CSI 5163 (95.573) ALGORITHM ANALYSIS AND DESIGN

- **CSI 5163 (95.5703) ALGORITHM ANALYSIS AND DESIGN** (3 cr.) (T)
  Topics of current interest in the design and analysis of computer algorithms for graph-theoretical applications; e.g. shortest paths, chromatic number, etc. Lower bounds, upper bounds, and average performance of algorithms. Complexity theory.

- PROFESSOR: Dr. Nejib Zaguia
  - SITE 5-031, 562-5800 ext.:6782
  - zaguia@site.uottawa.ca
  - Office Hours: Friday11:30-13:00

- LECTURES:
  - Thursday, 18:00-21:00, LMX 339
- REFERENCES
- **Textbook**: Kleinberg and Tardos, *Algorithm Design*, Addison Wesley, 7th edition.
- Introduction to algorithms, T.H. Cormen, C.E. Leiserson and R. L. Rivest, McGraw-Hill Book Company, 1990.
- The design and analysis of computer algorithms, A.V.Aha, M.R.Garey and J.D.Ullman, Addison-Wesley
- Combinatorial Optimization: algorithms and complexity, C.H. Papadimitriou and K. Steiglitz, Prentice Hall 1982.

CSI 5163 (95.573) ALGORITHM ANALYSIS AND DESIGN

COURSE OBJECTIVES

This is a course on the design and analysis of algorithms. Core results and techniques are introduced, which are useful to those planning to specialize in other areas in computer science. Moreover, some fairly advanced topics will be covered. This will provide an idea of the current research for the benefit of those who might wish to specialize in this area. In the last few lectures we will study a number of specific applications.

COURSE OUTLINE

- Introduction:
  - Introduction of the course, asymptotic notations, basic data structure
- Basic Graph algorithms and Basic algorithms design (greedy, divide and conquer, dynamic programming, …)
- More Graph algorithms:
  - Chromatic number, Network flow, ..

CSI 5163 (95.573) ALGORITHM ANALYSIS AND DESIGN

- **Marking scheme:**
  - Report on the topic and Research paper : 30%
  - Oral presentation on the Research paper : 15%
  - Participation during presentations 5%
  - 2 Midterm Exams in Class (February 13, march 13) 50%
- **General Information**
- A reading project is required to fulfill the class requirements. A list of papers and topics on algorithms will be posted by January 20th. You must write a report of about 15 pages about your topic and also explaining the results of the research paper and their proofs. You must read you paper in depth. Your report should not just be a condensed version of the original paper, but should explain why the paper is interesting and present the main ideas in the way you understand it best, perhaps using examples or special cases, or perhaps reworking a proof. Or you might fill in missing details of the original paper. You must give a talk of about 10-15 minutes explaining the most important and major points in the research paper.

Evaluation of reports.

- Evaluation will be based on the following criterion: (for the purpose of marking these criterion will be considered approximately equally, except where noted.)
  - Writing quality
  - Relevance and incorporation of the ideas presented during the lectures of this course
  - Depth of the analysis and understanding of the paper …

Evaluation of presentations.

- Presentations will be evaluated based on the format and quality of the presentation (organization, quality of overheads, colors, graphics, speaking clearly, managing the allotted time of the presentation, …)
- It should not be a repetition of materials from the paper, lecture notes or textbooks.

- **Important:**
  - All writing reports should be returned on March 20th before the beginning of the lecture.
  - A copy of your presentations (hardcopy) should be returned at the beginning of your talk. Please send me a soft copy of the reports and the presentations by e-mail.

Asymptotic performance

- How does the algorithm behave as the problem size gets very large?
  - Running time
  - Memory/storage requirements
  - Bandwidth/power requirements/logic gates/etc.

