SINGAPORE INSTITUTE OF TECHNOLOGY

Introduction to Algorithms

Algorithm Analysis

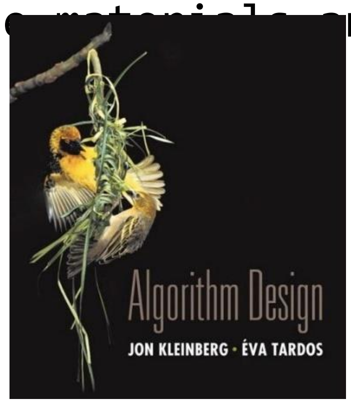
Learning Objectives

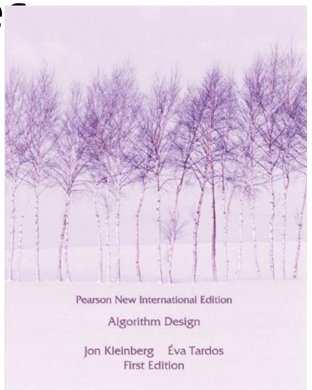
- **Overall**: To promote individual learning and **analytical thinking skills**.

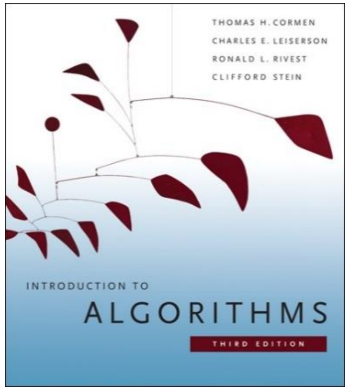
- More detailed objectives:
  1. Be able to **prove correctness** of algorithms.
     - Some people spend their whole career to prove something.
  2. Have deep understanding of differences and advantages of **iterative and recursive algorithms**.
  3. Have a knowledge base of (many) **existing algorithms**.
  4. Understand **speed vs. space trade-off**, importance of **data-structures,** and **data preprocessing**.
     - Search for an item: hash, tree, and traverse all.
  5. Be able to **design new algorithms**.
     - Correctness.
     - Efficiency.

Overview

- Acknowledgement: **Dr. Zhang Wei, Dr. LIU Fang, Dr. Donny** - course materials
- Appointment by email:
  - Pohkok.Loo@singaporetech.edu.sg
  - AF_Alfred.Whang@singaporetech.edu.sg
- References:
  - **Algorithm Design**, Jon Kleinberg and Eva Tardos.
  - **Introduction to Algorithms**, Thomas Cormen, Charles Lieserson, *et al*.
  - Introduction to the Design and Analysis of Algorithms, Anany Levitin.
  - Course materials and activities







Source: https://kwize.com/quote/9797; https://eden.uktv.co.uk/animals/birds/article/weaver-birds/

Tentative Dates and Grading Policy

- **Marks:**
  - 28% - Assignment (group of 5/6 students)
  - 12% - Tutorial participation/quiz – no makeup
  - 30% - Midterm
  - 30% - Final Exam

- Part 1: week 1 to 6 (Loo)
  - 1 assignment (group 14%), week 5, Due on 25/9/23 23:59
  - 5 tutorial participation/quiz (6%), during each tutorial
  - 1 midterm (30%), week 6, 3/10/23 2-3pm

- Part 2: week 8 to 13 (Alfred)
  - 1 assignment (group 14%), week 12, Due on 13/11/23
  - 5 tutorial participation/quiz (6%), during each tutorial
  - 1 Final (30%), week 13, 21/11/23 (TBA)

Tentative Dates and Grading Policy

- Blended learning
  - All lecture content/video/zoom recording will be posted before the tutorial.
  - The key face to face session is during the tutorial where you will work on tutorial questions, ask questions and complete quizzes.

- Enrol yourself in xSit follows the below guide. Minor variation may occur due to dynamic enrolment add/drop situation.
  - P1: 5 groups of 6 students, 2 groups of 5 students
  - P2: 4 groups of 6 students, 3 groups of 5 students
  - P3: 5 groups of 6 students, 2 groups of 5 students
  - P4: 6 groups of 6 students, 1 groups of 5 students
  - Can only join students within the same tutorial group

Tentative Dates and Grading Policy

- Peer review
  - If a group with 5 students received 72 for the assignment, then

Student	Contribution	Final score
S1	80/100	80/120*72=48.0
S2	95/100	95/120*72=50.5
S3	95/100	95/120*72=50.5
S4	110/100	110/120*72=66.0
S5	120/100	120/120*72=72.0
	Max=120	

- Be careful when allocating percentage for each member. It should not vary too much.
- A student contribution = average contribution given by all other members and yourself.

