Melbourne Bioinformatics
BIOINFORMATICS + DATA SERVICES + INFRASTRUCTURE, FOR LIFE SCIENCES TODAY

POSTERA CRESCAM LAUDE
THE UNIVERSITY OF MELBOURNE

COMP90014

Algorithms for Bioinformatics

Week 1A - Overview and Algorithms

Billiard balls

9 billiard balls

1 balance scale

One ball is heavier than all the others

How many tries do you need to find it?

Billiard balls

9 billiard balls

1 balance scale

One ball is heavier than all the others

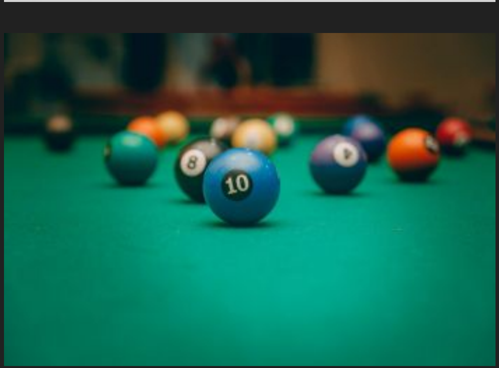
How many tries do you need to find it?

Split 8 / 1: either 1 try (lucky), or 3 tries

Billiard balls

分成8/1：这意味着你在天平的两边各放四个球，剩下一个球不放。如果你幸运，天平会平衡，那么没放在天平上的球就是较重的球，这只需要一次尝试。但如果天平倾斜，你知道较重的球在天平上的八个球中，这种情况下，你还需要两次尝试来确定哪一个球较重。所以，最坏的情况下，这种策略可能需要三次尝试。

分成3/3/3：这是最优策略。你将球分成三组，每组三个球，然后先比较两组。如果天平倾斜，你就知道哪一组有较重的球。如果平衡，则较重的球在你没有称重的那组中。无论哪种情况，你只需要再进行一次称重就能找到重球：比较重组中的两个球，把第三个球放在一边。如果天平倾斜，你就找到了较重的球；如果天平平衡，放在一边的球就是较重的球。这种方法保证你只需要两次尝试就能找到较重的球。



9 billiard balls

1 balance scale

One ball is heavier than all the others

How many tries do you need to find it?

Split 8 / 1:	either 1 try (lucky), or 3 tries
Split 3 / 3 / 3:	always 2 tries

Algorithms

Algorithms are everywhere

Cooking recipe

Instructions to assemble a desk

Driving directions from A to B

What are they?

An algorithm is a method [or a set of instructions] for solving a problem. —Niklaus Wirth

Act on data to produce an output.

Set of strategies to solve a problem.

Strategies form toolkit

Why learn about algorithms?

Write new tools and perhaps new algorithms

Understand the methods you are applying when you run existing tools

Plan your computational needs based on the tools you will run

Change the way you think
(largest 5 numbers in random list)

COMP90014

Aims

Explore algorithms *used in bioinformatics*

How they work; their advantages and limitations

Why, when and how to apply them

Not just efficiency! Simplicity is important too.

Lecture / Tutorial Format

The problem (biological context, data)

Algorithmic approaches

Assumptions, constraints, trade-offs

Evaluation (some weeks)

Fundamentals Revision

Modules -> Welcome to COMP90014

Pre-recorded lecture: genomics

Playlist of external videos & tutorials

How to Succeed

Attend the lectures

Attend the tutorials

Have fun!

Staff

Grace Hall (Subject Coordinator)

grace.hall1@unimelb.edu.au

Melbourne Bioinformatics

Office hour: TBA

Adam Taranto (Tutor)

adam.taranto@unimelb.edu.au

Office hour: TBA

Lectures & Tutorials

Class	Weekday	Time	Building	Room
Lecture 1	Tuesday	11:00am - 12:00pm	Biosciences 4 (Building 147)	G003 (Agar Theatre)
Lecture 2	Thursday	2:15pm - 3:15pm	Chemical Engineering 1 (Building 165)	G20 (Chemical and Biomolecular Engineering Theatre)
Tutorial 1	Tuesday	2:00pm - 3:00pm	Western Edge Biosciences (Building 125)	G04 (Collaborative Learning Centre)
	Thursday	12:00pm - 1:00pm	Medical (Building 181)	E307 (Frederic Wood Jones Room)

Lectures / tutorials available on Canvas in case of a timetable clash.

