COMP 3011
DESIGN AND ANALYSIS OF ALGORITHMS
FALL 2024

# Introduction

LI Bo
Department of Computing
The Hong Kong Polytechnic University

# INFORMATION

From the teaching team:
- Do not take attendance
- All lectures and tutorials will be recorded (for offline review use)

- ***Teaching Team***
  - Instructor: ***LI Bo (LI, Comp Bo)*** (PQ834, comp-bo.li@polyu.edu.hk )
    Feel free to contact me with whatever questions/concerns you have.
  - TA: ***XING Shiji*** (23038824r@connect.polyu.hk)

- ***Meeting Time***
  - Lectures: Wednesdays 11:30 – 13: 20 (Y306 & MS Teams) by the instructor
  - Tutorials: Mondays 11:30 – 12: 20 (Y306 & MS Teams) by TA (or the instructor)
  - Office Hours: Wednesdays 15:00 – 17:00 (PQ834) or by appointment

- ***References***
  - Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2022). Introduction to algorithms. MIT press.
  - Kleinberg J, Tardos E. Algorithm Design. Pearson Education, 2006.
  - Dasgupta S, Papadimitriou CH, Vazirani UV. Algorithms. New York: McGraw-Hill Higher Education; 2008.

# LEARNING OUTCOMES AND ASSESSMENT

***Learning Outcomes***

a. Understand basic and advanced techniques for designing algorithms;
b. Design algorithms for solving computing problems efficiently;
c. Analyze and compare the efficiency of algorithms; and
d. Implement efficient algorithms for solving computing problems in a high-level programming language (e.g., C++, Java, Phyton)*.

***Assessment***

- Continuous assessment (60%) + Final Examination (40%)
- Continuous assessment ***(tentative, from last year)***
  - Three Homework Assignments (30%)
  - One mini-project (e.g., Implement algorithms with real-world data) (10%)
  - One Midterm Exam (20%, Oct 30, 2024, tentatively)

# ASSIGNMENTS

Assignments are submitted in ***Blackboard (Learn@PolyU)***

- Assignments should be submitted ***before 11:59 pm*** on the due date.
- Scheduled assignment deadlines will be announced in Blackboard.

***Plagiarism***

- Plagiarism is the action of using or copying someone (including ChatGPT) else's idea or work and pretending that you thought of it or created it.
- Using ChatGPT to search study materials is allowed.
- Discussing with classmates/instructor/TA is encouraged!

***Late Policy*** (without excuse)

- 1 day late: 50% penalty
- 2 days late: no marks will be given

# TOPICS (TENTATIVE)

***Basic Algorithms*** (~ 6 Lec)

1) Algorithm Analysis
2) Graph Algorithms
3) Greedy Algorithms
4) Divide-and-Conquer
5) Dynamic Programming I
6) Dynamic Programming II

We may spend more time on the basic part.

***Advanced Algorithms*** (~ 6 Lec)

1) Geometry Algorithms
2) NP-Complete Problems
3) Approximation Algorithms
4) Randomized Algorithms
5) Advanced Data Structure
6) Linear Programming

We will choose some from the list.

# COMMENTS FROM PREVIOUS YEARS

***Before Midterm***

- Teach more new knowledge rather than shortest-path algorithms, sorting algorithms, divide-and-conquer algorithms that have been already taught several times in COMP2011, COMP2012, COMP2322, and other courses. As COMP2011 is a pre-requisite of this course, which is a compulsory for year-2 students.
- Add some difficulties
- To difficult

# COMMENTS FROM PREVIOUS YEARS

***After Final***

- For us exchange who were forced to join the department of computing (because of the contract with my home university) it was very hard to even follow the subject because we are not really computer science students so we had no idea about this subject
- BE Easier. The content before midterm is okay but after is soooooo difficult
- But still, it was too hard for us because of a lack of previous knowledge
- Lower the content coverage, and try to deep down the basics first
- There can be more practice examples and fewer contents in this lecture.

***Survey***: https://forms.gle/uDUNA6H7NiT2uuDe9
(Feel free to let me know your suggestions at the beginning of the subject!)

# TODAY' AGENDA

- ***Three examples:***
  - A math problem (Fibonacci numbers, how to save time)
  - A real-world problem (Counting handshakes, how to mathematically model a problem)
  - A Nobel prize problem (Stable matching, a taste of algorithm design and analysis)



- Algorithm Analysis, if time allows

# EXAMPLE 1
# A MATH PROBLEM

# REMEMBER FIBONACCI?

- ***Leonardo Fibonacci*** (Italian mathematician)
- But today Fibonacci is most widely known for his famous sequence of numbers

$$0, 1, 1, 2, 3, 5, 8, 13, 21, 34, \ldots$$

- More formally, the Fibonacci numbers $F_n$ are generated by the simple rule

$$F_n = \begin{cases} F_{n-1} + F_{n-2}, & n \geq 2 \\ 1, & n = 1 \\ 0, & n = 0 \end{cases}$$

- In fact, the Fibonacci numbers grow almost as fast as the powers of 2: for example, $F_{30}$ is over a million, and $F_{100}$ is already 21 digits long! In general, $F_n \approx 2^{0.694n}$.