Stable Matching: Example

- History: A self-enforcing college admission process, or job recruiting process.
  - Each **student** has his/her **preference list** of the universities.
  - Each **university** has her **preference list** of the students.
  - Normal that the preferences **do not align** perfectly.
  - **How** to make both sides **satisfied**?

Stable Matching: Example

- **Example** to be more intuitive (in time order):
  - (May 1) Jon accepted the offer from Apple.
  - (May 2) Google offers Jon and Jon likes Google more.
  - (May 3) Jon retracts his acceptance of Apple and accepts Google’s offer. (Apple has one free slot now.)
  - (May 4) Apple soon offers David and he likes Apple more.
  - (May 5) David retracks his Facebook offer and accepts Apple’s offer. (Facebook has one free slot now.)
  - (May 6) Facebook moves to X, and X turns down Y and accepts Facebook.
  - (May 7) Y moves to Z ... Situation gets **out of control**.

Stable Matching: Example

- Things can be different, possibly even worse.
  - Jon, David and the others have communication channels.
  - They hear something and accordingly do something.
  - The **network** of the communication will be **fully connected** with around $O(n^2)$ links.
    - Suppose 50 students, 50 companies, then we have 10,000 communication links.
    - Each link can trigger a change of the offer distribution.
    - **Chaos** happens. Everyone tired and unhappy.
- Basic issue?
  - People **cannot act in their** **self-interest**, otherwise system may crash.

Source: http://ictusmarketing.com/blog/2014/5/23/what-happens-when-the-buyer-knows-more-than-the-seller

Stable Matching: Example

- Basic issue?
  - People **cannot act in their** **self-interest**, otherwise system may crash.
- Our desired process – **stable situation**:
  - Use **intelligence** to **guide** the process.
  - **Idea: self-interest prevents retracting offers.**
    - Company, with its offers all being accepted, gets a call from another applicant
    - -> “No, we prefer each applicant we have accepted than you, so we will not change.” **Chaos avoided**.
    - Student, with an offer, gets a call from a university.
    - -> “No. I am happy where I am.” No chaos.

Source: http://ictusmarketing.com/blog/2014/5/23/what-happens-when-the-buyer-knows-more-than-the-seller

Stable Matching

- Question: given employer E and applicant $A \notin E$, can we have:
  - E **prefers her accepted applicants** than A (another applicant).
  - A **prefers his/her current employer** over working for E.
- Obj: design **algorithm** -> produce **stable situations**.

- Similar application: residents and hospitals matching.
  - Even a decade earlier than the employee and employer one.
  - Still widely used today in USA.
- Intuition: personal interest & system interest.
- **Problem formulation -> algorithm design -> algorithm analysis.**

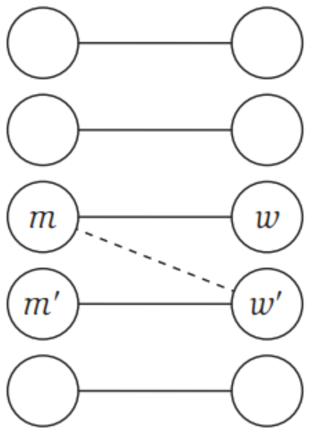
Source: https://www.channelfutures.com/files/2013/12/secretexchange_0.jpg; https://builtin.com/sites/default/files/styles/og/public/2019-04/Machine-Learning-in-Healthcare.png

Problem Formulation

- Problem: As **clean** as possible. As **simple** as possible. **Improve** step by step.
  - Apply to programming as well.
- Complex situation: a company can hire multiple employees.
  - 1-to-many relationship.
- Simple: **marriage problem**.
  - A man can only marry a woman.
  - A woman can only marry a man.
  - **1-to-1 relationship**.
- Math definition.
  - n men: $M = \{m_1, m_2, \cdots, m_n\}$.
  - n women: $W = \{w_1, w_2, \cdots, w_n\}$.
  - **Matching**: S, a set of pairs, pair $(m, w) \in S$ has a man $m \in M$ and a woman $w \in W$.
    - Requirement: each man/woman only appears **at most once** in S.
  - **Perfect matching**: everyone married & married to one person. Or $|M| = |W| = |S| = n$.

Problem Formulation

- **Preference**: each man m ranks all women. The order is m's **preference list.**
  - m prefers w to w' if he ranks w higher than w'.
  - Ties not allowed.
  - Each woman w also ranks all the men, also her preference list.
- Perfect matching all we want? (All men and women in the matching)
- **Unstable pair** (m, w) and (m', w') in a perfect matching S:
  - m prefers w'.
  - w' prefers m.
  - (m, w') is called an **instability**. The pair is not in S.
  - m and w' prefer each other but they are assigned to a different person
  - Objective: find a **perfect matching, without** any **instability** -> **stable matching**.
- Questions:
  - Is stable matching **possible?**
  - If true, **how** to find it fast?

