CS6161: Design and Analysis of Algorithms (Fall 2020)

Introduction

Instructor: Haifeng Xu

Outline

- A Brief Intro to Algorithms
- Administrivia
- An Example

What is an Algorithm?

Interpretation: Given any input problem instance, an algorithm is a finite procedure that consists of executable computer operations and outputs a correct answer for the problem

“ An algorithm is a finite, definite, effective procedure, with some input and some output. ” — Donald Knuth

A Simple Illustrative Example

<u>Interpretation</u>: Given any input problem instance, an algorithm is a finite procedure that consists of executable computer operations and outputs a correct answer for the problem

Problem: Calculate $3\times3\times3\times3\times3$

A Simple Illustrative Example

Interpretation: Given any input problem instance, an algorithm is a finite procedure that consists of executable computer operations and outputs a correct answer for the problem

Problem: Calculate $3 \times 3 \times 3 \times 3 \times 3$

- Computer can only calculate basic multiplication: $a \times b$

- In fact, computer can only do addition
- Operation $a \times b$ is also implemented by an algorithm, and stored as a basic module for other tasks

A Simple Illustrative Example

Interpretation: Given any input problem instance, an algorithm is a finite procedure that consists of executable computer operations and outputs a correct answer for the problem

Problem: Calculate $3 \times 3 \times 3 \times 3 \times 3$

- Computer can only calculate basic multiplication: $a \times b$
- Calculate $x_2 = 3 \times 3$, then $x_3 = x_2 \times 3$, …, $x_5 = x_4 \times 3$, which is 3^5
- Can be more cleanly written as follows

1) $x_1 = 3$;
2) for $i = 2$ to 5, $x_i = x_{i-1} \times 3$;
3) Output x_5

This is called pseudo-code

A Simple Illustrative Example

<u>Interpretation</u>: Given any input problem instance, an algorithm is a finite procedure that consists of executable computer operations and outputs a correct answer for the problem

Problem: Calculate $3 \times 3 \times 3 \times 3 \times 3$

- Computer can only calculate basic multiplication: $a \times b$
- Calculate $x_2 = 3 \times 3$, then $x_3 = x_2 \times 3$, …, $x_5 = x_4 \times 3$, which is 3^5
- Can be more cleanly written as follows

1) $x_1 = 3$;
2) for $i = 2$ to 5, $x_i = x_{i-1} \times 3$;
3) Output x_5

This course only requires writing pseudo-code

- Implementing it in computer understandable language (i.e., coding) is not required, but encouraged as your after-class exercise

Desirable Properties of Algorithms

- Correctness: The output is guaranteed to be correct
- (Running) Time efficiency: The algorithm runs fast, i.e., does not use "too many" operations
- Space efficiency*: The algorithm does not use "too many" computer storage space
- Generality*: How general is the problem instance that can be solved by your algorithm?

An algorithm that can only calculate $3\times3\times3\times3\times3$ is likely not useful – at least should be able to calculate a^b

Desirable Properties of Algorithms

- Correctness: The output is guaranteed to be correct
- (Running) Time efficiency: The algorithm runs fast, i.e., does not use “too many” operations
- Space efficiency*: The algorithm does not use “too many” computer storage space
- Generality*: How general is the problem instance that can be solved by your algorithm?
- . . . .

Desirable Properties of Algorithms

- Correctness: The output is guaranteed to be correct
- (Running) Time efficiency: The algorithm runs fast, i.e., does not use "too many" operations
- Space efficiency*: The algorithm does not use "too many" computer storage space
- Generality*: How general is the problem instance that can be solved by your algorithm?

This course aims to, given a problem as input,

(1) find algorithmic procedure to solve the problem (algorithm design)

(2) prove it has the above nice properties (algorithm analysis)

Types of Algorithmic Problems

- Decisions problem
  - Whether there is a path from your home to UVA?
  - Only need to answer *yes* or *no*
- Search problem
  - Find a path from your home to UVA
  - Need to output a feasible path (or output infeasible)
- Optimization problem
  - Find the fastest path from your home to UVA
  - Feasible + fastest
- Strategic games, puzzles, and interactions
  - Find the fastest path, taking into account that many other people may take the same path and cause high traffic delay
  - Have conflicts with others → game theory

How to Prove Correctness?

- We will introduce various techniques during the lectures: induction, contradiction argument, etc.

How to Measure (Time) Efficiency?

