Introduction to Algorithms

Topic 3 : Comparison Based Sorting Algorithms

Xiang-Yang Li and Haisheng Tan[1]

School of Computer Science and Technology
University of Science and Technology of China (USTC)

Fall Semester 2021

Outline

Basic Concepts

Simple Sorting Algorithms

Efficient Sorting Algorithms

Summary

Basic Concepts of Sorting Algorithm

Stability

Regardless of how the input data is distributed, the data objects of the same keyword will be kept in the same order as in the input during the sorting process, which is called *stable sorting*. Otherwise, called *unstable sorting*.

Example: $2, 2^*, 1 \rightarrow 1, 2^*, 2$ (unstable sorting)

Basic Concepts of Sorting Algorithm

Stability

Regardless of how the input data is distributed, the data objects of the same keyword will be kept in the same order as in the input during the sorting process, which is called *stable sorting*. Otherwise, called *unstable sorting*.

Example: $2, 2^*, 1 \rightarrow 1, 2^*, 2$ (unstable sorting)

Time Complexity

Usually measured by the number of data **comparisons** and the number of data **movements** in the algorithm execution.

Basic Concepts of Sorting Algorithm

Stability

Regardless of how the input data is distributed, the data objects of the same keyword will be kept in the same order as in the input during the sorting process, which is called *stable sorting*. Otherwise, called *unstable sorting*.

Example: $2, 2^*, 1 \rightarrow 1, 2^*, 2$ (unstable sorting)

Time Complexity

Usually measured by the number of data **comparisons** and the number of data **movements** in the algorithm execution.

In-place Sorting

only a constant of elements are stored outside the input array.

Outline
Basic Concepts
Simple Sorting Algorithms
Efficient Sorting Algorithms
Summary

Insertion Sort
Selection Sort
Bubble Sort

Contents

Outline
Basic Concepts
Simple Sorting Algorithms
Efficient Sorting Algorithms
Summary

Insertion Sort
Selection Sort
Bubble Sort

Insertion Sort

General idea: Maintain an ordered sequence.

Outline
Basic Concepts
Simple Sorting Algorithms
Efficient Sorting Algorithms
Summary

Insertion Sort
Selection Sort
Bubble Sort

Insertion Sort

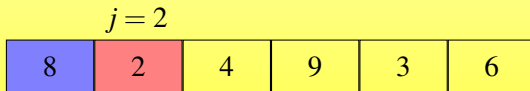**General idea:** Maintain an ordered sequence.

INSERTION-SORT(A)
1: **for** $j = 2$ to $A.length$ **do**
2:     $key = A[j]$
3:     // Insert $A[j]$ into the sorted sequence $A[1..j-1]$.
4:     $i = j - 1$
5:     **while** $i > 0$ and $A[i] > key$ **do**
6:         $A[i+1] = A[i]$
7:         $i = i - 1$
8:     $A[i+1] = key$

Outline
Basic Concepts
Simple Sorting Algorithms
Efficient Sorting Algorithms
Summary

Insertion Sort
Selection Sort
Bubble Sort

Example of Insertion Sort

8	2	4	9	3	6

Outline
Basic Concepts
Simple Sorting Algorithms
Efficient Sorting Algorithms
Summary

Insertion Sort
Selection Sort
Bubble Sort

Example of Insertion Sort

$j = 2$

8	2	4	9	3	6

Outline
Basic Concepts
Simple Sorting Algorithms
Efficient Sorting Algorithms
Summary

Insertion Sort
Selection Sort
Bubble Sort

Example of Insertion Sort

$$j = 2$$

8	2	4	9	3	6

Outline
Basic Concepts
Simple Sorting Algorithms
Efficient Sorting Algorithms
Summary

Insertion Sort
Selection Sort
Bubble Sort

Example of Insertion Sort

$j = 2$

8	2	4	9	3	6

Outline
Basic Concepts
Simple Sorting Algorithms
Efficient Sorting Algorithms
Summary

Insertion Sort
Selection Sort
Bubble Sort

Example of Insertion Sort

$$j = 3$$

2	8	4	9	3	6

Outline
Basic Concepts
Simple Sorting Algorithms
Efficient Sorting Algorithms
Summary

Insertion Sort
Selection Sort
Bubble Sort

Example of Insertion Sort

Outline
Basic Concepts
Simple Sorting Algorithms
Efficient Sorting Algorithms
Summary

Insertion Sort
Selection Sort
Bubble Sort

Example of Insertion Sort

$j = 3$

2	8	4	9	3	6

Outline
Basic Concepts
Simple Sorting Algorithms
Efficient Sorting Algorithms
Summary

Insertion Sort
Selection Sort
Bubble Sort

Example of Insertion Sort

Outline
Basic Concepts
Simple Sorting Algorithms
Efficient Sorting Algorithms
Summary

Insertion Sort
Selection Sort
Bubble Sort

Example of Insertion Sort

$$j = 4$$

2	4	8	9	3	6

Outline
Basic Concepts
Simple Sorting Algorithms
Efficient Sorting Algorithms
Summary

Insertion Sort
Selection Sort
Bubble Sort

Example of Insertion Sort

Outline
Basic Concepts
Simple Sorting Algorithms
Efficient Sorting Algorithms
Summary

Insertion Sort
Selection Sort
Bubble Sort

Example of Insertion Sort

$j = 4$

2	4	8	9	3	6

Outline
Basic Concepts
Simple Sorting Algorithms
Efficient Sorting Algorithms
Summary

Insertion Sort
Selection Sort
Bubble Sort

Example of Insertion Sort

$j = 5$

2	4	8	9	3	6

Outline
Basic Concepts
Simple Sorting Algorithms
Efficient Sorting Algorithms
Summary

Insertion Sort
Selection Sort
Bubble Sort

Example of Insertion Sort

$$j = 5$$

2	4	8	9	3	6

Outline
Basic Concepts
Simple Sorting Algorithms
Efficient Sorting Algorithms
Summary

Insertion Sort
Selection Sort
Bubble Sort

Example of Insertion Sort

Outline
Basic Concepts
Simple Sorting Algorithms
Efficient Sorting Algorithms
Summary

Insertion Sort
Selection Sort
Bubble Sort

Example of Insertion Sort

$j = 5$

2	4	8	9	3	6

Outline
Basic Concepts
Simple Sorting Algorithms
Efficient Sorting Algorithms
Summary

Insertion Sort
Selection Sort
Bubble Sort

Example of Insertion Sort

$j = 6$

2	3	4	8	9	6

Outline
Basic Concepts
Simple Sorting Algorithms
Efficient Sorting Algorithms
Summary

Insertion Sort
Selection Sort
Bubble Sort

Example of Insertion Sort

$$j = 6$$

| 2 | 3 | 4 | 8 | 9 | 6 |

Outline
Basic Concepts
Simple Sorting Algorithms
Efficient Sorting Algorithms
Summary

