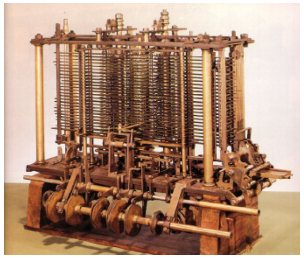
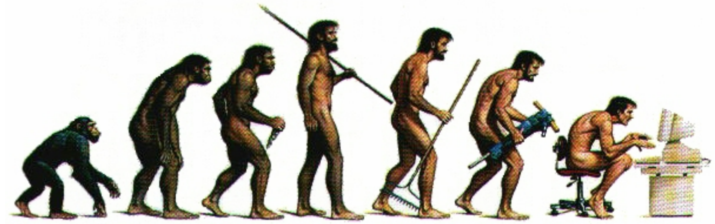
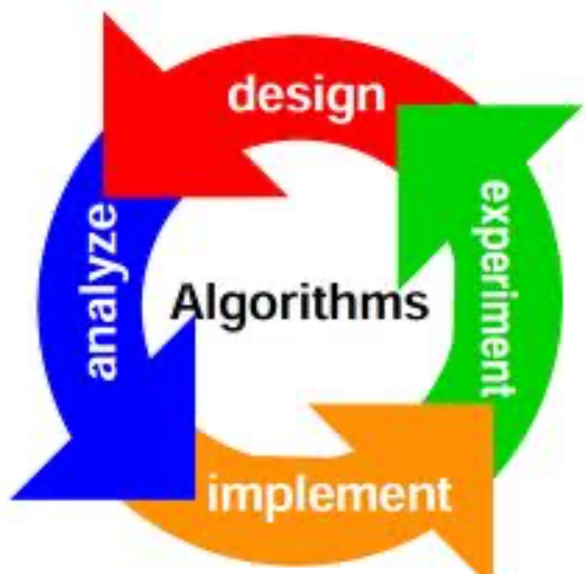




Data Structures and Algorithms II

CMPSC 130B



Course plan

- We are in the midst of a pandemic. Try your best.
- Instructor
  - Ambuj K Singh, ambuj@ucsb.edu
- Teaching Assistants
  - Kha-Dinh Luong, vluong@ucsb.edu
  - Chinmay Sonar, vaishali@ucsb.edu
- Undergraduate Learning Assistants
  - Gabriele Soule
  - Andrew Kraft
- "Welcome to UC Santa Barbara – where the land meets the sea, where brilliant minds <u>meet</u> each other, and where academic excellence and <u>social engagement</u> unite to spark creativity and discovery."

Remote live lectures using zoom

- Resource for remote learning: https://keeplearning.id.ucsb.edu/
- If you have logistical or technical issues with remote engagement, please email help@collaborate.ucsb.edu.
- Lectures and discussion sections will be recorded and made available on GauchoSpace for students who may not be able to attend at this time.
- By default, your microphone and camera will be muted when you join the session. If you do not want to be included in the recording, simply keep your camera and microphone off.
- You may ask questions by unmuting or in the chat window.

Catalog description

***Prerequisite**: Computer Science 130A*

Design and analysis of computer algorithms. Correctness proofs and solution of recurrence relations. Design techniques; divide and conquer, greedy strategies, dynamic programming, branch and bound, backtracking, and local search. Applications of techniques to problems from several disciplines. NP-completeness.

Prerequisite concepts

- Data structures:
  - Sets with insert/delete/member: Hashing
  - Sets with priority: Heaps, priority queues
  - Balanced search trees
  - Union/find
  - Graphs
  - Sorting
- Discrete mathematics
  - Functions, relations, recurrence equations, induction, logic, proofs, …
- Programming
  - C++
- Program/algorithm complexity analysis

Objectives

- The goal of this course is to introduce you to a systematic study of algorithmic design techniques and intractability using examples across many areas of computer science and related fields. We will cover the following topics:
  - Greedy algorithms
  - Divide and conquer
  - Dynamic programming
  - NP-completeness
  - Approximation algorithms