Choose a tutorial and sign up for it!

Tutorials generally run one week behind content.

Assessment

3x Assignments (10% 15% 15%) (50 hrs)

Weeks 4, 8, 11. Released 2 weeks prior.

Online submission, due Tuesdays @ midnight!

Code tasks, pseudocode, written answer

Take-home exam (20%) (2hrs)

Online submission

Code tasks, pseudocode, written answer

Final written exam (40%) (2 hrs)

In-person, paper

Multiple choice, short answer, long answer

Assignment Penalties

10% per day (not incl. weekends)

Hurdles

20 / 40 across assignments

30 / 60 across exams

Final exam info

Can bring 2 double sided pages of notes

Can bring English dictionary

All lectures, tutorials, and assignments are examinable (unless otherwise noted)

Academic Integrity

Plagiarism and Misconduct

The University takes plagiarism very seriously, whether deliberate or accidental.

It is your responsibility to understand how to avoid plagiarism and collusion.

You can find a guide here

UMSU has an advocacy service here

Please sign the Academic Integrity Statement on Canvas (requirement - due end week 2)

Academic honesty

Submitted work must be done independently.

Don’t copy your work from anywhere else.

If cheating or collusion is detected, all parties will be referred to MDHS for handling under the University discipline procedures

ChatGPT

You can use ChatGPT.

In assessments, you may only use ChatGPT to assist in grammar, clarity, and spelling. You must acknowledge use of ChatGPT. More info here.

Good ideas

As your personal tutor (learning)

Spelling / grammar check (assessment polish)

Bad ideas

Using ChatGPT to write your assessments.

Assessments are mostly code - ChatGPT will not give good answers. We will ensure that ChatGPT cannot answer our assessment questions, so don't bother. In addition, the final exam is in-person, written, so won't have access to ChatGPT.

ChatGPT

You can use ChatGPT.

In assessments, you may only use ChatGPT to assist in grammar, clarity, and spelling. You must acknowledge use of ChatGPT. More info here.

Good ideas

As your personal tutor (learning)

Spelling / grammar check (assessment polish)

Bad ideas

Using ChatGPT to write your assessments.

Assessments are mostly code - ChatGPT will not give good answers. We will ensure that ChatGPT cannot answer our assessment questions, so don't bother. In addition, the final exam is in-person, written, so won't have access to ChatGPT.

Create a personalized tutor to accelerate your learning

“As an AI tutor on the topic of *[insert topic]*, interactively assess my understanding through a series of probing questions. Analyze my responses to identify and clarify any misconceptions or knowledge gaps. Additionally suggest resources to improve my understanding in identified weak areas”

Spelling / Grammar Check

“Fix any spelling / grammar issues in the following text. Additionally improve cohesion and conciseness where necessary“

"""
[insert text]
"""

Networking & Culture

Networking

People in this room will be your future colleagues!

Make friends! Networking at university is very important.

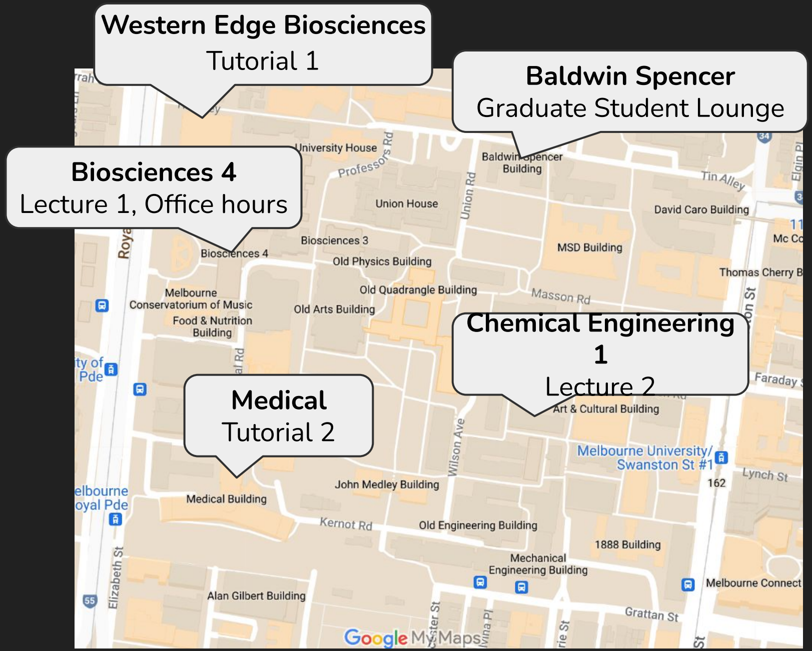
In the unlikely event you experience bullying or harassment, let staff know & we will resolve the situation.