Insertion Sort
Selection Sort
Bubble Sort

Example of Insertion Sort

$j = 6$

2	3	4	8	9	6

Outline
Basic Concepts
Simple Sorting Algorithms
Efficient Sorting Algorithms
Summary

Insertion Sort
Selection Sort
Bubble Sort

Example of Insertion Sort

$$j = 6$$

2	3	4	8	9	6

Outline
Basic Concepts
Simple Sorting Algorithms
Efficient Sorting Algorithms
Summary

Insertion Sort
Selection Sort
Bubble Sort

Example of Insertion Sort

2	3	4	6	8	9

Outline
Basic Concepts
Simple Sorting Algorithms
Efficient Sorting Algorithms
Summary

Insertion Sort
Selection Sort
Bubble Sort

Insertion Sort

- Time Complexity
    - Best: $O(n)$
    - Average: $O(n^2)$
    - Worst: $O(n^2)$

- Memory: 1

- Stable: Yes

Outline
Basic Concepts
Simple Sorting Algorithms
Efficient Sorting Algorithms
Summary

Insertion Sort
Selection Sort
Bubble Sort

Insertion Sort

- Time Complexity
  - Best: $O(n)$
  - Average: $O(n^2)$
  - Worst: $O(n^2)$
- Memory: 1
- Stable: Yes

INSERTION-SORT(A)

1: **for** $j = 2$ to $A.length$ **do**
2:     $key = A[j]$
3:     // Insert $A[j]$ into the sorted sequence $A[1..j-1]$.
4:     $i = j - 1$
5:     **while** $i > 0$ and $A[i] > key$ **do**
6:       $A[i+1] = A[i]$
7:       $i = i - 1$
8:     $A[i+1] = key$

Outline
Basic Concepts
Simple Sorting Algorithms
Efficient Sorting Algorithms
Summary

Insertion Sort
Selection Sort
Bubble Sort

Contents

Outline
Basic Concepts
Simple Sorting Algorithms
Efficient Sorting Algorithms
Summary

Insertion Sort
Selection Sort
Bubble Sort

Selection Sort

General idea: Select and remove the smallest element from unsorted set.

Outline
Basic Concepts
Simple Sorting Algorithms
Efficient Sorting Algorithms
Summary

Insertion Sort
Selection Sort
Bubble Sort

Selection Sort

General idea: Select and remove the smallest element from unsorted set.

SELECTION-SORT(A)
1: **for** $i = 1$ to $A.length - 1$ **do**
2:     $k = i$ ▷ k is the position of the **smallest** key.
3:     **for** $j = i + 1$ to $A.length$ **do**
4:         **if** $A[j] < A[k]$ **then**
5:             $k = j$
6:     **if** $k \neq i$ **then**
7:         $A[i] \leftrightarrow A[k]$

Outline
Basic Concepts
Simple Sorting Algorithms
Efficient Sorting Algorithms
Summary

Insertion Sort
Selection Sort
Bubble Sort

Example of Selection Sort

8	2	4	9	3	6

Outline
Basic Concepts
Simple Sorting Algorithms
Efficient Sorting Algorithms
Summary

Insertion Sort
Selection Sort
Bubble Sort

Example of Selection Sort

smallest

8	2	4	9	3	6

Outline
Basic Concepts
Simple Sorting Algorithms
Efficient Sorting Algorithms
Summary

Insertion Sort
Selection Sort
Bubble Sort

Example of Selection Sort

Outline
Basic Concepts
Simple Sorting Algorithms
Efficient Sorting Algorithms
Summary

Insertion Sort
Selection Sort
Bubble Sort

Example of Selection Sort

2	8	4	9	3	6

Outline
Basic Concepts
Simple Sorting Algorithms
Efficient Sorting Algorithms
Summary

Insertion Sort
Selection Sort
Bubble Sort

Example of Selection Sort

smallest

| 2 | 8 | 4 | 9 | 3 | 6 |

Outline
Basic Concepts
Simple Sorting Algorithms
Efficient Sorting Algorithms
Summary

Insertion Sort
Selection Sort
Bubble Sort

Example of Selection Sort

smallest

2	8	4	9	3	6

Outline
Basic Concepts
Simple Sorting Algorithms
Efficient Sorting Algorithms
Summary

Insertion Sort
Selection Sort
Bubble Sort

Example of Selection Sort

2	3	4	9	8	6

Outline
Basic Concepts
Simple Sorting Algorithms
Efficient Sorting Algorithms
Summary

Insertion Sort
Selection Sort
Bubble Sort

Example of Selection Sort

smallest

2	3	4	9	8	6

Outline
Basic Concepts
Simple Sorting Algorithms
Efficient Sorting Algorithms
Summary

Insertion Sort
Selection Sort
Bubble Sort

Example of Selection Sort

2	3	4	9	8	6

Outline
Basic Concepts
Simple Sorting Algorithms
Efficient Sorting Algorithms
Summary

Insertion Sort
Selection Sort
Bubble Sort

Example of Selection Sort

smallest

| 2 | 3 | 4 | 9 | 8 | 6 |

Outline
Basic Concepts
Simple Sorting Algorithms
Efficient Sorting Algorithms
Summary

Insertion Sort
Selection Sort
Bubble Sort

Example of Selection Sort

smallest

| 2 | 3 | 4 | 9 | 8 | 6 |

Outline
Basic Concepts
Simple Sorting Algorithms
Efficient Sorting Algorithms
Summary

Insertion Sort
Selection Sort
Bubble Sort

Example of Selection Sort

2	3	4	6	8	9

Outline
Basic Concepts
Simple Sorting Algorithms
Efficient Sorting Algorithms
Summary

Insertion Sort
Selection Sort
Bubble Sort

Example of Selection Sort

smallest

2	3	4	6	8	9

Outline
Basic Concepts
Simple Sorting Algorithms
Efficient Sorting Algorithms
Summary

Insertion Sort
Selection Sort
Bubble Sort

Example of Selection Sort

2	3	4	6	8	9

Outline
Basic Concepts
Simple Sorting Algorithms
Efficient Sorting Algorithms
Summary

Insertion Sort
Selection Sort
Bubble Sort

Selection Sort

- Time Complexity
  - Best: $O(n^2)$
  - Average: $O(n^2)$
  - Worst: $O(n^2)$

- Memory: 1

- Stable: No

Outline
Basic Concepts
Simple Sorting Algorithms
Efficient Sorting Algorithms
Summary

Insertion Sort
Selection Sort
Bubble Sort

Selection Sort

- Time Complexity
    - Best: $O(n^2)$
    - Average: $O(n^2)$
    - Worst: $O(n^2)$

- Memory: 1

- Stable: No

SELECTION-SORT(A)
1: **for** $i = 1$ to $A.length - 1$ **do**
2:     $k = i$
3:     **for** $j = i + 1$ to $A.length$ **do**
4:         **if** $A[j] < A[k]$ **then**
5:             $k = j$
6:     **if** $k \neq i$ **then**
7:         $A[i] \leftrightarrow A[k]$

Outline
Basic Concepts
Simple Sorting Algorithms
Efficient Sorting Algorithms
Summary

Insertion Sort
Selection Sort
Bubble Sort

Selection Sort

Stable sorting: How to revise the selection sorting to make it *stable*?

