

# **CS580: Algorithm Design, Analysis, and Implementation**

Gopal Pandurangan  
gopal@cs.purdue.edu  
Office: CS128

TA: Maleq Khan  
mmkhan@cs.purdue.edu  
Office: CS143

webpage: [www.cs.purdue.edu/homes/gopal/cs580-2004](http://www.cs.purdue.edu/homes/gopal/cs580-2004)  
newsgroup: [purdue.class.cs580](mailto:purdue.class.cs580)  
WebCT: [webct.ics.purdue.edu](http://webct.ics.purdue.edu)

# Theory of Algorithms

Many applications give rise to algorithmic problems.

How do we characterize a **good** solution (program/algorithm) for the problem?

How do we design a good algorithm?

How do we analyze algorithms? How do we show that an algorithm is good/efficient?

Can “efficient” algorithms be always found ?

We need a rigorous mathematical approach.

# Sorting

Input: An array  $X[1 \dots n]$  of  $n$  numbers.

Output: An ordering of elements of  $X$  such that  $X[1] \leq X[2] \leq \dots \leq X[n]$ .

**Algorithm** Simple\_Sort( $X, n$ );

```
for  $I := 1$  to  $n$  do
  for  $J := I + 1$  to  $n$  do
    if  $X[J] < X[I]$  then
       $TEMP := X[I];$ 
       $X[I] := X[J];$ 
       $X[J] := TEMP;$ 
```

**Algorithm** Insertion\_Sort( $X, n$ );

**for**  $J = 2$  **to**  $n$  **do**

$p := X[J];$

$I = J - 1;$

**while**  $I > 0$  **and**  $X[I] > p$  **do**

$A[I + 1] := A[I]$

$I := I - 1$

$A[I + 1] := p$

**Procedure MergeSort( $X, Left, Right$ )**

**if  $Left \neq Right$  then**

$Middle := \lceil \frac{1}{2}(Left + Right) \rceil;$

MergeSort( $X, Left, Middle - 1$ );

MergeSort( $X, Middle, Right$ );

Merge( $X, Left, Middle, Right$ )

**Procedure Merge( $X, Left, Middle, Right$ )**

$I := Left; J := Middle; K := 0;$

**while  $I \leq Middle - 1$  and  $J \leq Right$  do**

$K := K + 1;$

**if  $X[I] \leq X[J]$  then**

$Temp[K] := X[I]; I := I + 1;$

**else**

$Temp[K] := X[J]; J := J + 1;$

**if  $J > Right$  then**

**for  $T := 0$  to  $Middle - 1$  do**

$X[Right - T] := X[Middle - 1 - T]$

**for  $T := 0$  to  $K - 1$  do**

$X[Left + T] := Temp[T]$

**Algorithm** Merge\_Sort( $X, n$ )

MergeSort( $X, 1, n$ );

# Theory of Algorithms

- A solution to a problem can be computed in various ways - some are more efficient than others.
- We are interested in efficient solutions for large problems (asymptotic complexity).
- Mathematical approach:
  - Efficient algorithms are based on mathematical observations.
  - Mathematical characterization of algorithms' efficiency.
- “Machine independent” theory.

# Algorithms Everywhere

- Computational Biology, Bioinformatics
- Communication Networks
- Internet/Web Applications
- Electronic Commerce
- Optimization
- Data Processing
- Computer Graphics
- Quantum Computing
- More ....

# Course Outline

- Introduction: Mathematical concepts for Algorithm Analysis, Probability Theory.
- Algorithm Design Techniques: Divide and Conquer, Randomization, Dynamic Programming, Greedy Algorithms, Amortized Analysis.
- Data Structures: Hashing, Balanced Search Trees, Disjoint set union-find, Heaps.
- Graph Algorithms: Basic Algorithms, Minimum Spanning Trees, Shortest Paths, Network Flow, Matchings.
- Lower Bounds.
- NP-completeness: Complexity classes, Cook's Theorem, Reductions.