# A DIRECT ALGORITHM

- By the recursive definition of $F_n$,

  function $fib1(n)$

  if $n = 0$: return 0    1 step

  if $n = 1$: return 1    1 step

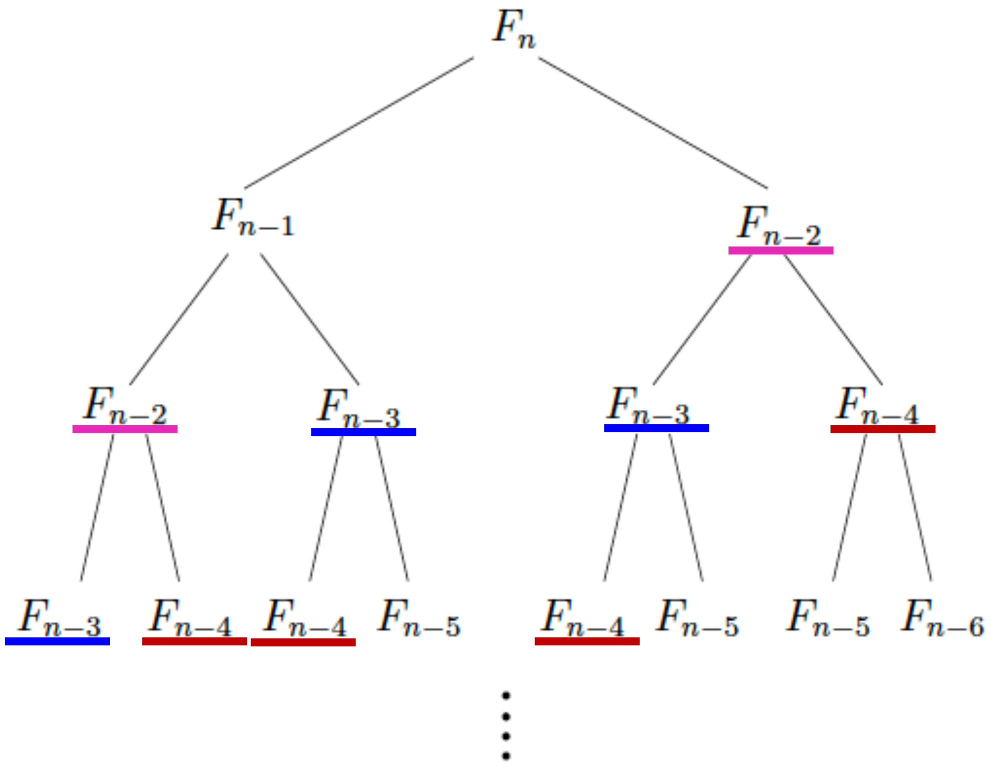
  If $n \leq 1$, $T(n) \leq 2$

  return $fib1(n-1) + fib1(n-2)$    $T(n) = T(n-1) + T(n-2) + 3$ for $n > 1$.

- Whenever we have an algorithm, there are three questions we always ask:
  - 1. Is it correct? ✓
  - 2. How much time does it take, as a function of $n$?    $T(n) \geq F_n \approx 2^{0.694n}$
  - 3. And can we do better?

# A DIRECT ALGORITHM

- For example, we want to know $F_{200}$, then $T(200) \geq F_{200} \geq 2^{138}$
- With the fastest computer, after $F_{100}$, we use one year to obtain $F_{101}$ and another year to get $F_{102}$, …
- Why is $fib1(n)$ slow?



# A POLYNOMIAL ALGORITHM

- Why not save the known results?

  function $fib2(n)$

  if $n = 0$ return 0    1 step

  create an array $f[0,1,\cdots,n]$    1 step

  $f[0] = 0$, $f[1] = 1$    2 steps

  for $i = 2,\cdots,n$:    $n-1$ rounds

       $f[i] = f[i-1] + f[i-2]$    1 step

  $\}$ $n-1$ steps

  return $f[n]$    1 step

- In total $\underline{T(n) = n + 4}$ steps.

  polynomial steps!!!

***Learning Objectives:***

b. Design algorithms for solving computing problems efficiently;

c. Analyze and compare the efficiency of algorithms; and

a. Understand basic (and advanced) techniques for designing algorithms;

# EXAMPLE 2
# A REAL-WORLD PROBLEM

# COUNTING PROBLEM

- People in a party may or may not shake hands with each other.
- Claim: There must be two people who made the same number of handshakes?

True or not?

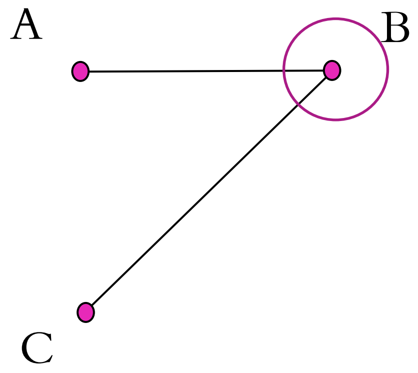
***Task 1:***
Model the problem mathematically

# PROBLEM MODELING

How can the problem be represented?

How are people represented?

How are handshakes represented?




- Person by a vertex (or node)
- Handshake between A and B by a line (edge) joining vertices A and B

"GRAPH" REPRESENTATION

"Graph" consists of vertices and edges (not charts).

For example: A, B, C are in the party

A shakes hands with B, and B with C

***Degree:*** the number of edges adjacent to B
= the number of handshakes made by that person

# COUNTING PROBLEM

- People in a party may or may not shake hands with each other.
- There must be two people who made the same number of handshakes?

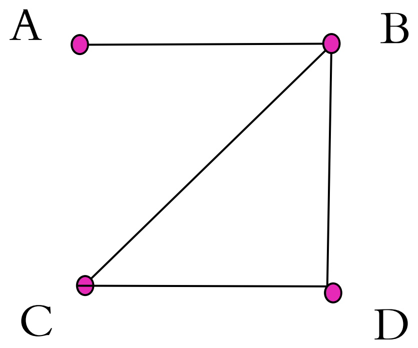
= Every graph (with at least 2 vertices) must have two vertices of equal degree.

***Task 1:***
Model the problem mathematically

# A TOY EXAMPLE

- A, B, C and D are in the party
  - A shakes hands with B
  - B shakes hands with D
  - D shakes hands with C
  - B shakes hands with C

GRAPH






- C and D have made the same number of handshakes or the same degree (degree-2 vertices), i.e, C and D shake hand with exactly 2 persons.