- Use running time $T(n) =$ # of basic operations as a fnc of input size n
  - Basic operations (addition, multiplication, comparisons, accessing an array, etc.) are assumed to take constant time
  - The larger $T(n)$ is, the worse it is

- When the input size n is large, $2n$ and n does not differ much but n^2 and n differ a lot → big O notation

How to Measure (Time) Efficiency?

- Big-O notation ("O" means "order"): ignore constant coefficients
  - E.g., $5n = O(n)$, $\frac{n}{2} = O(n)$, $3n^2 = O(n^2)$

Q: $n^2 + 100n + 1000 = O(??)$ Ans: $O(n^2)$

- $T(n) = O\big(f(n)\big) \Leftrightarrow \lim_{n\to\infty} \frac{T(n)}{f(n)} \leq C$ for some constant C independent of n

Q: Is $3n^2 = O(n^3)$?

- Yes, but if you proved $T(n) = 3n^2$ running time, you do not want to say it is a $O(n^3)$ time algorithm….

Will have more practice in HW 1

How to Measure (Time) Efficiency?

- Big-O notation ("O" means "order"): ignore constant coefficients
  - E.g., $5n = O(n), \quad \frac{n}{2} = O(n), \quad 3n^2 = O(n^2)$

- Standard in efficiency analysis, why?
  - Being 10 times slower may be remedied by a faster computer, but being 1000 times slower cannot
  - Makes analysis easier since difficult to quantify constant coefficients

This Course in a Nutshell

Basic algorithm design techniques, their applications and analysis

- Divide and conquer
- Greedy and local search
- Dynamic programming
- Optimization-based design techniques
- Intractability, NP-completeness and reductions
- Advanced topics: approximation algorithms, randomized algorithms, mechanism design (algorithm design in multi-agent decision making setup)

A more intense version of CS4102; about 50% new topics including local search, linear programming, approximation algorithms, etc.

Course Goal

- Formulate a problem as an algorithmic question
- Use the right design techniques to solve the problem
- Rigorously analyze the correctness and efficiency of your algorithm
- Understand limitations of algorithms, i.e., what problem cannot be “efficiently” solved by algorithms

Outline

- A Brief Intro to Algorithms
- Administrivia
- An Example

Basic Information

- Course time: Tuesday/Thursday, 5:00 pm – 6:15 pm
- Synchronous virtual lecture
- Instructor: Haifeng Xu
  - Email: hx4ad@virginia.edu
  - Office Hour: TuThu 6:15 – 7:15 pm (rightly after lectures)
- TAs
  - **Jibang Wu**: office hour Mon/Wed 4 – 5 pm
  - **Fan Yao**: office hour Tue/Thur 10 – 11 am
- Depending on demand, can add more office hours (let us know!)

Course Materials

- Couse website: https://www.haifeng-xu.com/cs6161fa20/
- Textbooks (recommended but not required)
  - **Algorithm Design** by Jon Kleinberg, Eva Tardos, main textbook
  - **Introduction to Algorithms** by Cormen, Leiserson, Rivest and Stein
- All lectures will be recorded and uploaded to Collab
  - Will try to upload slides to Collab before lectures
- Attending live lectures is not required but encouraged
  - Easier for you to focus

Course Communication

- Collab for course managements, including announcements, homework, slides, videos, etc.
- Piazza for discussions/questions/re-grading
  - https://piazza.com/virginia/fall2020/cs6161
  - Please definitely make sure to join!

Prerequisites

- Mathematically mature: be comfortable with proofs
- Asymptotic notation (Big-O, Omega, Theta)
- Basic data structures
  - Arrays, linked lists, trees, heaps (priority queues), graphs.
- Basic graph algorithms
  - Connected components, BFS, DFS.
- Discrete mathematics
  - Evaluating sums and simple recurrences.
- Basic probabilities
  - random variables, distributions, expectations, variance, etc.

Doing reasonably well in CS4102 or its equivalence should prepare you well for this course

Requirements and Grading

Grading consists of two components:

- 6 problem sets, 50% of grade
  - Only (the highest) 5 of them will count towards your grade
  - Written and proof-based, no coding tasks
  - Will likely be challenging
  - Discussion allowed, even encouraged, but must write up solutions independently
  - Must submit PDF doc, preferably in Latex; scanned hand-written solutions will not be accepted
  - 5 late days allowed; each HW can use at most 2 late days
- Two mid-term exams, 25% each
  - The exams cover the first and second half of the course, respectively
  - Each exam is designed for 2~3 hours, but you will be given 24 hours to complete it

Important Notes

- Due to pandemic, we have tried our best to make the course as flexible as we can
  - Please let us know if you have any further suggestions
- Final letter grades will be based on your standing in the class (i.e., your ranking) while not the absolute grades
  - HW/Exams might be challenging, but should not affect your letter grades
  - Letter grades will be generous

Outline

- A Brief Intro to Algorithms
- Administrivia
- An Example

Sorting: what is the problem?

