **2** **4** **5** **6** **9**

## Pass 6

0 0 **1** **2** **4** **5** **6** **9**

## Input

5 1 2 0 4 6 0 9

## Pass 1

1 2 0 4 5 0 6 9

## Pass 2

1 0 2 4 0 5 6 9

## Pass 3

0 1 2 0 4 5 6 9

## Pass 4

0 1 0 2 4 5 6 9

## Pass 5

0 0 1 2 4 5 6 9

## Pass 6

0 0 1 2 4 5 6 9

## Pass 7

0 0 1 2 4 5 6 9

## Input

5 1 2 0 4 6 0 9

## Pass 1

1 2 0 4 5 0 6 9

## Pass 2

1 0 2 4 0 5 6 9

## Pass 3

0 1 2 0 4 5 6 9

## Pass 4

0 1 0 2 4 5 6 9

## Pass 5

0 0 1 2 4 5 6 9

## Pass 6

0 0 1 2 4 5 6 9

## Pass 7

0 0 1 2 4 5 6 9

## Input

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 5 | 1 | 2 | 0 | 4 | 6 | 0 | 9 |

## Output

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 2 | 4 | 5 | 6 | 9 |

# Running Time

**Algorithm 1** BubbleSort algorithm

```
procedure BUBBLE(a[], n)
    for i ← n − 1 to 1 do
        for j ← 1 to i do
            if a[j − 1] > a[j] then
                swap(a[j − 1], a[j])
            end if
        end for
    end for
end procedure
```

# Running Time

**Algorithm 2** BubbleSort algorithm

**procedure** BUBBLE($a[], n$)
    **for** $i \leftarrow n - 1$ to 1 **do**
        **for** $j \leftarrow 1$ to $i$ **do**
            **if** $a[j-1] > a[j]$ **then**
                $swap(a[j-1], a[j])$
            **end if**
        **end for**
    **end for**
**end procedure**

} $\Theta(1)$

# Running Time

**Algorithm 3** BubbleSort algorithm

```
procedure BUBBLE(a[], n)
    for i ← n − 1 to 1 do
        for j ← 1 to i do
            if a[j − 1] > a[j] then
                swap(a[j − 1], a[j])
            end if
        end for
    end for
end procedure
```

Lines 4–6: $\Theta(1)$; lines 3–7: $\Theta(i)$

# Running Time

**Algorithm 4** BubbleSort algorithm

```
procedure BUBBLE(a[], n)
    for i ← n − 1 to 1 do
        for j ← 1 to i do
            if a[j − 1] > a[j] then
                swap(a[j − 1], a[j])
            end if
        end for
    end for
end procedure
```

$\Theta(1)$ $\Theta(i)$ $\Sigma_1^{n-1}\Theta(i)$

# Running Time

**Algorithm 5** BubbleSort algorithm

```
procedure BUBBLE(a[], n)
    for i ← n − 1 to 1 do
        for j ← 1 to i do
            if a[j − 1] > a[j] then
                swap(a[j − 1], a[j])
            end if
        end for
    end for
end procedure
```

$\Theta(1)$ $\Theta(i)$ $\Sigma_1^{n-1}\Theta(i)$

## Running Time

$\Sigma_0^{n-1}\Theta(i) = \Theta(\Sigma_1^{n-1} i) = \Theta(n^2)$

## Running Time

- Worst case:
- Reverse sorted:
- Already sorted:

## In-place

## Stable

## Adaptive

(Suggest the change in the previous implementation.)

## Running Time

- Worst case: $O(n^2)$
- Reverse sorted:
- Already sorted:

## In-place

## Stable

## Adaptive

(Suggest the change in the previous implementation.)

## Running Time

- Worst case: $O(n^2)$
- Reverse sorted: $\Theta(n^2)$
- Already sorted:

## In-place

## Stable

## Adaptive

(Suggest the change in the previous implementation.)

## Running Time

- Worst case: $O(n^2)$
- Reverse sorted: $\Theta(n^2)$
- Already sorted: Can be designed for $\Theta(n)$

## In-place

## Stable

## Adaptive

(Suggest the change in the previous implementation.)

## Running Time

- Worst case: $O(n^2)$
- Reverse sorted: $\Theta(n^2)$
- Already sorted: Can be designed for $\Theta(n)$

## In-place

## Stable

## Adaptive

(Suggest the change in the previous implementation.)

## Running Time

- Worst case: $O(n^2)$
- Reverse sorted: $\Theta(n^2)$
- Already sorted: Can be designed for $\Theta(n)$

## In-place

$O(1)$ extra space

## Stable

## Adaptive

(Suggest the change in the previous implementation.)

## Running Time

- Worst case: $O(n^2)$
- Reverse sorted: $\Theta(n^2)$
- Already sorted: Can be designed for $\Theta(n)$

## In-place

$O(1)$ extra space

## Stable

Yes

## Adaptive

(Suggest the change in the previous implementation.)

## Running Time

- Worst case: $O(n^2)$
- Reverse sorted: $\Theta(n^2)$
- Already sorted: Can be designed for $\Theta(n)$

## In-place

$O(1)$ extra space

## Stable

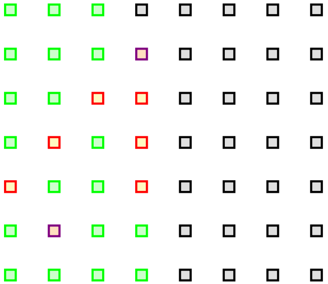
Yes

## Adaptive

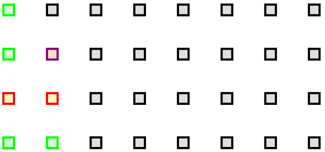
Yes (Suggest the change in the previous implementation.)