Spaces

Discord groups are awesome.

Office hours spaces can also be workspaces!

Pro tip: Science Graduate Student Lounge, Baldwin Spencer Building (Building 113) is great.



Western Edge Biosciences
Tutorial 1

Baldwin Spencer
Graduate Student Lounge

Biosciences 4
Lecture 1, Office hours

Chemical Engineering 1
Lecture 2

Medical
Tutorial 2

Discord

Discussion

Student representatives

- Represent student body
- Communication channel on behalf of students
- Needs, requests, feedback
- Email grace.hall1@unimelb.edu.au to apply

Discussion forums

- Use your preferred setup for discussing with each other
- Use the Canvas Discussion Board for course content questions
- If you have a question, other people are thinking the same thing
- If we use the discussion forum, everybody benefits

Get your phones out!

pollev.com/gracehall381

Discussion

Student representatives

Represent student body

Communication channel on behalf of students

Needs, requests, feedback

Email grace.hall1@unimelb.edu.au to apply

Discussion forums

Use your preferred setup for discussing with each other

Use the Canvas Discussion Board for course content questions

If you have a question, other people are thinking the same thing

If we use the discussion forum, everybody benefits

```
def select_student_reps(candidates: list[str]) -> list[str]:
    MAX_REPS: int = 3
    if len(candidates) == 0:
        return force_someone()
    return candidates[0:MAX_REPS]

candidates = [“Shinji Ikari”, “Asuka Langley”, “Rei Ayanami”, “Misato Katsuragi” ...]
selected = select_student_reps(candidates)
```

pollev.com/gracehall381

Discussion

Student representatives

Represent student body

Communication channel on behalf of students

Needs, requests, feedback

Email grace.hall1@unimelb.edu.au to apply

Discussion forums

Use your preferred setup for discussing with each other

Use the Canvas Discussion Board for course content questions

If you have a question, other people are thinking the same thing

If we use the discussion forum, everybody benefits

```
def select_student_reps(candidates: list[str]) -> list[str]:     * need at least 1
    MAX_REPS: int = 3                                              * max 3
    if len(candidates) == 0:                                       * first in, first served
        return force_someone()                                     * complexity? O(1): constant
    return candidates[0:MAX_REPS]

candidates = [“Shinji Ikari”, “Asuka Langley”, “Rei Ayanami”, “Misato Katsuragi” ...]
selected = select_student_reps(candidates)
```

Topics

Week	Topic
1	Indexing
2	Sequence Alignment & Mapping
3	Comparing Sequences
4	Advanced Indexing, Graphs & Trees
5	Evolutionary Trees
6	Genomic regions, features, intervals
7 - 8	De novo genome assembly
9	Dimensionality reduction
-	-
10	Unsupervised learning (clustering)
11	Supervised learning
12	Recap

This week

1. Sign up for a tutorial slot, attend tutorial
2. Read course information on Canvas
3. Complete the Academic Integrity statement
4. Background survey on Poll Everywhere.
   (available after class)
   pollev.com/gracehall381

Algorithms Primer

Algorithms & Categories
Data Types & Data Structures
Algorithms & Performance
Heuristics & Approximations

Algorithms

Algorithms

Set of strategies to solve a problem (sorting a list)

Algorithms generally fall into a few broad types

Keep these in mind when thinking about your problem

Most “new” algorithms aren’t new - combining other strategies to adapting to new problem

Common Classes

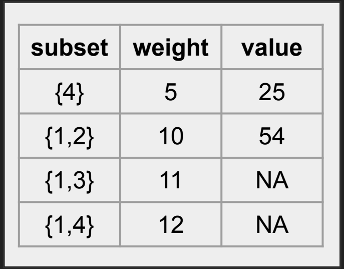
Brute force / exhaustive search (eg closest points, knapsack)