True or False?

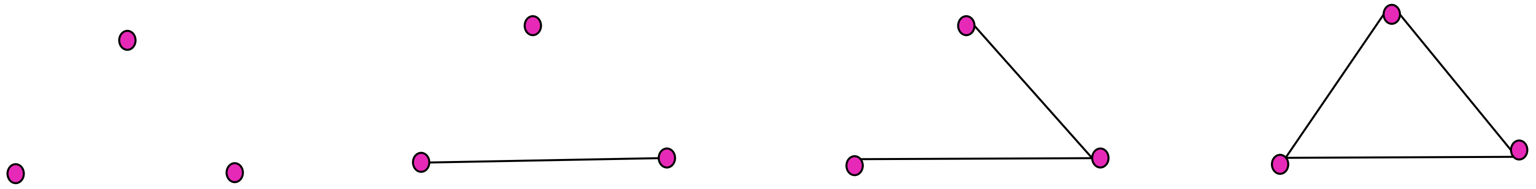
Every graph (with at least 2 vertices) must have two vertices of equal degree.

***Tips:***
Examples are helpful!

Proof by Exhaustive Enumeration?

# HANDSHAKING PROBLEM (CASE STUDY)

- Easy to prove for 2 persons
- For 3 persons, how many cases are there?



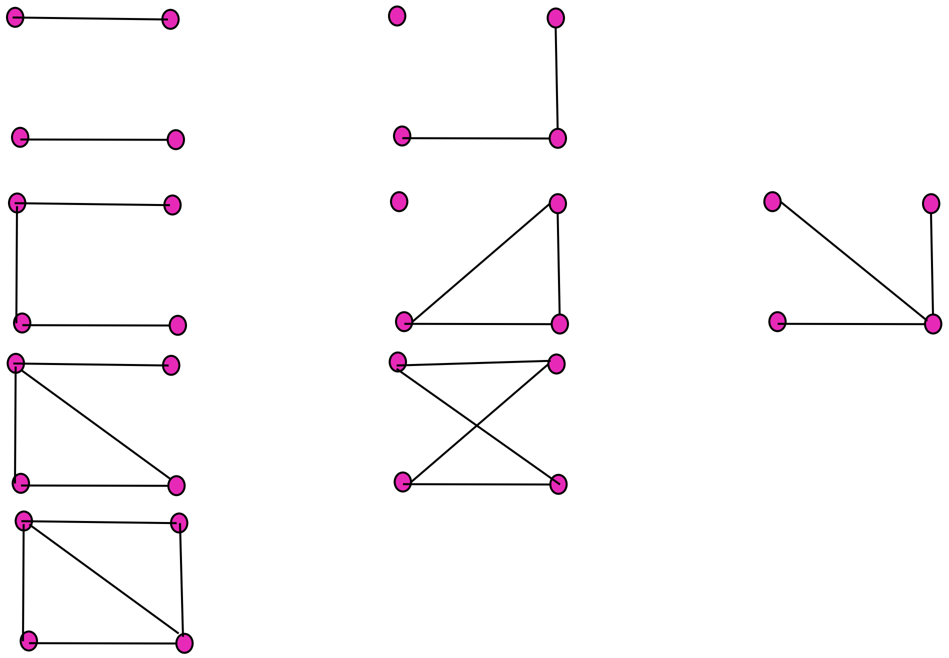
- There are 4 ***distinct*** cases: Proof by exhaustion (try all cases)
- For 4 persons, how many distinct cases are there?

  How can we ensure that we have exhausted all cases?

  A careful COUNTING is needed.

# EXHAUSTION FOR $n = 4$ PERSONS

- Easy to prove when the number of handshakes is 0 or 1.
- 2 handshakes:
- 3 handshakes:
- 4 handshakes:
- 5 handshakes:



| No of handshakes | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| No of cases | 1 | 1 | 2 | 3 | 2 | 1 | 1 |

# HANDSHAKING PROBLEM

- This claim is true
  - for any number of persons
  - for any number of handshakes
  - even some people do not shake hands with anyone



- Graph problem:

Given $n$ vertices, no matter how the edges are drawn, there are at least two vertices with the same number of degrees.

Proof?

# HANDSHAKING PROBLEM

***Tips:***
Proof by Contradiction!

- ***Theorem***. Given $n$ vertices, no matter how the edges are drawn, there are at least two vertices with the same number of degrees.

- ***Proof***. (By contradiction)
- For each vertex $v \in \{1,2,\cdots,n\}$, the degree of $v$ is at least 0 (i.e., no handshake) and at most $n-1$ (i.e., handshake with everyone else) → $n$ possibilities.
- If all $n$ vertices have different degrees, then the $n$ verrices and the $n$ possibilities have a one-to-one correspondence.
- Thus, there is one vertex with degree 0 (i.e., no handshake) and one vertex with degree $n-1$ (i.e., handshake with everyone else), which is a contradiction. ■

# EXAMPLE 3
# 2012 NOBEL PRIZE IN ECONOMICS

# STABLE MATCHING

- ***Lloyd Shapley***. Stable matching theory and Gale–Shapley algorithm.

**COLLEGE ADMISSIONS AND THE STABILITY OF MARRIAGE**

D. GALE* AND L. S. SHAPLEY, Brown University and the RAND Corporation

**1. Introduction.** The problem with which we shall be concerned relates to the following typical situation: A college is considering a set of $n$ applicants of which it can admit a quota of only $q$. Having evaluated their qualifications, the

- ***Alvin Roth***. Applied Gale–Shapley to matching med-school students with hospitals, students with schools, and organ donors with patients.