# Outline

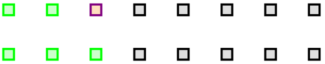
## An intermediate step



## Pass 1



## Pass 2



. . .

## Pass 7

## Input

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 5 | 1 | 2 | 0 | 4 | 6 | 0 | 9 |

## Input

5 1 2 0 4 6 0 9

## Pass 1

5 1 2 0 4 6 0 9

## Input

5 1 2 0 4 6 0 9

## Pass 1

1 5 2 0 4 6 0 9

## Input

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 5 | 1 | 2 | 0 | 4 | 6 | 0 | 9 |

## Pass 1

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | 5 | 2 | 0 | 4 | 6 | 0 | 9 |

## Input

5 1 2 0 4 6 0 9

## Pass 1

1 5 2 0 4 6 0 9

## Pass 2

1 5 2 0 4 6 0 9

## Input

5 1 2 0 4 6 0 9

## Pass 1

1 5 2 0 4 6 0 9

## Pass 2

1 5 2 0 4 6 0 9

## Input

5 1 2 0 4 6 0 9

## Pass 1

1 5 2 0 4 6 0 9

## Pass 2

1 2 5 0 4 6 0 9

## Input

5 1 2 0 4 6 0 9

## Pass 1

1 5 2 0 4 6 0 9

## Pass 2

1 2 5 0 4 6 0 9

## Input

5 1 2 0 4 6 0 9

## Pass 1

1 5 2 0 4 6 0 9

## Pass 2

1 2 5 0 4 6 0 9

## Pass 3

1 2 5 0 4 6 0 9

## Input

5 1 2 0 4 6 0 9

## Pass 1

1 5 2 0 4 6 0 9

## Pass 2

1 2 5 0 4 6 0 9

## Pass 3

1 2 5 0 4 6 0 9

## Input

5 1 2 0 4 6 0 9

## Pass 1

1 5 2 0 4 6 0 9

## Pass 2

1 2 5 0 4 6 0 9

## Pass 3

1 2 0 5 4 6 0 9

## Input

5 1 2 0 4 6 0 9

## Pass 1

1 5 2 0 4 6 0 9

## Pass 2

1 2 5 0 4 6 0 9

## Pass 3

1 0 2 5 4 6 0 9

## Input

5 1 2 0 4 6 0 9

## Pass 1

1 5 2 0 4 6 0 9

## Pass 2

1 2 5 0 4 6 0 9

## Pass 3

0 1 2 5 4 6 0 9

### Input

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 5 | 1 | 2 | 0 | 4 | 6 | 0 | 9 |

### Pass 1

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | 5 | 2 | 0 | 4 | 6 | 0 | 9 |

### Pass 2

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | 2 | 5 | 0 | 4 | 6 | 0 | 9 |