- Input: sequence $\langle a_1, a_2, \cdots, a_n \rangle$ of real numbers
- Output: permutation $\langle a'_1, a'_2, \cdots, a'_n \rangle$ such that

$$a'_1 \leq a'_2 \leq \cdots \leq a'_n$$

Called a problem description, or simply, a problem

Example:

Input: 8 2 4 9 3 6 ← Input instance

Output: 2 3 4 6 8 9

- An algorithm is correct if it outputs the correct answer for any valid input instance

What Is Your First Thought?

➢ Natural idea: if first k numbers already correctly sorted, adding the $(k+1)'th$ number is easy

- Insertion sort

8   2   4   9   3   6

What Is Your First Thought?

- Natural idea: if first k numbers already correctly sorted, adding the $(k+1)'th$ number is easy
  - Insertion sort

8 2 4 9 3 6

2 8 4 9 3 6

What Is Your First Thought?

➢ Natural idea: if first k numbers already correctly sorted, adding the $(k+1)'th$ number is easy

- Insertion sort

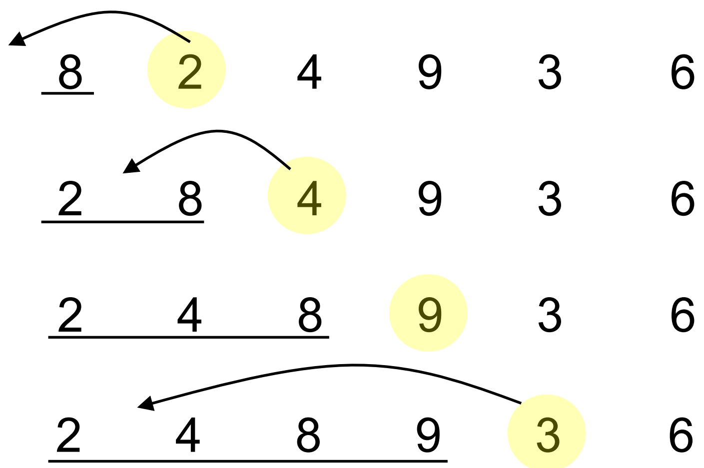
8 2 4 9 3 6

2 8 4 9 3 6

2 4 8 9 3 6

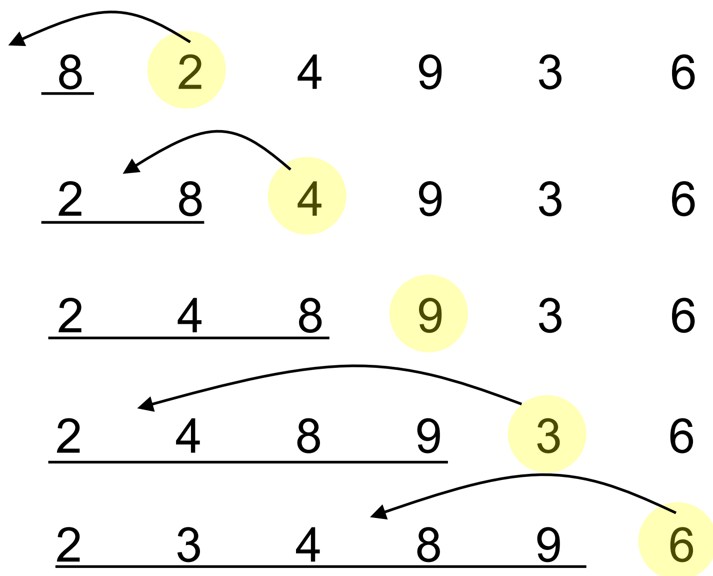
What Is Your First Thought?