Outline
Basic Concepts
Simple Sorting Algorithms
Efficient Sorting Algorithms
Summary

Insertion Sort
Selection Sort
Bubble Sort

Contents

Outline
Basic Concepts
Simple Sorting Algorithms
Efficient Sorting Algorithms
Summary

Insertion Sort
Selection Sort
Bubble Sort

Bubble Sort

General idea: From the back to the front, if some elements are smaller than their predecessor, then swap them.

Outline
Basic Concepts
Simple Sorting Algorithms
Efficient Sorting Algorithms
Summary

Insertion Sort
Selection Sort
Bubble Sort

Bubble Sort

General idea: From the back to the front, if some elements are smaller than their predecessor, then swap them.

BUBBLE-SORT(A)
1: **for** $i = 1$ to $A.length - 1$ **do**
2:      $noswap = TRUE$
3:      **for** $j = A.length - 1$ downto i **do**
4:          **if** $A[j + 1] < A[j]$ **then**
5:              $A[j] \leftrightarrow A[j + 1]$
6:              $noswap = FALSE$
7:      **if** $noswap$ **then** *break*

Outline
Basic Concepts
Simple Sorting Algorithms
Efficient Sorting Algorithms
Summary

Insertion Sort
Selection Sort
Bubble Sort

Example of Bubble Sort

8	2	4	9	3	6

Outline
Basic Concepts
Simple Sorting Algorithms
Efficient Sorting Algorithms
Summary

Insertion Sort
Selection Sort
Bubble Sort

Example of Bubble Sort

Outline
Basic Concepts
Simple Sorting Algorithms
Efficient Sorting Algorithms
Summary

Insertion Sort
Selection Sort
Bubble Sort

Example of Bubble Sort

Outline
Basic Concepts
Simple Sorting Algorithms
Efficient Sorting Algorithms
Summary

Insertion Sort
Selection Sort
Bubble Sort

Example of Bubble Sort

Outline
Basic Concepts
Simple Sorting Algorithms
Efficient Sorting Algorithms
Summary

Insertion Sort
Selection Sort
Bubble Sort

Example of Bubble Sort

Outline
Basic Concepts
Simple Sorting Algorithms
Efficient Sorting Algorithms
Summary

Insertion Sort
Selection Sort
Bubble Sort

Example of Bubble Sort

i	$j+1$				
8	2	3	4	9	6

Outline
Basic Concepts
Simple Sorting Algorithms
Efficient Sorting Algorithms
Summary

Insertion Sort
Selection Sort
Bubble Sort

Example of Bubble Sort

Outline
Basic Concepts
Simple Sorting Algorithms
Efficient Sorting Algorithms
Summary

Insertion Sort
Selection Sort
Bubble Sort

Example of Bubble Sort

Outline
Basic Concepts
Simple Sorting Algorithms
Efficient Sorting Algorithms
Summary

Insertion Sort
Selection Sort
Bubble Sort

Example of Bubble Sort

Outline
Basic Concepts
Simple Sorting Algorithms
Efficient Sorting Algorithms
Summary

Insertion Sort
Selection Sort
Bubble Sort

Example of Bubble Sort

Outline
Basic Concepts
Simple Sorting Algorithms
Efficient Sorting Algorithms
Summary

Insertion Sort
Selection Sort
Bubble Sort

Example of Bubble Sort

Outline
Basic Concepts
Simple Sorting Algorithms
Efficient Sorting Algorithms
Summary

Insertion Sort
Selection Sort
Bubble Sort

Example of Bubble Sort

Outline
Basic Concepts
Simple Sorting Algorithms
Efficient Sorting Algorithms
Summary

Insertion Sort
Selection Sort
Bubble Sort

Example of Bubble Sort

Outline
Basic Concepts
Simple Sorting Algorithms
Efficient Sorting Algorithms
Summary

Insertion Sort
Selection Sort
Bubble Sort

Example of Bubble Sort

Outline
Basic Concepts
Simple Sorting Algorithms
Efficient Sorting Algorithms
Summary

Insertion Sort
Selection Sort
Bubble Sort

Example of Bubble Sort

Outline
Basic Concepts
Simple Sorting Algorithms
Efficient Sorting Algorithms
Summary

Insertion Sort
Selection Sort
Bubble Sort

Example of Bubble Sort

2	3	4	6	8	9

Outline
Basic Concepts
Simple Sorting Algorithms
Efficient Sorting Algorithms
Summary

Insertion Sort
Selection Sort
Bubble Sort

Bubble Sort

- Time Complexity
  - Best: $O(n)$
  - Average: $O(n^2)$
  - Worst: $O(n^2)$

- Memory: 1

- Stable: Yes

Outline
Basic Concepts
Simple Sorting Algorithms
Efficient Sorting Algorithms
Summary

Insertion Sort
Selection Sort
Bubble Sort

Bubble Sort

- Time Complexity
  - Best: $O(n)$
  - Average: $O(n^2)$
  - Worst: $O(n^2)$
- Memory: 1
- Stable: Yes

BUBBLE-SORT(A)

1: **for** $i = 1$ to $A.length - 1$ **do**
2:     $noswap = TRUE$
3:     **for** $j = A.length - 1$ downto i **do**
4:         **if** $A[j+1] < A[j]$ **then**
5:             $A[j] \leftrightarrow A[j+1]$
6:             $noswap = FALSE$
7:     **if** $noswap$ **then** $break$

Outline
Basic Concepts
Simple Sorting Algorithms
Efficient Sorting Algorithms
Summary

Shellsort
Heapsort
Quicksort

Contents

Basic Concepts

Simple Sorting Algorithms

Efficient Sorting Algorithms
Shellsort
Heapsort
Quicksort

Summary

Outline
Basic Concepts
Simple Sorting Algorithms
Efficient Sorting Algorithms
Summary

Shellsort
Heapsort
Quicksort

Shellsort

General idea:

- Choose a descending gap sequence (*e.g., $D = [5, 3, 2, 1]$*).
- In each round, elements with the same gap d are in the same group.
- Apply INSERTION-SORT for each group.
- Reduce the amount of data migration that caused by insertion sort.