Computational model

- The assumption: generic uniprocessor random-access machine (RAM)
  - All memory equally expensive to access
  - No concurrent operations
  - All reasonable instructions take unit time
    - Except, of course, function calls
  - Constant word size
    - Unless we are explicitly manipulating bits

Input Size

- Time and space complexity
  - This is generally a function of the input size
    - e.g., sorting, multiplication
  - How we characterize input size depends:
    - Sorting: number of input items
    - Multiplication: total number of bits
    - Graph algorithms: number of nodes & edges
    - etc
  - Number of primitive steps that are executed
    - Except for time of executing a function call, most statements roughly require the same amount of time

Analysis

- Worst case
  - Provides an upper bound on running time
  - An absolute guarantee
- Average case
  - Provides the expected running time
  - Very useful, but treat with care: what is "average"?
    - Random (equally likely) inputs
    - Real-life inputs

An Example: Insertion Sort

```
InsertionSort(A, n) {
  for i = 2 to n {
      key = A[i]
      j = i - 1;
      while (j > 0) and (A[j] > key) {
           A[j+1] = A[j]
           j = j - 1
      }
      A[j+1] = key
  }
}
```

Insertion Sort

```
InsertionSort(A, n) {                          T(n)
for i = 2 to n {                               n-1
      key = A[i]                               1
      j = i - 1;                               1
      while (j > 0) and (A[j] > key) {         i-1
            A[j+1] = A[j]                      1
            j = j - 1                          1
      }                                        0
      A[j+1] = key                             1
   }                                           0
}
```

Average case???

Best case: while loop body never executed → T(n) is a linear function T(n)= n-1

Worst case: while loop body executed for all previous elements →

$T(n) = \sum_{2 \le i \le n}(2*(i-1)+1+1+1)= \sum_{2 \le i \le n}(2i+1)= 2(\sum_{2 \le i \le n} i) + (n-1)$

$= [(n+1)n/2] - 1 + (n-1) = [n^2 + 3n - 4]/2$ T(n) is a quadratic function

Analysis

Simplifications

- Ignore actual and abstract statement costs
- *Order of growth* is the interesting measure:
  - Highest-order term is what counts
    - Remember, we are doing asymptotic analysis
    - As the input size grows larger it is the high order term that dominates

A function f(n) is O(g(n)) if there exist positive constants c and n_0 such that

$f(n) \le c \cdot g(n)$ for all $n \ge n_0$

Big O Fact

We say Insertion Sort's run time is $O(n^2)$

- A polynomial of degree k is $O(n^k)$

Suppose $f(n) = b_k n^k + b_{k-1} n^{k-1} + \ldots + b_1 n + b_0$

Let $a_i = |b_i|$

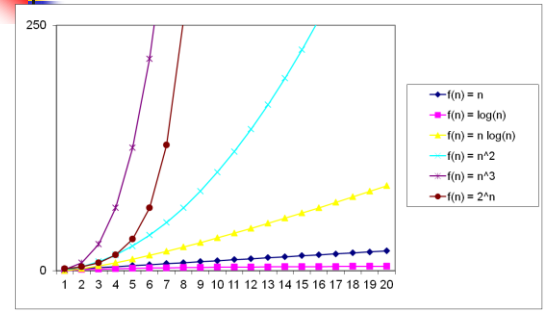
$f(n) \le a_k n^k + a_{k-1} n^{k-1} + \ldots + a_1 n + a_0$

$$\le \quad n^k \sum a_i \frac{n^i}{n^k} \quad \le \quad n^k \sum a_i \quad \le \quad cn^k$$