# STABLE MATCHING (MARRIAGE)

***Our Task:***
Assign students to hospitals!

***Problem Input:***

- A set of $n$ hospitals (girls) $H$ and a set of $n$ students (boys) $S$.
- Each hospital $h \in H$ ranks students.
- Each student $s \in S$ ranks hospitals.

$H$ ={Atlanta, Boston, Chicago}

$S$ ={Xavier, Yolanda, Zeus}.



| | 1st (favorite) | 2nd | 3rd (least favorite) |
|---|---|---|---|
| **Atlanta** | Xavier | Yolanda | Zeus |
| **Boston** | Yolanda | Xavier | Zeus |
| **Chicago** | Xavier | Yolanda | Zeus |

**hospitals' preference lists**

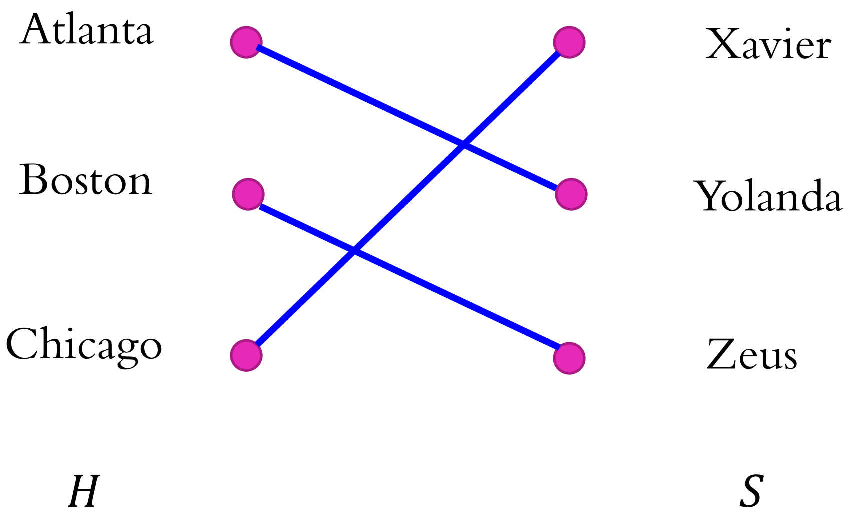
| | 1st (favorite) | 2nd | 3rd (least favorite) |
|---|---|---|---|
| **Xavier** | Boston | Atlanta | Chicago |
| **Yolanda** | Atlanta | Boston | Chicago |
| **Zeus** | Atlanta | Boston | Chicago |

**students' preference lists**

# PERFECT MATCHING

***Definition.*** A *matching* $M$ is a set of ordered pairs $h - s$ with $h \in H$ and $s \in S$ s.t.

- Each hospital $h \in H$ appears in at most one pair of $M$.
- Each student $s \in S$ appears in at most one pair of $M$.




“GRAPH” REPRESENTATION

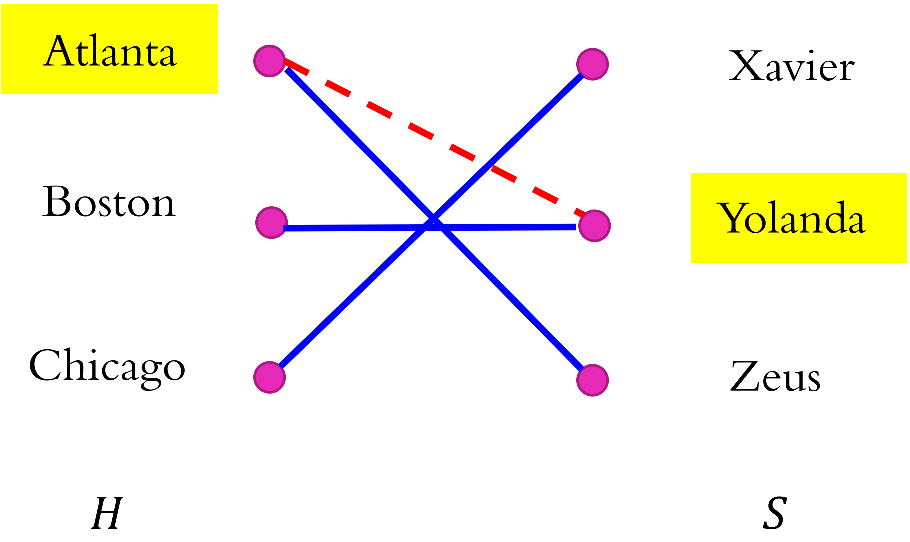
***How to evaluate the matching?***

- All hospitals/students are matched!
- Hospitals/students can get the assignment they like.

***Definition.*** A matching $M$ is *perfect* if $|M| = |H| = |S| = n$.

# UNSTABLE PAIR

(Atlanta – Yolanda) forms an unstable pair




| | favorite ↓ 1st | 2nd | least favorite ↓ 3rd |
|---|---|---|---|
| Atlanta | Xavier | Yolanda | Zeus |
| Boston | Yolanda | Xavier | Zeus |
| Chicago | Xavier | Yolanda | Zeus |

**hospitals' preference lists**

| | favorite ↓ 1st | 2nd | least favorite ↓ 3rd |
|---|---|---|---|
| Xavier | Boston | Atlanta | Chicago |
| Yolanda | Atlanta | Boston | Chicago |
| Zeus | Atlanta | Boston | Chicago |

**students' preference lists**

# UNSTABLE PAIR

***Definition.*** Given a perfect matching $M$, hospital $h$ and student $s$ form an *unstable pair* if both

- $h$ prefers $s$ to its matched student; and
- $s$ prefers $h$ to its matched hospital.