- Natural idea: if first k numbers already correctly sorted, adding the $(k+1)'th$ number is easy
  - Insertion sort



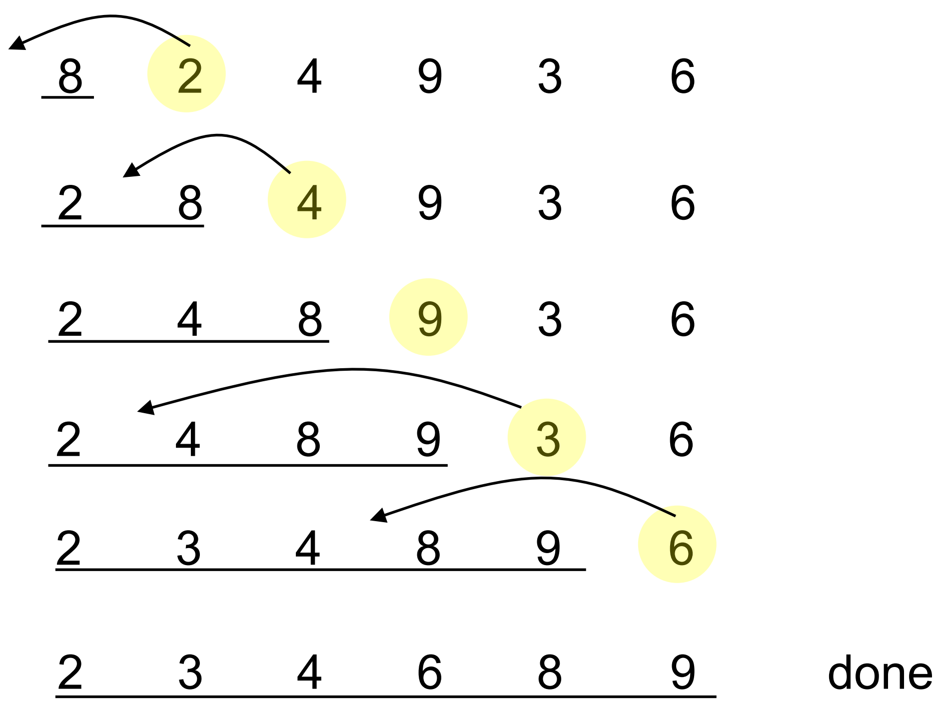
What Is Your First Thought?

- Natural idea: if first k numbers already correctly sorted, adding the $(k+1)'th$ number is easy
  - Insertion sort

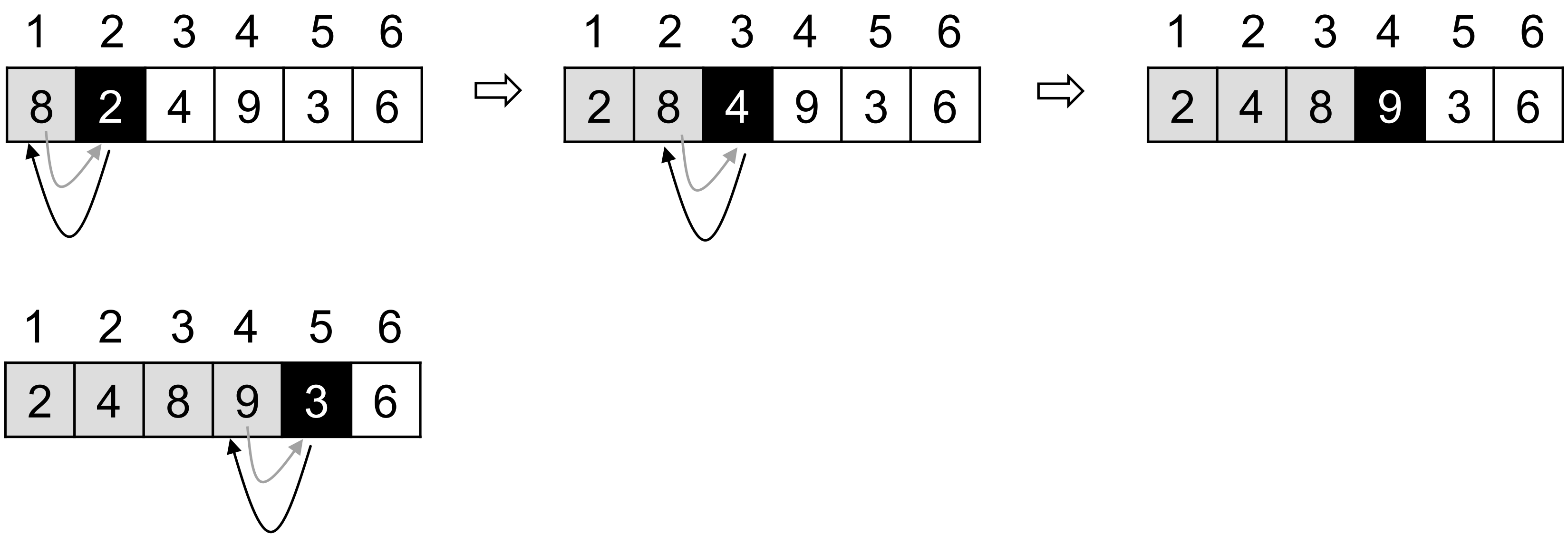


What Is Your First Thought?