- Approximation Algorithms.
- Probabilistic Algorithms.
- More ...

# References

**Text:** Introduction to Algorithms (second edition) by Cormen, Leiserson, Rivest, and Stein.

Lecture Slides (will be posted at WebCT).

Documents/papers posted in the course webpage and WebCT.

# Grading

Assignments (20%):

- Individually written - **non collaborative**.
- Must be in Latex.
- Concise and correct algorithms and proofs.
- Work **must** be submitted on time.

Midterm - 40% and Final - 40%.

**Academic Dishonesty policy:** All submitted work should be on your own. Copying or using other people's work (including from the Web) or using unauthorized material (the text, lecture slides, class notes, and papers/documents posted at the course webpage and WebCT) are the only material allowed) will result in  $-MAX$  points, where  $MAX$  is the maximum possible

number of points for that assignment. Repeat offense will result in getting a failure grade in the course and reporting to the Dean of students.

# Computation Model: RAM

RAM - Random Access Machine:

1. A computing unit plus a memory.
2. Instructions are executed sequentially.
3. Accessing (reading/writing) any one memory cell takes one step.
4. Any “reasonable” one operation takes one step.

# Asymptotic Notation

$g(n) = O(f(n))$  if there is a positive constant  $c$  (independent of  $n$ ), such that for any  $n \geq n_0$ ,  $g(n) \leq cf(n)$ .

$$\limsup_{n \rightarrow \infty} \frac{g(n)}{f(n)} \leq c.$$

$g(n) = \Omega(f(n))$  if there is a positive constant  $c$  (independent of  $n$ ), such that for any  $n \geq n_0$ ,  $g(n) \geq cf(n)$ .

$$\liminf_{n \rightarrow \infty} \frac{g(n)}{f(n)} \geq c.$$

$g(n) = \Theta(f(n))$  if  $g(n) = O(f(n))$  AND  $g(n) = \Omega(f(n))$ .

## Example

$$1) n^3 + 2^{1000}n^2 + 2^{100000}n = O(n^3) = \Omega(n^3) = \Theta(n^3)$$

2) In general for any polynomial

$$p(n) = \sum_{i=0}^d a_i n^i$$

where  $a_i$  are constants and  $a_d > 0$ , we have  $p(n) = \Theta(n^d)$ .

$$3) n = O(n^2)$$

$$4) n^{1000} = O(1.1^n)$$

$$5) e^x = 1 + x + \Theta(x^2), \text{ as } x \rightarrow 0.$$

6)

$$f(n) = \begin{cases} n^2 & : \text{ n is even} \\ n^3 & : \text{ n is odd} \end{cases}$$

$$g(n) = n^3$$

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

$$\limsup_{n \rightarrow \infty} \frac{f(n)}{g(n)} \leq 1.$$

## Example

**Theorem 1.** *Let  $T(n)$  be the best running time of an algorithm for finding the maximum of  $n$  numbers stored in an unsorted array, then  $T(n) = \Theta(n)$ .*

### Proof.

1. Lower Bound: If the algorithm makes less than  $n$  comparisons, then an *adversary* can make sure that an unseen element is the largest.

2. Upper Bound: The following algorithm finds the maximum in no more than  $n$  comparisons:

1.  $Max = A[1]; I = 1;$

2. While  $I < n$  do

(a)  $I := I + 1;$

(b) If  $MAX < A[I]$  then  $MAX := A[I];$

□

## More notation

$g(n) = o(f(n))$  if for any positive constant  $c$  there is a constant  $n_c$  such that for any  $n > n_c$ ,  $g(n) < cf(n)$ .

$$\limsup_{n \rightarrow \infty} \frac{g(n)}{f(n)} = 0.$$

$g(n) = \omega(f(n))$  if for any positive constant  $c$  there is a constant  $n_c$  such that for any  $n > n_c$ ,  $f(n) > cg(n)$ .

$$\liminf_{n \rightarrow \infty} \frac{f(n)}{g(n)} = \infty.$$

## Example

1)  $1/n = o(1)$

2)  $n^{1000} = o(1.1^n)$

3)  $n^2 = \omega(n)$

4)  $n = (?)n^{1+\sin n}$

$$\limsup_{n \rightarrow \infty} \frac{n}{n^{1+\sin n}} = \infty$$

$$\liminf_{n \rightarrow \infty} \frac{n}{n^{1+\sin n}} = 0$$

5) Finding the  $\lfloor \frac{n}{2} \rfloor$ -largest element in an array of  $n$  elements takes  $o(n^2)$  steps.