***Key point.*** An unstable pair $h - s$ could each improve by joint action.

| | 1st | 2nd | 3rd |
|---|---|---|---|
| **Atlanta** | Xavier | Yolanda | Zeus |
| **Boston** | Yolanda | Xavier | Zeus |
| **Chicago** | Xavier | Yolanda | Zeus |

| | 1st | 2nd | 3rd |
|---|---|---|---|
| **Xavier** | Boston | Atlanta | Chicago |
| **Yolanda** | Atlanta | Boston | Chicago |
| **Zeus** | Atlanta | Boston | Chicago |

**A–Y is an unstable pair for matching M = { A–Z, B–Y, C–X }**

# UNSTABLE PAIR

One more exercise:

➢ Which pair is unstable in the matching {A–X, B–Z, C–Y}?

| | 1st | 2nd | 3rd |
|---|---|---|---|
| Atlanta | Xavier | Yolanda | Zeus |
| Boston | Yolanda | Xavier | Zeus |
| Chicago | Xavier | Yolanda | Zeus |

| | 1st | 2nd | 3rd |
|---|---|---|---|
| Xavier | Boston | Atlanta | Chicago |
| Yolanda | Atlanta | Boston | Chicago |
| Zeus | Atlanta | Boston | Chicago |

# STABLE MATCHING PROBLEM

***Definition.*** A ***stable matching*** is a perfect matching with no unstable pairs.

***Stable matching problem.*** Given the preference lists of $n$ hospitals and $n$ students, find a stable matching (if one exists).



| | 1st | 2nd | 3rd |
|---|---|---|---|
| **Atlanta** | Xavier | Yolanda | Zeus |
| **Boston** | Yolanda | Xavier | Zeus |
| **Chicago** | Xavier | Yolanda | Zeus |

| | 1st | 2nd | 3rd |
|---|---|---|---|
| **Xavier** | Boston | Atlanta | Chicago |
| **Yolanda** | Atlanta | Boston | Chicago |
| **Zeus** | Atlanta | Boston | Chicago |

**a stable matching M = { A–X, B–Y, C–Z }**

## 1. DO STABLE MATCHINGS ALWAYS EXIST?

## 2. CAN WE COMPUTE SUCH A SOLUTION EFFICIENTLY?



If you are a hospital, what will you do?

- Propose to your favorite student

What will you do if you are the proposed student?

- If you do not have a better option, accept; otherwise, reject.

# GALE–SHAPLEY ALGORITHM

GALE–SHAPLEY (*preference lists for hospitals and students*)

---

INITIALIZE $M$ to empty matching. We usually initialize the output as empty in the first step.

WHILE (some hospital $h$ is unmatched and hasn't proposed to every student)

  $s \leftarrow$ first student on $h$'s list to whom $h$ has not yet proposed.

  IF ($s$ is unmatched)

    Add $h$–$s$ to matching $M$. $(h - s)$ form a temporary pair

  ELSE IF ($s$ prefers $h$ to current partner $h'$)

    Replace $h'$–$s$ with $h$–$s$ in matching $M$. If $h$ is a better man for $s$, she rejects $h'$ and matches to $h$.

  ELSE

    $s$ rejects $h$.

$$h\colon \underbrace{A > B > C >}\ \overset{\displaystyle s}{\overset{\downarrow}{D}} > E > F > \cdots$$

RETURN stable matching $M$.

We do not propose to people who rejected us.

# GALE–SHAPLEY ALGORITHM

***Observation 1***. *Hospitals propose to students in decreasing order of preference.*

$$h\colon A > B > C > D > E > F > \cdots$$

***Observation 2***. *Once a student is matched, the student never becomes unmatched; only "trades up".*



***Claim.*** *Algorithm terminates after at most $n^2$ iterations of WHILE loop.*

***Proof.***

- Each time through the WHILE loop, a hospital proposes to a new student.
- Thus, there are at most $n^2$ possible proposals. ■

# GALE–SHAPLEY ALGORITHM

***Claim.*** *Gale–Shapley outputs a matching.*

***Proof.***

- Hospital proposes only if unmatched $\Rightarrow$ matched to $\leq 1$ student
- Student keeps only best hospital $\Rightarrow$ matched to $\leq 1$ hospital



***Claim*.*** *In Gale–Shapley matching, all hospitals get matched.*

***Proof.*** [by contradiction]

- Suppose, for sake of contradiction, that some hospital $h \in H$ is unmatched upon termination of Gale–Shapley algorithm.
- Then some student, say $s \in S$, is unmatched upon termination.
- By ***Observation 2***, $s$ was never proposed to.
- But $h$ proposes to every student, since $h$ ends up unmatched. ■

# GALE–SHAPLEY ALGORITHM

***Claim.*** *In Gale–Shapley matching, all students get matched.*

***Proof.*** [by counting]

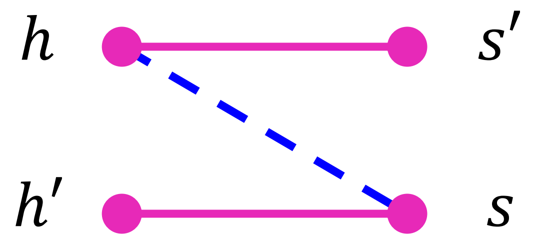
- By previous claim, all $n$ hospitals get matched.
- Thus, all $n$ students get matched. ■



***Claim*.*** *In Gale–Shapley matching $M^*$, there are no unstable pairs.*

***Ideas.***

- Consider any pair $h - s$ that is not in $M^*$.
- Need to show the pair $h - s$ is not unstable.
  - Case 1: $h$ never proposed to $s$.
  - Case 2: $h$ proposed to $s$.




