Lecture 02 — Examples of Algorithms Analysis

COMPSCI 308 — Design and Analysis of Algorithms

Xing Shi Cai https://newptcai.gitlab.io

Duke Kunshan University, Kunshan, China

Jan – Mar 2026

Summary

Running Time — Sorting Algorithms

- Sorting Problems
- Insertion Sort
- Radix Sort

Optimality — Robot Tour Optimization

Correctness — Prime Test

Granularity — Integer Multiplication

Assignments[1]

Practice makes perfect!

Introduction to Algorithms (ITA)

📖 Read —

- Section 2.1, 2.2, 2.3

✏️ Practice —

- Exercises 2.1: 1–4.
- Exercises 2.2: 1–2, 4.
- Exercises 2.3: 1–6.
- Problem 2-1.

[1] 💣 Assignments will not be collected; however, quiz problems will be selected from them. (This includes both Readings and Exercises.)

Running Time — Sorting Algorithms

Running Time — Sorting Algorithms

Sorting Problems

Sorting Problems

How to design an algorithm for

- Input: An array of n numbers $\langle a_1, a_2, \ldots, a_n \rangle$.
- Output: A permutation (of the original array) $\langle a'_1, a'_2, \ldots, a'_n \rangle$ such that

$$a'_1 \leq a'_2 \leq \ldots \leq a'_n.$$

Non-decreasing.

why ?
1. Easy to locate data.
2. More important things comes first.
3. Rank candidates

Sorting Problems

How to design an algorithm for

- Input: An array of n numbers $\langle a_1, a_2, \dots, a_n \rangle$.
- Output: A permutation (of the original array) $\langle a'_1, a'_2, \dots, a'_n \rangle$ such that

$$a'_1 \leq a'_2 \leq \dots \leq a'_n.$$

📌 Application

While the PageRank algorithm assigns a score to each web page, a sorting algorithm remains necessary to sort web pages before presenting them to users.

🍰 What are some other applications of sorting?

Instances

A particular input for a sorting algorithm is called an instance of the sorting problem.

In general, an instance of a problem consists of the input needed to compute a solution to the problem.

The running time of an algorithm often depends on the instance —

- good instances have short running times.
- bad instances have long running times.

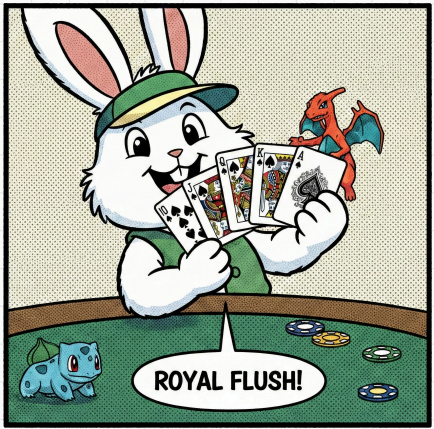


Figure 1: A good hand.

Instances

A particular input for a sorting algorithm is called an instance of the sorting problem.

In general, an instance of a problem consists of the input needed to compute a solution to the problem.

The running time of an algorithm often depends on the instance —

- good instances have short running times.
- bad instances have long running times.

Figure 1: A bad hand.

Average and Worst-Case Analysis

Average analysis focuses on the average running time.

Worst-case analysis focuses on the worst-case running time.

Which of these analyses is more "practical"?

The advantage of *average analysis*—

Better idea of how much time it usually take.

The disadvantage of *average analysis*—

Usually assume uniform distribution.

Average and Worst-Case Analysis

Average analysis focuses on the average running time.

Worst-case analysis focuses on the worst-case running time.

Which of these analyses is more "practical"?

The advantage of *worst-case analysis*—

The runtime is guaranteed to be less than worst-time.

The disadvantage of *worst-case analysis*—

It can be the case that worst-case is rare.

Keys and Records

The numbers/strings to be sorted are called keys.

Often they are paired with some (satellite) data.

Together, they form a record.

Student ID	Name	Age
12345	Jon Snow	14
67890	Cersei Lannister	32
11111	Sansa Stark	11
24680	Daenerys Targaryen	13
43210	Arya Stark	9

Table 1: Example: In this spreadsheet, the student IDs can be seen as keys and each row can be seen as a record.

Running Time — Sorting Algorithms

Insertion Sort

An efficient algorithm for sorting a small number of elements works the way you might sort a hand of playing cards.