We can sort in  $O(n \log n)$  time.

# Recurrences

Algorithmic analysis often involves solving recurrences. This is especially typical in recursive algorithms.

$$\text{Solve } T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + 1$$

Guess  $T(n) = O(n)$ . Let's use **induction** to show that  $T(n) \leq cn$ .

$$T(n) \leq c\lfloor n/2 \rfloor + c\lceil n/2 \rceil + 1$$

$= cn + 1$  which does not prove our hypothesis.

Guess  $T(n) \leq cn - b$ . Then

$$T(n) \leq (c\lfloor n/2 \rfloor - b) + (c\lceil n/2 \rceil - b) + 1$$

$$= cn - 2b + 1$$

$$\leq cn - b, \text{ if } b \geq 1.$$

And  $c$  should be chosen large enough to satisfy the boundary conditions.

## Another Example

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

**Guess and verify** that  $T(n) \leq cn \log n$ , for some constant  $c$ .

$$T(n) \leq 2cn/2 \log n/2 + n$$

$$= cn \log n - cn \log 2 + n$$

$$= cn \log n - cn + n$$

$$\leq cn \log n \text{ if } c \geq 1.$$

However, if you try to show  $T(n) \leq cn$  by arguing:

$$T(n) \leq 2cn/2 + n = (c + 1)n = O(n)$$

This is incorrect!

## Change of Variables

Solve  $T(n) = 2T(\sqrt{n}) + 1$

Renaming  $m = \log n$ :

$$T(2^m) = 2T(2^{m/2}) + 1$$

Renaming  $S(m) = T(2^m)$ :

$$S(m) = 2S(m/2) + 1$$

Thus,  $S(m) = O(m)$  and  $T(n) = T(2^m) = S(m) = O(m) = O(\log n)$ .

# A General Theorem for “Divide and Conquer” Recurrences

Consider recurrences of the form

$T(n) = aT(n/b) + f(n)$  where  $a$  and  $b > 1$  are constants and  $f(n)$  is some function.

**Theorem 2. [“Master” Theorem]** *The solution for the above recurrence  $T(n)$  is:*

1) *If  $af(n/b) = cf(n)$  for some constant  $c < 1$  then  $T(n) = \Theta(f(n))$ .*

2) *If  $af(n/b) = cf(n)$  for some constant  $c > 1$  then  $T(n) = \Theta(n^{\log_b a})$ .*

3) *If  $af(n/b) = f(n)$  then  $T(n) = \Theta(f(n) \log_b n)$ .*

**Proof.** Draw a “recursion tree” for  $T(n)$ :  $f(n)$  is the root and it has  $a$  children each of which is a recursion tree for  $T(n/b)$ . That is, a recursion tree is a complete  $a$ -ary tree where each node at depth  $i$  has the value  $a^i f(n/b^i)$ . The leaves of the tree contains the “base cases” of the recursion. Since we are looking at asymptotic bounds, we can assume without loss of generality (w.l.o.g) that  $T(1) = f(1)$ . Assuming each level of the tree is full, we have,

$$T(n) = f(n) + af(n/b) + a^2 f(n/b^2) + \dots + a^L f(n/b^L)$$

where  $L$  is the depth of the recursion tree.

$$L = \log_b n \text{ and since } f(1) = \Theta(1),$$

$$a^L f(n/b^L) = \Theta(a^{\log_b n}) = \Theta(n^{\log_b a})$$

1) If  $f(n)$  is a constant factor larger than  $af(n/b)$  then  $T(n)$  is a geometric series with largest term  $f(n)$ . Hence  $T(n) = \Theta(f(n))$ .

