CS 521 Lecture I

DREXEL UNIVERSITY
DEPT. OF COMPUTER SCIENCE

FALL 2011

CS 521

- This course is intended as a broad graduate-level introduction to the design and analysis of algorithms for some of the most frequently encountered combinatorial problems.
- No specific prior knowledge is necessary, but some exposure to the topic at the undergraduate level and some general mathematical background and maturity are assumed.
- The course aims to provide familiarity with general algorithmic techniques, performance measures, analysis tools, and problem areas.
- The topics covered by the course were selected according to practical importance, conceptual elegance, and breadth.

Text:

- Cormen, Leiserson, Rivest, and Stein Introduction to Algorithms MIT Press, 2001.
- Other references:
  - Kozen Design and Analysis of Algorithms Springer Verlag, 1992.
  - Tarjan, Data Structures and Network Algorithms, SIAM Series in Applied Mathematics 44, 1983.
  - Papadimitriou and Steiglitz Combinatorial Optimization: Algorithms and Complexity Prentice hall, 1982.

Syllabus

- The syllabus below is tentative and is subject to change as the course proceeds:
  - Asymptotic Analysis of algorithms.
  - Combinatorial Algorithms for searching and sorting.
  - Basic graph algorithms
    - depth-first and breadth-first search; Minimum Spanning trees; bi-connected components; shortest paths; transitive closure, maximum flow algorithms; etc.
  - Data Structures
    - Heaps; Binary Search Trees; Red-Black Trees; etc.
  - Introduction to intractability, NP-Completeness and Reductions.

What are you supposed to know?

- Basics about an algorithms.
- Growth Functions and Asymptotic notation
- Recurrences:
  - Substitution method; recursion tree method; master method.
- Probabilistic Analysis
- Sorting, Median and Order Statistics.
- Elementary data structures:
  - Stacks and linked lists (and its variations); rooted trees.

Grading:

- Home works and exercises: 30%
- Mid-term exam: 30%
- Final: 40%

Example:

- Insertion sort:

```
For j=2 to n
    key=A(j)
    i=j-1
    while i>0 and A(i)>key
            A(i+1)=A(i)
            A(i)=key
            i- -
    end
end
```

- **Execution:**

A: 7 3 5 8 1 2

3 **7**

3 5 **7**

3 5 7 8

1 **3 5 7 8**

1 2 **3 5 7 8**

Analysis

- Running time:
  - Depends on input size & input properties
- Want an upper bound on:
  - Worst case: Max T(n), any input size.
  - Expected: E[T(n)], input taken from a distribution (which?)
    - Example: Sorting arriving TCP/IP packets (they are mostly sorted already).
  - Best Case: Can be used to argue that the algorithm is really bad.
    - Any algorithm can be rewritten to have an excellent "best case" performance.

Example:

- **Insertion sort:**

For j=2 to n → *n times*

key=A(j) → *n-1 times*

i=j-1 → *n-1 times*

while i>0 and A(i)>key → *n-1 times*

A(i+1)=A(i)

A(i)=key → $$\sum_{j=2}^{n}\left(t_j-1\right)$$

i- -

end

end

Analysis

- Assume each operation takes 1 time unit (approximation):

$$T(n) = n + (n-1) + (n-1) + (n-1) + 3\sum_{j=2}^{n}\left(t_j - 1\right)$$

- tj is in the worst case j :

$$\sum_{j=2}^{n}\left(t_j - 1\right) = \sum_{j=2}^{n} j = \frac{n(n+1)}{2} - 1$$

- Would like to formalize this statement!
- Best running time:
  - Outer loop: always executed,
  - Inner loop: not executed if Key >=A(j), meaning that A is already sorted.

Formalization

- How to formalize that n(n+1)/2 was the main term?
- The answer is an asymptotic analysis:
  - Ignore machine-dependent constants.
  - Look at growth of T(*n*) as *n* approaches infinity.
- Intuition: drop low-order terms:

$$5n^4 + 10n^2 - 3n + 2 = \Theta(n^4)$$

$$\text{Ideas: as } n \to \infty,\ \Theta(n^2) \text{ grows slower than } \Theta(n^4).$$

Example:

- In Insertion Sort, the inner loop was $\Theta(j)$

$$T(n) \approx \sum_{j=2}^{n} \Theta(j) \approx \Theta\left(\sum_{j=2}^{n} j\right) \approx \Theta\left(n^2\right)$$

- Is this formal?

$$\Theta(1) + \Theta(1) = \Theta(1)$$

  - Seems to imply:

$$\sum_{j=1}^{n} \Theta(1) \approx \Theta(1) \qquad \Leftarrow \text{incorrect}$$

