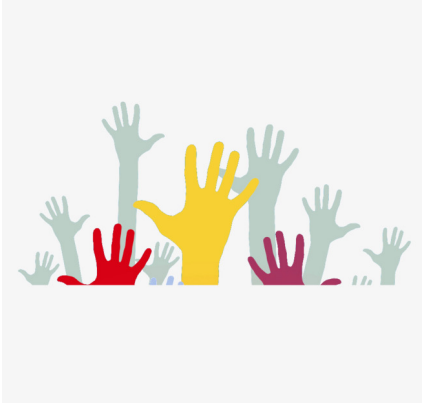
Advanced Algorithms

Lecture 1: Introduction, course overview

Backgrounds

- First year (MS/PhD)
- First Grad class
- UG
- Most recent algorithms class
- Discrete math/probability



al·go·rithm

/ˈalgəˌriT͟Həm/

noun

noun: **algorithm**; plural noun: **algorithms**

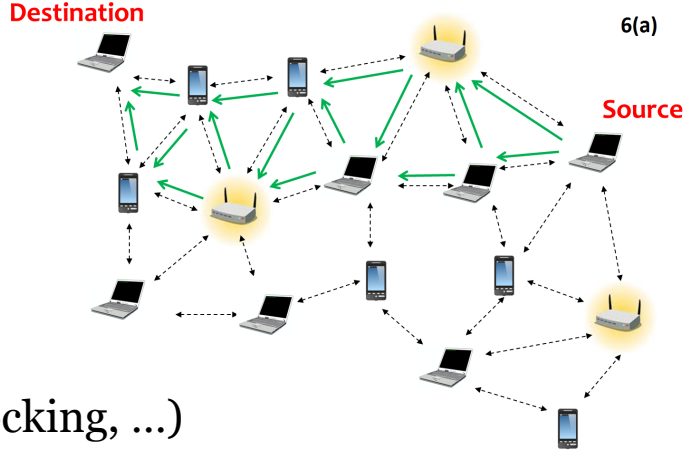
a process or set of rules to be followed in calculations or other problem-solving operations, especially by a computer.
"a basic **algorithm for** division"

Origin

late 17th century (denoting the Arabic or decimal notation of numbers): variant (influenced by Greek *arithmos* 'number') of Middle English *algorism*, via Old French from medieval Latin *algorismus* . The Arabic source, *al-K̲wārizmī* 'the man of K̲wārizm' (now Khiva), was a name given to the 9th-century mathematician Abū Ja'far Muhammad ibn Mūsa, author of widely translated works on algebra and arithmetic.

Algorithms

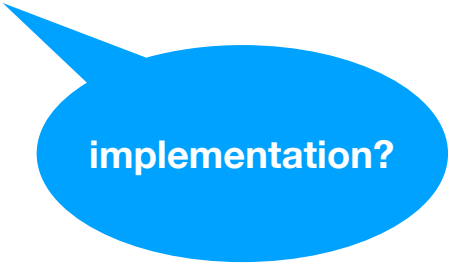
- Backbone of every computation



- Nature (e.g. evolution, bird flocking, …)

Goals of the course

- Principles of designing "efficient" algorithms
- Analyze the running time, memory utilization, etc. — **reasoning** about algorithms
- **Proving correctness** — why?
- Understanding the limits — are there tasks we *cannot* hope to perform efficiently?



Logistics

- Course homepage: **Canvas**
- Policies
- Homeworks submitted via canvas
- **Piazza** for discussions (create test post thread)
- Instructor: **Aditya Bhaskara** (bhaskara.course @ gmail)
  Office: MEB 3470