Outline
Basic Concepts
Simple Sorting Algorithms
Efficient Sorting Algorithms
Summary

Shellsort
Heapsort
Quicksort

Shellsort

SHELL-PASS(A, d)
1: **for** $i = d + 1$ to n **do**
2:     **if** $A[i] < A[i - d]$ **then**
3:         $key = A[i]$ //$A[i]$ is to inserted in the correct position
4:         $j = i - d$
5:         **while** $j > 0$ and $key < A[j]$ **do**
6:             $A[j + d] = A[j]$
7:             $j = j - d$
8:         $A[j + d] = key$

SHELLSORT(A, D)
1: **for** *increment* in D **do**
2:     SHELL-PASS($A, increment$)

Outline
Basic Concepts
Simple Sorting Algorithms
Efficient Sorting Algorithms
Summary

Shellsort
Heapsort
Quicksort

Example of Shellsort

21   25   49   <u>25</u>   16   08   27   04   55   48

Outline
Basic Concepts
Simple Sorting Algorithms
Efficient Sorting Algorithms
Summary

Shellsort
Heapsort
Quicksort

Example of Shellsort

21   25   49 $\underline{25}$ 16   08   27   04   55   48 $d = 3$

Outline
Basic Concepts
Simple Sorting Algorithms
Efficient Sorting Algorithms
Summary

Shellsort
Heapsort
Quicksort

Example of Shellsort

| 21 | 25 | 49 | <u>25</u> | 16 | 08 | 27 | 04 | 55 | 48 | $d = 3$ |

| 21 | 04 | 08 | <u>25</u> | 16 | 49 | 27 | 25 | 55 | 48 |

Outline
Basic Concepts
Simple Sorting Algorithms
Efficient Sorting Algorithms
Summary

Shellsort
Heapsort
Quicksort

Example of Shellsort

21	25	49	25	16	08	27	04	55	48	$d = 3$

21	04	08	25	16	49	27	25	55	48	$d = 2$

Outline
Basic Concepts
Simple Sorting Algorithms
Efficient Sorting Algorithms
Summary

Shellsort
Heapsort
Quicksort

Example of Shellsort

| 21 | 25 | 49 | <u>25</u> | 16 | 08 | 27 | 04 | 55 | 48 | $d = 3$ |

| 21 | 04 | 08 | <u>25</u> | 16 | 49 | 27 | 25 | 55 | 48 | $d = 2$ |

| 08 | 04 | 16 | <u>25</u> | 21 | 25 | 27 | 48 | 55 | 49 | |

Outline
Basic Concepts
Simple Sorting Algorithms
Efficient Sorting Algorithms
Summary

Shellsort
Heapsort
Quicksort

Example of Shellsort

| 21 | 25 | 49 | <u>25</u> | 16 | 08 | 27 | 04 | 55 | 48 | $d = 3$ |

| 21 | 04 | 08 | <u>25</u> | 16 | 49 | 27 | 25 | 55 | 48 | $d = 2$ |

| 08 | 04 | 16 | <u>25</u> | 21 | 25 | 27 | 48 | 55 | 49 | $d = 1$ |

Outline
Basic Concepts
Simple Sorting Algorithms
Efficient Sorting Algorithms
Summary

Shellsort
Heapsort
Quicksort

Example of Shellsort

| 21 | 25 | 49 | 25 | 16 | 08 | 27 | 04 | 55 | 48 | $d = 3$ |

| 21 | 04 | 08 | 25 | 16 | 49 | 27 | 25 | 55 | 48 | $d = 2$ |

| 08 | 04 | 16 | 25 | 21 | 25 | 27 | 48 | 55 | 49 | $d = 1$ |

| 04 | 08 | 16 | 21 | 25 | 25 | 27 | 48 | 49 | 55 |

Outline
Basic Concepts
Simple Sorting Algorithms
Efficient Sorting Algorithms
Summary

Shellsort
Heapsort
Quicksort

Shellsort

- Time Complexity
  - Best: depends on the gap sequence
  - Average: depends on the gap sequence
  - Worst: depends on the gap sequence, e.g., $O(n^{4/3})$, when the gap sequence is $4^k + 3 \cdot 2^{k-1} + 1$, prefixed with 1.
- Memory: 1
- Stable: No

Outline
Basic Concepts
Simple Sorting Algorithms
Efficient Sorting Algorithms
Summary

Shellsort
Heapsort
Quicksort

Shellsort

Shellsort

Why Shellsort typically performs faster?

- ▶ Insertion-Sorting small-sized array although costs $O(n^2)$ in the worst case, but it is similar to $O(n)$ in values.
- ▶ For large array, when we use a gap large enough (in the order of $O(n)$), each sub-array has a small size, thus efficient to sort.
- ▶ After enough iterations, when the gap is small, the majority part of the array is already sorted (thus the complexity is small again).

Outline
Basic Concepts
Simple Sorting Algorithms
Efficient Sorting Algorithms
Summary

Shellsort
Heapsort
Quicksort

Shellsort

How to select the gap sequence?

- $\lceil \frac{n}{2^k} \rceil$: time complexity $\Theta(n^2)$

- $2\lceil \frac{n}{2^{k+1}} \rceil + 1$: time complexity $\Theta(n^{\frac{3}{2}})$

- $2^k - 1$: time complexity $\Theta(n^{\frac{3}{2}})$

- $2^k + 1$ ($k \geq 0$): time complexity $\Theta(n^{\frac{3}{2}})$

- Successive numbers of the form $2^p 3^q$ for prime numbers p, q: time complexity $\Theta(n \log^2 n)$.

Outline
Basic Concepts
Simple Sorting Algorithms
Efficient Sorting Algorithms
Summary

Shellsort
Heapsort
Quicksort

Shellsort: the lowerbound on the time-complexity

The worst-case complexity of any version of Shellsort is of higher order: Plaxton, Poonen, and Suel showed that it grows at least as rapidly as $\Omega\left(n\left(\dfrac{\log n}{\log\log n}\right)^2\right)$.

Outline
Basic Concepts
Simple Sorting Algorithms
Efficient Sorting Algorithms
Summary

Shellsort
Heapsort
Quicksort

Contents

Outline
Basic Concepts
Simple Sorting Algorithms
Efficient Sorting Algorithms
Summary

Shellsort
Heapsort
Quicksort

Basic Concepts of Heap

Heap

A data structure which is an array object that can be viewed as a **nearly complete binary tree**.

The tree is completely filled on all levels except possibly the lowest, which is filled from the left up to a point.

Given the index i of a node, the indices of its parent $\text{PARENT}(i)$, left child $\text{LEFT}(i)$, and right child $\text{RIGHT}(i)$ can be computed simply:

$$
\begin{array}{lll}
\text{PARENT}(i) & \textbf{return} & \lfloor i/2 \rfloor \\
\text{LEFT}(i) & \textbf{return} & 2 * i \\
\text{RIGHT}(i) & \textbf{return} & 2 * i + 1
\end{array}
$$

Outline
Basic Concepts
Simple Sorting Algorithms
Efficient Sorting Algorithms
Summary

Shellsort
Heapsort
Quicksort

Example of Max-heap

max-heap: $A[\text{PARENT}(i)] \geq A[i]$, for all i other than the root.