Textbook

- Required:
  - *Algorithm Design*, Jon Kleinberg and Éva Tardos, 2006, Pearson.
- References:
  - *Algorithms,* Sanjoy Dasgupta, Christos Papadimitriou, Umesh Vazirani, McGraw-Hill
  - *Introduction to Algorithms*, Thomas H. Cormen, Charles Leiserson, Ronald Rivest, McGraw-Hill
  - *Data Structures and Algorithm Analysis in C++,* by Mark Allen Weiss (4th edition)
- For fun:
  - *Algorithms to Live By: The Computer Science of Human Decisions,* Brian Christian and Tom Griffiths, Holt & Co.

Assignments and grading

- Grades will be based on class participation, homework assignments, programming assignments, midterm exam, and final exam:
  - 7% class participation
  - 27% Homework assignments
  - 26% Programming assignments
  - 20% Midterm exam (Feb 3, during lecture)
  - 20% Final exam (March 17, 8-11 AM), optional
- Late assignments are not allowed.
- **All graded work turned in must be completely your own, including programming assignments.**
- There will be no makeup exams.

Topics covered

• Greedy Algorithms	4 Lectures
• Divide & Conquer	4L
• Dynamic Programming	3L
• NP-Hardness	4L
• Approximation Algorithms	2L

Syllabus discussion

This week's goals

- Assigned reading
  - Chapter 4
- Discussion section plan
  - Algorithm complexity analysis
  - Proof by induction
  - Graph algorithms

The Idea of an Algorithm

- 9th-century Persian mathematician Muḥammad ibn Mūsā al-Khwārizmī (latinized *Algoritmi)*.
- A sequence of unambiguous instructions for solving a problem.
- Finite – must eventually terminate.
- Complete – always gives a solution when there is one.
- Correct (sound) – always gives a "correct" solution.
- Efficient

Famous algorithms

- Constructions of Euclid
- Newton's root finding
- Fast Fourier Transform
- Compression (Huffman, Lempel-Ziv, GIF, MPEG)
- DES, RSA encryption
- Simplex for linear programming
- Shortest Path Algorithms (Dijkstra, Bellman-Ford)
- Error correcting codes (CDs, DVDs)
- TCP congestion control, IP routing
- Pattern matching (Genomics)
- Search Engines

Algorithms in modern world

- Enormous amount of data
  - E-commerce (Amazon, eBay)
  - Advertisement (Google)
  - Network traffic (telecom billing, monitoring)
  - Database transactions (Sales, inventory)
  - Scientific measurements (astrophysics, geology)
  - Sensor networks. RFID tags
  - Bioinformatics (genome, protein bank)
  - Drug discovery (high throughput screens)
  - Machine learning & AI
- Need for scalability

Why efficient algorithms matter?

- Suppose $N = 10^6$
- A PC can read/process N records in 1 sec.
- But if some algorithm does N*N computations, then it takes 1M seconds ≈ 11 days!!!
- 100 City Traveling Salesman Problem.
  - A supercomputer solved an instance with 85,900 points in 136 CPU-years.

$O(n^2 2^n)$

- Fast factoring algorithms can break encryption schemes. Research determines what is safe code length (> 100 digits).
- Advent of quantum computing

How to measure algorithm performance?

- What metric should be used to judge algorithms?
  - Length of the program (lines of code)
  - Ease of programming (bugs, maintenance)
  - Memory required

❑ Running time

- Running time is the dominant standard.
  - Quantifiable and easy to compare
  - Often the critical bottleneck

Average, Best, and Worst-Case

- On which input instances should the algorithm's performance be judged?
- Average case:
  - Real world distributions difficult to predict
  - Typically used in statistical machine learning
- Best case:
  - Seems unrealistic
- Worst case:
  - Gives an absolute guarantee
  - **Typical**

Caveats

- Follow the spirit, not the letter
  - A 100n algorithm is more expensive than an n^2 algorithm provided $n < 100$
- Other considerations:
  - a program used only a few times
  - a program run on small data sets
  - ease of coding, porting, maintenance
  - memory requirements

Worst case, Best case, and Average case

```
template<class T>
void SelectionSort(T a[], int n)
{   // Early-terminating version of selection sort
    bool sorted = false;
    for (int size=n; !sorted && (size>1); size--) {
        int pos = 0;
        sorted = true;
        // find largest in a[0,..,size-1]
        for (int i = 1; i < size; i++)
            if (a[pos] <= a[i]) pos = i;
            else sorted = false; // out of order
        Swap(a[pos], a[size - 1]);
        }
}
```

Worst case?