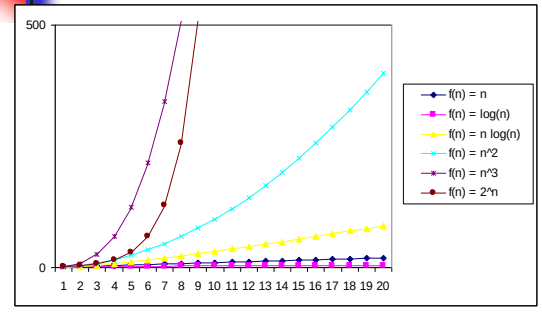
Lower Bound Notation

- A function f(n) is $\Omega(g(n))$ if there exists $c \geq 0$ and n_0 such that
  $$0 \leq c \cdot g(n) \leq f(n) \quad \forall\ n \geq n_0$$
- A function f(n) is $\Theta(g(n))$ if there exists $c_1 \geq 0$, $c_2 \geq 0$, and $n_0 \geq 0$ such that
  $$c_1\, g(n) \leq f(n) \leq c_2\, g(n) \quad \forall\ n \geq n_0$$
- f(n) is $\Theta(g(n))$ iff f(n) is both $O(g(n))$ and $\Omega(g(n))$

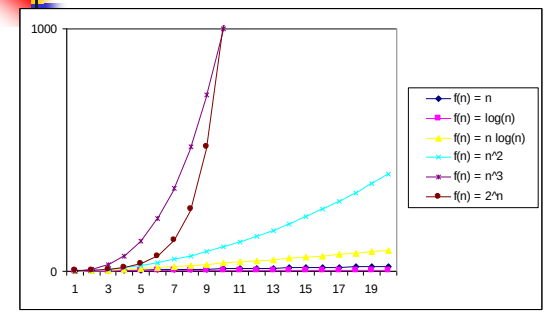
Practical Complexity



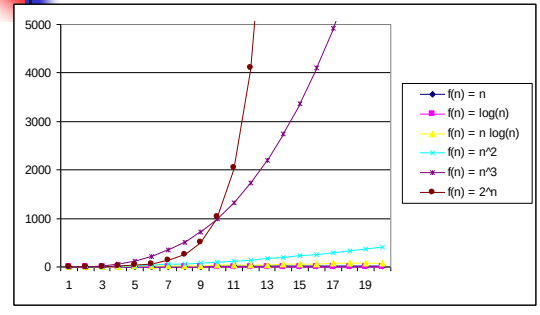
Practical Complexity



Practical Complexity



Practical Complexity



Practical Complexity

Order of complexity	10	20	30	40	50	60
n	.00001 seconds	.00002 seconds	.00003 seconds	.00004 seconds	.00005 seconds	.00006 seconds
n^2	.0001 seconds	.0004 seconds	.0009 seconds	.0016 seconds	.0025 seconds	.0036 seconds
n^3	.001 seconds	.008 seconds	.027 seconds	.064 seconds	.125 seconds	.216 seconds
n^5	.1 seconds	3.2 seconds	24.3 seconds	1.7 minutes	5.2 minutes	13.0 minutes
2^n	.001 seconds	1.0 second	17.9 minutes	12.7 days	35.7 years	366 centuries
3^n	.059 seconds	58 minutes	6.5 years	3855 centuries	$2x10^8$ centuries	$1.3x10^{13}$ centuries

Other Asymptotic Notations

- A function f(n) is o(g(n)) if there exists positive constants c and n_0 such that
  $f(n) < c\,g(n)\ \forall\ n \geq n_0$
- A function f(n) is ω(g(n)) if there exists positive constants c and n_0 such that
  $c\,g(n) < f(n)\ \forall\ n \geq n_0$
- Intuitively,
  - o() is like <
  - ω() is like >
  - Θ() is like =
  - O() is like ≤
  - Ω() is like ≥

Analysis of Merge Sort

Solving recurrences

```
MergeSort(A, left, right) {                    T(n)
   if (left < right) {                         Θ(1)
      mid = floor((left + right) / 2);         Θ(1)
      MergeSort(A, left, mid);                 T(n/2)
      MergeSort(A, mid+1, right);              T(n/2)
      Merge(A, left, mid, right);              Θ(n)
   }
}
```

So T(n) = Θ(1) when n = 1, and
2T(n/2) + Θ(n) when n > 1

Review: Solving Recurrences

The expression:

$$T(n) = \begin{cases} c & n = 1 \\ 2T\left(\frac{n}{2}\right) + cn & n > 1 \end{cases}$$

is a *recurrence*.