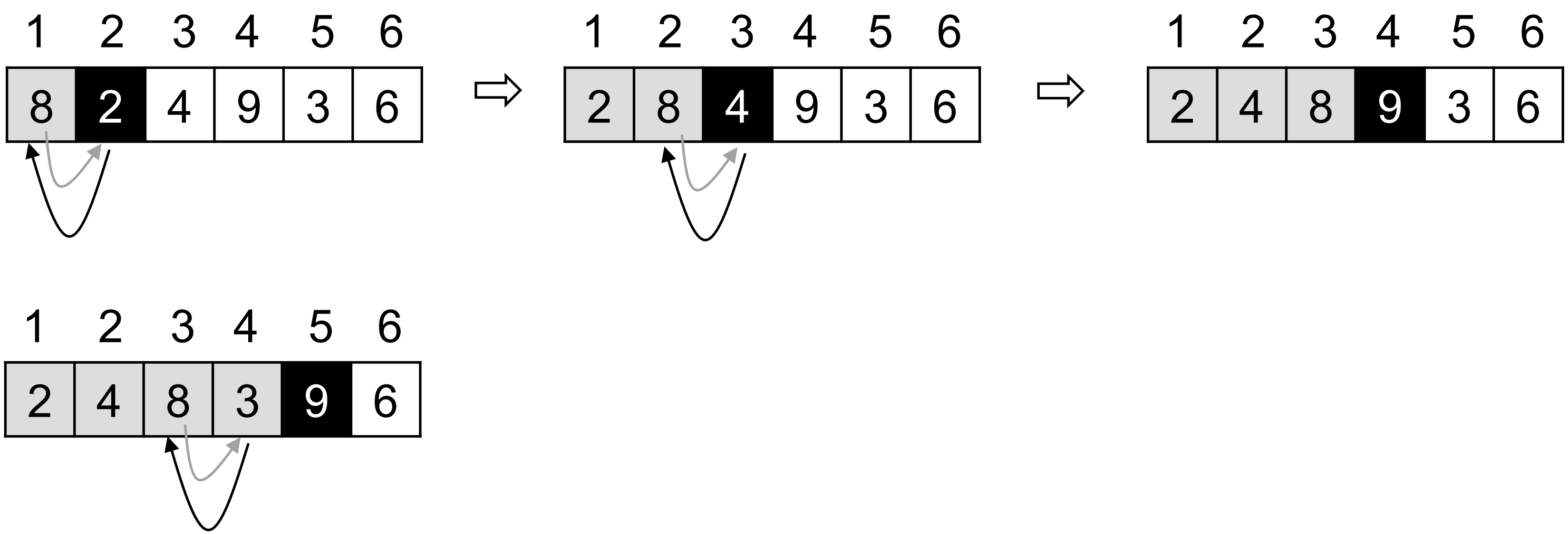
- Natural idea: if first k numbers already correctly sorted, adding the $(k+1)'th$ number is easy
  - Insertion sort