Outline
Basic Concepts
Simple Sorting Algorithms
Efficient Sorting Algorithms
Summary

Shellsort
Heapsort
Quicksort
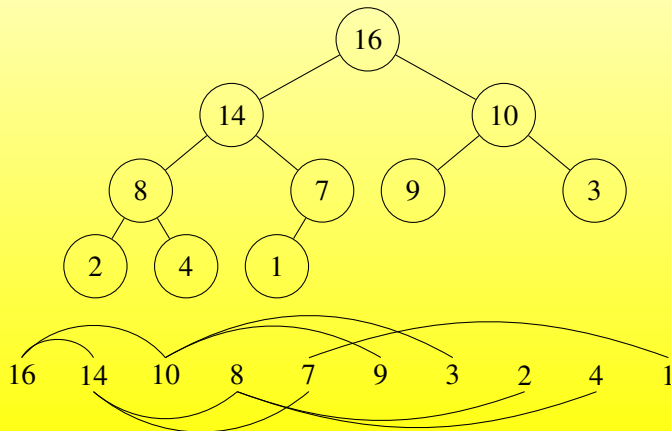
Maintaining the Heap Property

Assumption: sub-trees rooted at LEFT(i) & RIGHT(i) are max-heaps.

MAX-HEAPIFY(A, i) // Input an an array and an index i

1: $l = \text{LEFT}(i)$;
2: $r = \text{RIGHT}(i)$
3: **if** $l \leq A.heap\text{-}size$ and $A[l] > A[i]$ **then**
4:     $largest = l$
5: **else** $largest = i$
6: **if** $r \leq A.heap\text{-}size$ and $A[r] > A[largest]$ **then**
7:     $largest = r$
8: **if** $largest \neq i$ **then**
9:     $A[i] \leftrightarrow A[largest]$
10:     MAX-HEAPIFY($A, largest$)

Outline
Basic Concepts
Simple Sorting Algorithms
Efficient Sorting Algorithms
Summary

Shellsort
Heapsort
Quicksort

Maintaining the Heap Property

Example: MAX-HEAPIFY(A, 2)

Outline
Basic Concepts
Simple Sorting Algorithms
Efficient Sorting Algorithms
Summary

Shellsort
Heapsort
Quicksort

Maintaining the Heap Property

Example: MAX-HEAPIFY(A, 2)

Outline
Basic Concepts
Simple Sorting Algorithms
Efficient Sorting Algorithms
Summary

Shellsort
Heapsort
Quicksort

Maintaining the Heap Property

Example: MAX-HEAPIFY(A, 2)

Outline
Basic Concepts
Simple Sorting Algorithms
Efficient Sorting Algorithms
Summary

Shellsort
Heapsort
Quicksort

Maintaining the Heap Property

Example: MAX-HEAPIFY(A, 2)

Outline
Basic Concepts
Simple Sorting Algorithms
Efficient Sorting Algorithms
Summary

Shellsort
Heapsort
Quicksort

Maintaining the Heap Property

Example: MAX-HEAPIFY(A, 2)

Outline
Basic Concepts
Simple Sorting Algorithms
Efficient Sorting Algorithms
Summary

Shellsort
Heapsort
Quicksort

Building a Heap

Fact: with the array representation of an n-element heap, the leaves are the nodes indexed from $\lfloor A.length/2 \rfloor + 1$ to n, and each leaf is a 1-element max-heap to begin with.

Outline
Basic Concepts
Simple Sorting Algorithms
Efficient Sorting Algorithms
Summary

Shellsort
Heapsort
Quicksort
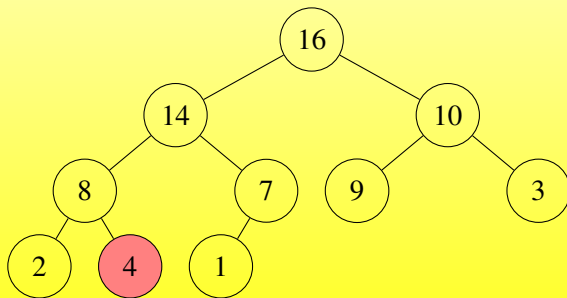
Building a Heap

Fact: with the array representation of an n-element heap, the leaves are the nodes indexed from $\lfloor A.length/2 \rfloor + 1$ to n, and each leaf is a 1-element max-heap to begin with.

BUILD-MAX-HEAP(A)
1: $A.heap\text{-}size = A.length$
2: **for** $i = \lfloor A.length/2 \rfloor$ downto 1 **do**
3:     MAX-HEAPIFY(A, i)

Outline
Basic Concepts
Simple Sorting Algorithms
Efficient Sorting Algorithms
Summary

Shellsort
Heapsort
Quicksort

Building a Heap

$A.length = 10$

| 4 | 1 | 3 | 2 | 16 | 9 | 10 | 14 | 8 | 7 |

Outline
Basic Concepts
Simple Sorting Algorithms
Efficient Sorting Algorithms
Summary

Shellsort
Heapsort
Quicksort

Building a Heap

$A.length = 10$

| 4 | 1 | 3 | 2 | 16 | 9 | 10 | 14 | 8 | 7 |

Outline
Basic Concepts
Simple Sorting Algorithms
Efficient Sorting Algorithms
Summary

Shellsort
Heapsort
Quicksort

Building a Heap

$A.length = 10$

| 4 | 1 | 3 | 2 | 16 | 9 | 10 | 14 | 8 | 7 |

Outline
Basic Concepts
Simple Sorting Algorithms
Efficient Sorting Algorithms
Summary

Shellsort
Heapsort
Quicksort

Building a Heap

$A.length = 10$

| 4 | 1 | 3 | 2 | 16 | 9 | 10 | 14 | 8 | 7 |

Outline
Basic Concepts
Simple Sorting Algorithms
Efficient Sorting Algorithms
Summary

Shellsort
Heapsort
Quicksort

Building a Heap

$A.length = 10$

| 4 | 1 | 3 | 14 | 16 | 9 | 10 | 2 | 8 | 7 |

Outline
Basic Concepts
Simple Sorting Algorithms
Efficient Sorting Algorithms
Summary

Shellsort
Heapsort
Quicksort

Building a Heap

$A.length = 10$

4   1   3   14   16   9   10   2   8   7

Outline
Basic Concepts
Simple Sorting Algorithms
Efficient Sorting Algorithms
Summary

Shellsort
Heapsort
Quicksort

Building a Heap

$A.length = 10$

| 4 | 1 | 10 | 14 | 16 | 9 | 3 | 2 | 8 | 7 |

Outline
Basic Concepts
Simple Sorting Algorithms
Efficient Sorting Algorithms
Summary

Shellsort
Heapsort
Quicksort

Building a Heap

$A.length = 10$

| 4 | 1 | 10 | 14 | 16 | 9 | 3 | 2 | 8 | 7 |

Outline
Basic Concepts
Simple Sorting Algorithms
Efficient Sorting Algorithms
Summary