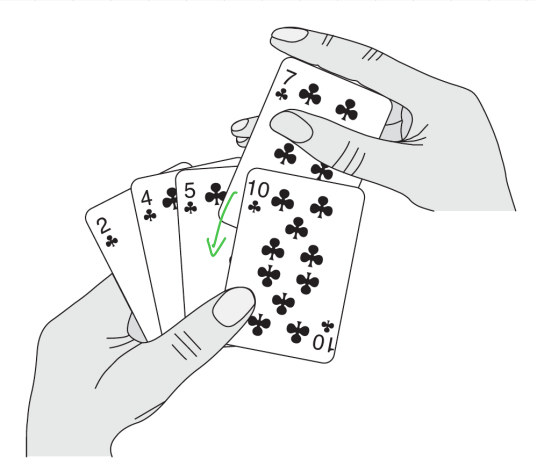


Figure 2: Insertion sort

Insertion Sort

An efficient algorithm for sorting a small number of elements works the way you might sort a hand of playing cards.

1. 🤚 — ⟨ 4 ⟩

• ♣ — ⟨4 5 2 10 7⟩

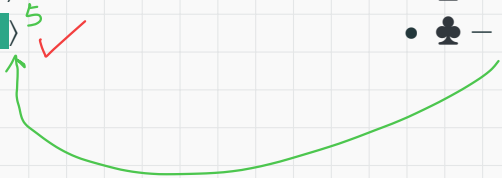
↑

top of deck.

Insertion Sort

An efficient algorithm for sorting a small number of elements works the way you might sort a hand of playing cards.

1. ✋ — ⟨ ⟩
2. ✋ — ⟨ 4 ⟩



- ♣ — ⟨4 5 2 10 7⟩
- ♣ — ⟨5 2 10 7⟩

Insertion Sort

An efficient algorithm for sorting a small number of elements works the way you might sort a hand of playing cards.

1. ✋ — ⟨ ⟩
2. ✋ — ⟨ 4 ⟩
3. ✋ — ⟨4 5 ⟩

- ♣ — ⟨4 5 2 10 7⟩
- ♣ — ⟨5 2 10 7⟩
- ♣ — ⟨2 10 7⟩

Insertion Sort

An efficient algorithm for sorting a small number of elements works the way you might sort a hand of playing cards.

1. 🤚 — ⟨ ⟩
2. 🤚 — ⟨ 4 ⟩
3. 🤚 — ⟨4 5⟩
4. 🤚 — ⟨ 2 4 5⟩

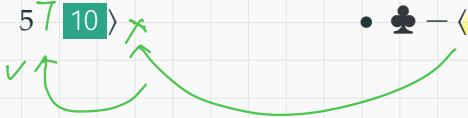
- ♣ — ⟨4 5 2 10 7⟩
- ♣ — ⟨5 2 10 7⟩
- ♣ — ⟨2 10 7⟩
- ♣ — ⟨10 7⟩

Insertion Sort

An efficient algorithm for sorting a small number of elements works the way you might sort a hand of playing cards.

1. ✋ — ⟨ ⟩
2. ✋ — ⟨ 4 ⟩
3. ✋ — ⟨4 5 ⟩
4. ✋ — ⟨ 2 4 5⟩
5. ✋ — ⟨2 4 5 10 ⟩

- ♣ — ⟨4 5 2 10 7⟩
- ♣ — ⟨5 2 10 7⟩
- ♣ — ⟨2 10 7⟩
- ♣ — ⟨10 7⟩
- ♣ — ⟨7⟩



Insertion Sort

An efficient algorithm for sorting a small number of elements works the way you might sort a hand of playing cards.

1. 🤚 — ⟨ ⟩
2. 🤚 — ⟨ 4 ⟩
3. 🤚 — ⟨4 5 ⟩
4. 🤚 — ⟨ 2 4 5⟩
5. 🤚 — ⟨2 4 5 10 ⟩
6. 🤚 — ⟨2 4 5 7 10⟩

- ♣ — ⟨4 5 2 10 7⟩
- ♣ — ⟨5 2 10 7⟩
- ♣ — ⟨2 10 7⟩
- ♣ — ⟨10 7⟩
- ♣ — ⟨7⟩
- ♣ — ⟨⟩

