Introduction

• Instructors
  ➢ Nisarg Shah
    o cs.toronto.edu/~nisarg, nisarg@cs, SF 2301C
    o LEC 0101 and 0102

• TAs: Too many to list

• Disclaimer!
  ➢ First online version of the course, so expect a bumpy ride at the start, but hopefully, we’ll get through together
  ➢ Use any of the feedback mediums (email, Piazza, ...) to let me know if you have any suggestions for improvement
Course Information

• **Course Page** [www.cs.toronto.edu/~nisarg/teaching/373f20/](http://www.cs.toronto.edu/~nisarg/teaching/373f20/)
  - All the information below is in the course information sheet, available on Piazza

• **Discussion Board** [piazza.com/utoronto.ca/fall2020/csc373](http://piazza.com/utoronto.ca/fall2020/csc373)

• **Grading** – MarkUs
  - Link will be distributed after about a week or two
  - LaTeX preferred, scans are OK!

• All times in **Eastern time zone**, all zoom links on the course page
Lectures

• **Time & Place:** Tue 4-5pm, Thu 1-3pm, Zoom

• **Details**
  - Delivered live
  - 10 minute break after every 50 minutes of lecture
  - Students can ask questions using Zoom’s chat feature
  - One TA will be present to continuously answer questions
  - I might also answer questions once in a while
Tutorials

• **Time & Place:** Tue 5-6pm, Zoom

• **Details**
  - Delivered live by TAs
  - Problem sets will be posted early on the course webpage
    - Easier problems that are warm-up to assignments/exams
  - Please try them before coming to the tutorials
  - TAs will explain the problems, allow you to discuss them in breakout rooms, and then go over key parts of the solutions
  - Solutions will be posted later on the course webpage
Tutorials

• Further details
  ➢ Each section is divided into three parts (A,B,C)
  ➢ Students divided by birth month: A = Jan-Apr, B = May-Aug, C = Sep-Dec
  ➢ Feel free to attend a different tutorial than the one you’re assigned
    o EXCEPT when the tutorial slot is being used for a test
  ➢ If the attendance is low, the number of tutorials per section may be reduced
Office Hours

• **Time & Place:** Wed 4-5pm, Fri 10-11am, Zoom
  ➢ Do you have conflicts with these slots? Poll!

• **Details**
  ➢ I will conduct them
  ➢ Use the “raise hand” feature
  ➢ I will call upon the raised hands in order
  ➢ When called upon, unmute and ask the question
  ➢ Always phrase your question in a way that doesn’t give away your solutions or approach to an assignment problem
    o Just like in a physical office
Tests

• 2 term tests, one end-of-term test (final exam)

• Time & Place: Tue 5-6pm (tutorial slot)
  ➢ Need to be able to attend live!
  ➢ I’m considering using part of the Tue 4-5pm lecture slot to give you more time

• Tentative Plan
  ➢ Open book, closed internet
  ➢ You may be asked to join a zoom link and keep your video on
  ➢ If you have a question, you can “raise hand”, and I or a TA can take you to a breakout room to answer your question
  ➢ Upload scanned answer sheet at the end (we’ll do a mock run of this)
Assignments

• 4 assignments, best 3 out of 4

• Group work
  ➢ In groups of up to three students
  ➢ Best way to learn is for each member to try each problem

• Questions will be more difficult
  ➢ May need to mull them over for several days; do not expect to start and finish the assignment on the same day!
  ➢ May include bonus questions

• Submission on MarkUs, more details later
  ➢ May need to compress the PDF
Grading Policy

• 3 homeworks  *  10%  =  30%

• 2 term tests   *  20%  =  40%

• Final exam     *  30%  =  30%

• NOTE: To pass, you must earn at least 40% on the final exam
Approximate Due Dates

• Please note the word approximate!
  ➢ Assignment 1: Apx. Oct 9
  ➢ Assignment 2: Apx. Oct 30
  ➢ Assignment 3: Apx. Nov 13
  ➢ Assignment 4: Apx. Nov 27
  ➢ Midterm 1: Apx. Oct 20
  ➢ Midterm 2: Apx. Nov 17

• Conflicts
  ➢ The tests are during the tutorial slot, so there should ideally be no conflict
  ➢ That said, if you think you’ll have a conflict, let me know at the earliest
Textbook

• Primary reference: lecture slides

• Primary textbook (required)

• Supplementary textbooks (optional)
  ➢ [KT] Kleinberg; Tardos: *Algorithm Design*. 
Other Policies

• Collaboration
  ➢ Free to discuss with classmates or read online material
  ➢ Must write solutions in your own words
    o Easier if you do not take any pictures/notes from discussions

• Citation
  ➢ For each question, must cite the peer (write the name) or the online sources (provide links), if you obtained a significant insight directly pertinent to the question
  ➢ Failing to do this is plagiarism!
Other Policies

• “No Garbage” Policy
  ➢ Borrowed from: Prof. Allan Borodin (citation!)