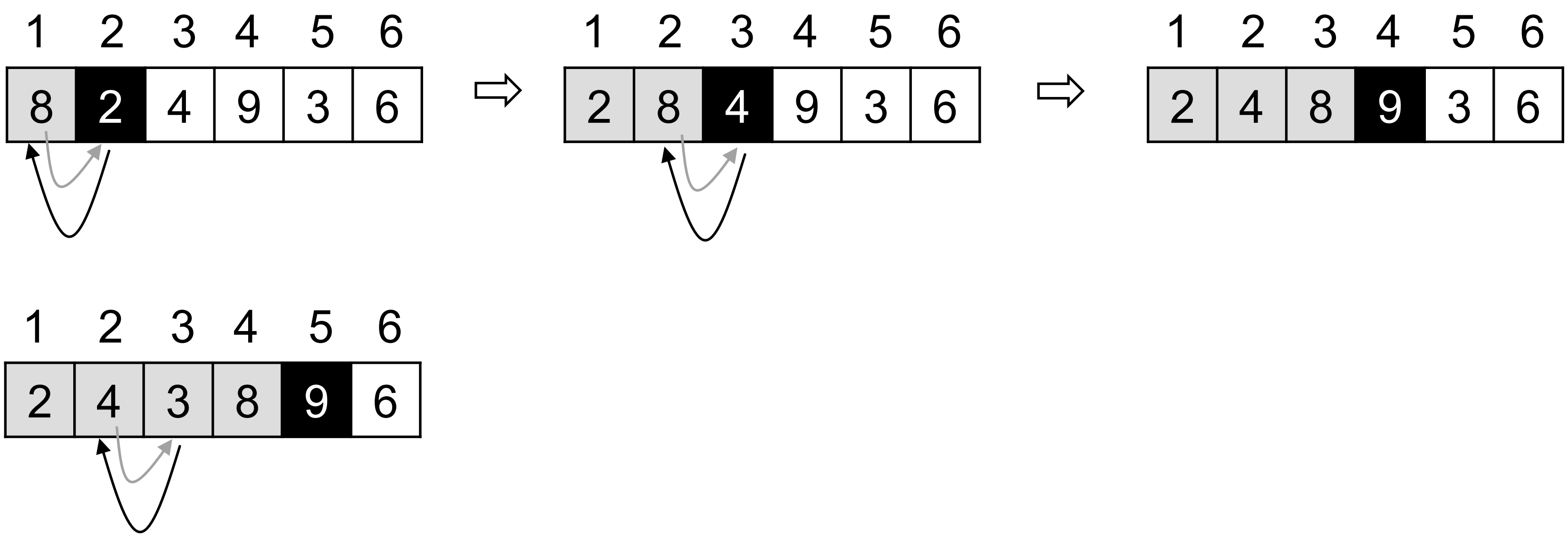
Viewed from Array Data Structure



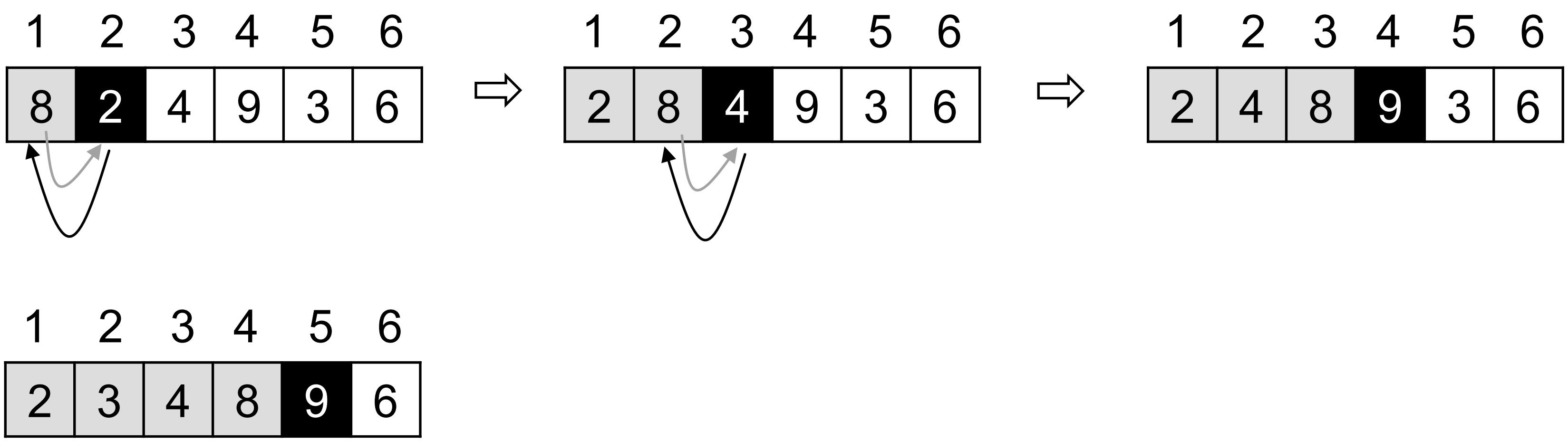
Viewed from Array Data Structure



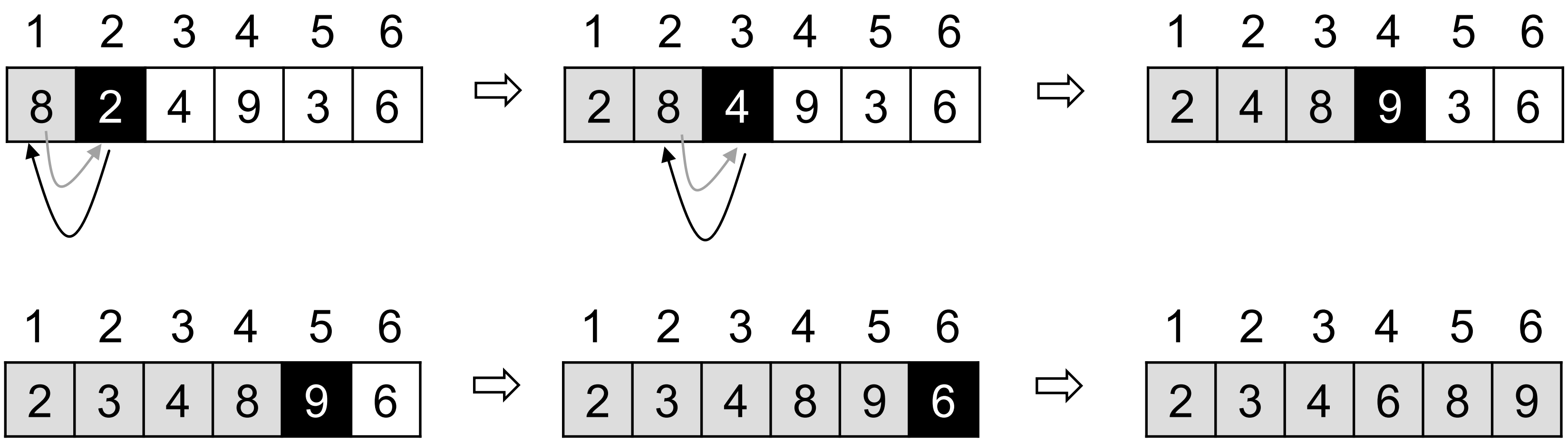
Viewed from Array Data Structure



Viewed from Array Data Structure



Viewed from Array Data Structure



Pseudo-code

INSERTION-SORT(A)

for $j = 2$ **to** $A.length$

  $key = A[j]$

  // Insert $A[j]$ into the sorted sequence $A[1:j-1]$

Comments

  $i = j - 1$

  while $i > 0$ and $A[i] > key$

    $A[i+1] = A[i]$

    $i = i - 1$

  $A[i+1] = key$

1	2	3	4	5	6
2	8	4	9	3	6

$j = 3, key = 4, i = 2$

Pseudo-code

INSERTION-SORT(A)

for $j = 2$ **to** $A.length$

  $key = A[j]$

  // Insert $A[j]$ into the sorted sequence $A[1{:}j-1]$

Comments

  $i = j - 1$

  while $i > 0$ and $A[i] > key$

    $A[i+1] = A[i]$

    $i = i - 1$

  $A[i+1] = key$

1	2	3	4	5	6
2	8	8	9	3	6

$j = 3, key = 4, i = 2$

Analysis: Correctness?

INSERTION-SORT(A)

for $j = 2$ **to** $A.length$
    $key = A[j]$
    // Insert $A[j]$ into the sorted sequence $A[1{:}j-1]$