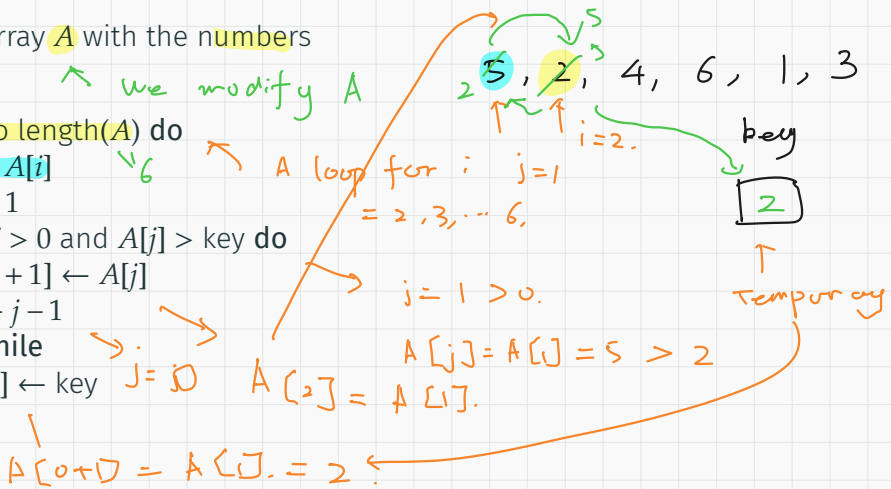
Pseudocode of Insertion Sort

Input: an array A containing the numbers to be sorted.

Output: the array A with the numbers sorted.

```
1: for i = 2 to length(A) do
2:     key ← A[i]
3:     j ← i − 1
4:     while j > 0 and A[j] > key do
5:         A[j + 1] ← A[j]
6:         j ← j − 1
7:     end while
8:     A[j + 1] ← key
9: end for
```

Let's try out insertion sort on $A = \langle 5, 2, 4, 6, 1, 3 \rangle$.



Why do we care?

Instead of banging our heads on the ⌨ and trying to find better algorithms, why don't we just buy a better 🖥?

- Save money.
- Works on all hardware.
- Less energy consumption.

The Efficiency of Sorting Algorithms

Different algorithms devised to solve the same problem often differ dramatically in their efficiency.

For example, to sort n numbers,

- Insertion Sort takes time roughly $c_1 n^2$,
- Merge Sort takes time roughly $c_2 n \log n$,

where c_1 and c_2 are constants that do not depend on n.

💡 Insertion sort typically has a smaller constant factor than merge sort, in other words, $c_1 < c_2$.

🍰 Which algorithm is faster?

When n Is Small

According to this 👆 experiment, insertion sort is faster when n is small.

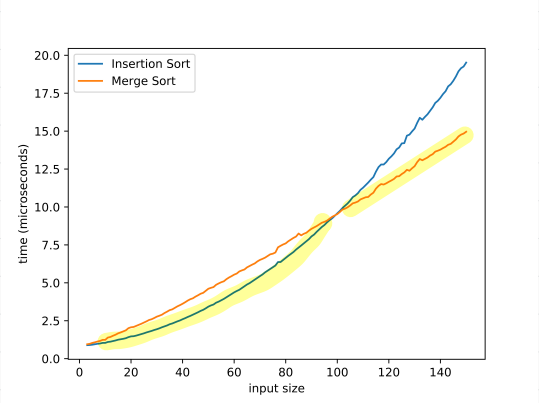


Figure 2: Comparison of sorting algorithms

When n Is Large

Note that

$$\lim_{n\to\infty} \frac{c_1 n^2}{c_2 n \log n} = \infty \tag{1}$$

insertion sort

merge sort.

Thus, no matter what c_1, c_2 are, for any $M > 0$, there exists an n_0, such that

$$\frac{c_1 n^2}{c_2 n \log n} > M$$

for all $n \geq n_0$.

🧩 Can you prove (1)?

A Thought Experiment

Let's dive deeper with a practical example:

- Consider $n = 10^7$.
- Merge sort: $50n \log(n)$ instructions.[2]
- Insertion sort: $2n^2$ instructions.

Execution environment:

- Merge sort on a 💻: 10^7 instructions/sec.
- Insertion sort on a 🖥️: 10^{10} instructions/sec.

🍰 Which algorithm will be faster?

$\log_e x$

[2] In this course, $\log x$ and $\ln x$ both denote the natural logarithm of x.

Beyond Performance

Besides running time, there are many aspects of an algorithm that can be analyzed:

- 🏆 Optimality: Produces the optimal results.
- ✅ Correctness: Produces expected results.
- 🔗 Modularity: Independent, manageable code units.
- 🛠️ Maintainability: Easily modifiable code.
- 🎛️ Functionality: Set of capabilities.
- 🛡️ Robustness: Handles unexpected inputs gracefully.
- 😃 User-friendliness: Intuitive for users.
- ⏱️ Programmer time: Time needed for coding tasks.
- 🧩 Simplicity: Least complex solution.
- ⚙️ Extensibility: Adaptable for new features.
- 💹 Reliability: Consistent performance.

Which Algorithm to Choose?

Among many sorting algorithms, deciding the best algorithm for a given application depends on:

- The number of items to be sorted.
- The extent of pre-sorting among items.
- Restrictions on item values.
- The computer's architecture.
- Storage devices: main memory, disks, or tapes.
- ... (other factors as applicable).