Divide and Conquer (eg mergesort)

Recursive (eg tree traversal)

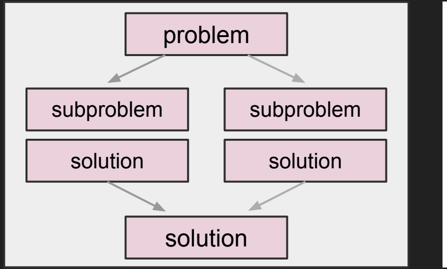
Dynamic Programming (eg local alignment)

Greedy (eg Dijkstra Shortest Path)



subset	weight	value
{4}	5	25
{1,2}	10	54
{1,3}	11	NA
{1,4}	12	NA

Brute force



problem

subproblem

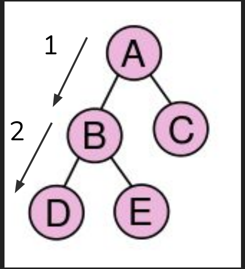
subproblem

solution

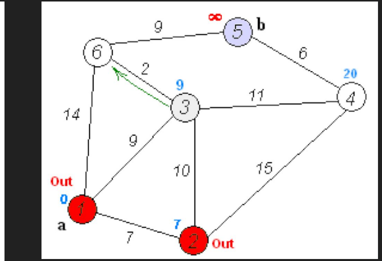
solution

solution

Divide & Conquer



Recursive



Greedy

Data Types & Data Structures

Algorithms act on data

- Data: raw material
- Steel plates enter factory, gears exit factory
- Need to understand the different raw material we will deal with

Data in bioinformatics

- FASTA - sequences (eg genomes)
- FASTQ - sequences w/ quality (eg reads)
- BAM / SAM - alignments (eg reads -> genome)
- Matrix - 2D arrays (eg gene expression counts)
- VCF - genetic variants

Data Types & Data Structures

Algorithms act on data

- Data: raw material
- Steel plates enter factory, gears exit factory
- Need to understand the different raw material we will deal with

Data in bioinformatics

- FASTA - sequences (eg genomes)
- FASTQ - sequences w/ quality (eg reads)
- BAM / SAM - alignments (eg reads -> genome)
- Matrix - 2D arrays (eg gene expression counts)
- VCF - genetic variants

Data types

- Primitives
- Collections
- Derived types (classes)

Data structures

- Arrangements of data for our algorithms
- Higher order structures
- eg graph:
  - Nodes and vertices
  - Relationships between data (social media)

Python Types

Primitives

String - multiple characters (eg “NLGN1”)

Integer - whole number (eg 1, -29)

Float - decimal number (eg 43.34)

Boolean - two possible values (eg True, False)

List

Ordered values of any type

```
mylist = [“Grace Hall”, 92325, 4.0, [100, 100]]
```

Access data via index

Can modify elems, append, slice

Can loop (iterate) over each element

Tuple

Same as list except immutable

Array (numpy etc)

Store data sequentially in memory

All values are same type

Can be multidimensional (2D array = matrix)

Dict (hash table)

key: value pairs

Great for providing a mapping between an identifier and some data

Set

Collection of unindexed, unordered, unique items

Data Structures (common)

Arranging data for use in algorithms

Queue

First-in, first-out principle.

Add at the end, remove from the beginning.

```
enqueue(item, my_queue)

dequeue(my_queue)
```

Stack

First-in, last-out principle.

Add at the top, remove from the top

```
push(item, my_stack)

pop(my_stack)
```

Data Structures (common)

Arranging data for use in algorithms

Queue

First-in, first-out principle.

Add at the end, remove from the beginning.

```
enqueue(item, my_queue)
dequeue(my_queue)
```

Stack

First-in, last-out principle.