### Pass 3

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 5 | 4 | 6 | 0 | 9 |

## Input

5 1 2 0 4 6 0 9

## Pass 1

1 5 2 0 4 6 0 9

## Pass 2

1 2 5 0 4 6 0 9

## Pass 3

0 1 2 5 4 6 0 9

## Pass 4

0 1 2 5 4 6 0 9

### Input

5 1 2 0 4 6 0 9

### Pass 1

1 5 2 0 4 6 0 9

### Pass 2

1 2 5 0 4 6 0 9

### Pass 3

0 1 2 5 4 6 0 9

### Pass 4

0 1 2 5 4 6 0 9

## Input

5 1 2 0 4 6 0 9

## Pass 1

1 5 2 0 4 6 0 9

## Pass 2

1 2 5 0 4 6 0 9

## Pass 3

0 1 2 5 4 6 0 9

## Pass 4

0 1 2 4 5 6 0 9

## Input

5 1 2 0 4 6 0 9

## Pass 1

1 5 2 0 4 6 0 9

## Pass 2

1 2 5 0 4 6 0 9

## Pass 3

0 1 2 5 4 6 0 9

## Pass 4

0 1 2 4 5 6 0 9

## Input

5 1 2 0 4 6 0 9

## Pass 1

1 5 2 0 4 6 0 9

## Pass 2

1 2 5 0 4 6 0 9

## Pass 3

0 1 2 5 4 6 0 9

## Pass 4

0 1 2 4 5 6 0 9

## Pass 5

0 1 2 4 5 6 0 9

### Input

5 1 2 0 4 6 0 9

### Pass 1

1 5 2 0 4 6 0 9

### Pass 2

1 2 5 0 4 6 0 9

### Pass 3

0 1 2 5 4 6 0 9

### Pass 4

0 1 2 4 5 6 0 9

### Pass 5

0 1 2 4 5 6 0 9

## Input

5 1 2 0 4 6 0 9

## Pass 1

1 5 2 0 4 6 0 9

## Pass 2

1 2 5 0 4 6 0 9

## Pass 3

0 1 2 5 4 6 0 9

## Pass 4

0 1 2 4 5 6 0 9

## Pass 5

0 1 2 4 5 6 0 9

## Input

5 1 2 0 4 6 0 9

## Pass 1

1 5 2 0 4 6 0 9

## Pass 2

1 2 5 0 4 6 0 9

## Pass 3

0 1 2 5 4 6 0 9

## Pass 4

0 1 2 4 5 6 0 9

## Pass 5

0 1 2 4 5 6 0 9

## Pass 6

0 1 2 4 5 6 0 9

### Input

5 1 2 0 4 6 0 9

### Pass 1

1 5 2 0 4 6 0 9

### Pass 2

1 2 5 0 4 6 0 9

### Pass 3

0 1 2 5 4 6 0 9

### Pass 4

0 1 2 4 5 6 0 9

### Pass 5

0 1 2 4 5 6 0 9

### Pass 6

0 1 2 4 5 6 0 9

## Input

5 1 2 0 4 6 0 9

## Pass 1

1 5 2 0 4 6 0 9

## Pass 2

1 2 5 0 4 6 0 9

## Pass 3

0 1 2 5 4 6 0 9

## Pass 4

0 1 2 4 5 6 0 9

## Pass 5

0 1 2 4 5 6 0 9

## Pass 6

0 1 2 4 5 0 6 9

## Input

5 1 2 0 4 6 0 9

## Pass 1

1 5 2 0 4 6 0 9

## Pass 2

1 2 5 0 4 6 0 9

## Pass 3

0 1 2 5 4 6 0 9

## Pass 4

0 1 2 4 5 6 0 9

## Pass 5

0 1 2 4 5 6 0 9

## Pass 6

0 1 2 4 0 5 6 9

### Input

5 1 2 0 4 6 0 9

### Pass 1

1 5 2 0 4 6 0 9

### Pass 2

1 2 5 0 4 6 0 9

### Pass 3

0 1 2 5 4 6 0 9

### Pass 4

0 1 2 4 5 6 0 9

### Pass 5

0 1 2 4 5 6 0 9

### Pass 6

0 1 2 0 4 5 6 9

## Input

5 1 2 0 4 6 0 9

## Pass 1

1 5 2 0 4 6 0 9

## Pass 2

1 2 5 0 4 6 0 9

## Pass 3

0 1 2 5 4 6 0 9

## Pass 4

0 1 2 4 5 6 0 9

## Pass 5

0 1 2 4 5 6 0 9

## Pass 6

0 1 0 2 4 5 6 9

## Input

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 5 | 1 | 2 | 0 | 4 | 6 | 0 | 9 |

## Pass 1

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | 5 | 2 | 0 | 4 | 6 | 0 | 9 |

## Pass 2

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | 2 | 5 | 0 | 4 | 6 | 0 | 9 |

## Pass 3

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 5 | 4 | 6 | 0 | 9 |

## Pass 4

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 4 | 5 | 6 | 0 | 9 |

## Pass 5

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 4 | 5 | 6 | 0 | 9 |

## Pass 6

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 2 | 4 | 5 | 6 | 9 |

## Input

5 1 2 0 4 6 0 9

## Pass 1

1 5 2 0 4 6 0 9

## Pass 2

1 2 5 0 4 6 0 9

## Pass 3

0 1 2 5 4 6 0 9

## Pass 4

0 1 2 4 5 6 0 9

## Pass 5

0 1 2 4 5 6 0 9

## Pass 6

0 0 1 2 4 5 6 9

## Input

5 1 2 0 4 6 0 9

## Pass 1

1 5 2 0 4 6 0 9

## Pass 2

1 2 5 0 4 6 0 9

## Pass 3

0 1 2 5 4 6 0 9

## Pass 4

0 1 2 4 5 6 0 9

## Pass 5

0 1 2 4 5 6 0 9

## Pass 6

0 0 1 2 4 5 6 9

## Pass 7

0 0 1 2 4 5 6 9

**Input**

5 1 2 0 4 6 0 9

**Pass 1**

1 5 2 0 4 6 0 9

**Pass 2**

1 2 5 0 4 6 0 9

**Pass 3**

0 1 2 5 4 6 0 9

**Pass 4**

0 1 2 4 5 6 0 9

**Pass 5**

0 1 2 4 5 6 0 9

**Pass 6**

0 0 1 2 4 5 6 9

**Pass 7**

0 0 1 2 4 5 6 9

**Input**

5 1 2 0 4 6 0 9

**Pass 1**

1 5 2 0 4 6 0 9

**Pass 2**

1 2 5 0 4 6 0 9

**Pass 3**

0 1 2 5 4 6 0 9

**Pass 4**

0 1 2 4 5 6 0 9

**Pass 5**

0 1 2 4 5 6 0 9

**Pass 6**

0 0 1 2 4 5 6 9

**Pass 7**

0 0 1 2 4 5 6 9

## Input

5 1 2 0 4 6 0 9

## Output

0 0 1 2 4 5 6 9

**Algorithm 6** Insertion algorithm

```
procedure INERTION(a[], n)
    for i ← 1 to n − 1 do
        key ← a[i]
        j ← i
        while j > 0 and a[j − 1] > key do
            a[j] ← a[j − 1]
            j ← j − 1
        end while
        a[j] ← key
    end for
end procedure
```

**Algorithm 7** Insertion algorithm

**procedure** INERTION($a[], n$)
  **for** $i \leftarrow 1$ to $n - 1$ **do**
    $key \leftarrow a[i]$
    $j \leftarrow i$
    **while** $j > 0$ and $a[j-1] > key$ **do**
      $a[j] \leftarrow a[j-1]$
      $j \leftarrow j - 1$
    **end while**
    $a[j] \leftarrow key$
  **end for**
**end procedure**

$\{$ $O(1)$

**Algorithm 8** Insertion algorithm

```
procedure INERTION(a[], n)
    for i ← 1 to n − 1 do
        key ← a[i]
        j ← i
        while j > 0 and a[j − 1] > key do
            a[j] ← a[j − 1]
            j ← j − 1
        end while
        a[j] ← key
    end for
end procedure
```

$\{$ $O(1)$ $\}$ $t_i$ times

**Algorithm 9** Insertion algorithm

```
procedure INERTION(a[], n)
    for i ← 1 to n − 1 do
        key ← a[i]
        j ← i
        while j > 0 and a[j − 1] > key do
            a[j] ← a[j − 1]
            j ← j − 1
        end while
        a[j] ← key
    end for
end procedure
```

$t_i$ times

$O(1)$

$\sum_1^{n-1} t_i$

**Algorithm 10** Insertion algorithm

```
procedure INERTION(a[], n)
    for i ← 1 to n − 1 do
        key ← a[i]
        j ← i
        while j > 0 and a[j − 1] > key do
            a[j] ← a[j − 1]
            j ← j − 1
        end while
        a[j] ← key
    end for
end procedure
```

$O(1)$

$t_i$ times

$\Sigma_1^{n-1} t_i$

## Running Time

$\Sigma_1^{n-1} t_i$ where $t_i$ is the time taken in $i^{th}$ loop.