1. Partial marks for viable approaches
2. Zero marks if the answer makes no sense
3. 20% marks if you admit to not knowing how to approach the question (“I do not know how to approach this question”)

• 20% > 0% !!
Other Policies

• Late Days
  ➢ 4 total late days across all 4 assignments
  ➢ Managed by MarkUs
  ➢ At most 2 late days can be applied to a single assignment
  ➢ Already covers legitimate reasons such as illness, university activities, etc.
    o Petitions will only be granted for circumstances which cannot be covered by this
Zoom Features

• Just to get acquainted, let’s try out the following features:

  ➢ Polls (already tried)
  ➢ Chat
  ➢ Reactions
  ➢ Raise hand
  ➢ Yes/No
  ➢ Breakout rooms
Enough with the boring stuff.
What will we study?

Why will we study it?
Muhammad ibn Musa al-Khwarizmi
c. 780 – c. 850
What is this course about?

• **Algorithms**
  - Ubiquitous in the real world
    - From your smartphone to self-driving cars
    - From graph problems to graphics problems
    - ...
  - Important to be able to design and analyze algorithms
  - For some problems, good algorithms are hard to find
    - For some of these problems, we can formally establish complexity results
    - We’ll often find that one problem is easy, but its minor variants are suddenly hard
What is this course about?

- **Algorithms**
  - Algorithms in specialized environments or using advanced techniques
    - Distributed, parallel, streaming, sublinear time, spectral, genetic...
  
- Other concerns with algorithms
  - Fairness, ethics, ...

- ...mostly beyond the scope of this course
What is this course about?

• **Topics in this course**
  - Divide and Conquer
  - Greedy
  - Dynamic programming
  - Network flow
  - Linear programming
  - NP-completeness (not really an algorithm design paradigm)
  - Approximation algorithms (if time permits)
  - Randomized algorithms (if time permits)
What is this course about?

• How do we know which paradigm is right for a given problem?
  ➢ A very interesting question!
  ➢ Subject of much ongoing research...
    o Sometimes, you just know it when you see it...

• How do we analyze an algorithm?
  ➢ Proof of correctness
  ➢ Proof of running time
    o We’ll try to prove the algorithm is efficient in the worst case
    o In practice, average case matters just as much (or even more)
What is this course about?

• What does it mean for an algorithm to be efficient in the worst case?
  ➢ Polynomial time
  ➢ It should use at most poly(n) steps on any n-bit input
    o $n, n^2, n^{100}, 100n^6 + 237n^2 + 432, ...$

  ➢ If the input to an algorithm is a number $x$, the number of bits of input is $\log x$
    o This is because it takes $\log x$ bits to represent the input $x$ in binary
    o So the running time should be polynomial in $\log x$, not in $x$

  ➢ How much is too much?
What is this course about?

Picture-Hanging Puzzles*

Erik D. Demaine†  Martin L. Demaine†  Yair N. Minsky†  Joseph S. B. Mitchell§
Ronald L. Rivest†  Mihai Pătrașcu†

Theorem 7 For any $n \geq k \geq 1$, there is a picture hanging on $n$ nails, of length $n^{c'}$ for a constant $c'$, that falls upon the removal of any $k$ of the nails.

$n^{6,100 \log_2 c}$. Using the $c \leq 1,078$ upper bound, we obtain an upper bound of $c' \leq 6,575,800$. Using

So, while this construction is polynomial, it is a rather large polynomial. For small values of $n$, we can use known small sorting networks to obtain somewhat reasonable constructions.
What is this course about?

Better Balance by Being Biased:
A 0.8776-Approximation for Max Bisection

Per Austrin*, Siavosh Benabbas*, and Konstantinos Georgiou†

has a lot of flexibility, indicating that further improvements may be possible. We remark that, while polynomial, the running time of the algorithm is somewhat abysmal; loose estimates places it somewhere around $O(n^{10^{100}})$; the running time of the algorithm of [RT12] is similar.
What is this course about?