There are basic techniques to solve recurrence relations.

- Substitution method
- Iteration method
- Master method

The substitution method : *Guess the form of the answer, then use induction to find the constants and show that guessed solution works*

- *Example*: merge sort
  - T(n) = 2T(n/2) + cn
  - We guess that the answer is O(n lg n)
  - Prove it by induction
  - Can similarly show T(n) = Ω(n lg n), thus Θ(n lg n)

The Iteration method

1. Expand the recurrence
2. Work some algebra to express as a summation
3. Evaluate the summation

s(n) = c + s(n-1)
= c + c + s(n-2)
= 2c + s(n-2)
= 2c + c + s(n-3)
= 3c + s(n-3)
...
= cn + s(0) = cn

$$s(n) = \begin{cases} 0 & n = 0 \\ c + s(n-1) & n > 0 \end{cases}$$

s(n) = cn

Review: Solving Recurrences

s(n)

$$s(n) = \begin{cases} 0 & n = 0 \\ n + s(n-1) & n > 0 \end{cases}$$

= n + s(n-1)

= n + n-1 + s(n-2)

...

= n + n-1 + n-2 + n-3 + ... + n-(k-1) + s(n-k)

...

= n + n-1 + n-2 + n-3 + ... + 2 + 1 + s(0)

$$= \sum_{i=1}^{n} i + s(0) \qquad s(n) = n\frac{n+1}{2}$$

Review: Solving Recurrences

T(n) = 2T(n/2) + c

$$T(n) = \begin{cases} c & n = 1 \\ 2T\left(\frac{n}{2}\right) + c & n > 1 \end{cases}$$

= 2(2T(n/2/2) + c) + c

= 2^2T(n/2^2) + 2c + c

= 2^2(2T(n/2^2/2) + c) + 3c

= 2^3T(n/2^3) + 4c + 3c .....

= 2^kT(n/2^k) + (2^k - 1)c

.....

= $2^{\lg n}$ T(n/$2^{\lg n}$) + ($2^{\lg n}$ - 1)c

= n T(n/n) + (n - 1)c

= n T(1) + (n-1)c

= nc + (n-1)c = (2n - 1)c

Review: Solving Recurrences

What about this relation:

$$T(n) = \begin{cases} c & n = 1 \\ aT\left(\frac{n}{b}\right) + cn & n > 1 \end{cases}$$

Review: Solving Recurrences

$$T(n) = \begin{cases} c & n = 1 \\ aT\left(\frac{n}{b}\right) + cn & n > 1 \end{cases}$$

T(n) =
aT(n/b) + cn
a(aT(n/b/b) + cn/b) + cn
$a^2T(n/b^2)$ + cn(a/b + 1)
$a^2(aT(n/b^2/b) + cn/b^2)$ + cn(a/b + 1)
$a^3T(n/b^3)$ + cn(a^2/b^2 + a/b + 1)
...
$a^kT(n/b^k)$ + cn(a^{k-1}/b^{k-1} + a^{k-2}/b^{k-2} + ... + a^2/b^2 +a/b +1)

For $k = \log_b n$, that is $n = b^k$

T(n) = $a^kT(1)$ + cn(a^{k-1}/b^{k-1} + ... + a^2/b^2 + a/b + 1)
= cna^k/b^k + cn(a^{k-1}/b^{k-1} + ... + a^2/b^2 + a/b + 1)
= cn(a^k/b^k + ... + a^2/b^2 + a/b + 1)

Review: Solving Recurrences

T(n) = cn(a^k/b^k + ... + a^2/b^2 + a/b + 1, ($k=\log_b n$, $n=b^k$)

Recall that $(x^k + x^{k-1} + \ldots + x + 1) = (x^{k+1} - 1)/(x-1)$ *and* $a^{\log n} = n^{\log a}$

$$\frac{a^k}{b^k} + \frac{a^{k-1}}{b^{k-1}} + \cdots + \frac{a}{b} + 1 = \frac{(a/b)^{k+1} - 1}{(a/b) - 1} = \frac{1 - (a/b)^{k+1}}{1 - (a/b)}$$