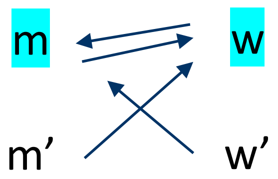


https://lh3.googleusercontent.com/proxy/mSGoaVr1Xw8QKWjYbqN6EUi_IgUtadnluD1kz455cZCHIeuoO8mt49NtMr6FxIl2r-BhqM_0pf_ThtI9rG7ls4D58WILssI-SxTq9LWy6KyV

Possible – Proven with Examples

SINGAPORE INSTITUTE OF TECHNOLOGY

- Two men m, m' and two women w, w'.
  - m prefers w to w'
  - m' prefers w to w'
  - w prefers m to m'
  - w' prefers m to m'
- Do we have complete agreement?
  - The men agree on the order of women.
  - The women agree on the order of men.
  - **Unique stable matching**: (m, w) and (m', w').
    - Although m' prefers w, w didn't prefer m'
- Another perfect matching (m, w') and (m', w).
  - Is this stable? No (m and w prefer each other)
- A more complicated example in the next slide.



Focus is the pair where they want each other (m,w)

$n = 2$, n^2 possibilities, $2^2=4$
(m,w), (m,w'), (m',w), (m',w')
6 possible combinations
(m,w), (m,w') – not possible
(m,w), (m',w) – not possible
(m,w), (m',w') - ok
(m,w'), (m',w) – ok, but unstable
(m,w'), (m',w') – not possible
(m',w), (m',w') – not possible

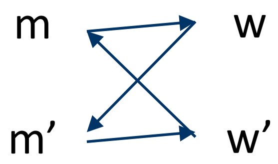
m and w wants each other but now they are both refused their choice and with others

We have a situation where 2 parties want each other but yet they have someone else

https://movies.disney.com/aladdin-2019

Possible – Proven with Examples

- A more complicated example.
  - m prefers w to w'
  - m' prefers w' to w
  - w prefers m' to m
  - w' prefers m to m'



No situation where both parties want each other but yet get someone else

- Men's preferences meshes well -> (m, w) and (m', w') perfect for men.
- Women's preferences meshes well -> (m, w') and (m', w) perfect for women.
- Any problem?
  - Both are **stable matchings**, but they **clash**.
  - No happy ending for all men and women.
- Conclusion: $\geq$ **1 stable matching possible**.

(m,w), (m,w'), (m',w'), (m',w)
6 possible combinations
(m,w), (m,w') – not possible
(m,w), (m',w') – ok (fr men)
(m,w), (m',w) – not possible
(m,w'), (m',w') – not possible
(m,w'), (m',w) – ok (fr woman)
(m',w'), (m',w) – not possible

https://i.pinimg.com/originals/be/2c/73/be2c73938c39dfa2a8701fedc6492b91.gif

Design the Algorithm

- Our plan:
  - Show that a **stable matching exists**.
  - Propose an **efficient algorithm** accordingly.
- Some basic ideas – 3 Stages.
  - **Free**. In the beginning.
  - **Engagement**. Intermediate state. (Still can change)
  - **Married**. The algorithm or matching terminates. Engagements declared final.
- Why engage?
  - Bob likes Alice the most and proposes to her.
  - Shall Alice accept?
    - Accepts: what if Alice loves Jon the most and Jon proposes to her later? Regret?
    - If not, what if Bob is the best she can get? Lonely in the end?
  - **Engagement** is helpful to deal with the above situation.
- Procedure: all free in the beginning, eventually get engaged, and finally married.

https://www.cartier.sg/?gclid=Cj0KCQjw2PP1BRCiARIsAEqv-pQfx7w6lmf_nHEQkUPGq77fndswenymmP3fzaWULRVlLTxOOsWuo3gaAtVCEALw_wcB&gclsrc=aw.ds

Gale-Shapley Algorithm

```
Initially all m ∈ M and w ∈ W are free
While there is a man m who is free and hasn't proposed to
every woman
   Choose such a man m
   Let w be the highest-ranked woman in m's preference list
      to whom m has not yet proposed
   If w is free then
      (m, w) become engaged
   Else w is currently engaged to m'
      If w prefers m' to m then
         m remains free
      Else w prefers m to m'
         (m, w) become engaged
         m' becomes free
      Endif
   Endif
Endwhile
Return the set S of engaged pairs
```

- Can return a **stable matching?**
- How to **prove** our judgement?