## Running Time

- Worst case:
- Best case:
- Reverse sorted:
- Already sorted:

## In-place

## Stable

## Adaptive

## Running Time

- Worst case: $t_i = i \Rightarrow$ Running time $\Sigma_1^{n-1} i = O(n^2)$
- Best case: $t_i = 1 \Rightarrow$ Running time $\Sigma_1^{n-1} 1 = O(n)$
- Reverse sorted:
- Already sorted:

## In-place

## Stable

## Adaptive

## Running Time

- Worst case: $t_i = i \Rightarrow$ Running time $\Sigma_1^{n-1} i = O(n^2)$
- Best case: $t_i = 1 \Rightarrow$ Running time $\Sigma_1^{n-1} 1 = O(n)$
- Reverse sorted: $\Theta(n^2)$
- Already sorted:

## In-place

## Stable

## Adaptive

## Running Time

- Worst case: $t_i = i \Rightarrow$ Running time $\Sigma_1^{n-1} i = O(n^2)$
- Best case: $t_i = 1 \Rightarrow$ Running time $\Sigma_1^{n-1} 1 = O(n)$
- Reverse sorted: $\Theta(n^2)$
- Already sorted: $\Theta(n)$

## In-place

## Stable

## Adaptive

## Running Time

- Worst case: $t_i = i \Rightarrow$ Running time $\Sigma_1^{n-1} i = O(n^2)$
- Best case: $t_i = 1 \Rightarrow$ Running time $\Sigma_1^{n-1} 1 = O(n)$
- Reverse sorted: $\Theta(n^2)$
- Already sorted: $\Theta(n)$

## In-place

## Stable

## Adaptive

## Running Time

- Worst case: $t_i = i \Rightarrow$ Running time $\Sigma_1^{n-1} i = O(n^2)$
- Best case: $t_i = 1 \Rightarrow$ Running time $\Sigma_1^{n-1} 1 = O(n)$
- Reverse sorted: $\Theta(n^2)$
- Already sorted: $\Theta(n)$

## In-place

$O(1)$ extra space

## Stable

## Adaptive

## Running Time

- Worst case: $t_i = i \Rightarrow$ Running time $\Sigma_1^{n-1} i = O(n^2)$
- Best case: $t_i = 1 \Rightarrow$ Running time $\Sigma_1^{n-1} 1 = O(n)$
- Reverse sorted: $\Theta(n^2)$
- Already sorted: $\Theta(n)$

## In-place

$O(1)$ extra space

## Stable

Yes

## Adaptive

## Running Time

- Worst case: $t_i = i \Rightarrow$ Running time $\Sigma_1^{n-1} i = O(n^2)$
- Best case: $t_i = 1 \Rightarrow$ Running time $\Sigma_1^{n-1} 1 = O(n)$
- Reverse sorted: $\Theta(n^2)$
- Already sorted: $\Theta(n)$

## In-place

$O(1)$ extra space

## Stable

Yes

## Adaptive

Yes, and less overhead

## Running Time

- Worst case:
- Best case:
- Reverse sorted:
- Already sorted:

## In-place

## Stable

## Adaptive

What if binary search is used?

## Running Time

- Worst case:
- Best case:
- Reverse sorted:
- Already sorted:

## In-place

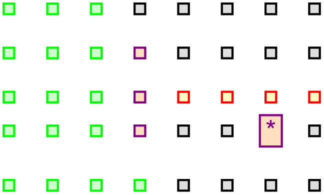
## Stable

## Adaptive

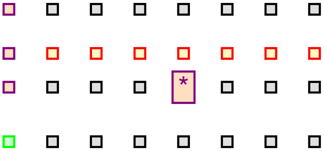
What if a linked list is used instead of an array?

# Outline

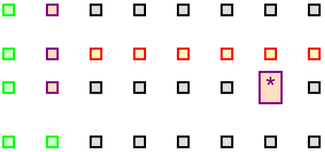
## An intermediate step




## Pass 1

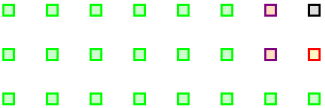



## Pass 2




. . .

## Pass 7



## Input

5 1 2 0 4 6 0 9

## Input

5 1 2 0 4 6 0 9

## Pass 1

5 1 2 0 4 6 0 9

## Input

5 1 2 0 4 6 0 9

## Pass 1

5 1 2 0 4 6 0 9

### Input

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 5 | 1 | 2 | 0 | 4 | 6 | 0 | 9 |

### Pass 1

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 5 | 1 | 2 | 0 | 4 | 6 | 0 | 9 |

## Input

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 5 | 1 | 2 | 0 | 4 | 6 | 0 | 9 |