Asymptotics

- Big-Oh notation:

$$f(n) = O(g(n)) \Leftrightarrow \exists \text{ constant } c, n_0 \text{ s.t. } \forall n \geq n_0 : 0 \leq f(n) \leq cg(n)$$

- Example 2n2=O(n6) but not vice versa!
- "=" is not equality, but membership in a set.
- Set notation is cumbersome:

$$O(g(n)) = \left\{ f(n) \mid \exists \text{ constant } c, n_0 \text{ s.t. } \forall n \geq n_0 : 0 \leq f(n) \leq cg(n) \right\}$$

- What does it mean to say f(n)=O(n)+n2

Asymptotics

- Small-oh notation

$$f(n) = o(g(n)) \Leftrightarrow \forall \text{ constant } c, \exists n_0 \text{ s.t. } \forall n \geq n_0 : 0 \leq f(n) < cg(n)$$

- Example:
  - Prove that n=o(n2):
  - Given c any constant >0, choose n0=2/c. Then we have:

$$\text{for } n \geq n_0,\ n^2 \geq \frac{2}{c} n \Rightarrow cn^2 \geq c\left(\frac{2}{c} n\right) > n.$$

Omega Notation

- Big Omega:

$$f(n) = \Omega(g(n)) \Leftrightarrow \exists \text{ constant } c, n_0 \text{ s.t. } \forall n \geq n_0 : 0 \leq cg(n) \leq f(n)$$

- Small-omega

$$f(n) = \omega(g(n)) \Leftrightarrow \forall \text{ constant } c, \exists n_0 \text{ s.t. } \forall n \geq n_0 : 0 \leq cg(n) < f(n)$$

Transivity

- Most rules apply:
  - Example: transitivity

$$a \le b \;\&\; b \le c \Rightarrow a \le c$$

  - Proof:

$$f = O(g) \;\&\; g = O(h) \Rightarrow f = O(h)$$

$$g(n) = O(h(n)) \Leftrightarrow \exists \text{ constant } c', n'_0 \text{ s.t. } \forall n \ge n'_0 : 0 \le g(n) \le c' h(n)$$

Take $n_0'' = \max(n_0, n_0')$, and $c'' = cc'$. Then

$$\forall n \ge n_0'' : 0 \le f(n) \le cg(n) \le cc' h(n) = c'' h(n)$$

$$\Rightarrow f(n) = O(g(n))$$

Theta Notation

- Theta:

$$f(n) = \Theta(g(n)) \Leftrightarrow \exists \text{ cons. } c_1, c_2, n_0 \text{ s.t. } \forall n \ge n_0 : 0 \le c_1 g(n) \le f(n) \le c_2 g(n)$$

- Often mistaken for Big-Oh notation.
- Example: $\frac{n^2}{2} - 2n = \Theta\left(n^2\right)$
- Proof:

$$\text{Take } \; n_0 = 8, \text{ then for all } n \ge n_0 \text{ we have:}$$

$$n^2/2 - 2n \ge n^2/4 + 8n/4 - 2n = n^2/4$$

$$\text{On the other hand } n^2/2 - 2n < n^2/2.$$

$$\text{Thus: } \; n^2/4 \le n^2/2 - 2n \le n^2/2, \text{ i.e. } c_1 = 1/4, c_2 = 1/2.$$

Simple Theorem

- Claim:

  If $f(n) = O(g(n))$ and $g(n) = O(f(n))$ then $f(n) = \Theta\left(g(n)\right)$

- Proof:

$$\exists n_1, c_1, \forall n \geq n_1 : 0 \leq f(n) \leq c_1 g(n)$$

$$\exists n_2, c_2, \forall n \geq n_2 : 0 \leq g(n) \leq c_1 f(n)$$

$$\Rightarrow \forall n \geq \max(n_1, n_2) : 0 \leq \frac{1}{c_2} g(n) \leq f(n) \leq c_1 g(n)$$

Divide and Conquer paradigm

- Solving a problem for input of size n can be reduced to same problem on inputs of sizes $n_1, n_2, \ldots, n_k$ such that:

$$n = n_1 + n_2 + \ldots + n_k$$

- Example: Towers of Hanoi

  Goals: Transfer all n disks from peg A to peg C

  Rules:
    - Move one disk at a time
    - Never place larger disks above smaller one

  Recursive solution:
    - transfer n-1 disks from A to B
    - move largest disk from A to C
    - Transfer n-1 disks from B to C

  Total Number of moves:
    - $T(n)=2T(n-1)+1$

Recurrence for Tower of Hanoi

- Recurrence Relation:
  - $T(n)=2T(n-1)+1$
  - $T(1)=1$

1. Solution by unfolding:

$$
\begin{aligned}
T(n) &= 2(2T(n-2)+1)+1 \\
&= 4T(n-2)+2+1 \\
&= 4(2T(n-3)+1)+2+1 \\
&= 8T(n-3)+4+2+1= \\
&\ldots \\
&= 2^i T(n-i)+2^{i-1}+2^{i-2}+\ldots+2^1+2^0 \\
&= \ldots
\end{aligned}
$$