Best case?

Average case?

Breakout

Asymptotic Notations

- Big-O, "bounded above by": $T(n) = O(f(n))$
  - For some c and N, $T(n) \le c \cdot f(n)$ whenever $n > N$.

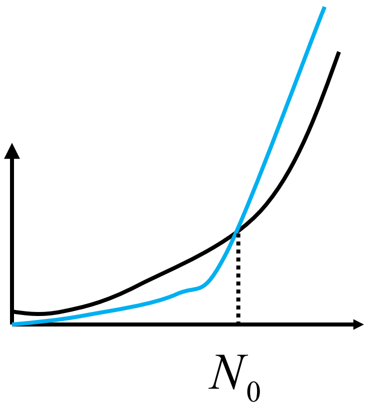
- Big-Omega, "bounded below by": $T(n) = \Omega(f(n))$
  - For some $c>0$ and N, $T(n) \ge c \cdot f(n)$ whenever $n > N$.
  - Same as $f(n)$ is $O(T(n))$.

- Big-Theta, "bounded above and below by": $T(n) = \Theta(f(n))$
  - $T(n)$ is $O(f(n))$ and also $T(n)$ is $\Omega(f(n))$

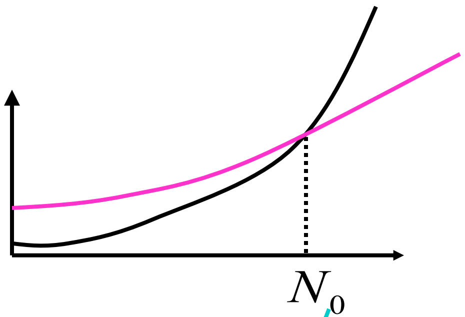
- Little-o, "strictly bounded above by": $T(n) = o(f(n))$
  - $T(n)/f(n) \to 0$ as $n \to \infty$

In pictures

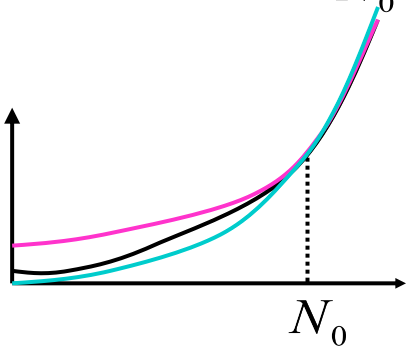
- Big-Oh (most commonly used)
  - bounded above