## Pass 1

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 5 | 4 | 6 | 0 | 9 |

## Input

5 1 2 0 4 6 0 9

## Pass 1

0 1 2 5 4 6 0 9

## Pass 2

0 1 2 5 4 6 0 9

## Input

5 1 2 0 4 6 0 9

## Pass 1

0 1 2 5 4 6 0 9

## Pass 2

0 1 2 5 4 6 0 9

## Input

5 1 2 0 4 6 0 9

## Pass 1

0 1 2 5 4 6 0 9

## Pass 2

0 1 2 5 4 6 0 9

## Input

5 1 2 0 4 6 0 9

## Pass 1

0 1 2 5 4 6 0 9

## Pass 2

0 0 2 5 4 6 1 9

## Input

5 1 2 0 4 6 0 9

## Pass 1

0 1 2 5 4 6 0 9

## Pass 2

0 0 2 5 4 6 1 9

## Pass 3

0 0 2 5 4 6 1 9

## Input

5 1 2 0 4 6 0 9

## Pass 1

0 1 2 5 4 6 0 9

## Pass 2

0 0 2 5 4 6 1 9

## Pass 3

0 0 2 5 4 6 1 9

## Input

5 1 2 0 4 6 0 9

## Pass 1

0 1 2 5 4 6 0 9

## Pass 2

0 0 2 5 4 6 1 9

## Pass 3

0 0 2 5 4 6 1 9

## Input

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 5 | 1 | 2 | 0 | 4 | 6 | 0 | 9 |