Running Time — Sorting Algorithms

Radix Sort

📌 Example: Exams in Sweden

In 🇸🇪, exams are anonymously graded using a random 5-digit ID for each student.

Fifty papers need to be sorted by IDs for efficient grade entry.

🥹 Which algorithm should you use? Merge sort or insertion sort?

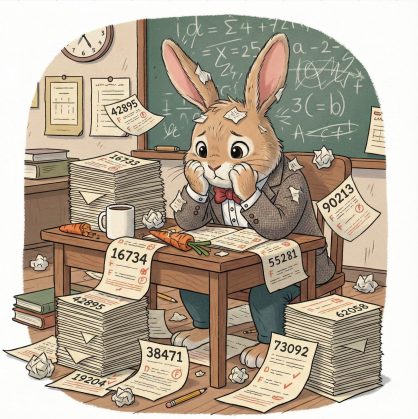


Figure 3: 😩 How to sort these papers?

📌 Example: Exams in Sweden

In 🇸🇪, exams are anonymously graded using a random 5-digit ID for each student.

Fifty papers need to be sorted by IDs for efficient grade entry.

😳 Which algorithm should you use? Merge sort or insertion sort?

😯 The problem with both algorithms is that they both need to compare 5-digit numbers.

😩 It is easy for a 💻 but hard for a 👷.

Figure 3: 😩 How to sort these papers?

Radix Sort

The best way to do this is thus to use radix sort (bucket sort):

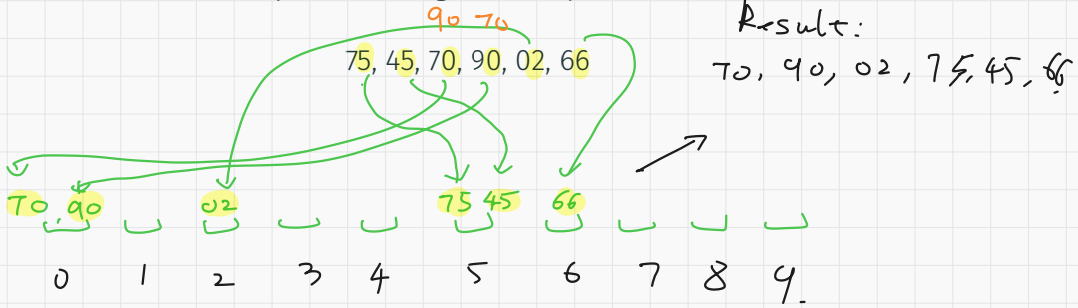
- A non-comparative sorting algorithm 👍
- Sorts numbers by processing each digit at a time from the right to the left (using any sorting algorithm you like)
- Efficient for sorting fixed-length numbers

Radix Sort

The best way to do this is thus to use radix sort (bucket sort):

- A non-comparative sorting algorithm 👍
- Sorts numbers by processing each digit at a time from the right to the left (using any sorting algorithm you like)
- Efficient for sorting fixed-length numbers

📌 Let's see how it works by considering an example —

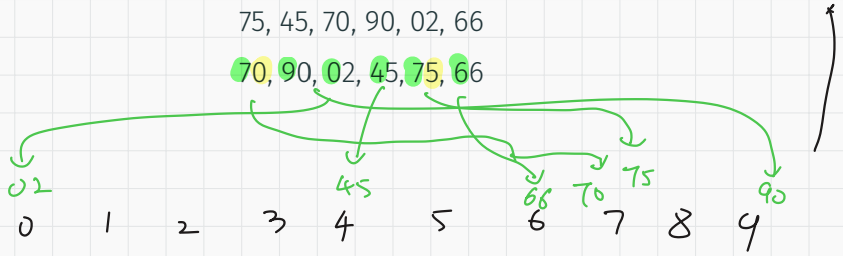


Radix Sort

The best way to do this is thus to use radix sort (bucket sort):

- A non-comparative sorting algorithm 👍
- Sorts numbers by processing each digit at a time from the right to the left (using any sorting algorithm you like)
- Efficient for sorting fixed-length numbers

📌 Let's see how it works by considering an example —



Result:
02, 45, 66, 70, 75, 90

Radix Sort

The best way to do this is thus to use radix sort (bucket sort):

- A non-comparative sorting algorithm 👍
- Sorts numbers by processing each digit at a time from the right to the left (using any sorting algorithm you like)
- Efficient for sorting fixed-length numbers

📌 Let's see how it works by considering an example —

75, 45, 70, 90, 02, 66

70, 90, 02, 45, 75, 66

02, 45, 66, 70, 75, 90

Radix Sort

The best way to do this is thus to use radix sort (bucket sort):