TAs

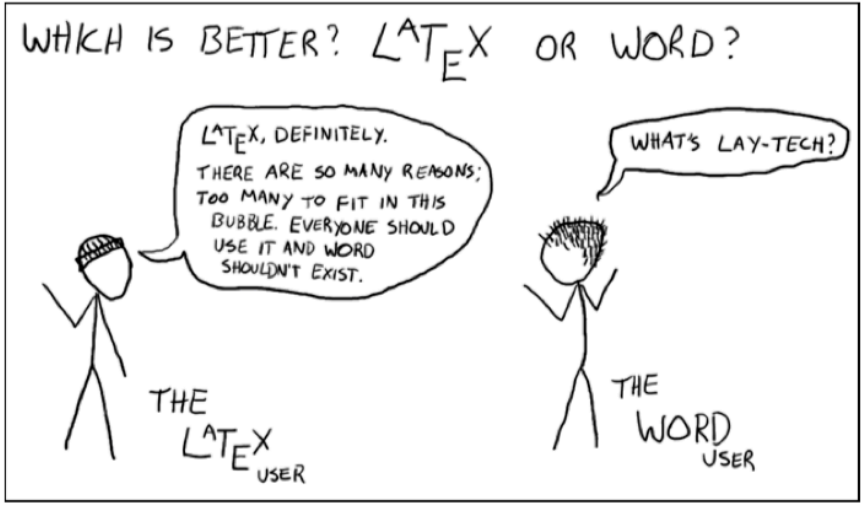
- Vivek Gupta — contact person for grading questions
- Pegah Nokhiz
- Kanchana Ruwanpathirana

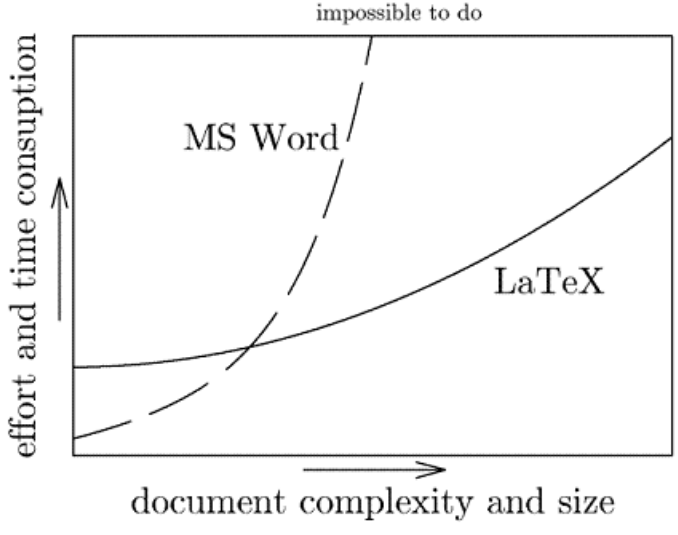
Contact information and office hours will be posted on canvas today

Grading

- Homeworks (60%): best 5 out of 6
- Mid-term exam (15%)
- Final exam (25%)

HW submission





All HWs must be prepared in LaTeX or Markdown and submitted as PDF

→ Typora

- New to LaTeX?
  - Start HW early
  - Try Markdown (e.g. Typora)

- Submit HW0 this Friday
- Citation, collaboration policy

Pre-requisites

Basic data structures, analysis

different ways of storing sets of #s.

- Arrays, lists, binary search trees → allows you to quickly tell if a "query" x is present in the set or not.
- Big-Oh notation (don't care about constants), Little-oh, ...
- T(n) = 4 n^2 + 10 n + 15

$O(n^2)$ → $o(n^3)$; $o(n^2 \log n)$; $10^{100} \cdot n$

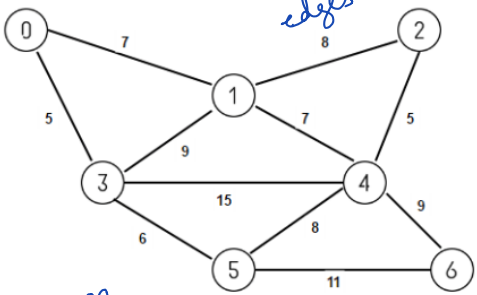
We Defn: Let $f(n)$ and $g(n)$ be two functions. Then we say that $f(n) \in o(g(n))$ if for any constant $c > 0$, $f(n) < c\,g(n)$ for all large enough n.