## Pass 1

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 5 | 4 | 6 | 0 | 9 |

## Pass 2

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 2 | 5 | 4 | 6 | 1 | 9 |

## Pass 3

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 5 | 4 | 6 | 2 | 9 |

## Input

5 1 2 0 4 6 0 9

## Pass 1

0 1 2 5 4 6 0 9

## Pass 2

0 0 2 5 4 6 1 9

## Pass 3

0 0 1 5 4 6 2 9

## Pass 4

0 0 1 5 4 6 2 9

## Input

5 1 2 0 4 6 0 9

## Pass 1

0 1 2 5 4 6 0 9

## Pass 2

0 0 2 5 4 6 1 9

## Pass 3

0 0 1 5 4 6 2 9

## Pass 4

0 0 1 5 4 6 2 9

## Input

5 1 2 0 4 6 0 9

## Pass 1

0 1 2 5 4 6 0 9

## Pass 2

0 0 2 5 4 6 1 9

## Pass 3

0 0 1 5 4 6 2 9

## Pass 4

0 0 1 5 4 6 2 9

## Input

5 1 2 0 4 6 0 9

## Pass 1

0 1 2 5 4 6 0 9

## Pass 2

0 0 2 5 4 6 1 9

## Pass 3

0 0 1 5 4 6 2 9

## Pass 4

0 0 1 2 4 6 5 9

## Input

5 1 2 0 4 6 0 9

## Pass 1

0 1 2 5 4 6 0 9

## Pass 2

0 0 2 5 4 6 1 9

## Pass 3

0 0 1 5 4 6 2 9

## Pass 4

0 0 1 2 4 6 5 9

## Pass 5

0 0 1 2 4 6 5 9

### Input

5 1 2 0 4 6 0 9

### Pass 1

0 1 2 5 4 6 0 9

### Pass 2

0 0 2 5 4 6 1 9

### Pass 3

0 0 1 5 4 6 2 9

### Pass 4

0 0 1 2 4 6 5 9

### Pass 5

0 0 1 2 4 6 5 9

## Input

5 1 2 0 4 6 0 9

## Pass 1

0 1 2 5 4 6 0 9

## Pass 2

0 0 2 5 4 6 1 9

## Pass 3

0 0 1 5 4 6 2 9

## Pass 4

0 0 1 2 4 6 5 9

## Pass 5

0 0 1 2 4 6 5 9

## Input

5 1 2 0 4 6 0 9

## Pass 1

0 1 2 5 4 6 0 9

## Pass 2

0 0 2 5 4 6 1 9

## Pass 3

0 0 1 5 4 6 2 9

## Pass 4

0 0 1 2 4 6 5 9

## Pass 5

0 0 1 2 4 6 5 9

### Input

5 1 2 0 4 6 0 9

### Pass 1

0 1 2 5 4 6 0 9

### Pass 2

0 0 2 5 4 6 1 9

### Pass 3

0 0 1 5 4 6 2 9

### Pass 4

0 0 1 2 4 6 5 9

### Pass 5

0 0 1 2 4 6 5 9

### Pass 6

0 0 1 2 4 6 5 9

## Input

5 1 2 0 4 6 0 9

## Pass 1

0 1 2 5 4 6 0 9

## Pass 2

0 0 2 5 4 6 1 9

## Pass 3

0 0 1 5 4 6 2 9

## Pass 4

0 0 1 2 4 6 5 9

## Pass 5

0 0 1 2 4 6 5 9

## Pass 6

0 0 1 2 4 6 5 9

## Input

5 1 2 0 4 6 0 9

## Pass 1

0 1 2 5 4 6 0 9

## Pass 2

0 0 2 5 4 6 1 9

## Pass 3

0 0 1 5 4 6 2 9

## Pass 4

0 0 1 2 4 6 5 9

## Pass 5

0 0 1 2 4 6 5 9

## Pass 6

0 0 1 2 4 6 5 9

## Input

5 1 2 0 4 6 0 9

## Pass 1

0 1 2 5 4 6 0 9

## Pass 2

0 0 2 5 4 6 1 9

## Pass 3

0 0 1 5 4 6 2 9

## Pass 4

0 0 1 2 4 6 5 9

## Pass 5

0 0 1 2 4 6 5 9

## Pass 6

0 0 1 2 4 5 6 9

## Input

5 1 2 0 4 6 0 9

## Pass 1

0 1 2 5 4 6 0 9

## Pass 2

0 0 2 5 4 6 1 9

## Pass 3

0 0 1 5 4 6 2 9

## Pass 4

0 0 1 2 4 6 5 9

## Pass 5

0 0 1 2 4 6 5 9

## Pass 6

0 0 1 2 4 5 6 9

## Pass 7

0 0 1 2 4 5 6 9

## Input

5 1 2 0 4 6 0 9

## Pass 1

0 1 2 5 4 6 0 9

## Pass 2

0 0 2 5 4 6 1 9

## Pass 3

0 0 1 5 4 6 2 9

## Pass 4

0 0 1 2 4 6 5 9

## Pass 5

0 0 1 2 4 6 5 9

## Pass 6

0 0 1 2 4 5 6 9

## Pass 7

0 0 1 2 4 5 6 9

## Input

5 1 2 0 4 6 0 9

## Pass 1

0 1 2 5 4 6 0 9

## Pass 2

0 0 2 5 4 6 1 9

## Pass 3

0 0 1 5 4 6 2 9

## Pass 4

0 0 1 2 4 6 5 9

## Pass 5

0 0 1 2 4 6 5 9

## Pass 6

0 0 1 2 4 5 6 9

## Pass 7

0 0 1 2 4 5 6 9

## Input

5 1 2 0 4 6 0 9

## Output

0 0 1 2 4 5 6 9

**Algorithm 11** Selection algorithm

```
procedure SELECTION(a[], n)
    for i ← 0 to n − 2 do
        minPos ← i
        for j ← i + 1 to n − 1 do
            if a[j] < a[minPos] then
                minPos ← j
            end if
        end for
        swap(a[i], a[minPos])
    end for
end procedure
```

**Algorithm 12** Selection algorithm

```
procedure SELECTION(a[], n)
    for i ← 0 to n − 2 do
        minPos ← i
        for j ← i + 1 to n − 1 do
            if a[j] < a[minPos] then  ┐
                minPos ← j            ├ Θ(1)
            end if                    ┘
        end for
        swap(a[i], a[minPos])
    end for
end procedure
```

**Algorithm 13** Selection algorithm

```
procedure SELECTION(a[], n)
    for i ← 0 to n − 2 do
        minPos ← i
        for j ← i + 1 to n − 1 do
            if a[j] < a[minPos] then
                minPos ← j
            end if
        end for
        swap(a[i], a[minPos])
    end for
end procedure
```

$\Theta(1)$

$\Theta(n - i)$ times

**Algorithm 14** Selection algorithm

```
procedure SELECTION(a[], n)
    for i ← 0 to n − 2 do
        minPos ← i
        for j ← i + 1 to n − 1 do
            if a[j] < a[minPos] then
                minPos ← j
            end if
        end for
        swap(a[i], a[minPos])
    end for
end procedure
```

$\Theta(1)$ (lines 5–7)

$\Theta(n-i)$ times (lines 4–8)

$\sum_0^{n-2}\Theta(n-i)$ (lines 2–10)

**Algorithm 15** Selection algorithm

```
procedure SELECTION(a[], n)
    for i ← 0 to n − 2 do
        minPos ← i
        for j ← i + 1 to n − 1 do
            if a[j] < a[minPos] then
                minPos ← j
            end if
        end for
        swap(a[i], a[minPos])
    end for
end procedure
```

$\Theta(1)$

$\Theta(n-i)$ times

$\Sigma_0^{n-2}\Theta(n-i)$

## Running Time

$\Sigma_0^{n-2}\Theta(n-i) = \Theta(\Sigma_1^{n-1}(n-i)) = \cdots = \Theta(\Sigma_1^{n} i) = \Theta(n^2)$

## Running Time

- Worst case:
- Best case:
- Reverse sorted:
- Already sorted:

## In-place

## Stable

## Adaptive

## Running Time

- Worst case: $O(n^2)$
- Best case: $O(n^2)$
- Reverse sorted:
- Already sorted:

## In-place

## Stable

## Adaptive

## Running Time

- Worst case: $O(n^2)$
- Best case: $O(n^2)$
- Reverse sorted: $\Theta(n^2)$
- Already sorted:

## In-place

## Stable

## Adaptive

## Running Time

- Worst case: $O(n^2)$
- Best case: $O(n^2)$
- Reverse sorted: $\Theta(n^2)$
- Already sorted: $\Theta(n^2)$

## In-place

## Stable

## Adaptive

## Running Time

- Worst case: $O(n^2)$
- Best case: $O(n^2)$
- Reverse sorted: $\Theta(n^2)$
- Already sorted: $\Theta(n^2)$

## In-place

## Stable

## Adaptive

## Running Time

- Worst case: $O(n^2)$
- Best case: $O(n^2)$
- Reverse sorted: $\Theta(n^2)$
- Already sorted: $\Theta(n^2)$

## In-place

$O(1)$ extra space

## Stable

## Adaptive

## Running Time

- Worst case: $O(n^2)$
- Best case: $O(n^2)$
- Reverse sorted: $\Theta(n^2)$
- Already sorted: $\Theta(n^2)$

## In-place

$O(1)$ extra space

## Stable

No

## Adaptive

## Running Time

- Worst case: $O(n^2)$
- Best case: $O(n^2)$
- Reverse sorted: $\Theta(n^2)$
- Already sorted: $\Theta(n^2)$

## In-place

$O(1)$ extra space

## Stable

No

## Adaptive

NO

## Running Time

- Worst case:
- Best case:
- Reverse sorted:
- Already sorted:

## In-place

## Stable

## Adaptive

Is there anything good about this algo?
What if a heap is used to find the minimum? (Heap Sort)

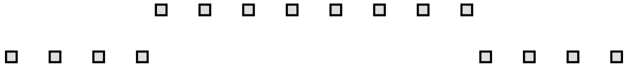
# Outline

## Divide and Conquer

- Divide the $n$-elements sequence into 2 subsequences of $n/2$ elements.
- **Recursively** sort each subsequence using merge sort.
- **Merge** the two sorted subsequences to produce the sorted answer.

