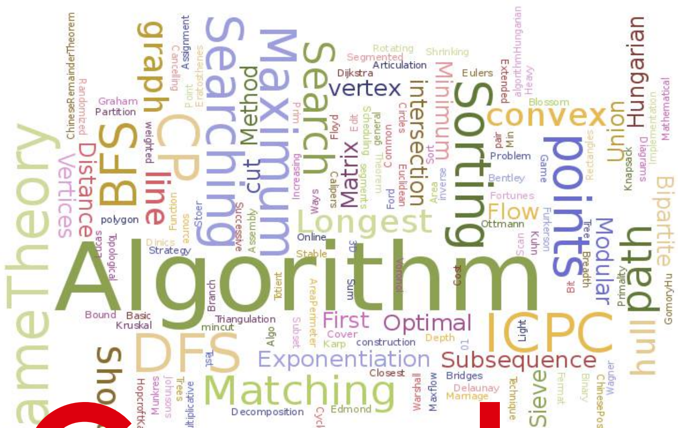
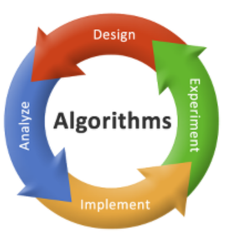


Complexity Analysis of Algorithms

Course: Algorithms



Faculty: Dr. Rajendra Prasath

Autumn 2018

Computational Complexity

This class covers many important aspects of **Computational Complexity** to be considered while solving a specific problem. This lecture illustrates the Best, Average and Best cases of algorithms

Recap: Types of Algorithms

- **Algorithms are classified**
  - **By implementation**
  - **By Design Paradigm**
  - **By Domain Specific study**
  - **By Computational Complexity**
- There are different variations of algorithms and their purpose might be different from the classical theory of algorithms

Computational Complexity

- Running Times of the algorithms may vary based on n.
  - Fastest Algorithms – Less time
  - Slowest Algorithms – More time

- Order of Growth (Magnitude)
  - **How fast the running time grows with the input size, say n?**

- How to perform accurate analysis of algorithms in terms of its computational complexities:
  - Worst, average and Best cases

Running Time Estimation

SIZE / COMPLEXITY	20	50	100	200	500	1000
1000n	.02 sec	.05 sec	.1 sec	.2 sec	.5 sec	1 sec
1000n lg n	.09 sec	.3 sec	.6 sec	1.5 sec	4.5 sec	10 sec
$100n^2$	.04 sec	.25 sec	1 sec	4 sec	25 sec	2 min
$10n^3$	.02 sec	1 sec	10 sec	1 min	21 min	2.7 hr
$n^{\lg n}$	.4 sec	1.1 hr	220 DAYS	125 CENT	5×10^8 CENT	
$2^{n/3}$	.0001 sec	.1 sec	2.7 hr	3×10^4 CENT		
2^n	1 sec	35 YR	3×10^4 CENT			
3^n	58 min	2×10^9 CENT				

- One step takes one Microsecond. lg n denotes $\log_2 n$

Computational Complexity

TIME / COMPLEXITY	1sec	10^2 sec (1.7 min)	10^4 sec (2.7 hr)	10^6 sec (12 DAYS)	10^8 sec (3 YEARS)	10^{10} sec (3 CENT.)
$1000n$	10^3	10^5	10^7	10^9	10^{11}	10^{13}
$1000n \lg n$	1.4×10^2	7.7×10^3	5.2×10^5	3.9×10^7	3.1×10^9	2.6×10^{11}
$100n^2$	10^2	10^3	10^4	10^5	10^6	10^7
$10n^3$	46	2.1×10^2	10^3	4.6×10^3	2.1×10^4	10^5
$n^{\lg n}$	22	36	54	79	112	156
$2^{n/3}$	59	79	99	119	139	159
2^n	19	26	33	39	46	53
3^n	12	16	20	25	29	33

- As the problem size increases, polynomial time algorithm become unusable gradually.
- A factor of ten increase in machine speed corresponds to a factor of ten increase in time.

The Spectrum of Computational Complexity

TRACKTABLE

Polynomial	P
	Linear Programming
n^3	
	Matrix Multiplication
n^2	
n logn	Sorting
n	Selection

The Spectrum of Computational Complexity

INTRACKTABLE

Undecidable (with no algorithms)	Hilbert's Tenth Problem
Superexponential	Presburger Arithmetic
Exponential	Circularity of Attribute Grammers
	NP-Complete Problems

Hilbert's Tenth Problem

- Hilbert Spaces
  - Vibrating String can be modeled as a point in Hilbert Space (foundations of Functional Analysis)
- In 1900, Hilbert proposed 23 hard problems:

- The 10th Hard Problem:
  - Determining the solvability of a Diophantus equation:
  - Given a Diophantus equation with any number of unknowns and with rational integer coefficients: Devise a process, which could determine by a finite number of operations whether the equation is solvable in rational integers?