Gale–Shapley matching $M^*$

# GALE–SHAPLEY ALGORITHM

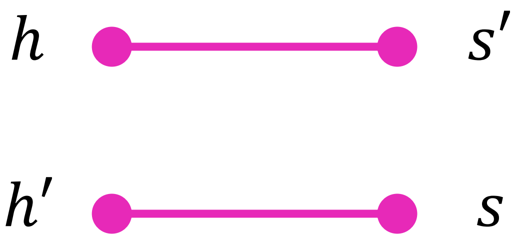
***Claim*.*** *In Gale–Shapley matching $M^*$, there are no unstable pairs.*

***Proof.***

- Consider any pair $h - s$ that is not in $M^*$.
- Case 1: $h$ never proposed to $s$.
  - $\Rightarrow$ $h$ prefers its Gale–Shapley partner $s'$ to $s$.
  - $\Rightarrow$ $h - s$ is not unstable.
- Case 2: $h$ proposed to $s$.
  - $\Rightarrow$ $s$ rejected $h$ (either right away or later)
  - $\Rightarrow$ $s$ prefers Gale–Shapley partner $h'$ to $h$.
  - $\Rightarrow$ $h - s$ is not unstable.
- In either case, the pair $h - s$ is not unstable. ■

hospitals propose in decreasing order of preference

students only trade up




Gale–Shapley matching $M^*$

# SUMMARY

***Stable matching problem.***

*Given $n$ hospitals and $n$ students, and their preference lists, find a stable matching if one exists.*

***Theorem.*** [Gale–Shapley 1962]

*The Gale–Shapley algorithm guarantees to find a stable matching for any problem instance.*

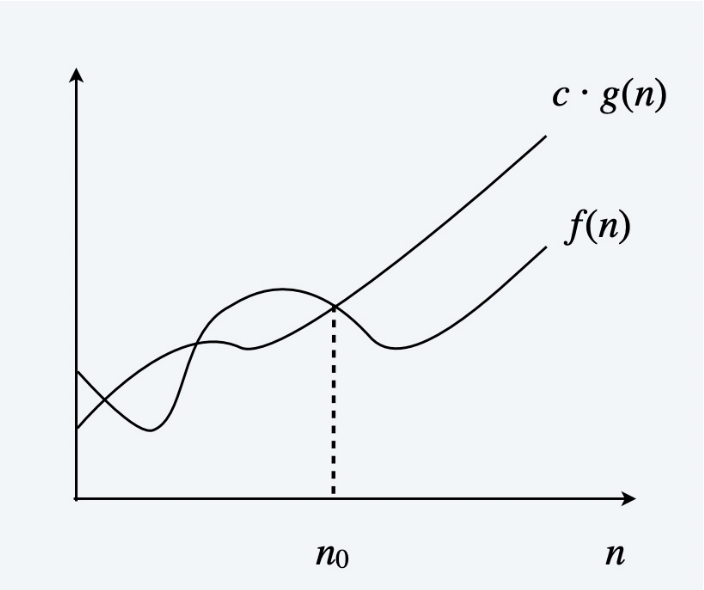
# ORDER OF GROWTH (BIG O NOTATION)

# BIG O NOTATION

*Upper bounds*

- $f(n)$ is $O(g(n))$ if there exist constants $c > 0$ and $n_0 \geq 0$ such that

$$0 \leq f(n) \leq c \cdot g(n) \text{ for all } n \geq n_0.$$




- Examples: $f(n) = 32n^2 + 17n + 1$
  - $f(n) = O(n^2)$ ✓
  - $f(n) = O(n^3)$ ✓
  - $f(n) = O(n)$ ✗

Set $n_0 = 1$, then for $n \geq n_0$

$$f(n) = 32n^2 + 17n + 1 \leq 32n^2 + 17n^2 + n^2 \leq 50n^2$$

$$\leq 50n^3$$

# BIG O NOTATION

$f(n) = 32n^2 + 17n + 1$

$f(n) \neq O(n)$

- $f(n) = O(g(n))$ if there exist constants $c > 0$ and $n_0 \geq 0$ such that

$$0 \leq f(n) \leq c \cdot g(n) \text{ for all } n \geq n_0.$$

- $f(n) \neq O(g(n))$ if for any constants $c > 0$ and $n_0 \geq 0$ such that

$$f(n) > c \cdot g(n) \text{ for some } n \geq n_0.$$

For any constants $c > 0$ and $n_0 \geq 0$, let $n = (c + n_0)$

$$f(n) = 32n^2 + 17n + 1 \geq 32n^2 \geq 32(c + n_0)^2 > c \cdot (c + n_0) = c \cdot n.$$

# BIG O NOTATION

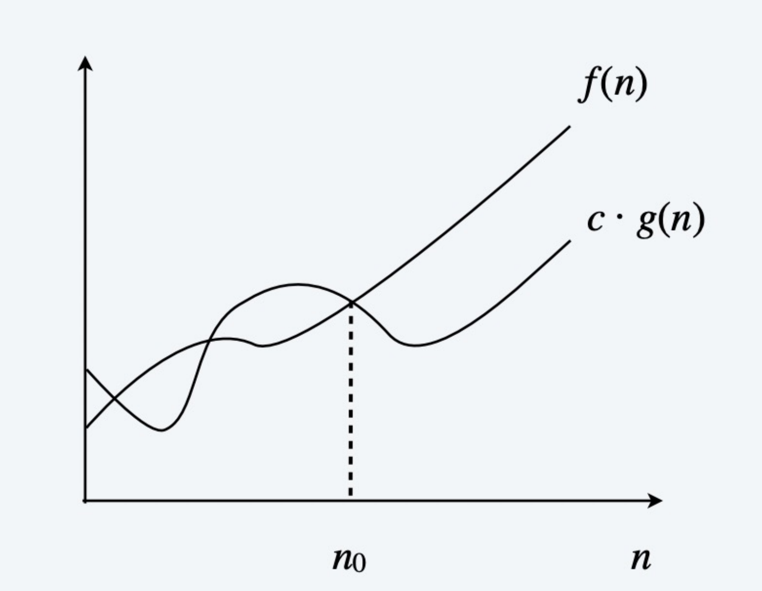
***Lower bounds***

- $f(n)$ is $\Omega(g(n))$ if there exist constants $c > 0$ and $n_0 \geq 0$ such that

$$f(n) \geq c \cdot g(n) \geq 0 \text{ for all } n \geq n_0.$$

- Examples: $f(n) = 32n^2 + 17n + 1$
  - $f(n) = \Omega(n^2)$ ✓
  - $f(n) = \Omega(n)$ ✓
  - $f(n) = \Omega(n^3)$ ✗