Add at the top, remove from the top

```
push(item, my_stack)
pop(my_stack)
```

Tree (weeks 4, 5)

Data has hierarchy

eg. Html

eg. Family tree

Can efficiently search trees

Graphs (weeks 4, 5)

Relationships between data

eg. Exploring a maze

eg. Routes from Footscray to CBD

eg. Solving a rubix cube

Data & Algorithms

Raw Data - Inputs we will use

Data types - Types of the raw data (primitives, collections, derived types)

Data structures - Arranging our data for use in algorithm

Algorithm classes - Broad types of problems, strategies to solve

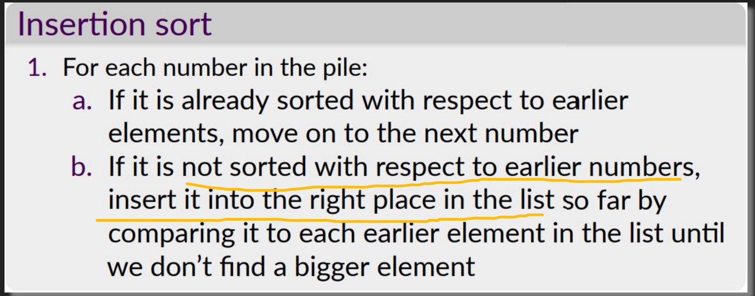
Algorithms - Act on the data structure to solve a problem

Algorithms

Insertion sort

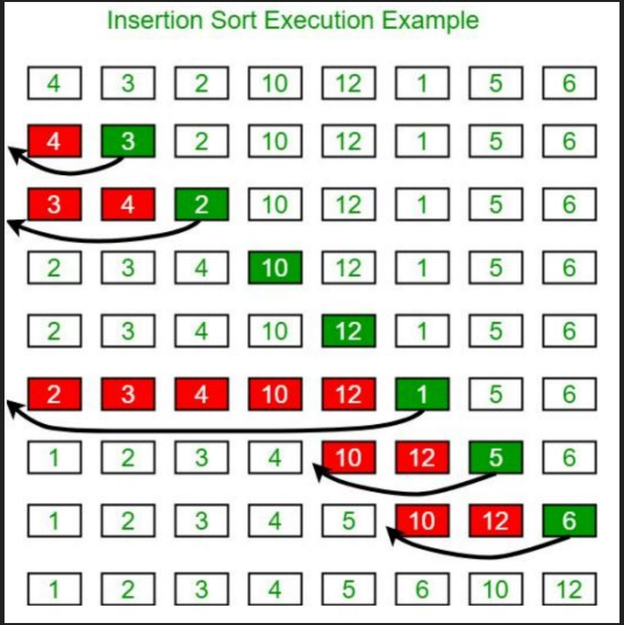
Input: pile of cards with numbers written on them `list[int]`
Output: pile of cards sorted numerically

Data types - integers
Data structure - array / linked list
Algorithm class - brute force
Algorithm - insertion sort



Insertion sort

1. For each number in the pile:
   a. If it is already sorted with respect to earlier elements, move on to the next number
   b. If it is not sorted with respect to earlier numbers, insert it into the right place in the list so far by comparing it to each earlier element in the list until we don't find a bigger element



Insertion Sort Execution Example

4	3	2	10	12	1	5	6
4	3	2	10	12	1	5	6
3	4	2	10	12	1	5	6
2	3	4	10	12	1	5	6
2	3	4	10	12	1	5	6
2	3	4	10	12	1	5	6
1	2	3	4	10	12	5	6
1	2	3	4	5	10	12	6
1	2	3	4	5	6	10	12

Image: geeksforgeeks.com

Algorithms

Correctness

Does the algorithm give the right answer for all inputs?

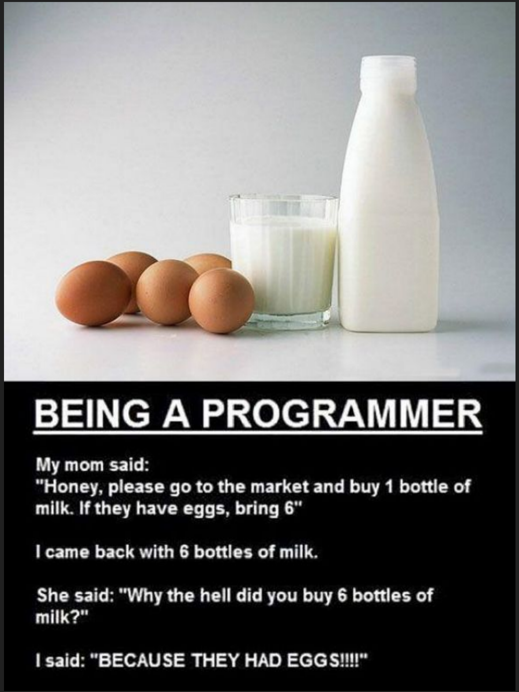
We use logic to implement algorithms

Efficiency

Does the algorithm produce an output in an acceptable amount of time, and use an acceptable amount of memory (space), for a given sized input?