  - **Diophantine equation** $3x^2 - 2xy - y^2z - 7 = 0$ **has an integer solution:** $x = 1, \quad y = 2, \quad z = -2$
  - But the Diophantine equation $x^2 + y^2 + 1 = 0$ has no such solution.

Efficiency of Algorithms

- Assume that we have a processor that executes a million high-level instructions per second and we have algorithms with polynomial running-time

How to define Efficiency?

- An algorithm is efficient if, when implemented, it runs quickly on real input instances.

- An algorithm is efficient if it achieves qualitatively better worst-case performance, at an analytical level, than brute-force search

- An algorithm is efficient if it has a polynomial running time

  - **Growing Polynomials:**
    $1 < n < n \log_2 n < n^2 < n^3 < 1.5n < 2^n < n!$

Asymptotic Upper Bounds

"Big-O" Notation

(introduced in P. Bachmann's 1892 book Analytische Zahlentheorie_

- Let $f(n)$ be a function
  - Say the worst case running time of a certain algorithm on an input of size n
- and $g(n)$ be another function

- We say that $f(n)$ is O ($g(n)$) for sufficiently large n, the function $f(n)$ is bounded by a constant multiple of $g(n)$

- More Precisely, $f(n)$ is O ($g(n)$) if **there exist** constants c > 0 and $n_0 \geq 0$ so that for all $n \geq n_0$, we have $f(n) \leq c \cdot g(n)$

- The constant c can not depend on n

Asymptotic Lower Bounds

"Omega (Ω)" Notation

- Let $f(n)$ be a function and $g(n)$ be another function

- We say that $f(n)$ is $\Omega(g(n))$ if there exist constants $c > 0$ and $n_0 \geq 0$ so that for all $n \geq n_0$, we have $f(n) \geq c \cdot g(n)$

- Note that the constant c must be fixed, independent of n.

Asymptotic Tight Bounds

"Theta (Θ)" Notation

- Let $f(n)$ be a function and let $g(n)$ be another function.
- We say that $f(n)$ is $\Theta(g(n))$ if $f(n)$ if both $O(g(n))$ and $\Omega(g(n))$.
- Asymptotically the tight bounds characterize the worst case performance of an algorithm precisely upto constant factors.
- It closes the gap between an upper bound and a lower bound.

Asymptotic Growth Rates

- A way of comparing functions that ignores constant factors and small input sizes
- $O(g(n))$: class of functions $f(n)$ that grow no faster than $g(n)$
- $\Theta\ (g(n))$: class of functions $f(n)$ that grow at same rate as $g(n)$
- $\Omega\ (g(n))$: class of functions $f(n)$ that grow at least as fast as $g(n)$

Computational Complexity

Functions

Constant << Logarithmic << Poly Logarithmic << Polynomial << Exponential << Factorial

$5 \ll 5 \log n \ll (\log n)^5 \ll n^5 \ll 2^{5n} \ll n!$

Identify the Constants

- Yes • 5
- Yes • 1,000,000,000,000
- Yes • 0.0000000000001
- Yes • -5
- Yes • 0
- No • 8 + sin(n)

The running time of the algorithm is a "Constant" if it does not depend significantly on the input size

Quadratic Functions?

- n^2
- $0.001\ n^2$
- $1000\ n^2$
- $5n^2 + 3n + 2\log n$

Ignore low-order terms
Ignore multiplicative constants
Ignore "small" values of n