- A non-comparative sorting algorithm 👍
- Sorts numbers by processing each digit at a time from the right to the left (using any sorting algorithm you like)
- Efficient for sorting fixed-length numbers

💣 Why cannot we sort from the left to the right? Let's see —

75, 45, 70, 90, 02, 66

Radix Sort

The best way to do this is thus to use radix sort (bucket sort):

- A non-comparative sorting algorithm 👍
- Sorts numbers by processing each digit at a time from the right to the left (using any sorting algorithm you like)
- Efficient for sorting fixed-length numbers

💣 Why cannot we sort from the left to the right? Let's see —

75, 45, 70, 90, 02, 66

02, 45, 66, 75, 70, 90

Radix Sort

The best way to do this is thus to use radix sort (bucket sort):

- A non-comparative sorting algorithm 👍
- Sorts numbers by processing each digit at a time from the right to the left (using any sorting algorithm you like)
- Efficient for sorting fixed-length numbers

💣 Why cannot we sort from the left to the right? Let's see —

75, 45, 70, 90, 02, 66

02, 45, 66, 75, 70, 90

70, 90, 02, 45, 75, 66

Exercise

Try to sort the following 3-digit IDs with radix sort.

530, 976, 297, 155, 266

Write down the result of sorting each digit.

Optimality — Robot Tour Optimization

Robot Tour Problem

烙铁

A 🤖 arm with a soldering iron needs to —

- Order contact points for sequential soldering.
- Minimize travel distance for assembly.

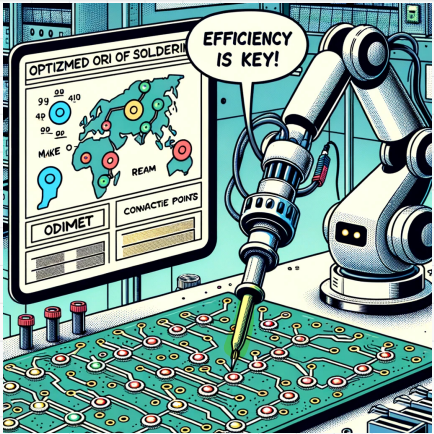


Figure 4: Robot arm tour

Robot Tour Optimization Algorithm

How can we develop an algorithm that provides:

- Input: A set S of n points in the plane.
- Output: The shortest cycle that visits each point in S.

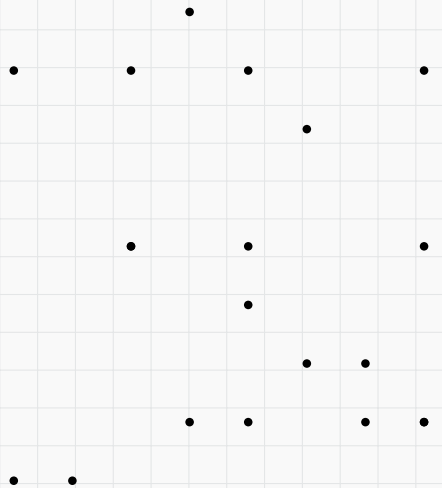


Figure 5: Optimal robot arm tour

Robot Tour Optimization Algorithm

How can we develop an algorithm that provides:

- Input: A set S of n points in the plane.
- Output: The shortest cycle that visits each point in S.

🍰 Why are all the connections in the optimal solution straight line segments?

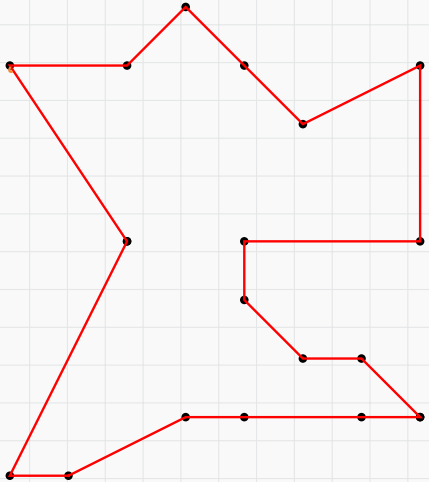


Figure 5: Optimal robot arm tour

The nearest neighbour algorithm works as follows:

1. Start at some point p_0.
2. Go to the nearest point p_1 which has not been visited.
3. Repeat until finished.

This works well for the following instance.