2) If  $f(n)$  is a constant factor smaller than  $af(n/b)$  then  $T(n)$  is a geometric series with largest term  $a^L f(n/b^L) = \Theta(n^{\log_b a})$ .

3) If  $af(n/b) = f(n)$  then there are  $L + 1$  levels each level summing to  $f(n)$  and hence  $\Theta(f(n) \log_b n)$ .

□

# Geometric Series

For real  $x \neq 1$ , the summation

$$\sum_{k=0}^n x^k = 1 + x + x^2 + \dots + x^n$$

is a geometric series and has the value

$$\sum_{k=0}^n x^k = \frac{x^{n+1} - 1}{x - 1}$$

Thus  $\sum_{k=0}^n x^k = \Theta(x^n)$  if  $x > 1$  (increasing series)

$\sum_{k=0}^n x^k = \Theta(1)$  if  $x < 1$  (decreasing series)

If the series is infinite and  $|x| < 1$ , then

$$\sum_{k=0}^{\infty} x^k = \frac{1}{1 - x}$$

## Examples

1. Randomized Selection:  $T(n) = T(3n/4) + 2n$

Here  $af(n/b) = 2(3n/4) = (3/4)f(n)$

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

2. Strassen's Matrix Multiplication:  $T(n) = 7T(n/2) + \Theta(n^2)$

That is,  $T(n) = 7T(n/2) + c_1n^2$ , for some positive constant  $c_1$ .

$af(n/b) = 7c_1(n/2)^2 = (7/4)c_1n^2 = (7/4)f(n)$

Hence,  $T(n) = \Theta(n^{\log_2 7}) = O(n^{2.81})$

3. MergeSort:  $T(n) = 2T(n/2) + n$

Here  $af(n/b) = f(n)$  and hence  $T(n) = \Theta(n \log n)$ .

## Other Examples

Some recurrences do not exactly fit the Master Theorem.

1.  $T(n) = 2T(n/2) + n/\log n$

Can't apply the Master Theorem directly since  $af(n/b) = n/(\log n - 1)$  is not equal to a constant factor times  $n/\log n$ .

We compute the recurrence directly.

The sum of the nodes in the  $i$ th level is  $n/(\log n - i)$ . Thus, the depth of recursion is at most  $\log n - 1$ .

$$T(n) = \sum_{i=0}^{\log n - 1} n/(\log n - i) = \sum_{j=1}^{\log n} n/j = nH_{\log n} = \Theta(n \log \log n),$$

where  $H_k = \sum_{i=1}^k 1/i$  is the *harmonic function* and  $H_k = \Theta(\log k)$ .

$$2. T(n) = T(3n/4) + T(n/4) + n$$

Each complete level in the tree adds up to  $n$  and each leaf has depth between  $\log_4 n$  and  $\log_{4/3} n$ .

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

## Handling Floors and Ceilings

The MergeSort recurrence is really

$$T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + n$$

We want to show that  $T(n) = \Theta(n \log n)$ .

We show the upper bound, lower bound is similar.

$$(1) T(n) \leq 2T(\lceil n/2 \rceil) + n \leq 2T(n/2 + 1) + n$$

We do a “domain transformation” as follows.

Let  $S(n) = T(n + \alpha)$  where  $\alpha$  is a constant chosen such that

$$(2) S(n) \leq 2S(n/2) + \Theta(n).$$

To find  $\alpha$ , we compare (1) and (2).

$$T(n + \alpha) \leq 2T((n + \alpha)/2 + 1) + n$$

$$T(n + \alpha) \leq 2T(n/2 + \alpha) + \Theta(n)$$

which gives  $\alpha = 2$ .

By Master Theorem,  $S(n) = O(n \log n)$ , and hence  $T(n) = S(n - \alpha) = O(n \log n)$ .