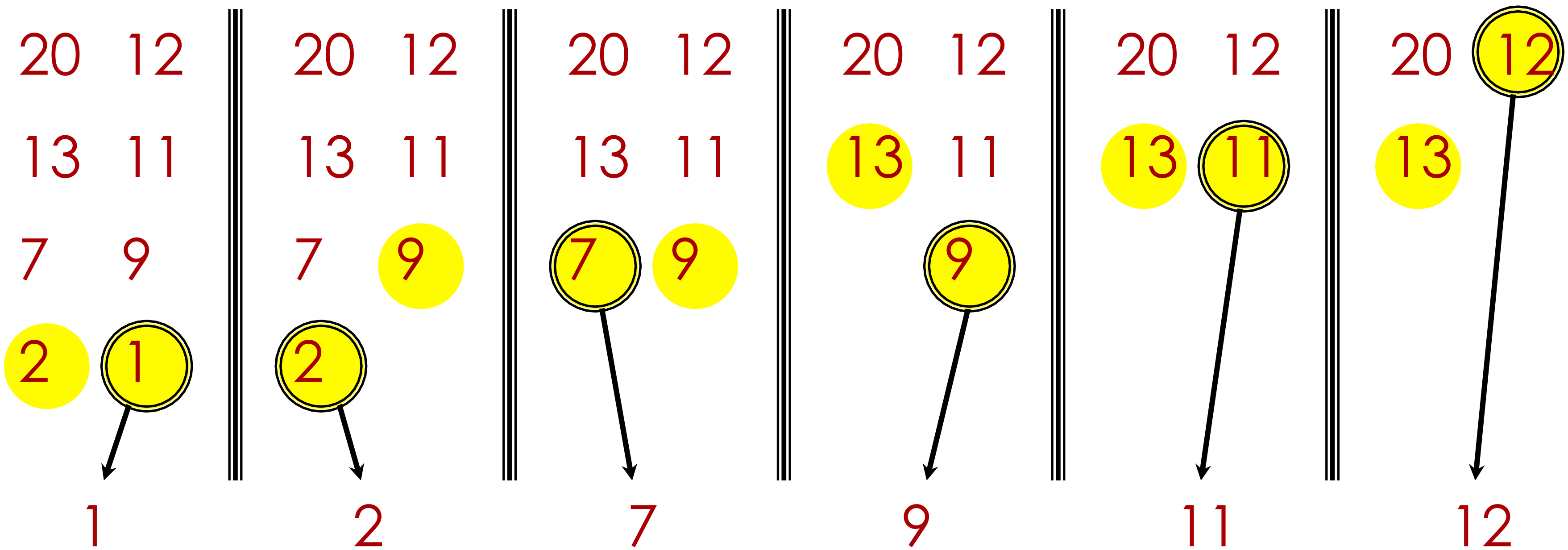
Write $\Theta(n^2)$

Time Efficiency of Non-recursive Algorithms

- Decide on parameter n indicating input size
- Identify algorithm's basic operation
- Determine worst, average, and best cases for input of size n
- Set up a sum for the number of times the basic operation is executed
- Simplify the sum using standard formulas and rules

An Example

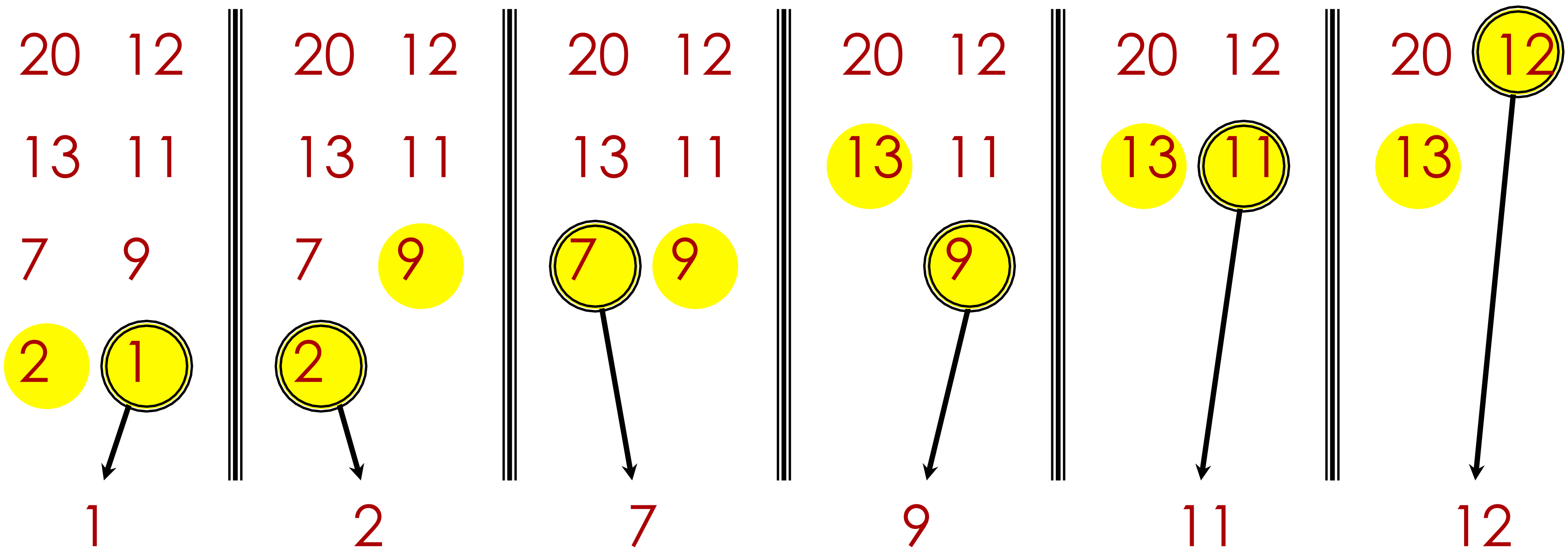
- How do we merge Two Sorted Arrays?