Computational power has followed Moore’s law We always find a way to run out of power!

Big Data & Petascale



BEING A PROGRAMMER

My mom said:
"Honey, please go to the market and buy 1 bottle of milk. If they have eggs, bring 6"

I came back with 6 bottles of milk.

She said: "Why the hell did you buy 6 bottles of milk?"

I said: "BECAUSE THEY HAD EGGS!!!!"

Performance

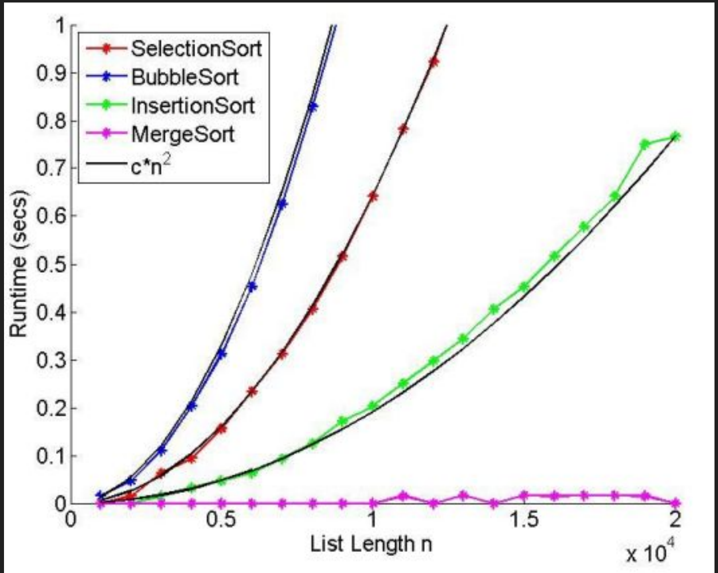
There are two ways to reason about the runtime performance of algorithms.

1. Empirically (by measurement)
2. Theoretically (by mathematical reasoning)

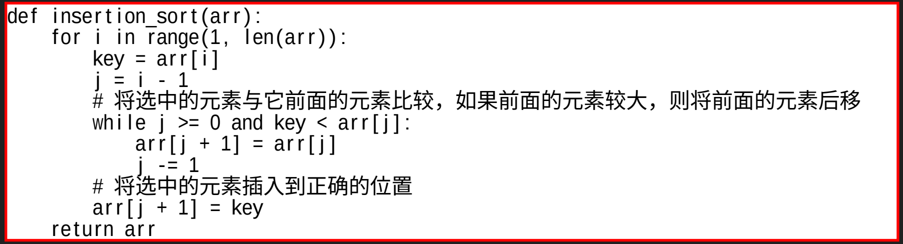
Both are equally important.

Empirical analysis will give information on real data, however only for the selected data.

Can we make more general statements about runtime complexity which cover a larger set of conditions?



Theoretical analysis



```
def insertion_sort(arr):
    for i in range(1, len(arr)):
        key = arr[i]
        j = i - 1
        # 将选中的元素与它前面的元素比较，如果前面的元素较大，则将前面的元素后移
        while j >= 0 and key < arr[j]:
            arr[j + 1] = arr[j]
            j -= 1
        # 将选中的元素插入到正确的位置
        arr[j + 1] = key
    return arr
```

Asymptotic complexity

How does the runtime of an algorithm (ultimately) scale proportionally to increases in input size?