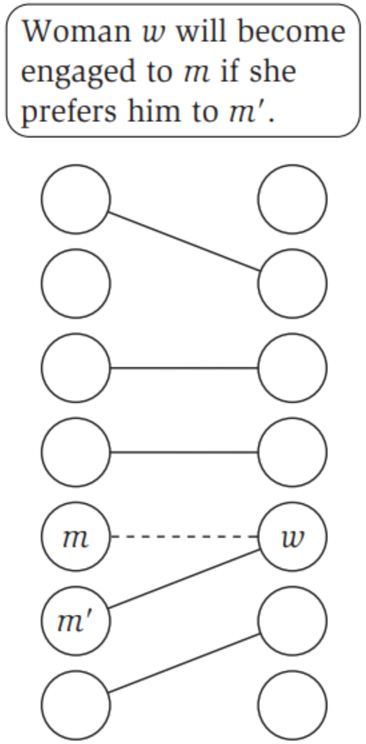


Figure 1.2 An intermediate state of the G-S algorithm when a free man m is proposing to a woman w.

Kleinberg J, Tardos E. Algorithm design. Pearson Education; 2006.

Demo 1

Iteration	Man Proposes to	Woman	Engagement	Free Males	Reason
0				A, B, C	
1					
2					
3					
4					
5					
6					

Man	Preference List	Woman	Preference List
A	w_1, w_2, w_3	w_1	C, B, A
B	w_1, w_2, w_3	w_2	C, B, A
C	w_1, w_2, w_3	w_3	C, B, A

Demo 1 - Result

Male proposes, but woman get to select the most preferred man

Iteration	Man Proposes to	Woman	Engagement	Free Males	Reason
0				A, B, C	
1	A	w_1	(A, w_1)	B, C	1st proposal to w_1
2	B	w_1	(B, w_1)	A, C	w_1 likes B more than A
3	A	w_2	(B, w_1), (A, w_2)	C	1st proposal to w_2
4	C	w_1	(C, w_1), (A, w_2)	B	w_1 likes C more than B
5	B	w_2	(C, w_1), (B, w_2)	A	w_2 likes B more than A
6	A	w_3	(C, w_1), (B, w_2), (A, w_3)	END	1st proposal to w_3

Man	Preference List	Woman	Preference List
A	w_1, w_2, w_3	w_1	C, B, A
B	w_1, w_2, w_3	w_2	C, B, A
C	w_1, w_2, w_3	w_3	C, B, A

Demo 2 (Let Women propose for fairness)

Iteration	Wm Proposes to	Man	Engagement	Free Wms	Reason
0				A, B, C	
1					
2					
3					
4					
5					
6					

Wms	Preference List	Man	Preference List
A	m_1, m_2, m_3	m_1	A, B, C
B	m_1, m_2, m_3	m_2	A, B, C
C	m_1, m_2, m_3	m_3	A, B, C

Demo 2 - Result

Woman proposes, but man get to select the most preferred woman

SINGAPORE INSTITUTE OF TECHNOLOGY

Iteration	Wm Proposes to	Man	Engagement	Free Wms	Reason
0				A, B, C	
1	A	m_1	(A, m_1)	B, C	1st proposal to m_1
2	B	m_1	(A, m_1)	B, C	m_1 (with A) rejects B
3	B	m_2	(A, m_1), (B, m_2)	C	1st proposal to m_2 (SUC)
4	C	m_1	(A, m_1), (B, m_2)	C	m_1 (with A) rejects C
5	C	m_2	(A, m_1), (B, m_2)	C	m_2 (with B) rejects C
6	C	m_3	(A, m_1), (B, m_2), (C, m_3)	END	1st proposal to m_3 (SUC)

Wms	Preference List	Man	Preference List
A	m_1, m_2, m_3	m_1	A, B, C
B	m_1, m_2, m_3	m_2	A, B, C
C	m_1, m_2, m_3	m_3	A, B, C

Analyze the Algorithm

- (Assume men proposing)
- **Fact 1.1**. **Woman** w remains **engaged** from the point at which she receives her first proposal; and the sequence of **partners** to which she is engaged **gets better** and better.
  - From the woman's perspective.
  - Only changes her mind if a **more preferred man proposes**.
- **Fact 1.2**. The seq. of women to whom **man** m proposes **gets worse** and worse.
  - From the man's perspective.
  - Only proposes again if he is free or his engaged **more preferred woman left** him.
- **Fact 1.3**. The algorithm **terminates** after **at most** $\boldsymbol{O(n^2)}$ **iterations**.
  - Proof **trick**: focus on the **terminate condition** of the while loop.
    - One man proposes to one different woman (he never proposed to) in each iteration.
    - How many men? How many woman? n x n

n = 2, n^2 possibilities, $2^2=4$
(m,w), (m,w'), (m',w), (m',w')