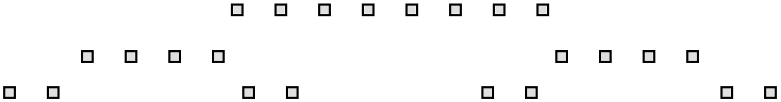
## Divide and Conquer

- Divide the $n$-elements sequence into 2 subsequences of $n/2$ elements.
- **Recursively** sort each subsequence using merge sort.
- **Merge** the two sorted subsequences to produce the sorted answer.



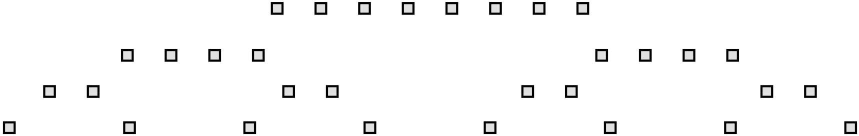
## Divide and Conquer

- Divide the $n$-elements sequence into 2 subsequences of $n/2$ elements.
- **Recursively** sort each subsequence using merge sort.
- **Merge** the two sorted subsequences to produce the sorted answer.



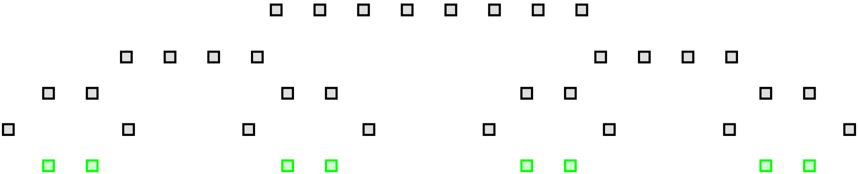
## Divide and Conquer

- Divide the $n$-elements sequence into 2 subsequences of $n/2$ elements.
- **Recursively** sort each subsequence using merge sort.
- **Merge** the two sorted subsequences to produce the sorted answer.



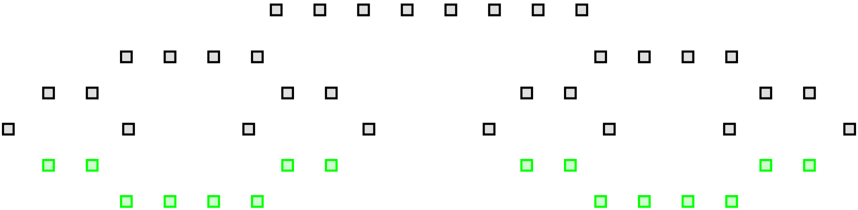
## Divide and Conquer

- Divide the $n$-elements sequence into 2 subsequences of $n/2$ elements.
- **Recursively** sort each subsequence using merge sort.
- **Merge** the two sorted subsequences to produce the sorted answer.



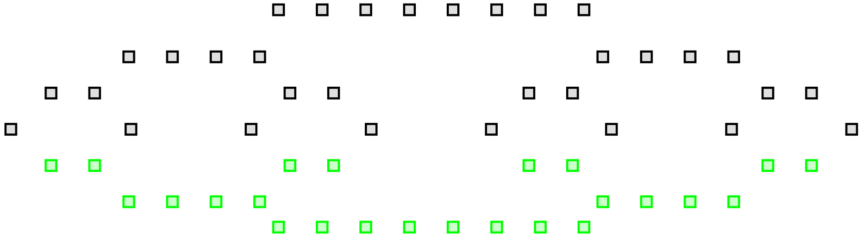
## Divide and Conquer

- Divide the $n$-elements sequence into 2 subsequences of $n/2$ elements.
- **Recursively** sort each subsequence using merge sort.
- **Merge** the two sorted subsequences to produce the sorted answer.



## Divide and Conquer

- Divide the $n$-elements sequence into 2 subsequences of $n/2$ elements.
- **Recursively** sort each subsequence using merge sort.
- **Merge** the two sorted subsequences to produce the sorted answer.



## Merging

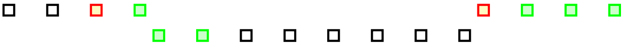
# Merging

# Merging

# Merging

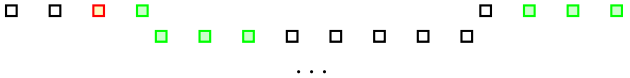
# Merging

## Merging



# Merging

## Merging



## Input

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 5 | 1 | 2 | 0 | 4 | 6 | 0 | 9 |