    $i = j - 1$
    while $i > 0$ and $A[i] > key$
        $A[i+1] = A[i]$
        $i = i - 1$
    $A[i+1] = key$

- The algorithm is indeed correct – prove by induction
  - A simple exercise for you to recap induction proof

Analysis: Running Time

INSERTION-SORT(A)

```
for j = 2 to A.length
    key = A[j]
    // Insert A[j] into the sorted sequence A[1:j − 1]
    i = j − 1
    while i > 0 and A[i] > key
        A[i + 1] = A[i]
        i = i − 1
    A[i + 1] = key
```

- Well, depend on input instance
- What if input is: 2 4 6 7 8 9 ?

	1	2	3	4	5	6
	2	4	6	7	8	9
Time cost	1	1	1	1	1	1

Total cost = $O(n)$ where $n = \mathrm{A.length}$

Analysis: Running Time

INSERTION-SORT(A)

for $j = 2$ **to** $A.length$
    $key = A[j]$
    // Insert $A[j]$ into the sorted sequence $A[1{:}j-1]$
    $i = j - 1$
    while $i > 0$ and $A[i] > key$
        $A[i+1] = A[i]$
        $i = i - 1$
    $A[i+1] = key$

- Well, depend on input instance
- What if input is: 2 4 6 7 8 9 ?
- What about: 9 8 7 6 4 2 ?