Analyze the Algorithm

- **Fact 1.4**. If **man** m is **free** at some point in the execution of the algorithm, then there is a **woman** to whom he has not **yet proposed**.
  - Proof idea: Contradiction analysis. Suppose the man has proposed to all, but now he is free. The only possibility is that all his proposed women are engaged with other men.
  - We have n women, engaged to n men, which does not include the man m. Now, we have $n + 1$ men. But it cannot be true.
- **Fact 1.5**. The set S returned at termination is a **perfect matching**.
  - Proof idea: The algorithm terminates only if there is no free man. So, all men and women are in the matching.

Analyze the Algorithm

- **Fact 1.6**. The algorithm returns a **stable matching**.
  - Proof idea: at least know it is perfect, so prove there is **no instability**.
    - **Contradiction** analysis again. Suppose we really have one:
    - (m, w) and (m', w') that:
    - Man m prefers w' more && woman w' prefers m more → (m, w') ?
    - If man m is together with w, he must have proposed to w' before, as he like w' more.
    - So w' rejected m for m'? No way, as w' like m more than m'. (Fact 1.1)

Wms	Preference List	Man	Preference List
w		m	w', w
w'	m, m'	m'	

Computational Tractability

Computational Tractability

- **Resources** in computer systems?
  - **Time, space, communication**. [https://ieeexplore.ieee.org/document/9063714]
  - Which one is the most important?
    - We **focus** on **time**, and space is also considered.
    - Why do we consider space?
- Obj: **quantify** computational tractability. How it **changes**?
  - Proposing an algorithm is easy.
  - An **efficient** (and correct) algorithm will be challenging.

https://analyticsindiamag.com/wp-content/uploads/2019/10/computer-types-10.jpg

https://i.pcmag.com/imagery/lineups/02SYcRWCQLDk6G1bnereOVv-2.fit_lim.v_1585232946.jpg

https://miro.medium.com/max/600/0*AgiPFgJrm9GQfzKJ.jpg

Computational Tractability

- Informal efficiency def.: **runs quickly** on real input.
  - Problem 1: run the algorithm in what environment?
  - Problem 2: how fast is fast?
    - Special case: a poor-designed algorithm runs super fast with tiny data input
    - Special case: a well-designed algorithm runs super slow with huge data
  - Problem 3: what is real input? Crash with large input?
- We want: **platform-independent** and **instance-independent**.

https://analyticsindiamag.com/wp-content/uploads/2019/10/computer-types-10.jpg

https://i.pcmag.com/imagery/lineups/02SYcRWCQLDk6G1bnereOVv-2.fit_lim.v_1585232946.jpg

https://miro.medium.com/max/600/0*AgiPFgJrm9GQfzKJ.jpg

Worst-Case Running Time

- Quantify the input size: N
  - n men and n women (eg. n=3)
  - $2n$ preference lists, each list of length n (eg. 2x3 plist x 3)
  - $N = 2n \cdot n = 2n^2$
- Task: analyze the **running time as a function of N.**
- **Worst-case** running time: the largest possible running time with input N.
- **Benchmark**: **brute-force** algorithm (brainless design).

Man	Preference List	Woman	Preference List
A	w_1, w_2, w_3	w_1	C, B, A
B	w_1, w_2, w_3	w_2	C, B, A
C	w_1, w_2, w_3	w_3	C, B, A

https://stickershop.line-scdn.net/stickershop/v1/product/1510436/LINEStorePC/main.png;compress=true

Redefine Efficiency

- Informal efficiency def.: achieves **better worst-case** performance, at an **analytical** level, than **brute-force** search.
- **Polynomial** time: quantify "reasonable" running time.
  - **Search space**: often grows **exponentially**.
  - **Reasonable** may be: from n to $n+1$, running time increases by a **constant**.
- Informal efficiency def.: an alg. is **efficient** if runs in **polynomial** time.
  - Extreme cases exists like the worst-case concept, i.e., $n^{10000000}$?
  - Again, in reality, the polynomial concept works well.