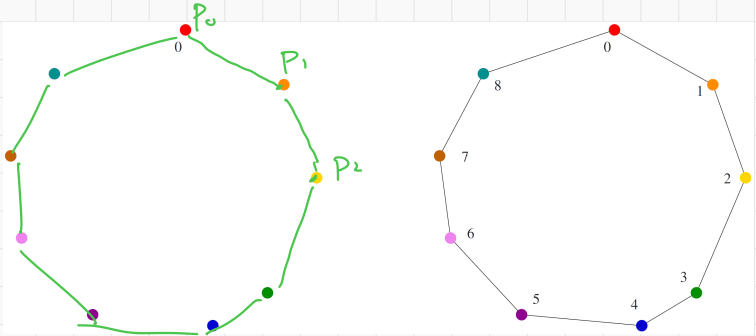


Figure 6: A good instance for the nearest neighbour algorithm

The nearest neighbour algorithm works as follows:

1. Start at some point p_0.
2. Go to the nearest point p_1 which has not been visited.
3. Repeat until finished.

This works badly for the following instance.

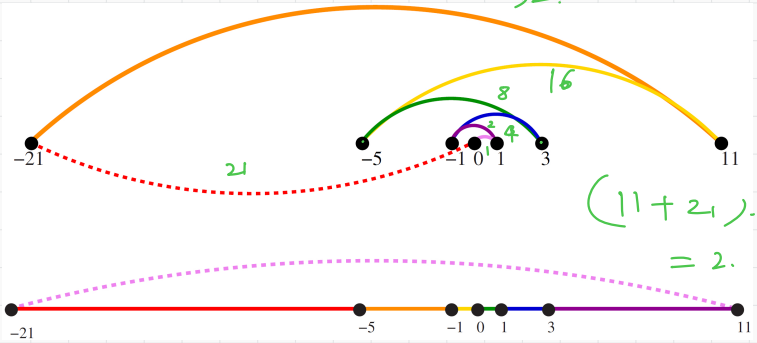


Figure 6: A bad instance for the nearest neighbour algorithm

This algorithm connects the closest pair of points, whose connection will not cause a cycle or a 3-way branch, until all points are in one tour.

This gives the optimal solution to the previous problem.

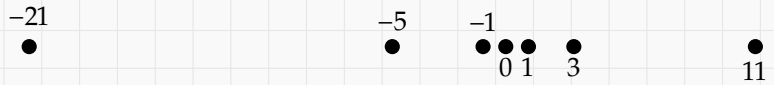


Figure 7: 📌 Applying the closest pair algorithm

When the Closest Pair Algorithm Fails

However, the closest pair algorithm fails in the following instance.

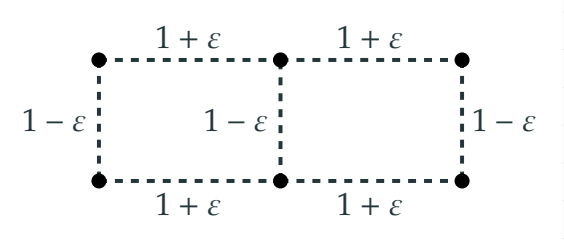


Figure 8: 📌 The closest pair algorithm produces the left tour. However, the one on the right is shorter.

🍰 Let us try the closest pair algorithm.

When the Closest Pair Algorithm Fails

However, the closest pair algorithm fails in the following instance.

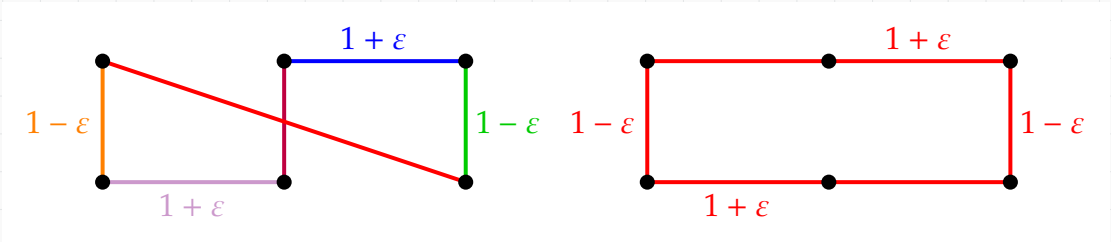


Figure 8: 📌 The closest pair algorithm produces the left tour. However, the one on the right is shorter.

🍰 How long are the two cycles?

The brute force algorithm simply tries every possible tour and finds the shortest one.

🏆 This guarantees the optimal solution to the previous problem.

But it is too slow for most practical use.