Shellsort
Heapsort
Quicksort

Building a Heap

$A.length = 10$

| 4 | 16 | 10 | 14 | 7 | 9 | 3 | 2 | 8 | 1 |

Outline
Basic Concepts
Simple Sorting Algorithms
Efficient Sorting Algorithms
Summary

Shellsort
Heapsort
Quicksort

Building a Heap

$A.length = 10$

| 4 | 16 | 10 | 14 | 7 | 9 | 3 | 2 | 8 | 1 |

Outline
Basic Concepts
Simple Sorting Algorithms
Efficient Sorting Algorithms
Summary

Shellsort
Heapsort
Quicksort

Building a Heap

$A.length = 10$

| 16 | 14 | 10 | 8 | 7 | 9 | 3 | 2 | 4 | 1 |
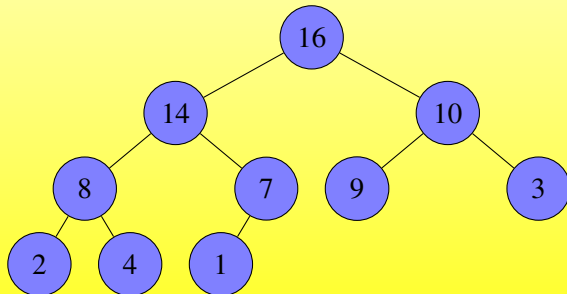
Outline
Basic Concepts
Simple Sorting Algorithms
Efficient Sorting Algorithms
Summary

Shellsort
Heapsort
Quicksort

The Heapsort Algorithm

General idea: Same as selection sort, maintain the minimum (maximum) element by using heap.

MAX-HEAP: $A[1]$ always stores the largest number.

Outline
Basic Concepts
Simple Sorting Algorithms
Efficient Sorting Algorithms
Summary

Shellsort
Heapsort
Quicksort

The Heapsort Algorithm

General idea: Same as selection sort, maintain the minimum (maximum) element by using heap.

MAX-HEAP: $A[1]$ always stores the largest number.

HEAPSORT(A)
1: BUILD-MAX-HEAP(A)
2: **for** $i = A.length$ downto 2 **do**
3:     $A[1] \leftrightarrow A[i]$
4:     $A.heap\text{-}size = A.heap\text{-}size - 1$
5:     MAX-HEAPIFY($A, 1$)

Outline
Basic Concepts
Simple Sorting Algorithms
Efficient Sorting Algorithms
Summary

Shellsort
Heapsort
Quicksort

Example of Heapsort

Outline
Basic Concepts
Simple Sorting Algorithms
Efficient Sorting Algorithms
Summary

Shellsort
Heapsort
Quicksort

Example of Heapsort

Outline
Basic Concepts
Simple Sorting Algorithms
Efficient Sorting Algorithms
Summary

Shellsort
Heapsort
Quicksort

Example of Heapsort

Outline
Basic Concepts
Simple Sorting Algorithms
Efficient Sorting Algorithms
Summary

Shellsort
Heapsort
Quicksort

Example of Heapsort

Outline
Basic Concepts
Simple Sorting Algorithms
Efficient Sorting Algorithms
Summary

Shellsort
Heapsort
Quicksort

Example of Heapsort

Outline
Basic Concepts
Simple Sorting Algorithms
Efficient Sorting Algorithms
Summary

Shellsort
Heapsort
Quicksort

Example of Heapsort

Outline
Basic Concepts
Simple Sorting Algorithms
Efficient Sorting Algorithms
Summary

Shellsort
Heapsort
Quicksort

Example of Heapsort

Outline
Basic Concepts
Simple Sorting Algorithms
Efficient Sorting Algorithms
Summary

Shellsort
Heapsort
Quicksort

Example of Heapsort

Outline
Basic Concepts
Simple Sorting Algorithms
Efficient Sorting Algorithms
Summary

Shellsort
Heapsort
Quicksort

Example of Heapsort

Outline
Basic Concepts
Simple Sorting Algorithms
Efficient Sorting Algorithms
Summary

Shellsort
Heapsort
Quicksort

Example of Heapsort

Outline
Basic Concepts
Simple Sorting Algorithms
Efficient Sorting Algorithms
Summary

Shellsort
Heapsort
Quicksort

Example of Heapsort

Outline
Basic Concepts
Simple Sorting Algorithms
Efficient Sorting Algorithms
Summary

Shellsort
Heapsort
Quicksort

Heapsort

- Time Complexity
  - MAX-HEAPIFY: $O(\log n)$ – Why?
  - BUILD-MAX-HEAP: $O(n)$ – Why?
  - Best: $O(n \log n)$
  - Average: $O(n \log n)$
  - Worst: $O(n \log n)$

- Memory: 1

- Stable: No

Outline
Basic Concepts
Simple Sorting Algorithms
Efficient Sorting Algorithms
Summary

Shellsort
Heapsort
Quicksort

Priority Queues

A *priority queue* is a data structure for maintaining a set S of elements, each with an associated value called a *key*. A *max-priority queue* supports the following operations:

- INSERT(S, x) inserts the element x into the set S, which is equivalent to the operation $S = S \cup \{x\}$.

- MAXIMUM(S) returns the element of S with the largest key.

- EXTRACT-MAX(S) removes and returns the element of S with the largest key.

- INCREASE-KEY(S, x, k) increases the value of element x's key to the new value k, which is assumed to be at least as large as x's current key value.

Outline
Basic Concepts
Simple Sorting Algorithms
Efficient Sorting Algorithms
Summary

Shellsort
Heapsort
Quicksort

Priority Queues

HEAP-MAXIMUM(A)
1: **return** $A[1]$

HEAP-EXTRACT-MAX(A)
1: **if** $A.heap\text{-}size < 1$ **then**
2:     **error** "heap underflow"
3: $max = A[1]$
4: $A[1] = A[A.heap\text{-}size]$
5: $A.heap\text{-}size = A.heap\text{-}size - 1$
6: MAX-HEAPIFY($A, 1$)
7: **return** max

Outline
Basic Concepts
Simple Sorting Algorithms
Efficient Sorting Algorithms
Summary

Shellsort
Heapsort
Quicksort

Priority Queues

HEAP-INCREASE-KEY(A, i, key)

1: **if** $key < A[i]$ **then**
2:     **error** "new key is smaller than current key"
3: $A[i] = key$
4: **while** $i > 1$ and $A[\text{PARENT}(i)] < A[i]$ **do**
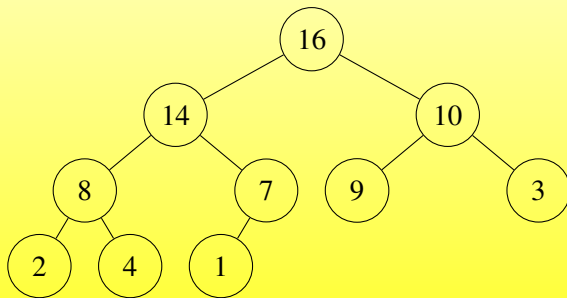5: $A[i] \leftrightarrow A[\text{PARENT}(i)]$
6:     $i = \text{PARENT}(i)$