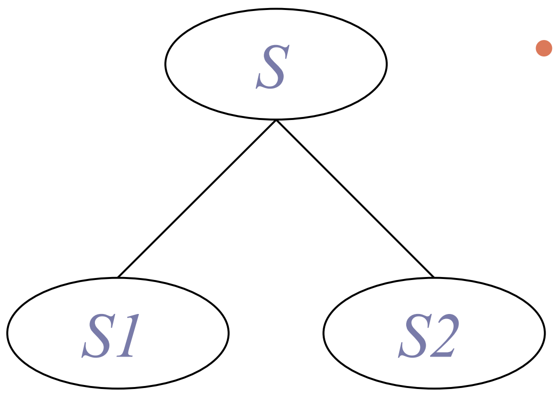
The expansion stops for $i=n-1$

$$T(n) = 2^{n-1}+2^{n-2}+2^{n-3}+\ldots+2^1+2^0$$

Total Number of moves:

- $T(n)=2^n-1=O(2^n)$

Merge-Sort

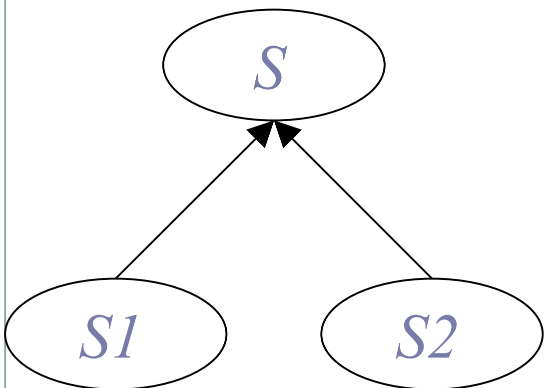


- Problem: Given a list *S* of *n* integers, create a sorted list of elements in *S*.
- Merge-sort Algorithm:

  Divide: If ***S*** has at lest two elements (nothing needs to be done if ***S*** is empty or has only one element), remove all the elements from ***S*** and put them into two sequences, ***S1*** and ***S2***, each containing about half of the elements of ***S***.

  - **Recursion:** Sort sequences *S1* and *S2*.
  - **Conquer:** Put back the elements into *S* by merging the sorted sequences *S1* and *S2* into a unique sorted sequence.

Merging Two Sorted Sequences



- Problem: Given two sequences S_1 and S_2 of sizes n_1 and n_2, create a (union) sorted list S (of size $n=n_1+n_2$).
- Algorithm Merge(S_1,S_2,S):
  - ***top(S_i)*** = first element in $\boldsymbol{S_i}$, for $\boldsymbol{i}$ in **{1,2}.**
  - **While** $\boldsymbol{S_1}$ is not empty and $\boldsymbol{S_2}$ is not empty do

    if ***top(S_1)*** < ***top(S_2)*** **then**

    move ***top(S_1)*** at the end of ***S***

    advance ***top(S_1)***

    else

    move ***top(S_2)*** at the end of ***S***

    advance ***top(S_2)***

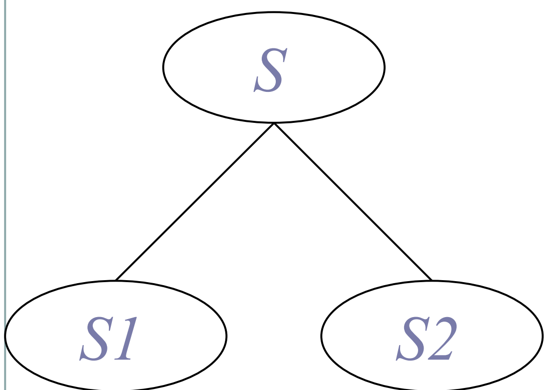
    While $\boldsymbol{S_1}$ is not empty **do**

    move the remaining *of* $\boldsymbol{S_1}$ *to* ***S***

    While $\boldsymbol{S_2}$ is not empty **do**

    move the remaining *of* $\boldsymbol{S_2}$ *to* ***S***

Recurrence for Merge Sort:



- Recurrence Relation:

$$T(n)=2T(n/2)+n$$
$$T(1)=1$$

- Solution by unfolding:

$$\begin{aligned} T(n) &= 2(2T(n/4)+(n/2))+n \\ &= 4T(n/2)+2n \\ &= 4(2T(n/8)+(n/4))+2n \\ &= 8T(n/8)+3n= \\ &\dots \\ &= 2^iT(n/2^{i})+i.n \\ &= \dots \end{aligned}$$