🍰 If there are n points, how many tours does the brute force algorithm need to check?

$$\frac{n!}{2 \cdot n}$$

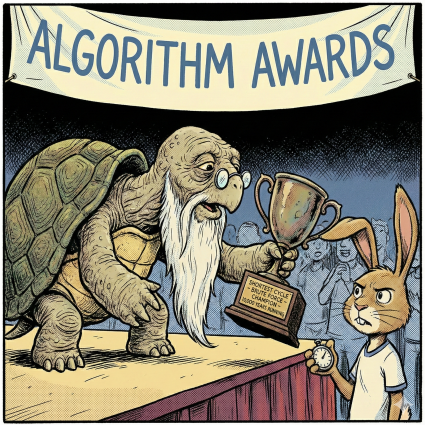


Figure 9: 🐢 I've checked every circle. Every. Single. One.

Travelling Salesman Problem

Imagine you need to visit multiple cities.

The times it takes to travel from one city to another are given

How do we find the quickest route to visit each city exactly once and then return to the starting point.

Figure 10: Travelling salesman problem

📌 Example of Travelling Salesman Problem

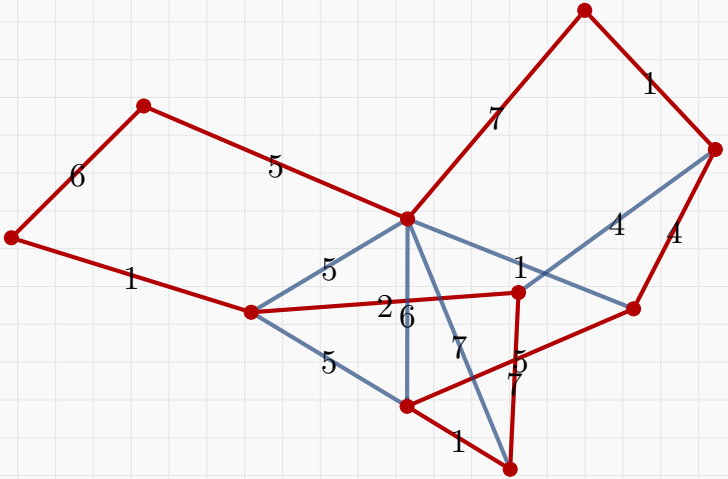


Figure 11: 📌 The shortest travelling salesman tour is highlighted with the red line.

Travelling Salesman Problem — Formal Description

This problem asks for the shortest possible route that visits each vertex exactly once and returns to the original vertex.

- Input: A graph $G = (V, E)$ where V is the set of vertices (cities) and E is the set of edges with associated weights (distances).
- Output: The shortest possible cycle that visits each vertex in V exactly once and returns to the starting vertex.

😢 The only known algorithm that guarantees the optimal solution is brute force.

Correctness — Prime Test

The Correctness of an Algorithm

An algorithm is said to be correct if, for every input instance, it stops with the correct output.

An incorrect algorithm might not stop on some input instances, or it might stop with an incorrect answer

😲 Incorrect algorithms can still be useful.

🍰 Can you think of some examples of incorrect algorithms that we use?

What Constitutes a "Correct" Output?

The definition of a "correct" output is context-dependent and may vary based on specific requirements or objectives.

For instance, in the context of a map application, "correctness" could signify various outcomes:

- A route that minimizes travel time.
- A route that is fastest while also avoiding tolls.
- A route that covers the shortest physical distance.
- And so on ...

Incorrect Algorithms Can be Useful

The commonly used public key cryptography system RSA relies on determining whether a large number n is prime.

🍰 How can we do this by brute force?

Incorrect Algorithms Can be Useful

The commonly used public key cryptography system RSA relies on determining whether a large number n is prime.

How can we do this by brute force?

```
1:  procedure IsPrime(n)
2:      Input: An integer n > 1
3:      Output: True if n is prime, False otherwise
4:      for i = 2 to n − 1 do
5:          if n ≡ 0 (mod i) then
6:              return False
7:          end if
8:      end for
9:      return True
10: end procedure
```

Any space for improvement?

Incorrect Algorithms Can be Useful

The commonly used public key cryptography system RSA relies on determining whether a large number n is prime.

🍰 How can we do this by brute force?

This is difficult to do efficiently.

Instead RSA relies on an algorithm that may return incorrect results:

- If it declares n is composite, then it is definitely true.
- If it declares n is prime, then it is true most of the time.

😎 Works pretty well in practice.

Prime Test in RSA

The algorithm in RSA that tests whether n is prime is based on the following:

Fermat's little theorem

If p is prime, then for every $1 \leq a < p$,

$$a^{p-1} \equiv 1 \pmod{p}.$$

$a \in \{1, \cdots p-1\}$.

There exists a $k \in \mathbb{N}$, such that $a^{p-1} = p \cdot k + 1$.

Prime Test in RSA

The algorithm in RSA that tests whether n is prime is based on the following:

Fermat's little theorem

If p is prime, then for every $1 \leq a < p$,

$$a^{p-1} \equiv 1 \pmod{p}.$$

Is this algorithm correct? No.