Running Times

n is the number of data points

	n	$n \log_2 n$	n^2	n^3	1.5^n	2^n	$n!$
$n = 10$	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec	4 sec
$n = 30$	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec	18 min	10^{25} years
$n = 50$	< 1 sec	< 1 sec	< 1 sec	< 1 sec	11 min	36 years	very long
$n = 100$	< 1 sec	< 1 sec	< 1 sec	1 sec	12,892 years	10^{17} years	very long
$n = 1{,}000$	< 1 sec	< 1 sec	1 sec	18 min	very long	very long	very long
$n = 10{,}000$	< 1 sec	< 1 sec	2 min	12 days	very long	very long	very long
$n = 100{,}000$	< 1 sec	2 sec	3 hours	32 years	very long	very long	very long
$n = 1{,}000{,}000$	1 sec	20 sec	12 days	31,710 years	very long	very long	very long

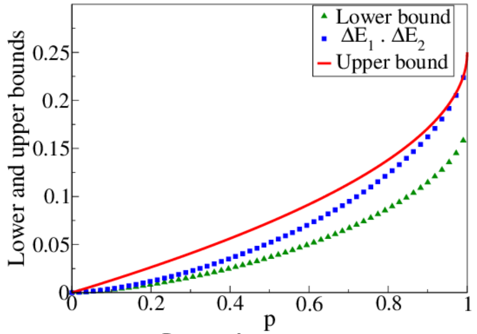
- Assumptions:
  - 1m instructions per second
  - Very long -> runs over 10^{25} years.
- Intelligence does not necessarily produce polynomial algorithms.

1,000,000 instructions per second (1mins = 60secs)
$n^3 = 1{,}000^3 = 1{,}000{,}000{,}000$
1,000,000,000 / 1,000,000 = 1,000secs x 1/60 = 17mins

Kleinberg J, Tardos E. Algorithm design. Pearson Education; 2006.

Asymptotic Order of Growth – Upper Bound

- Target: **worst-case** time, given input n, grows at a rate **at most proportional to** $f(n)$.
  - We also call $f(n)$ a **bound**.
- **Too specific** expressions **unnecessary**, i.e., runs in $1.62n^2 + n + 4$ steps.
  - Constants vary for different architectures
  - Determined by the highest order (e.g. n^2)
- How do you plan to quantify the above terms concisely?
  - We **ignore** the **constants** and We **ignore** the **lower order** items.



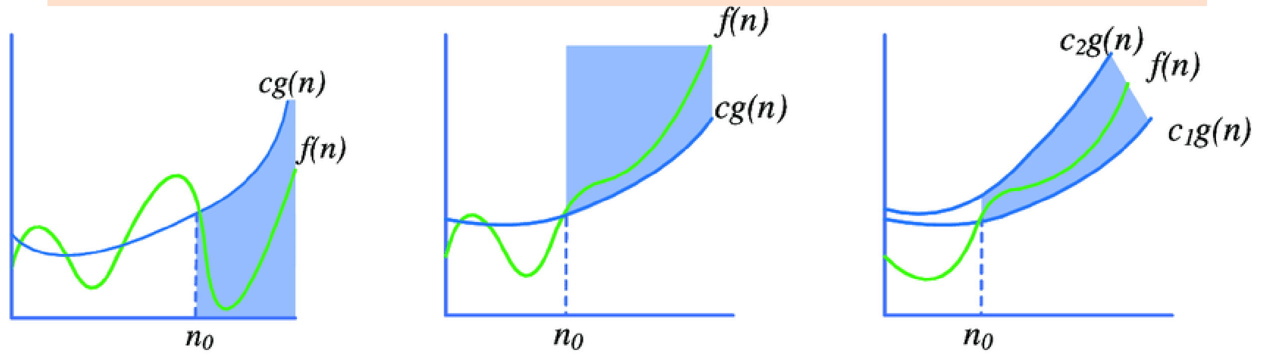
- Asymptotic Upper Bounds $O(\cdot)$:
  - The **worst-case** running time $T(n)$ of a certain algorithm on an input of size n.
    - Expresses only an **upper bound**.
  - $f(n) = pn^2 + qn + r$ for constants p, q, and r → $O(n^2)$.

https://www.researchgate.net/profile/Uttam_Singh7/publication/263048941/figure/fig3/AS:667047482630161@1536047921910/Color-online-The-lower-and-upper-bounds-on-E1E2-Here-both-the-axes-are.png (Tue. Till Here)