• What if we can’t find an efficient algorithm for a problem?
  ➢ Try to prove that the problem is hard
  ➢ Formally establish complexity results
  ➢ NP-completeness, NP-hardness, ...

• We’ll often find that one problem may be easy, but its simple variants may suddenly become hard
  ➢ Minimum spanning tree (MST) vs bounded degree MST
  ➢ 2-colorability vs 3-colorability
I’m not convinced.

Will I really ever need to know how to design abstract algorithms?
At the very least...

This will help you prepare for your technical job interview!

Real Microsoft interview question:

• Given an array $a$, find indices $(i, j)$ with the largest $j - i$ such that $a[j] > a[i]$
• Greedy? Divide & conquer?
Disclaimer

• The course is **theoretical in nature**
  - You’ll be working with abstract notations, proving correctness of algorithms, analyzing the running time of algorithms, designing new algorithms, and proving complexity results.

• **Something for everyone...**
  - If you’re somewhat scared going into the course
  - If you’re already comfortable with the proofs, and want challenging problems
Related/Follow-up Courses

• Direct follow-up
  ➢ CSC473: Advanced Algorithms
  ➢ CSC438: Computability and Logic
  ➢ CSC463: Computational Complexity and Computability

• Algorithms in other contexts
  ➢ CSC304: Algorithmic Game Theory and Mechanism Design (self promotion!)
  ➢ CSC384: Introduction to Artificial Intelligence
  ➢ CSC436: Numerical Algorithms
  ➢ CSC418: Computer Graphics
Divide & Conquer
History?

• Maybe you saw a subset of these algorithms?
  - Mergesort - $O(n \log n)$
  - Karatsuba algorithm for fast multiplication - $O(n^{\log_2 3})$ rather than $O(n^2)$
  - Largest subsequence sum in $O(n)$
  - ...

• Have you seen some divide & conquer algorithms before?
  - Maybe in CSC236/CSC240 and/or CSC263/CSC265
  - Write “yes”/”no” in chat
Divide & Conquer

• General framework
  - Break (a large chunk of) a problem into two smaller subproblems of the same type
  - Solve each subproblem recursively and independently
  - At the end, quickly combine solutions from the two subproblems and/or solve any remaining part of the original problem

• Hard to formally define when a given algorithm is divide-and-conquer...

• Let’s see some examples!
Master Theorem

• Here’s the master theorem, as it appears in CLRS
  ➢ Useful for analyzing divide-and-conquer running time
  ➢ If you haven’t already seen it, please spend some time understanding it

Theorem 4.1 (Master theorem)
Let $a \geq 1$ and $b > 1$ be constants, let $f(n)$ be a function, and let $T(n)$ be defined on the nonnegative integers by the recurrence

$$T(n) = aT(n/b) + f(n),$$

where we interpret $n/b$ to mean either $\lfloor n/b \rfloor$ or $\lceil n/b \rceil$. Then $T(n)$ has the following asymptotic bounds:

1. If $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.
2. If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a \lg n})$.
3. If $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some constant $\epsilon > 0$, and if $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large $n$, then $T(n) = \Theta(f(n))$. ■
Master Theorem

Intuition: Compare $f(n)$ with $n^{\log_b a}$. The larger determines the recurrence solution.
Counting Inversions

• Problem
  ➢ Given an array $a$ of length $n$, count the number of pairs $(i, j)$ such that $i < j$ but $a[i] > a[j]$

• Applications
  ➢ Voting theory
  ➢ Collaborative filtering
  ➢ Measuring the “sortedness” of an array
  ➢ Sensitivity analysis of Google's ranking function
  ➢ Rank aggregation for meta-searching on the Web
  ➢ Nonparametric statistics (e.g., Kendall's tau distance)
Counting Inversions

• Problem
  ➢ Count \((i, j)\) such that \(i < j\) but \(a[i] > a[j]\)

• Brute force
  ➢ Check all \(\Theta(n^2)\) pairs

• Divide & conquer
  ➢ Divide: break array into two equal halves \(x\) and \(y\)
  ➢ Conquer: count inversions in each half recursively
  ➢ Combine:
    o Solve (we’ll see how): count inversions with one entry in \(x\) and one in \(y\)
    o Merge: add all three counts
Counting Inversions