Set $n_0 = 1$, then for $n \geq n_0$

$$f(n) = 32n^2 + 17n + 1 \geq 32 \cdot n^2$$

$$\geq 32 \cdot n$$

# BIG O NOTATION

$f(n) = 32n^2 + 17n + 1$

To show $f(n) \neq \Omega(n^3)$

- $f(n) = \Omega(g(n))$ if there exist constants $c > 0$ and $n_0 \geq 0$ such that

$$f(n) \geq c \cdot g(n) \geq 0 \text{ for all } n \geq n_0.$$

- $f(n) \neq \Omega(g(n))$ if for any constants $c > 0$ and $n_0 \geq 0$ such that

$$f(n) < c \cdot g(n) \text{ for some } n \geq n_0.$$

- *For any constants $c > 0$ and $n_0 \geq 0$, let $n = c + n_0 + 50$

$$f(n) = 32n^2 + 17n + 1 \leq 50n^2 \leq 50(c + n_0 + 50)^2 < (c + n_0 + 50)^3 = n^3.$$

# BIG O NOTATION

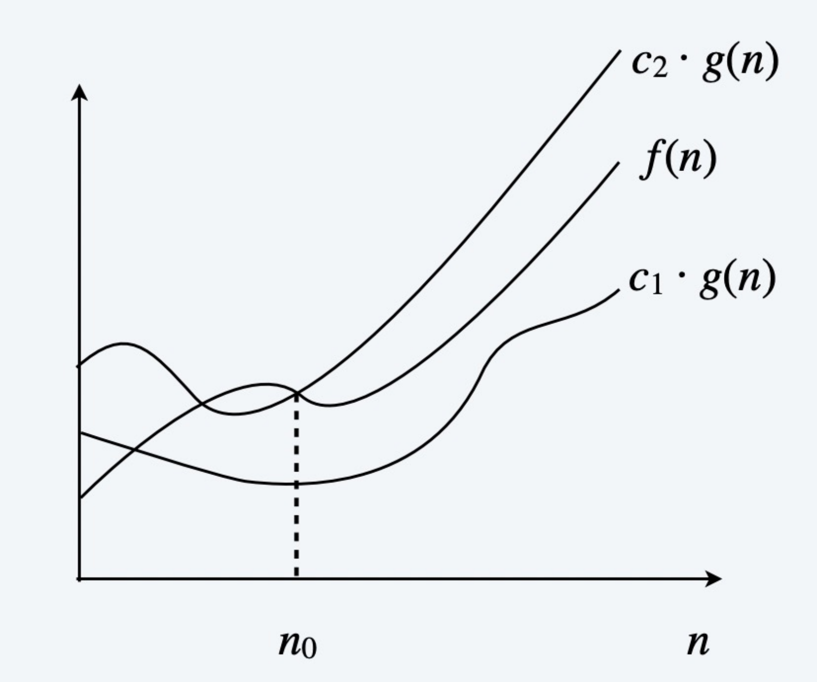
$f(n) = \Theta(g(n))$ if and only if
$f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$

***Tight bounds***

- $f(n)$ is $\Theta(g(n))$ if there exist constants $c_1 > 0, c_2 > 0$ and $n_0 \geq 0$ such that

$$c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n) \text{ for all } n \geq n_0.$$

- Examples: $f(n) = 32n^2 + 17n + 1$
  - $f(n) = \Theta(n^2)$ ✓
  - $f(n) = \Theta(n^3)$ ✗
  - $f(n) = \Theta(n)$ ✗



# ASYMPTOTIC BOUNDS AND LIMITS

***Propositions***

- If $\lim_{n\to\infty} \frac{f(n)}{g(n)} = c$ for some constant $0 < c < \infty$ then $f(n) = \Theta(g(n))$.

***Proof:***

- By definition of the limit, for any $\epsilon > 0$, there exists $n_0$ such that

$$c - \epsilon \leq \frac{f(n)}{g(n)} \leq c + \epsilon \text{ for all } n \geq n_0.$$

- Let $\epsilon = \frac{1}{2}c$. Then

$$\boxed{\frac{1}{2}c} \cdot g(n) \leq f(n) \leq \boxed{\frac{3}{2}c} \cdot g(n) \text{ for all } n \geq n_0. \blacksquare$$

- If $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$ for some constant $0 < c < \infty$ then $f(n) = O(g(n))$ but not $\Omega(g(n))$.
- If $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$ for some constant $0 < c < \infty$ then $f(n) = \Omega(g(n))$ but not $O(g(n))$.

# ASYMPTOTIC BOUNDS FOR SOME COMMON FUNCTIONS

***Polynomials.***

- Let $f(n) = a_0 + a_1 n + \cdots + a_d n^d$ with $a_d > 0$. Then, $f(n) = \Theta(n^d)$.

***Logarithms.***

- $\log_a n = \Theta(\log_b n)$ for every constants $a > 1$ and $b > 1$.

***Logarithms and polynomials.***

- $\log_a n = O(n^d)$ for every $a > 1$ and $b > 0$.