Graphs

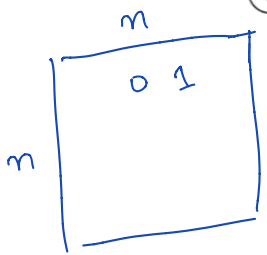
for each vertex, we have a list of neighbors.

vertices / nodes

edges.



- Storing graphs (adjacency lists, matrix)
- Breadth/depth first search
- "Reachability"



linear in the size of the graph. (#vertices + #edges).

Probability

- Random variables, expected values 
- Computing probabilities of simple events
- Homework 0

Code vs pseudocode

```
bool sorted(vector<int> &A) {
        for (int i=0; i<N-1; i++)
                if (A[i] > A[i+1]) return false;
        return true;
}

void sort(vector<int> &A) {
        while (!sorted(A)) {
                for (int i=0; i<N; i++) {
                        if (A[i] > A[i+1]) swap(A, i, i+1);
                }
        }
}
```

Code vs pseudocode

```
input: array A[0, …, N-1]

procedure sort(A):
        while A is not sorted do:
                for i from 0… N-1:
                        if (A[i] > A[i+1]), swap them;
```

*Use text, not code; assume **basic** sub-routines…*

Analysis example

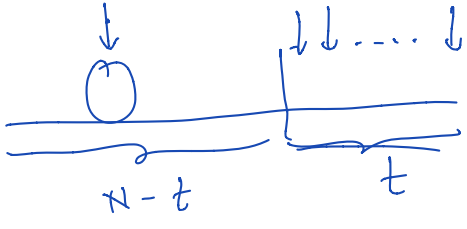
```
input: array A[0, …, N-1]

procedure sort(A):
        while A is not sorted do:
                for i from 0… N-1:
                        if (A[i] > A[i+1]), swap them;
```

Bubble-sort.

Time complexity:

- What is the worst-case running time? space?
- Are there inputs where it runs faster?
- How fast is it “typically”?



Worst-case running time

Claim: For every input, the .while. loop runs at most n times.

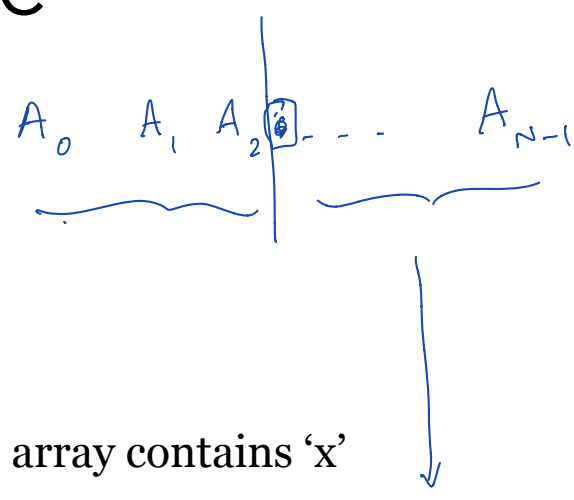
Obsn: After t iterations, the largest 't' elements are all in their correct positions.

Obsn: After first iter, largest element moves its correct position

Proof by induction:
- base case: $t = 1$
- inductive step: if statement is true for t, it is true for $t+1$.

"Proof by picture"

Example



Given an array of N integers:

A[0] < A[1] < A[2] < …. < A[N-1]

Question: given an integer 'x', find if the array contains 'x'

What if array isn't sorted?

reduces the "size of array" by 1/2.

$N \xrightarrow{1} \frac{N}{2} \xrightarrow{2} \frac{N}{4} \rightsquigarrow \dots$ (t)

$\frac{N}{2^t} = 1$

$N = 2^t \Rightarrow t = \log_2 N$

Algorithm

Running time

Caveat: reading input?

Correctness

Three key steps

- Describe algorithm
- Analyze run time/complexity
- Prove correctness