• From Kevin Wayne’s slides

\[\text{SORT-AND-COUNT} (L)\]

\[\text{IF list } L \text{ has one element}\]
\[\text{RETURN } (0, L).\]

\[\text{DIVIDE} \text{ the list into two halves } A \text{ and } B.\]
\[r_A, A \leftarrow \text{SORT-AND-COUNT} (A).\]
\[r_B, B \leftarrow \text{SORT-AND-COUNT} (B).\]
\[r_{AB}, L' \leftarrow \text{MERGE-AND-COUNT} (A, B).\]

\[\text{RETURN } (r_A + r_B + r_{AB}, L').\]
Counting Inversions

<table>
<thead>
<tr>
<th>Count Inversions in Left Half A</th>
<th>Count Inversions in Right Half B</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 5 4 8 10</td>
<td>2 6 9 3 7</td>
</tr>
<tr>
<td>5-4</td>
<td>6-3 9-3 9-7</td>
</tr>
</tbody>
</table>

Count Inversions (a, b) with a ∈ A and b ∈ B

<table>
<thead>
<tr>
<th>Count Inversions (a, b) with a ∈ A and b ∈ B</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 5 4 8 10</td>
</tr>
<tr>
<td>4-2 4-3 5-2 5-3 8-2 8-3 8-6 8-7 10-2 10-3 10-6 10-7 10-9</td>
</tr>
</tbody>
</table>

Output: 1 + 3 + 13 = 17
Counting Inversions

Q. How to count inversions \((a, b)\) with \(a \in A\) and \(b \in B\)?

A. Easy if \(A\) and \(B\) are sorted!

Count inversions \((a, b)\) with \(a \in A\) and \(b \in B\), assuming \(A\) and \(B\) are sorted.

- Scan \(A\) and \(B\) from left to right.
- Compare \(a_i\) and \(b_j\).
- If \(a_i < b_j\), then \(a_i\) is not inverted with any element left in \(B\).
- If \(a_i > b_j\), then \(b_j\) is inverted with every element left in \(A\).
- Append smaller element to sorted list \(C\).
Counting Inversions

• How do we formally prove correctness?
  ➢ Induction on $n$ is usually very helpful
  ➢ Allows you to assume correctness of subproblems

• Running time analysis
  ➢ Suppose $T(n)$ is the running time for inputs of size $n$
  ➢ Our algorithm satisfies $T(n) = 2T(n/2) + O(n)$
  ➢ Master theorem says this is $T(n) = O(n \log n)$
Without Master Theorem

Let’s say $T(n) = 2 \ T\left(\frac{n}{2}\right) + 2n$

Overall: $2n \log n$
Closest Pair in $\mathbb{R}^2$

• Problem:
  ➢ Given $n$ points of the form $(x_i, y_i)$ in the plane, find the closest pair of points.

• Applications:
  ➢ Basic primitive in graphics and computer vision
  ➢ Geographic information systems, molecular modeling, air traffic control
  ➢ Special case of nearest neighbor

• Brute force: $\Theta(n^2)$
Intuition from 1D?

• In 1D, the problem would be easily $O(n \log n)$
  ➢ Sort and check!

• Sorting attempt in 2D
  ➢ Find closest points by x coordinate
  ➢ Find closest points by y coordinate

• Non-degeneracy assumption
  ➢ No two points have the same x or y coordinate
Intuition from 1D?

• **Sorting attempt in 2D**
  - Find closest points by x or y coordinate
  - Doesn’t work!
Closest Pair in $\mathbb{R}^2$

• Let’s try divide-and-conquer!
  ➢ Divide: points in equal halves by drawing a vertical line $L$
  ➢ Conquer: solve each half recursively
  ➢ Combine: find closest pair with one point on each side of $L$
  ➢ Return the best of 3 solutions

Seems like $\Omega(n^2)$ 😞
Closest Pair in $\mathbb{R}^2$

• Combine
  ➢ We can restrict our attention to points within $\delta$ of $L$ on each side, where $\delta = \min(12, 21)$ is the best of the solutions in two halves
Closest Pair in $\mathbb{R}^2$

- Combine (let $\delta = \text{best of solutions in two halves}$)
  - Only need to look at points within $\delta$ of $L$ on each side,
  - Sort points on the strip by $y$ coordinate
  - Only need to check each point with next 11 points in sorted list!

Wait, what? Why 11?
Why 11?