Time = $\Theta(n)$ to merge a total of n elements (linear time)

An Example

- How do we merge Two Sorted Arrays?



Time = Θ(n) to merge a total of n elements (linear time)

20

Help among Yourselves?

- **Perspective Students** (having CGPA above 8.5 and above)
- **Promising Students** (having CGPA above 6.5 and less than 8.5)

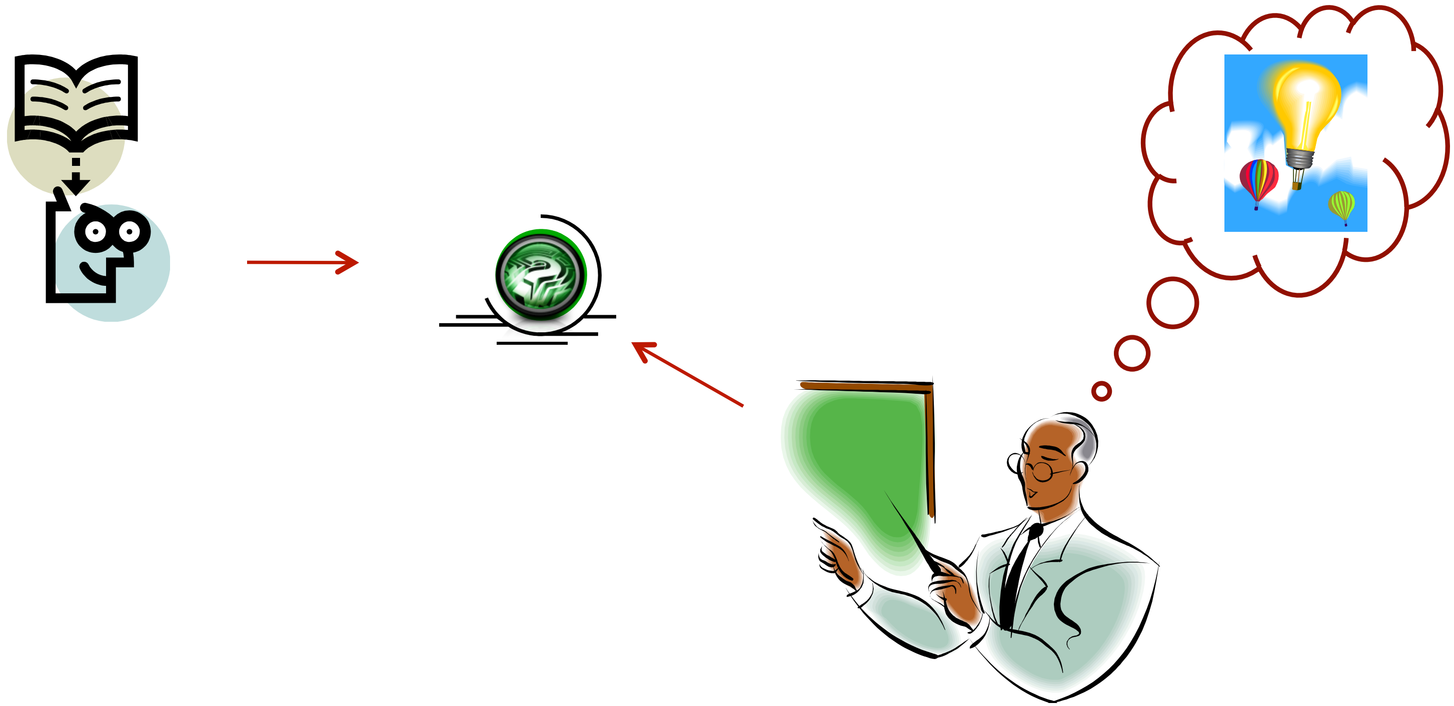
- **Needy Students** (having CGPA less than 6.5)
  - Can the above group help these students? (Your work will also be rewarded)

- You may grow a culture of **collaborative learning** by helping the needy students

Assistance

- You may post your questions to me at any time
- You may meet me in person on available time or with an appointment
- TA s would assist you to clear your doubts.
- You may leave me an email any time (email is the best way to reach me faster)

Thanks …



… Questions ???