If a = b then T(n) = cn(k + 1) =cn($\log_b$ n + 1)=Θ(n log n)

If a < b then $\frac{(a/b)^{k+1} - 1}{(a/b) - 1} = \Theta\left((a/b)^k\right)$ T(n) = cn ·Θ(1) = Θ(n)

If a > b then $\frac{(a/b)^{k+1} - 1}{(a/b) - 1} < \frac{1}{1 - a/b}$ T(n) = $\Theta(n^{\log a})$

T(n)=cn·Θ(a^k / b^k) =cn·Θ($a^{\log n}$ / $b^{\log n}$) = cn·Θ($a^{\log n}$ / n)
= cn · Θ($n^{\log a}$ / n) = Θ(cn · $n^{\log a}$ / n) = $\Theta(n^{\log a})$

Review: Solving Recurrences

So...

$$T(n) = \begin{cases} c & n = 1 \\ aT\left(\frac{n}{b}\right) + cn & n > 1 \end{cases}$$

$$T(n) = \begin{cases} \Theta(n) & a < b \\ \Theta(n \log_b n) & a = b \\ \Theta\left(n^{\log_b a}\right) & a > b \end{cases}$$

The Master Theorem

The Master Theorem is a very useful generalization of the previous theorem:

Consider any *divide and conquer* algorithm

- Suppose the algorithm divides the problem of size *n* into *a* subproblems, each of size *n*/*b*
- *f*(n) is the cost of subdividing the problem into the subroblems plus the cost of combining the solutions of subproblems

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

The Master Theorem

- if T(n) = aT(n/b) + f(n) then

$$T(n) = \begin{cases} \Theta\left(n^{\log_b a}\right) & f(n) = O\left(n^{\log_b a - \varepsilon}\right) \\ \Theta\left(n^{\log_b a} \log n\right) & f(n) = \Theta\left(n^{\log_b a}\right) \\ \Theta(f(n)) & f(n) = \Omega\left(n^{\log_b a + \varepsilon}\right) \text{AND} \\ & af(n/b) < cf(n) \text{ for large } n \end{cases} \quad \begin{matrix} \varepsilon > 0 \\ c < 1 \end{matrix}$$

Using The Master Method

- T(n) = 9T(n/3) + n

a=9, b=3, f(n) = n $n^{\log_b a} = n^{\log_3 9} = \Theta(n^2)$

Need to compare f(n)=n with $n^{\log_b a} = \Theta(n^2)$,

Since f(n)= $O(n^{2-\varepsilon})$, where ε=1, case 1 applies:

$$T(n) = \Theta\left(n^{\log_b a}\right) \text{when } f(n) = O\left(n^{\log_b a - \varepsilon}\right)$$

- Thus the solution is T(n) = $\Theta(n^2)$

Using The Master Method

- $T(n) = T(3n/4) + 1$ and $T(1)= \Theta(1)$

$a=1$, $b=4/3$, $f(n) = 1$ $\quad n^{\log_b a} = n^{\log_{4/3} 1} = (n^0) = 1$

Need to compare $f(n)=1$ with $n^{\log_b a} = 1$,

Since $f(n)= \Theta\ O(n^{\log_b a})$, case 2 applies:

- Thus the solution is $T(n) = \Theta(\log n)$

Using The Master Method

- $T(n) = 3T(n/3) + n \log n$ and $T(1)= \Theta(1)$

$a=3$, $b=3$, $f(n) = n \log n$ $\quad n^{\log_b a} = n^{\log_3 3} = \Theta(n)$

Need to compare $f(n)=n \log n$ with $n^{\log_b a} = \Theta(n)$,

Since $n \log n = \Omega(n)$, case 3 is the best possibility.

However there is no ε such that $n \log n = \Omega(n^{1+\varepsilon})$.

Thus the Master theorem cannot be applied to this recurrence.