- Big-Omega
  - bounded below



- Big-Theta
  - exactly

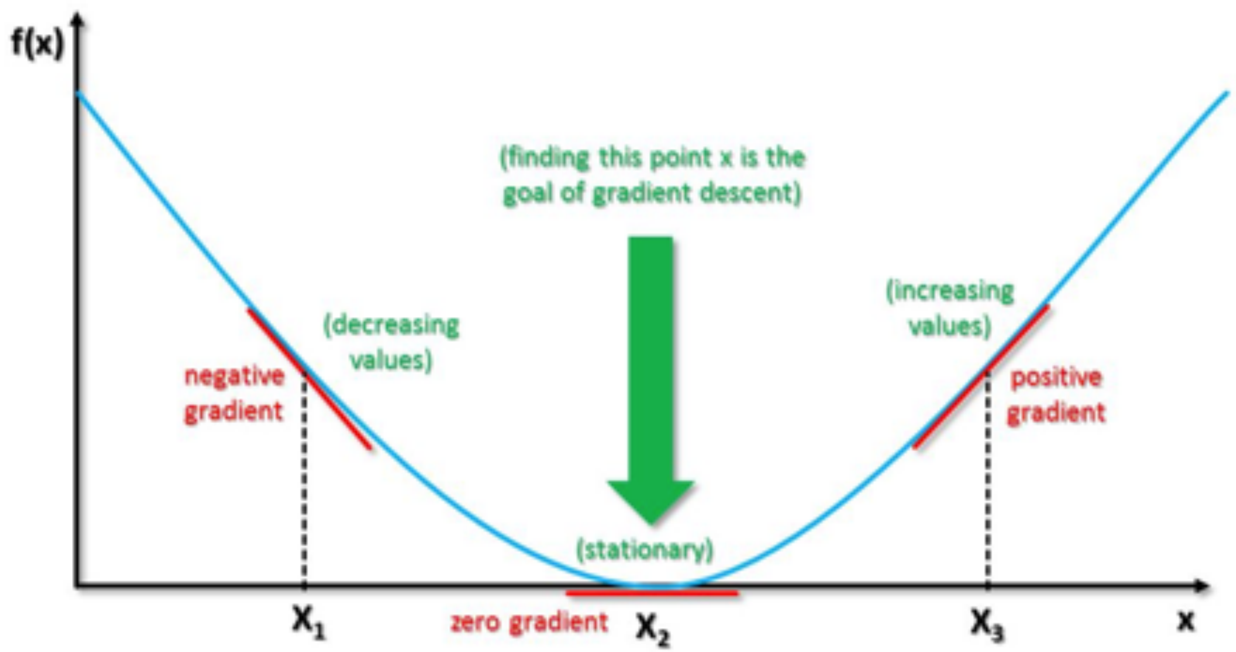


- Small-o
  - not as expensive as ...

Design of algorithms

- Understand the problem, assess its difficulty
- Choose an approach (e.g., exact/approximate, deterministic/probabilistic)
- Choose strategy and appropriate data structures
- Prove
  - termination
  - correctness and completeness
- Evaluate complexity
  - We wish to not only find a solution, but to find the best or optimal solution.
  - Or, show that no efficient solution exists.
- Compare to other known approaches

Gradient descent in Deep Learning



How to tell man from mouse?

- Dynamic programming of DNA/protein sequences
- Global alignment algorithm of Needleman and Wunsch (1970)
- Local alignment algorithm of Smith and Waterman (1981)

```
 1 MVHLTPEEKSAVTALWGKV--NVDEVGGEALGRLLVVYPWTQRFFESFGD     48
   || |:|.:|:.|.|.||||  :..|.|.|||.|:.:.:|.|:.:|..| |
 1 MV-LSPADKTNVKAAWGKVGAHAGEYGAEALERMFLSFPTTKTYFPHF-D     48

49 LSTPDAVMGNPKVKAHGKKVLGAFSDGLAHLDNLKGTFATLSELHCDKLH     98
   ||     .|:.:||.|||||..|.::.:||:|::....:.||:||..||.
49 LS-----HGSAQVKGHGKKVADALTNAVAHVDDMPNALSALSDLHAHKLR     93

99 VDPENFRLLGNVLVCVLAHHFGKEFTPPVQAAYQKVVAGVANALAHKYH    147
   |||.||:||.:.|:..||.|...||||.|.|:..|.:|.|:..|..||.
94 VDPVNFKLLSHCLLVTLAAHLPAEFTPAVHASLDKFLASVSTVLTSKYR    142
```

Matching of medical residents

- Residents rank colleges
- Colleges rank residents
- Find a stable matching:
  - There is no pair such that flipping the assignments leads to a “better” outcome.
- Solution runs in $O(n^2)$ time.
- Nobel prize in Economics 2012 to Shapley and Roth
- Many stable matchings are possible
  - Choosing the one that is best or that cannot be manipulated has been researched extensively.

Which webpages to display in a search

- Ranking based on keywords/topics
- Ranking based on importance of pages
- Examine the link structure of pages by random walk
- Display the central pages first
- Good authorities should be pointed by good authorities
  - The value of a node is determined by the value of the nodes that point to it.
- Google's PageRank