Mathematical function to describe runtime

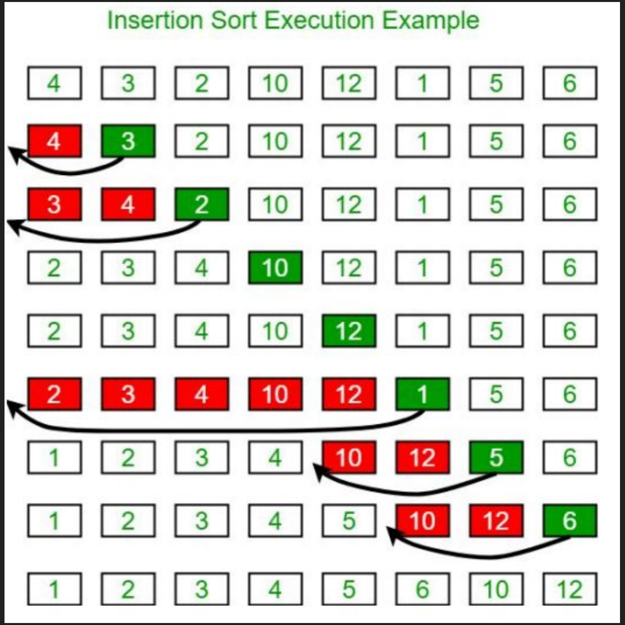
Big-O notation (upper bound)

Describes the limiting behavior of a function

Used to classify algorithms in complexity theory
Insertion sort is $O(n)$ in best case (already sorted), $O(n^2)$ in average & worst case

Merge sort is $O(n \log_2 n)$ in the best, worst and average cases

Insertion Sort Execution Example



4 3 2 10 12 1 5 6
4 3 2 10 12 1 5 6
3 4 2 10 12 1 5 6
2 3 4 10 12 1 5 6
2 3 4 10 12 1 5 6
2 3 4 10 12 1 5 6
1 2 3 4 10 12 5 6
1 2 3 4 5 10 12 6
1 2 3 4 5 6 10 12

Image: geeksforgeeks.com

Big O notation

Example

We propose a new sorting algorithm

The number of steps/comparisons required to sort a vector of size n is given by:

$F(n) = 2n^2 + 2n + 2$

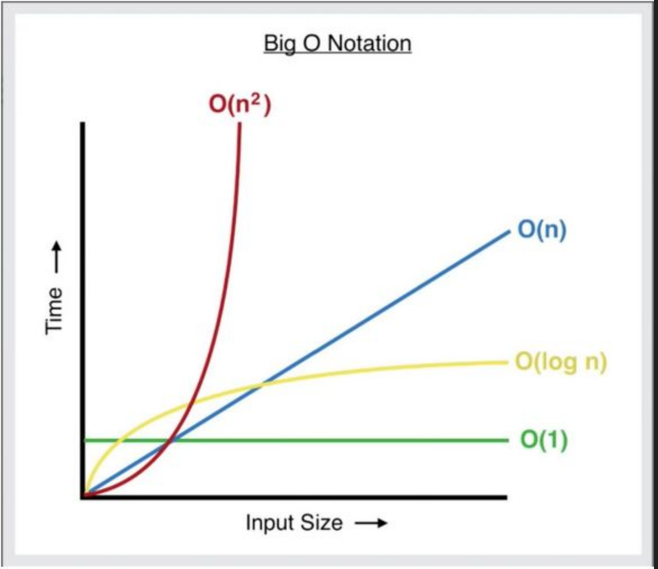
Ignore constants

Ignore slower growing terms

F(n) grows on the order of n^2

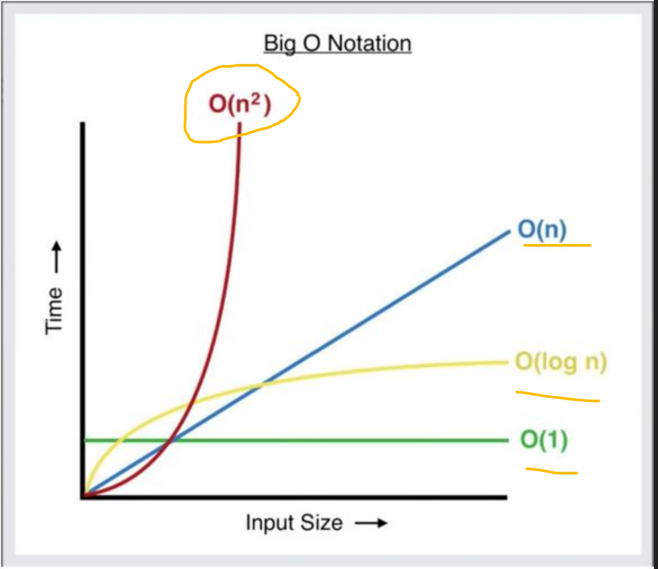
$F(n) \sim O(n^2)$

$n^2 \gg n \log n$



Big O notation

Notation	Name	n = 10	n = 100	n = 1000
O(1)	Constant	2	2	2
O(log n)	Logarithmic	~3	~7	~10
O(n)	Linear	10	100	1000
O(n log n)	Log-Linear	~300	~700	~10 000
$O(n^2)$	Quadratic	100	10 000	10^6
$O(n^c)$	Polynomial	1000	10^6	10^9
$O(c^n)$	Exponential	~1000	~10^{30}	~10^{300}



Need to understand how your algorithm behaves to plan your experiments.

Also valid for memory usage

Tractability

易处理性指的是那些我们可以在多项式时间内解决的问题。
也就是说，这类算法的时间复杂度可以表示为 n^c

是一个常数。多项式时间算法被认为是 “ 易解的 ” 或 “ 有效的 ”，因为它们的运行时间随着输入大小的增加而以可预测和合理的方式增长。在实践中，我们通常可以为这些问题找到精确的解决方案。

Tractable (easy)

Algorithms with polynomial complexity $O(n^c)$

We generally use exact solutions.

Intractable (hard)

Algorithms with super-polynomial complexity, such as exponential algorithms $O(c^n)$

We generally use approximate solutions.

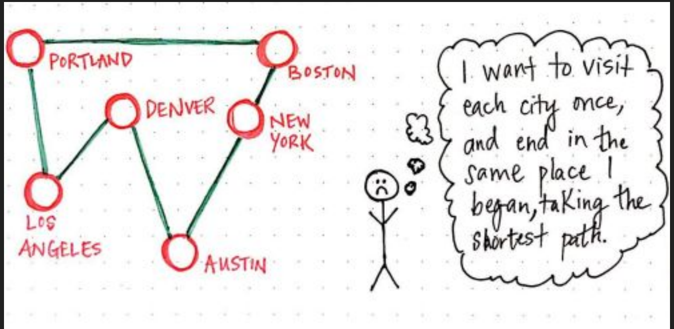
难处理性指的是那些算法的时间复杂度超过多项式时间，比如指数时间
c^n
c 是一个常数。对于这类问题，随着输入大小的增长，解决问题所需的时间增长得非常快，以至于对于较大的输入，它们变得几乎不可能在实际时间内解决。这些问题通常需要使用近似算法或启发式方法来找到解决方案，而不是精确解决方案。

Travelling salesperson problem

NP-complete

Reducible to another in polynomial time

What if your problem falls under this class?


PORTLAND
BOSTON
DENVER
NEW YORK
LOS ANGELES
AUSTIN
I want to visit each city once, and end in the same place I began, taking the shortest path.

Heuristics and Approximations

What can we do when complexity is high?

- Looking for food to eat in the fridge
- Picking an outfit

We manage to get by as best we can. How?

- We trade correctness for efficiency.
- “It’s good enough”
- Generally want to strike a balance.

In general this is done using heuristics
and/or approximation.

Heuristics and Approximations

What can we do when complexity is high?

Looking for food to eat in the fridge

Picking an outfit

We manage to get by as best we can. How?

We trade correctness for efficiency.

“It’s good enough”

Generally want to strike a balance.

In general this is done using heuristics and/or approximation.

Heuristic

A “sufficient” strategy

More efficient, but not always correct

Reducing search space is a common heuristic

eg. Looking for vegetables in the fridge

Approximation

Guarantees the solution is “close” to the optimal

eg. Booking optimal travel itinerary

Due to large data size in bioinformatics, many algorithms employ these strategies

Thank you!

Welcome to the subject!

Don’t forget your signed academic integrity statement

Background survey on Poll Everywhere.
pollev.com/gracehall381

Today: Overview & Algorithms

Next time: Indexing

Melbourne Bioinformatics
BIOINFORMATICS + DATA SERVICES + INFRASTRUCTURE, FOR LIFE SCIENCES TODAY

POSTERA CRESCAM LAUDE
THE UNIVERSITY OF
MELBOURNE