The expansion stops for $i=\log n$

$$T(n) = 2^{\log n}+n \log n$$

Total Number of moves:

$$T(n)= n + n \log n = O(n\log n)$$

Iterative recurrences

- Example: $$\begin{aligned} T(n) &= 4T(n/4) + n \\ &= n + 4(n/2 + 4T(n/4)) \\ &= n + 2n + 16T(n/4) \\ &= n + 2n + 16\left[n/4 + 4T(n/8)\right] \\ &= n + 2n + 4n + 4T(n/8) \\ &= n + 2n + 4n + \ldots \\ &= n\sum_{i=0}^{\log n - 1} 2^i + 4^{\log n} T(1) \\ &= \Theta(n^2) + \Theta(n^2) \end{aligned}$$
- Disadvantage:
  - Tedious
  - Error-Prone
- Use to generate initial guess, and then prove by induction.

Master Theorem

- Let a and b be constants, and let $f(n)$ be a nonnegative function defined on integral powers of b. Let $T(n)$ be defined on the integral powers of b as

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ aT\left(\frac{n}{b}\right) + f(n) & \text{if } n = b^k \end{cases}$$

Then we have:

•If $f(n) = O\left(n^{\log_b a - \varepsilon}\right)$ for some constant $\varepsilon > 0$, then $T(n) = O\left(n^{\log_b a}\right)$

•If $f(n) = \Theta\left(n^{\log_b a}\right)$ for some constant $\varepsilon > 0$, then $T(n) = O\left(n^{\log_b a} \log n\right)$

•If $f(n) = \Omega\left(n^{\log_b a + \varepsilon}\right)$ for some constant $\varepsilon > 0$, and if $n \geq b \Rightarrow af(n/b) \leq cf(n)$

for some positive constant $c \geq 0$,then $T(n) = \Theta\left(f(n)\right)$

Example

- Consider the recurrence $T(n)=T(n/2)+1$ (binary search)

  Then $a=1, b=2$ and $f(n)=1=n^{\log_2 1}$, so by case 2 of Master Theorem

  $T(n)=\Theta\left(n^{\log_2 1}\log n\right)=\Theta(\log n)$.

- Consider the recurrence $T(n)=2T(n/2)+n$ (merge sort)

  Then $a=2, b=2$ and $f(n)=n=n^{\log_2 2}$, so by case 2 of Master Theorem

  $T(n)=\Theta(n\log n)$.

- Consider the recurrence $T(n)=T(n/4)+n^{1/2}$

  Then $a=1, b=4$ and $f(n)=n^{1/2}=\Omega(n^{\log_2 1})$,

  and $af(n/b)=(n/4)^{1/2}=n^{1/2}/2=0.5f(n)$.

  So by case 3 of Master Theorem $T(n)=\Theta\left(n^{1/2}\right)$.

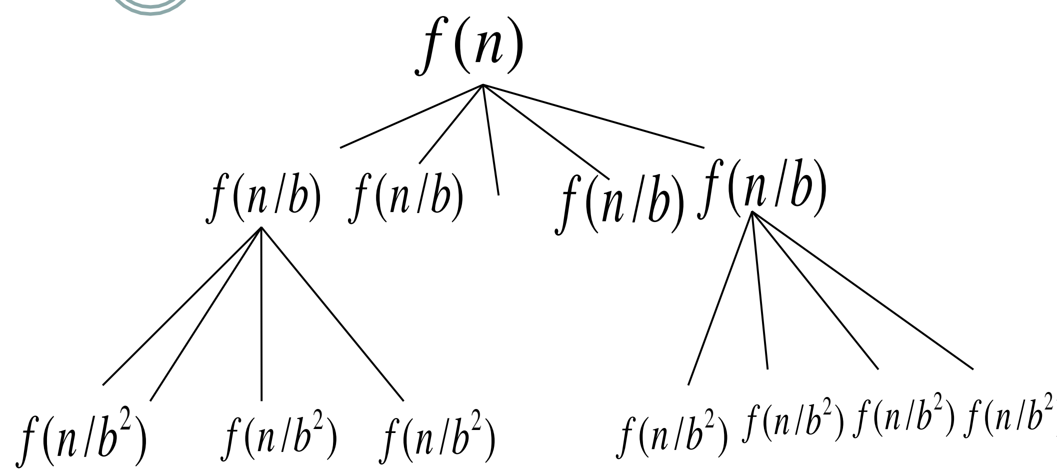
Build recursive tree

- The tree:

$f(n)$

$af(n/b)$

$a^2 f(n/b^2)$



Last row : $\Theta(a^{\log_b n}) = \Theta(n^{\log_b a})$ elemenst, each one $\Theta(1)$.

Total : $\Theta(n^{\log_b a}) + \sum_{i=1}^{\log_b n - 1} a^i f(n/b^i)$

Which term dominates?