	1	2	3	4	5	6
	7	8	9	6	4	2
Time cost	1	2	3	4	5	6

Total cost = $O(n^2)$

Analysis: Running Time

INSERTION-SORT(A)

```
for j = 2 to A.length
    key = A[j]
    // Insert A[j] into the sorted sequence A[1: j − 1]
    i = j − 1
    while i > 0 and A[i] > key
        A[i + 1] = A[i]
        i = i − 1
    A[i + 1] = key
```

- Well, depend on input instance
- What if input is: 2 4 6 7 8 9 ?
- What about: 9 8 7 6 4 2 ?
- Algorithm analysis uses worst-case cost

	1	2	3	4	5	6
	7	8	9	6	4	2
Time cost	1	2	3	4	5	6

Total cost = $O(n^2)$ → Running time of insertion sort

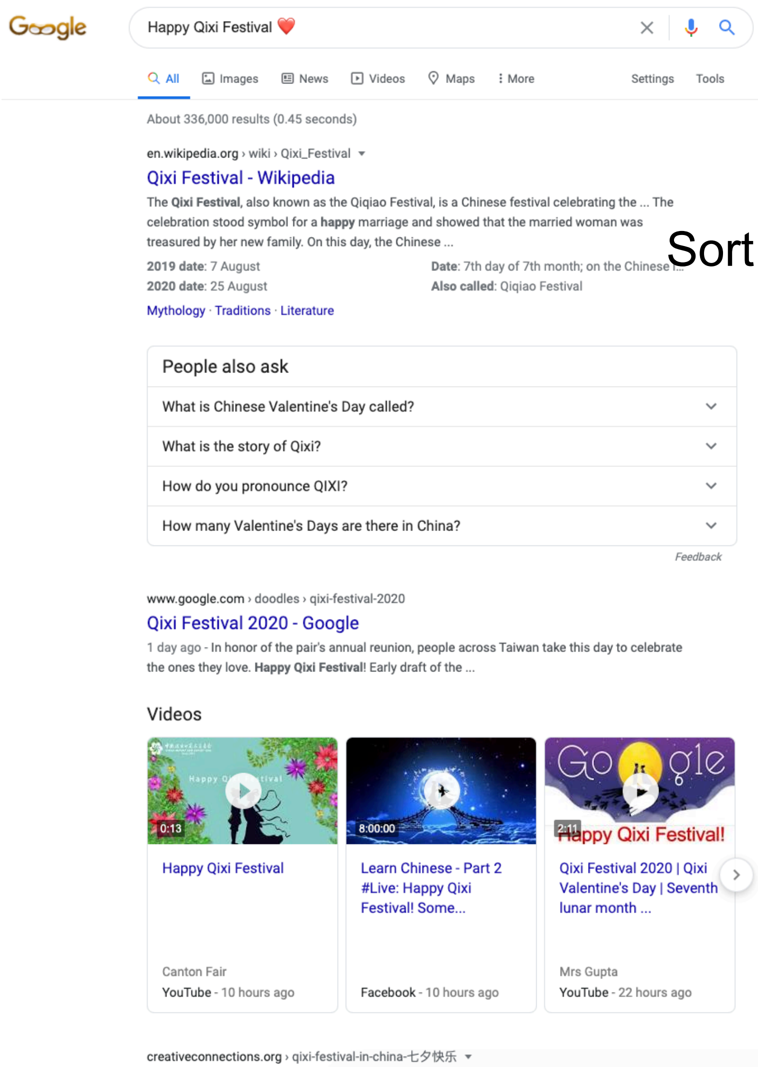
Why Worst-Case Analysis?

- Because you want the efficiency guarantee to hold for arbitrary input instance
- Remedy via *randomization*
  - For example, can first permute given array A uniformly at random
  - Leads to average-case analysis
  - Can reduce the cost by a constant, but still $O(n^2)$
  - (Will cover more when discussing randomized algorithms)

Need new algorithm design techniques to get a significant boost

- There are $O(n \log n)$ time algorithm for sorting (next lecture)

The Power of Algorithm Design



Sort 100K web links based on relevance scores

$O(n^2)$ vs. $O(n \log n)$

10^{10} vs. 10^6

Thank You

Haifeng Xu

University of Virginia

hx4ad@virginia.edu