Asymptotic Order of Growth – More bounds

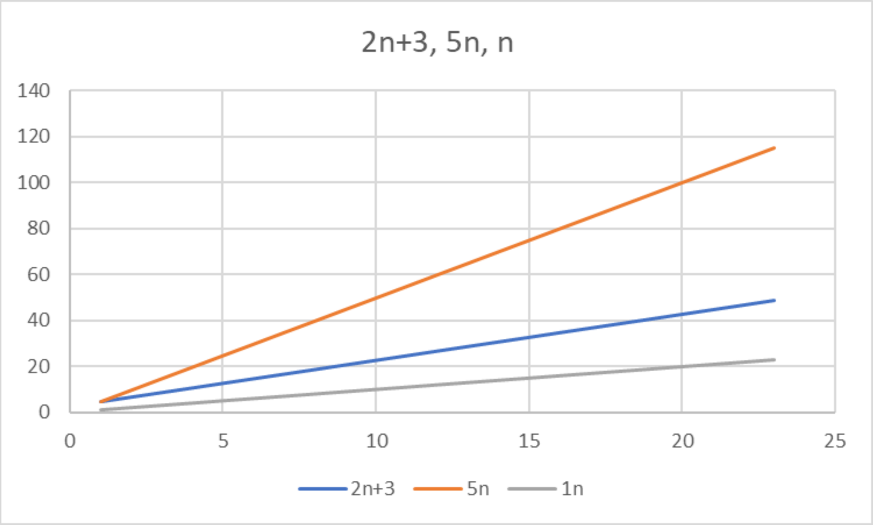
- How to justify our proven bound $O(\cdot)$ is good?
  - i.e., we are almost always true if we say it is $O(n^{100000n})$.
- Asymptotic Lower Bounds : $\Omega(n)$.
  - The func. $T(n)$ is **at least a constant "c" multiple** of $f(n) = c.g(n)$.
  - $T(n) = pn^2 + qn + r$ for constants p, q, and r → $\Omega(n^2)$.
- Asymptotically Tight Bounds $\Theta(n)$: $f(n) = \Omega(g(n)) = O(g(n))$ (optimal bound)
  - i.e., $f(n) = pn^2 + qn + r = \Omega(n^2) = O(n^2) = \Theta(n^2)$.
- Note: consider $n \to \infty$.
  - Only matters when n is large.
  - $f(n) = \Theta(g(n))$ if $\lim_{n\to\infty} \frac{f(n)}{g(n)} = c$.
  - c is a positive constant.

c.g(n) is the estimated bound



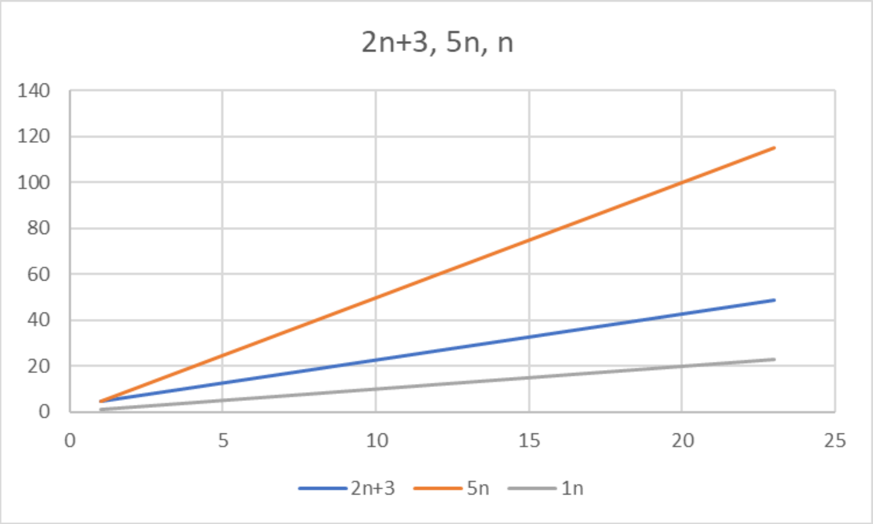
(a) Big-O Notation (b) Big-Omega Notation (c) Big-Theta Notation

https://www.researchgate.net/publication/329846158/figure/fig2/AS:706473407565824@1545447794620/Asymptotic-upper-lower-and-tight-bounds-of-a-function-from-a-c.png



- $f(n) = 2n + 3$
- $f(n) = O(g(n))$ where $g(n) = n$
- $f(n) <= c_1.g(n) = c_1.\ n$ where $n>n_o$, $c_1>0$, $n_o>=1$
- $2n+3 <= c_1.n$ where the RHS, big-O is larger than actual function
- If $n_o=1$ and $c_1 = 5$, then $5 <= 5$: ok
- If $n_o=2$ and $c_1 = 5$, then $7 <= 10$: ok
- Conclude that the big-oh of the function 2n+3 is O(n) where $c_1=5$, $n_o>=1$. This is the best worst case of this algorithm
- $f(n) = \Omega(g(n))$ where $g(n) = n$
- $f(n) >= c_2.g(n) = c_2.\ n$ where $n>n_o$, $c_2>0$, $n_o>=1$
- $2n+3 >= c_2.n$ where RHS, big-omega is smaller than the actual function
- If $n_o=1$ and $c_2 = 1$, then $5 >= 1$: ok
- If $n_o=2$ and $c_2 = 1$, then $7 >= 2$: ok
- Conclude that the big-omega of the function 2n+3 is $\Omega(n)$ where $c_2=1$, $n_o>=1$. This is the best case of this algorithm (the algo will never be better than this)