Outline
Basic Concepts
Simple Sorting Algorithms
Efficient Sorting Algorithms
Summary

Shellsort
Heapsort
Quicksort

Priority Queues

MAX-HEAP-INSERT(A, key)
1: $A.heap\text{-}size = A.heap\text{-}size + 1$
2: $A[A.heap\text{-}size] = -\infty$
3: HEAP-INCREASE-KEY($A, A.heap\text{-}size, key$)

Outline
Basic Concepts
Simple Sorting Algorithms
Efficient Sorting Algorithms
Summary

Shellsort
Heapsort
Quicksort

Example of HEAP-INCREASE-KEY

Outline
Basic Concepts
Simple Sorting Algorithms
Efficient Sorting Algorithms
Summary

Shellsort
Heapsort
Quicksort

Example of HEAP-INCREASE-KEY

Outline
Basic Concepts
Simple Sorting Algorithms
Efficient Sorting Algorithms
Summary

Shellsort
Heapsort
Quicksort

Example of HEAP-INCREASE-KEY

Outline
Basic Concepts
Simple Sorting Algorithms
Efficient Sorting Algorithms
Summary

Shellsort
Heapsort
Quicksort

Example of HEAP-INCREASE-KEY

Outline
Basic Concepts
Simple Sorting Algorithms
Efficient Sorting Algorithms
Summary

Shellsort
Heapsort
Quicksort

Example of HEAP-INCREASE-KEY

Outline
Basic Concepts
Simple Sorting Algorithms
Efficient Sorting Algorithms
Summary

Shellsort
Heapsort
Quicksort

Contents

Basic Concepts

Simple Sorting Algorithms

Efficient Sorting Algorithms
  Shellsort
  Heapsort
  Quicksort

Summary

Outline
Basic Concepts
Simple Sorting Algorithms
Efficient Sorting Algorithms
Summary

Shellsort
Heapsort
Quicksort

Quicksort

General idea:

- ▶ **Arbitrarily choose** an element x in the unsorted set for comparison.
- ▶ **Divide** the unsorted elements into two parts: $\leq x$ and $> x$.
- ▶ **Recursively** use QUICKSORT for the above two parts.

Outline
Basic Concepts
Simple Sorting Algorithms
Efficient Sorting Algorithms
Summary

Shellsort
Heapsort
Quicksort

Quicksort

General idea:

- ▶ **Arbitrarily choose** an element x in the unsorted set for comparison.
- ▶ **Divide** the unsorted elements into two parts: $\leq x$ and $> x$.
- ▶ **Recursively** use QUICKSORT for the above two parts.
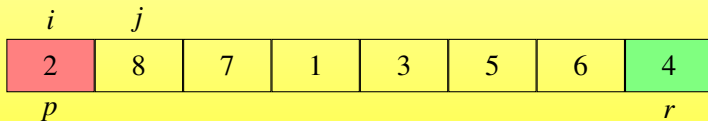
QUICKSORT(A, p, r)
1: **if** $p < r$ **then**
2:     $q = $ PARTITION(A, p, r)
3:     QUICKSORT($A, p, q - 1$)
4:     QUICKSORT($A, q + 1, r$)

Outline
Basic Concepts
Simple Sorting Algorithms
Efficient Sorting Algorithms
Summary

Shellsort
Heapsort
Quicksort

Partition
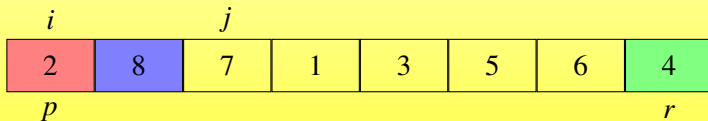
PARTITION(A, p, r)
1: $x = A[r]$ // pivot element
2: $i = p - 1$
3: **for** $j = p$ to $r - 1$ **do**
4:     **if** $A[j] \leq x$ **then**
5:         $i = i + 1$
6:         $A[i] \leftrightarrow A[j]$
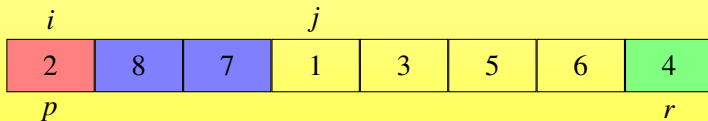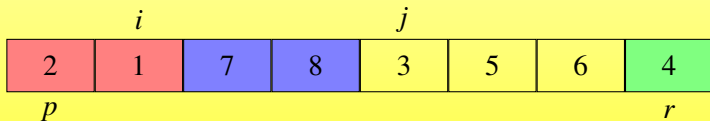7: $A[i + 1] \leftrightarrow A[r]$
8: **return** $i + 1$

Outline
Basic Concepts
Simple Sorting Algorithms
Efficient Sorting Algorithms
Summary

Shellsort
Heapsort
Quicksort

Example of Partition

Outline
Basic Concepts
Simple Sorting Algorithms
Efficient Sorting Algorithms
Summary

Shellsort
Heapsort
Quicksort

Example of Partition

Outline
Basic Concepts
Simple Sorting Algorithms
Efficient Sorting Algorithms
Summary

Shellsort
Heapsort
Quicksort

Example of Partition

Outline
Basic Concepts
Simple Sorting Algorithms
Efficient Sorting Algorithms
Summary

Shellsort
Heapsort
Quicksort

Example of Partition

Outline
Basic Concepts
Simple Sorting Algorithms
Efficient Sorting Algorithms
Summary

Shellsort
Heapsort
Quicksort

Example of Partition

Outline
Basic Concepts
Simple Sorting Algorithms
Efficient Sorting Algorithms
Summary

Shellsort
Heapsort
Quicksort

Example of Partition

Outline
Basic Concepts
Simple Sorting Algorithms
Efficient Sorting Algorithms
Summary

Shellsort
Heapsort
Quicksort

Example of Partition

Outline
Basic Concepts
Simple Sorting Algorithms
Efficient Sorting Algorithms
Summary

Shellsort
Heapsort
Quicksort

Example of Partition

Outline
Basic Concepts
Simple Sorting Algorithms
Efficient Sorting Algorithms
Summary

Shellsort
Heapsort
Quicksort

Example of Partition

Outline
Basic Concepts
Simple Sorting Algorithms
Efficient Sorting Algorithms
Summary