• Claim:
  ➢ If two points are at least 12 positions apart in the sorted list, their distance is at least $\delta$

• Proof:
  ➢ No two points lie in the same $\frac{\delta}{2} \times \frac{\delta}{2}$ box
  ➢ Two points that are more than two rows apart are at distance at least $\delta$
Recap: Karatsuba’s Algorithm

• Fast way to multiply two $n$ digit integers $x$ and $y$
• Brute force: $O(n^2)$ operations
• Karatsuba’s observation:
  ➢ Divide each integer into two parts
    o $x = x_1 \cdot 10^{n/2} + x_2, y = y_1 \cdot 10^{n/2} + y_2$
    o $xy = (x_1 y_1) \cdot 10^n + (x_1 y_2 + x_2 y_1) \cdot 10^{n/2} + (x_2 y_2)$
  ➢ Four $\frac{n}{2}$-digit multiplications can be replaced by three
    o $x_1 y_2 + x_2 y_1 = (x_1 + x_2)(y_1 + y_2) - x_1 y_1 - x_2 y_2$
  ➢ Running time
    o $T(n) = 3 T(\frac{n}{2}) + O(n) \Rightarrow T(n) = O(n^{\log_2 3})$
Strassen’s Algorithm

• Generalizes Karatsuba’s insight to design a fast algorithm for multiplying two $n \times n$ matrices
  ➢ Call $n$ the “size” of the problem
  
  \[
  \begin{bmatrix}
  C_{11} & C_{12} \\
  C_{21} & C_{22}
  \end{bmatrix} =
  \begin{bmatrix}
  A_{11} & A_{12} \\
  A_{21} & A_{22}
  \end{bmatrix} \ast
  \begin{bmatrix}
  B_{11} & B_{12} \\
  B_{21} & B_{22}
  \end{bmatrix}
  \]

  ➢ Naively, this requires $8$ multiplications of size $n/2$
    ◦ $A_{11} \ast B_{11}, A_{12} \ast B_{21}, A_{11} \ast B_{12}, A_{12} \ast B_{22}, ...$

  ➢ Strassen’s insight: replace $8$ multiplications by $7$
    ◦ Running time: $T(n) = 7 T(n/2) + O(n^2) \Rightarrow T(n) = O(n^{\log_2 7})$
Strassen’s Algorithm

\[
\begin{bmatrix}
C_{11} & C_{12} \\
C_{21} & C_{22}
\end{bmatrix} =
\begin{bmatrix}
A_{11} & A_{12} \\
A_{21} & A_{22}
\end{bmatrix} \times
\begin{bmatrix}
B_{11} & B_{12} \\
B_{21} & B_{22}
\end{bmatrix}
\]

**STRASSEN**(n, A, B)

*IF* (n = 1) *RETURN* A × B.

Partition A and B into 2-by-2 block matrices.

\[P_1 \leftarrow \text{STRASSEN}(n/2, A_{11}, (B_{12} - B_{22})).\]
\[P_2 \leftarrow \text{STRASSEN}(n/2, (A_{11} + A_{12}), B_{22}).\]
\[P_3 \leftarrow \text{STRASSEN}(n/2, (A_{21} + A_{22}), B_{11}).\]
\[P_4 \leftarrow \text{STRASSEN}(n/2, (B_{31} - B_{11})).\]
\[P_5 \leftarrow \text{STRASSEN}(n/2, (A_{11} + A_{12}) \times (B_{11} + B_{22})).\]
\[P_6 \leftarrow \text{STRASSEN}(n/2, (A_{12} - A_{22}) \times (B_{21} + B_{22})).\]
\[P_7 \leftarrow \text{STRASSEN}(n/2, (A_{11} - A_{21}) \times (B_{11} + B_{12})).\]

\[C_{11} = P_5 + P_4 - P_2 + P_6.\]
\[C_{12} = P_1 + P_2.\]
\[C_{21} = P_3 + P_4.\]
\[C_{22} = P_1 + P_5 - P_3 - P_7.\]

*RETURN* C.
Median & Selection

• **Selection:**
  - Given array $A$ of $n$ comparable elements, find $k$th smallest
  - $k = 1$ is min, $k = n$ is max, $k = [(n + 1)/2]$ is median
  - $O(n)$ is easy for min/max

• What about $k$-selection?
  - $O(nk)$ by modifying bubble sort
  - $O(n \log n)$ by sorting
  - $O(n + k \log n)$ using min-heap
  - $O(k + n \log k)$ using max-heap

• **Q:** What about just $O(n)$?
• **A:** Yes! Selection is easier than sorting.
QuickSelect