The algorithm is as follows:

1: Input: Positive integer n
2: Output: True if n is prime, False otherwise
3: Pick a random positive integer $a < n$
4: **if** $a^{n-1} \equiv 1 \pmod{n}$ **then**
5: **return** True
6: **else**
7: **return** False
8: **end if**

$\{1, \ldots n-1\}$

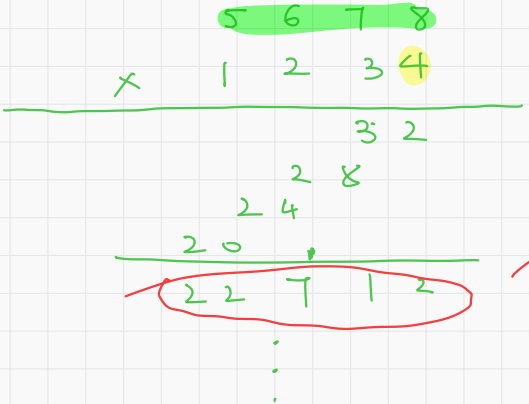
When n is prime, we always get True.

When n is not prime, sometimes we also get True.

Granularity — Integer Multiplication

Integer Multiplication

🍰 Do you remember how to compute 5678×1234 with ✏️?



How many x ?

4.

🍰 Do you remember how to compute 5678×1234 with ✏️?

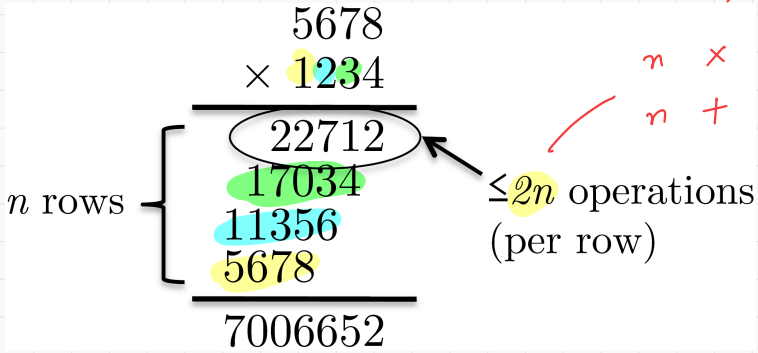


Figure 12: Multiplication as in elementary school

🍰 How many multiplications do we need to compute $x \cdot y$ given two n-digit numbers x and y?

Another Way to Compute $5678 \cdot 1234$

We can write

$$5678 = 56 \cdot 10^2 + 78, \qquad 1234 = 12 \cdot 10^2 + 34.$$

Then

$$\begin{aligned} 5678 \cdot 1234 &= (56 \cdot 10^2 + 78)(12 \cdot 10^2 + 34) \\ &= 56 \cdot 12 \cdot 10^4 + (56 \cdot 34 + 78 \cdot 12)10^2 + 78 \cdot 34 \end{aligned}$$

🍰 How many multiplications do we need to compute $5678 \cdot 1234$?

$4 \cdot 4 = 16$ multiplications

Another Way to Compute $5678 \cdot 1234$

We can write

$$5678 = 56 \cdot 10^2 + 78, \qquad 1234 = 12 \cdot 10^2 + 34.$$

Then

$$\begin{aligned} 5678 \cdot 1234 &= (56 \cdot 10^2 + 78)(12 \cdot 10^2 + 34) \\ &= 56 \cdot 12 \cdot 10^4 + (56 \cdot 34 + 78 \cdot 12)10^2 + 78 \cdot 34 \end{aligned}$$

Karatsuba noticed that we can use fewer multiplications by applying

$$(56 \cdot 34 + 78 \cdot 12) = (56 + 78)(34 + 12) - 56 \cdot 12 - 78 \cdot 34.$$

4 + 4 + 4 = 12 multiplications

🍰 How many multiplications do we need to compute $5678 \cdot 1234$ using this?

Karatsuba's Algorithm

To compute $x \cdot y$, we use

$$xy = x_1y_1 10^n + (x_1y_0 + x_0y_1)10^{n/2} + x_0y_0$$

and

$$(x_1y_0 + x_0y_1) = (x_1 + x_0)(y_1 + y_0) - x_1y_1 - x_0y_0.$$

This involves three multiplications of two $n/2$-digit numbers:

$$x_1y_1, \qquad x_0y_0, \qquad (x_1 + x_0)(y_1 + y_0)$$

We use the same method to compute them recursively.

Later we will see that the total number of multiplications needed is at most $cn^{\log_2 3}$.

What are your main takeaways today? Any questions?