Shellsort
Heapsort
Quicksort

Performance of Quicksort

Worst-case partitioning

The worst-case behavior for quicksort occurs when the partitioning routine produces one subproblem with $n-1$ **elements and one with** 0 **elements**. The partitioning costs $\Theta(n)$ time. the recurrence for the running time is

$$
\begin{aligned}
T(n) &= T(n-1) + T(0) + \Theta(n) \\
&= T(n-1) + \Theta(n) \\
&= \Theta(n^2).
\end{aligned}
$$

Outline
Basic Concepts
Simple Sorting Algorithms
Efficient Sorting Algorithms
Summary

Shellsort
Heapsort
Quicksort

Performance of Quicksort

Best-case partitioning

In the most even possible split, PARTITION produces two subproblems, each of size no more than $n/2$, since **one is of size $\lfloor n/2 \rfloor$ and one of size $\lceil n/2 \rceil - 1$**. In this case, quicksort runs much faster. The recurrence for the running time is then

$$T(n) = 2T(n/2) + \Theta(n)$$
$$= \Theta(n \lg n).$$

Outline
Basic Concepts
Simple Sorting Algorithms
Efficient Sorting Algorithms
Summary

Shellsort
Heapsort
Quicksort

Performance of Quicksort

Balanced partitioning

What if the split is always $\frac{1}{10} : \frac{9}{10}$? The recurrence for the running time is

$$T(n) = T(\frac{1}{10}n) + T(\frac{9}{10}n) + \Theta(n)$$
$$= \Theta(n \lg n).$$

Outline
Basic Concepts
Simple Sorting Algorithms
Efficient Sorting Algorithms
Summary

Shellsort
Heapsort
Quicksort

A Randomized Version of Quicksort

RANDOMIZED-PARTITION(A, p, r)

1: $i =$ RANDOM(p, r)
2: $A[r] \leftrightarrow A[i]$
3: **return** PARTITION(A, p, r)

RANDOMIZED-QUICKSORT(A, p, r)

1: **if** $p < r$ **then**
2:     $q =$ RANDOMIZED-PARTITION(A, p, r)
3:     RANDOMIZED-QUICKSORT($A, p, q - 1$)
4:     RANDOMIZED-QUICKSORT($A, q + 1, r$)

Outline
Basic Concepts
Simple Sorting Algorithms
Efficient Sorting Algorithms
Summary

Shellsort
Heapsort
Quicksort

Analysis of Quicksort

Worst-case analysis

     We saw that a worst-case split at every level of recursion in quicksort produces a $\Theta(n^2)$ running time, which, intuitively, is the worst-case running time of the algorithm.

     Using the **substitution method** (see Section 4.3), we can show that the running time of quicksort is $O(n^2)$.

Outline
Basic Concepts
Simple Sorting Algorithms
Efficient Sorting Algorithms
Summary

Shellsort
Heapsort
Quicksort

Analysis of Quicksort

Let $T(n)$ be the worst-case time for the procedure QUICKSORT on an input of size n. We have

$$\begin{aligned}
T(n) &= \max_{0 \leq q \leq n-1} (T(q) + T(n-q-1)) + \Theta(n) \\
&\leq \max_{0 \leq q \leq n-1} (cq^2 + c(n-q-1)^2 + \Theta(n)) \\
&= c \cdot \max_{0 \leq q \leq n-1} (q^2 + (n-q-1)^2 + \Theta(n)) \\
&\leq cn^2 - c(2n-1) + \Theta(n) \leq cn^2.
\end{aligned}$$

Outline
Basic Concepts
Simple Sorting Algorithms
Efficient Sorting Algorithms
Summary

Shellsort
Heapsort
Quicksort

Analysis of Quicksort

Running time and comparisons

Rename the elements of the array A as $z_1, z_2, ..., z_n$, with z_i being the ith smallest element (assuming distinct elements).
$Z_{ij} = \{z_i, z_{i+1}, ..., z_j\}$ to be the set of elements between z_i and z_j.

We define

$$X_{ij} = I\{z_i \text{ is compared to } z_j\}.$$

Since each pair is compared at most once, we can easily characterize **the total number of comparisons** performed by the algorithm:

$$X = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} X_{ij}.$$

Outline
Basic Concepts
Simple Sorting Algorithms
Efficient Sorting Algorithms
Summary

Shellsort
Heapsort
Quicksort

Analysis of Quicksort

$$E[X] = E\left[\sum_{i=1}^{n-1}\sum_{j=i+1}^{n} X_{ij}\right] = \sum_{i=1}^{n-1}\sum_{j=i+1}^{n} E[X_{ij}]$$

$$= \sum_{i=1}^{n-1}\sum_{j=i+1}^{n} Pr\{z_i \text{ is compared to } z_j\}$$

Outline
Basic Concepts
Simple Sorting Algorithms
Efficient Sorting Algorithms
Summary

Shellsort
Heapsort
Quicksort

Analysis of Quicksort

$$E[X] = E\left[\sum_{i=1}^{n-1}\sum_{j=i+1}^{n} X_{ij}\right] = \sum_{i=1}^{n-1}\sum_{j=i+1}^{n} E[X_{ij}]$$

$$= \sum_{i=1}^{n-1}\sum_{j=i+1}^{n} Pr\{z_i \text{ is compared to } z_j\}$$

$$= \sum_{i=1}^{n-1}\sum_{j=i+1}^{n} Pr\{z_i \text{ or } z_j \text{ is first pivot chosen from } Z_{ij}\}$$

$$= \sum_{i=1}^{n-1}\sum_{j=i+1}^{n} \frac{2}{j-i+1} = \sum_{i=1}^{n-1}\sum_{k=1}^{n-i} \frac{2}{k+1}$$

$$< \sum_{i=1}^{n-1}\sum_{k=1}^{n} \frac{2}{k} = \sum_{i=1}^{n-1} O(\lg n) = O(n\lg n).$$

Outline
Basic Concepts
Simple Sorting Algorithms
Efficient Sorting Algorithms
Summary

Shellsort
Heapsort
Quicksort

Quicksort

- Time Complexity
  - Best: $O(n \log n)$
  - Average: $O(n \log n)$
  - Worst: $O(n^2)$

- Memory: $O(\log n)$ on average, worst case space complexity is $O(n)$

- Stable: stable versions exist

Summary

Name	Average	Worst	Stable	Method
Insertion Sort	$O(n^2)$	$O(n^2)$	Yes	Insertion
Selection Sort	$O(n^2)$	$O(n^2)$	No	Selection
Bubble Sort	$O(n^2)$	$O(n^2)$	Yes	Exchanging
Merge sort	$O(n\log n)$	$O(n\log n)$	Yes	Merging
Shellsort	$(*)$	$O(n^{4/3})$ $(*)$	No	Insertion
Heapsort	$O(n\log n)$	$O(n\log n)$	No	Selection
Quicksort	$O(n\log n)$	$O(n^2)$	Exist	Partitioning

*The time complexity of shellsort depends on the selected gap sequence.

A sorting algorithm animation website:

https://www.toptal.com/developers/sorting-algorithms