• Find a pivot $p$

• Divide $A$ into two sub-arrays
  ➢ $A_{\text{less}} =$ elements $\leq p$, $A_{\text{more}} =$ elements $> p$
  ➢ If $|A_{\text{less}}| \geq k$, return $k$th smallest in $A_{\text{less}}$, otherwise return $(k - |A_{\text{less}}|)$th smallest in $A_{\text{more}}$

• Problem?
  ➢ If pivot is close to the min or the max, then we basically get $T(n) \leq T(n - 1) + O(n)$, which only gives $T(n) = O(n^2)$
  ➢ Want to reduce $n - 1$ to a fraction of $n$ (like $n/2$, $5n/6$, etc)
Finding a Good Pivot

• Divide $n$ elements into $\frac{n}{5}$ groups of 5 each

\[ n = 54 \]
Finding a Good Pivot

• Divide \( n \) elements into \( n/5 \) groups of 5 each
• Find the median of each group
Finding a Good Pivot

• Divide $n$ elements into $n/5$ groups of 5 each
• Find the median of each group
• Find the median of $n/5$ medians
Finding a Good Pivot

• Divide $n$ elements into $n/5$ groups of 5 each
• Find the median of each group
• Find the median of $n/5$ medians
• Use this median of medians as the pivot in quickselect

• Q: Why does this work?
Analysis

• How many elements can be $\leq p^*$?
  ➢ Out of $n/5$ medians, $n/10$ are $> p^*$
Analysis

• How many elements can be \( \leq p^* \)?
  ➢ Out of \( n/5 \) medians, \( n/10 \) are \( > p^* \)
Analysis

• $n/10$ of the $n/5$ medians are $\leq p^*$
  ➢ For each such median, there are 3 elements $\leq p^*$
  ➢ So there can be at most $7n/10$ elements that can be $> p^*$
Analysis

• Thus, $|A_{more}| \leq \frac{7n}{10}$
  ➢ Similarly, $|A_{less}| \leq \frac{7n}{10}$
  ➢ (These are rough calculations...)

• How does this factor into overall algorithm analysis?
Analysis

• Divide $n$ elements into $n/5$ groups of 5 each
• Find the median of each group
• Find $p^* = \text{median of } n/5 \text{ medians}$
• Create $A_{less}$ and $A_{more}$ according to $p^*$
• Run selection on one of $A_{less}$ or $A_{more}$

$T(n) \leq T(n/5) + T(7n/10) + O(n)$

• Note: $n/5 + 7n/10 = 9n/10$
  ➢ Only a fraction of $n$, so by the Master theorem, $T(n) = O(n)$
• **Best algorithm for a problem?**
  - Typically hard to determine
  - We still don’t know best algorithms for multiplying two $n$-digit integers or two $n \times n$ matrices
    - **Integer multiplication**
      - Breakthrough in March 2019: first $O(n \log n)$ time algorithm
      - It is conjectured that this is asymptotically optimal
    - **Matrix multiplication**
      - 1969 (Strassen): $O(n^{2.807})$
      - 1990: $O(n^{2.376})$
      - 2013: $O(n^{2.3729})$
      - 2014: $O(n^{2.3728639})$
Residual Notes

• Best algorithm for a problem?
  ➢ Usually, we design an algorithm and then analyze its running time

  ➢ Sometimes we can do the reverse:
    o E.g., if you know you want an $O(n^2 \log n)$ algorithm
    o Master theorem suggests that you can get it by
      \[ T(n) = 4 \, T\left(\frac{n}{2}\right) + O(n^2) \]
    o So maybe you want to break your problem into 4 problems of size $n/2$ each, and then do $O(n^2)$ computation to combine
Residual Notes

• Access to input
  ➢ For much of this analysis, we are assuming random access to elements of input
  ➢ So we’re ignoring underlying data structures (e.g. doubly linked list, binary tree, etc.)

• Machine operations
  ➢ We’re only counting the number of comparison or arithmetic operations
  ➢ So we’re ignoring issues like how real numbers are stored in the closest pair problem
  ➢ When we get to P vs NP, representation will matter
Residual Notes

• Size of the problem
  ➢ Can be any reasonable parameter of the problem
  ➢ E.g., for matrix multiplication, we used $n$ as the size
  ➢ But an input consists of two matrices with $n^2$ entries
  ➢ It doesn’t matter whether we call $n$ or $n^2$ the size of the problem
  ➢ The actual running time of the algorithm won’t change