## Input

5 1 2 0 4 6 0 9

5 1 2 0 4 6 0 9

## Input

5 1 2 0 4 6 0 9

5 1 2 0 4 6 0 9

## Pass 1

1 5

## Input

5 1 2 0 4 6 0 9

5 1 2 0 4 6 0 9

## Pass 1

1 5 0 2

## Input

5 1 2 0 4 6 0 9

5 1 2 0 4 6 0 9

## Pass 1

1 5 0 2 4 6

## Input

5 1 2 0 4 6 0 9

5 1 2 0 4 6 0 9

## Pass 1

1 5 0 2 4 6 0 9

## Input

5 1 2 0 4 6 0 9

## Pass 1

1 5 | 0 2 | 4 6 | 0 9

## Pass 2

1 5 | 0 2

## Input

5 1 2 0 4 6 0 9

## Pass 1

1 5    0 2    4 6    0 9

## Pass 2

1 5    □ 2

0 □ □ □

## Input

5 1 2 0 4 6 0 9

## Pass 1

1 5    0 2    4 6    0 9

## Pass 2

1 5    □ 2

0 □ □ □

## Input

5 1 2 0 4 6 0 9

## Pass 1

1 5 | 0 2 | 4 6 | 0 9

## Pass 2

□ 5 □ 2

0 1 □ □

## Input

5 1 2 0 4 6 0 9

## Pass 1

1 5    0 2    4 6    0 9

## Pass 2

□ 5    □ 2

0 1 □ □

## Input

5 1 2 0 4 6 0 9

## Pass 1

1 5    0 2    4 6    0 9

## Pass 2

□ 5    □ □

0 1 2 □

## Input

5 1 2 0 4 6 0 9

## Pass 1

1 5    0 2    4 6    0 9

## Pass 2

0 1 2 5

## Input

5 1 2 0 4 6 0 9

## Pass 1

1 5    0 2    4 6    0 9

## Pass 2

4 6    0 9

0 1 2 5

## Input

5 1 2 0 4 6 0 9

## Pass 1

1 5    0 2    4 6    0 9

## Pass 2

0 1 2 5    0 4 6 9

## Input

5 1 2 0 4 6 0 9

## Pass 1

1 5 | 0 2 | 4 6 | 0 9

## Pass 2

0 1 2 5 | 0 4 6 9

## Pass 3

0 1 2 5 | 0 4 6 9

## Input

5 1 2 0 4 6 0 9

## Pass 1

1 5 | 0 2 | 4 6 | 0 9

## Pass 2

0 1 2 5 | 0 4 6 9

## Pass 3

0 0 1 2 4 5 6 9

## Input

5 1 2 0 4 6 0 9

## Output

**Algorithm 16** MergeSort algorithm

```
procedure MergeSort(a[], left, right)
    if left < right then
        mid ← ⌊(left + right)/2⌋
        MergeSort(a, left, mid)
        MergeSort(a, mid + 1, right)
        Merge(a, left, mid, right)
    end if
end procedure
procedure Merge(a[], left, mid, right)
    L ← a[left..mid]
    R ← a[mid + 1..right]
    k ← left
    while there are still elements in L or R do
        Compare the 'first' elements in L and R
        Move the minimum of them from its corresponding list to a[k]
        k ← k + 1
    end while
end procedure
```

# Time analysis

- If the problem size is small, say $n \leq c$ for some constant $c$, we can solve the problem in constant time, i.e.,$\Theta(1)$.
- Let $T(n)$ be the time needed to sort for input of size $n$.
- Let $C(n)$ be the time needed to merge 2 lists of total size $n$. We know that $C(n) = \Theta(n)$.
- Assume that the problem can be split into 2 subproblems in constant time and $c = 1$. then

$$T(n) = \begin{cases} \Theta(1) & \text{if } n \leq 1 \\ 2T(\frac{n}{2}) + \Theta(n) & \text{if } n > 1. \end{cases}$$

$$
\begin{aligned}
T(n) &= 2T(\tfrac{n}{2}) && +cn \\
&= 2[2T(\tfrac{n}{4}) + c\tfrac{n}{2}] && +cn \\
&= 4T(\tfrac{n}{4}) && +2cn \\
&= 4[2T(\tfrac{n}{8}) + c\tfrac{n}{4}] && +2cn \\
&= 8T(\tfrac{n}{8}) && +3cn \\
&= \vdots && \\
&= 2^k T(\tfrac{n}{2^k}) && +kcn
\end{aligned}
$$

w.l.o.g. , assume $2^k = n \qquad \Rightarrow k = \log_2 n$

$$T(n) = cn\log_2 n + n = \Theta(n\log_2 n)$$

## Running Time

- Worst case:
- Best case:
- Reverse sorted:
- Already sorted:

## In-place

## Stable

## Adaptive

## Running Time

- Worst case: $O(n \log_2 n)$
- Best case: $O(n \log_2 n)$
- Reverse sorted:
- Already sorted:

## In-place

## Stable

## Adaptive

## Running Time

- Worst case: $O(n \log_2 n)$
- Best case: $O(n \log_2 n)$
- Reverse sorted: $\Theta(n \log_2 n)$
- Already sorted:

## In-place

## Stable

## Adaptive

## Running Time

- Worst case: $O(n \log_2 n)$
- Best case: $O(n \log_2 n)$
- Reverse sorted: $\Theta(n \log_2 n)$
- Already sorted: Current implementation: $\Theta(n \log_2 n)$. Can be made linear. Any suggestion?

## In-place

## Stable

## Adaptive

## Running Time

- Worst case: $O(n \log_2 n)$
- Best case: $O(n \log_2 n)$
- Reverse sorted: $\Theta(n \log_2 n)$
- Already sorted: Current implementation: $\Theta(n \log_2 n)$. Can be made linear. Any suggestion?

## In-place

## Stable

## Adaptive

## Running Time

- Worst case: $O(n \log_2 n)$
- Best case: $O(n \log_2 n)$
- Reverse sorted: $\Theta(n \log_2 n)$
- Already sorted: Current implementation: $\Theta(n \log_2 n)$. Can be made linear. Any suggestion?

## In-place

$O(1)$ extra space

## Stable

## Adaptive

## Running Time

- Worst case: $O(n \log_2 n)$
- Best case: $O(n \log_2 n)$
- Reverse sorted: $\Theta(n \log_2 n)$
- Already sorted: Current implementation: $\Theta(n \log_2 n)$. Can be made linear. Any suggestion?

## In-place

$O(1)$ extra space

## Stable

Yes

## Adaptive

## Running Time

- Worst case: $O(n \log_2 n)$
- Best case: $O(n \log_2 n)$
- Reverse sorted: $\Theta(n \log_2 n)$
- Already sorted: Current implementation: $\Theta(n \log_2 n)$. Can be made linear. Any suggestion?

## In-place

$O(1)$ extra space

## Stable

Yes

## Adaptive

No (Current implementation)/ Yes (After a modification)

# References


- Cormen, Leiserson, Rivest : *Introduction to Algorithms*
- http://www.sorting-algorithms.com/