***Exponentials and polynomials.***

- $n^d = O(r^n)$ for every $r > 1$ and every $d > 0$.

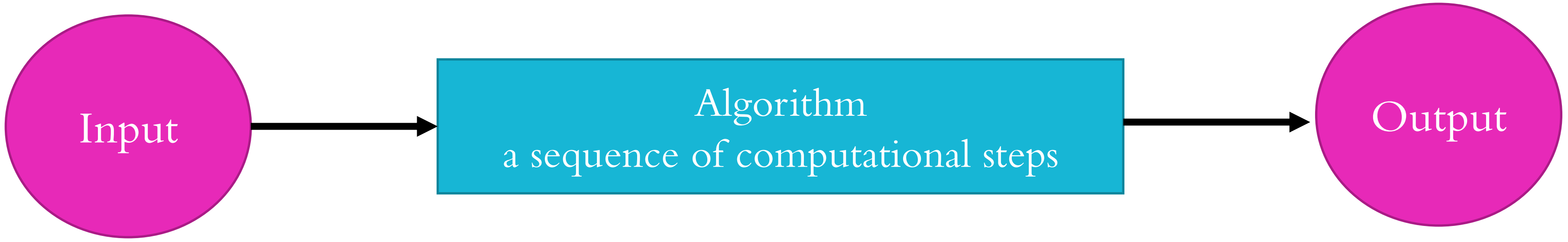
***Factorials.***

- $n! = 2^{\Theta(n \log n)}$.

# ALGORITHM ANALYSIS

# WHAT ARE ALGORITHMS?

➢ Informally, an ***algorithm*** is any well-defined computational procedure that takes some value, or set of values, as input and produces some value, or set of values, as output.





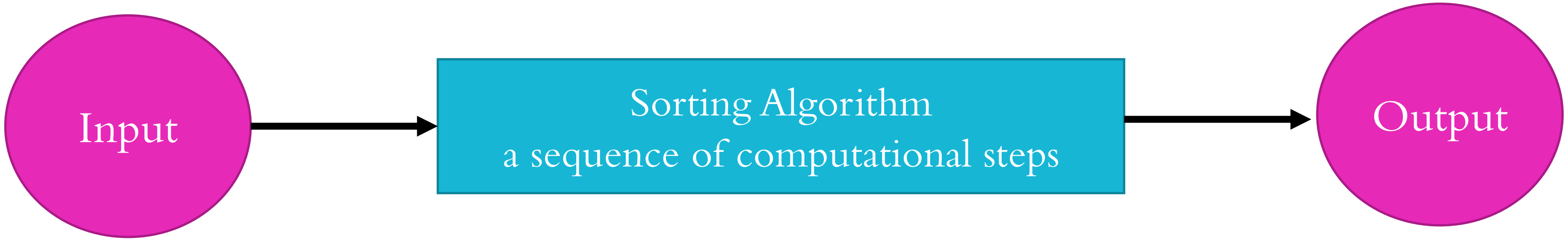

➢ E.g., ***Sorting problem***: we need to sort a sequence of numbers into nondecreasing order.

➢ ***Input:*** A sequence of $n$ numbers $(a_1, \cdots, a_n)$.

➢ ***Output:*** A permutation (reordering) $(a_1', \cdots, a_n')$ of the input sequence such that

$$a_1' \leq a_2' \leq \cdots \leq a_n'.$$

# WHAT ARE ALGORITHMS?




$(31, 41, 59, 26, 41, 58)$ (26, 31, 41, 41, 58, 59)

- A (Computational) ***Problem***

  A statement of the problem specifies in general terms the desired input/output relationship.

- An ***Instance*** (of a problem)

  Consists of the input (satisfying whatever constraints are imposed in the problem statement) needed to compute a solution to the problem.

# ANALYZING ALGORITHMS

- Space Complexity: Memory
- Time Complexity: Running time ???

- Primitive Operations:
  - Arithmetic (such as add, subtract, multiply, divide, remainder, floor, ceiling),
  - Logic operations (and, or)
  - Read/write memory
  - Array indexing
  - Following a pointer
  - Data movement (load, store, copy)
  - Control (conditional and unconditional branch, subroutine call and return)
- Each such instruction takes a constant amount of time.

# ANALYZING ALGORITHMS

The ***running time*** of an algorithm on a particular input is the number of primitive operations or "steps" executed:

- A constant amount of time is required to execute each line of our pseudocode.
- As machine-independent as possible.
- A function of the input size $n$.

# ANALYZING ALGORITHMS

GALE–SHAPLEY (*preference lists for hospitals and students*)

---

INITIALIZE $M$ to empty matching.

WHILE (some hospital $h$ is unmatched and hasn't proposed to every student)

  $s \leftarrow$ first student on $h$'s list to whom $h$ has not yet proposed.

  IF ($s$ is unmatched)

    Add $h$–$s$ to matching $M$.

  ELSE IF ($s$ prefers $h$ to current partner $h'$)

    Replace $h'$–$s$ with $h$–$s$ in matching $M$.

  ELSE

    $s$ rejects $h$.

RETURN stable matching $M$.

**Claim.** *The algorithm terminates after at most $n^2$ iterations of WHILE loop.*

# ANALYZING ALGORITHMS

The ***running time*** of an algorithm on a particular input is the number of primitive operations or "steps" executed:

- A constant amount of time is required to execute each line of our pseudocode.
- As machine-independent as possible.
- A function of the input size $n$.

We use ***worst-case running time***: the longest running time for any input of size $n$.

- Worst-case running time gives us an upper bound on the running time for any input (the algorithm will never take any longer).
- The worst case occurs fairly often.
- The "average case" is often roughly as bad as the worst case.

***Thank you!***