- $f(n) = 2n + 3$
- $f(n) = \theta(g(n))$ where $g(n) = n$
- $c_2. n = c_2.g(n) <= f(n) <= c_1.g(n) = c_1. n$ where $n>n_o$, $c_1,c_2 >0$, $n_o>=1$
- $2n+3 <= c_1.n$ where the RHS is the worst case
- Conclude that the worst case of the function 2n+3 is $\theta(n)$ where $c_1=5$, $n_o>=1$. This is the best worst case of this algorithm
- $c_2.n <= 2n+3$ where LHS, is the best case
- Conclude that the best case of the function 2n+3 is $\theta(n)$ where $c_2=1$, $n_o>=1$. This is the best case of this algorithm (the algo will never be better than this)
- theta is the average case of the algorithm



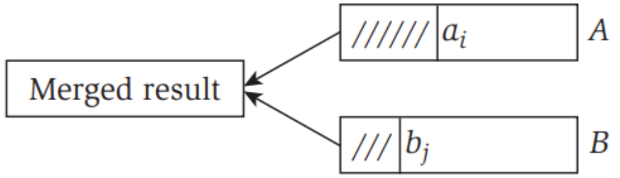
Common Times - $O(n)$

- Long-time goal: recognize **common styles** of analyzing the running time.
- **Linear time**: $\boldsymbol{O(n)}$, time scales by a constant factor times the input size n.
- **Find the maximum**: $a_1, a_2, \cdots, a_n$.
  - Each needs a comparison, in constant time.
  - $O(n) * O(1) = O(n)$.

```
max = a_1
For i = 2 to n
   If a_i > max then
      set max = a_i
   Endif
Endfor
```

- **Merge sorted lists**. The same $O(n)$, but more complex.
  - List $a_1, a_2, \cdots, a_n$ and list $b_1, b_2, \cdots, b_n$, both in ascending order.
  - Objective: $c_1, c_2, \cdots, c_{2n}$.
    - i.e., 2, 3, 11, 19 and 4, 9, 16, 25 → 2, 3, 4, 9, 11, 16, 19, 25.
  - A non-intelligent solution: concatenate the lists → sort all the items.
  - Record pa, pb, and pc, where $pa = pb = pc = 1$ in the beginning.
  - While $pa \leq n$ and $pb \leq n$:
    - If $a_{pa} > b_{pb}$ → smaller b_{pb} to c_{pc} → $pb{+}{+}$ and $pc{+}{+}$.
    - If $a_{pa} \leq b_{pb}$ → smaller a_{pa} to c_{pc} → $pa{+}{+}$ and $pc{+}{+}$.
  - Whatever left, append to c.

Append the smaller of a_i and b_j to the output.



Common Times - $O(n\log n)$, $O(n^2)$, $O(n^3)$, $O(n^k)$

- $O(n\log n)$: the time of any alg. splits input into **equal-sized pieces**, **solves** each **recursively**, and **combines** the two **solutions** in linear time.
  - Popular example: **sorting**.
  - Analyze the basic idea: sorting is expensive -> sort less -> combine.
    - i.e., $100^2 = 10{,}000$ -> $2 \times 50^2 = 5{,}000$ -> $5{,}000 + 100 = 5{,}100$, around half time.
- $O(n^2)$: i.e., n points in a plane, what the **nearest points**?
  - Brute-force sol.: for each pt, compute the dist. between it and another pt.
  - $O(n)$ points, each $O(n)$ distances, each distance $O(1)$ -> $O(n^2)$.
- $O(n^3)$: nested loops, i.e., 3-variable solver: $2x + 5y + 9z = 100$.
  - For x -> for y -> for z -> check the sum.
- $O(n^k)$: Any idea on getting $O(n^k)$?
  - Independent sets in a graph with k nodes.
  - Brute-force, choose k from n: $\binom{n}{k} = \frac{n(n-1)\cdots(n-k+1)}{1\cdot 2\cdots k} \le \frac{n^k}{k!} \le n^k$.

Beyond Polynomial Time

- $\boldsymbol{O(2^n)}$ vs. $\boldsymbol{O(n!)}$. Which one is faster?
  - $\boldsymbol{O(2^n)}$: the number of the subsets of n nodes.
  - $\boldsymbol{O(n!)}$: the number of perfect matchings; traveling salesman problem.
- **Sublinear** time: better than linear, i.e., $O(\log n)$.
  - Binary search.
  - **Query** the inputs instead of traversing them.
  - Half → half → half → half …