Lecture 01

Algorithmics
→ study of Algos
↳ Analysis
↳ Design.
"You teach a computer"

problem
↓
Algo.
I/P → [Computer] → O/P

Ch1
5,6.
4
1:2
1,4

Ex
① GCD(m,n)
→ the greatest no. that completely divide m and n.
② → Euclid gcd(m,n)
= gcd(n, m mod n)
till m mod n = 0
also gcd(m,0) = m.

gcd(60,24) =
(24, 60 mod 24)
gcd(24, 12) =
(12, 24 mod 12)
gcd(12,0) = 12

③ Prime factorization Method
→ Seive of Eratosthenes

Problem 6 Amany L.

Understanding the problem
You know the algo
input to algo is instance
- define the set of instances
- Algo work well for all instances but crash for boundary values
correct all the time ✓

Understand Problem
↓
- Decide: Computational means
- exact vs approx solving
- algorithmic design techniques
↓
Design an algo.
↓
Prove correctness
↓
Analyze the Algo
↓
Code the Algo

Algo design work on Von Neuman machine
→ Architecture.
RAM.
"instructions are executed one after other one operation at time"
Sequential Algo

→ New machines.
- Concurrent operations in parallel.
parallel algo

Exact Algo / exact soln
Approx Algo / App soln.

Speed and Memory of Computer -
↳ Compler prob
↳ huge volume of data.
↳ time critical app
- Mathematical / Scientific Exercise.
vs
algorithm as practical tool.

→ Algo. design techniques
→ Exhaustive
→ Brute force.
→ DP
→ Greedy

Algo. Design + DS. =
→ Program.
↓
Choice of DS.

Representing Algo
- Pseudocode
- Flowcharts

Algo correctness
yield a req. result in finite amt of time

Analysis: efficiency → time
→ space.
↓
Simplicity
generality

optimality.

Induction
* Types of Problem
* Data Structure

quadratic formula cannot be gen for polynomials of arbitrary degree

- Sorting (in-place, stable) 01, 10 Lecture 1
- Search (search-key)
- String processing (txt, alphanumeric, gene seq, bitstring)
  - String matching
- Graph problems: (TSP, Graph Coloring problems)
- Combinatorial problem.
- Geometric (Convex Hull, closest pair)
- Numerical

---

**Linear DS** array - linked list < SLL / DLL
String - char array
- binary / bit

List
Stack
Queue
PQ
Heap

**Graph**
- directed
- undirected
- cyclic
- acyclic
- adj list
- adj matrix
- inc. ~~dict~~
- complete dense
- Sparse
- weighted

**Tree** - forest
- Rooted
Ordered rooted (BT)

7 9 10

**Vectors**, **Set, Dictionaries**, bags

---

**Sorting Problem:** I/P $\xrightarrow{\text{permute}}$ O/P
- Cases $\leq$ B, A, W.

Correct algo halt for every instance.

time not known are hard problem NP-complete

| IS | MS |
|---|---|
| $an^2$ | $c_2 n \log n$ |

find n large enough where MS beat IS

Sort array of $10^7$ no.

| Computer A | Comp B |
|---|---|
| $10^{10}$ ins/s | $10^7$ ins/s |
| In s. $2n^2$ | $50n \log n$ |

Complete Exercise.
Cormen ch 1.

~~$c_1 n^2 \leq c_2 n \log n$~~

~~$c_1 n^2 \leq c_2 \log n$~~
$n$

$c_1 n \leq c_2 \log_2 n$

$n \leq \frac{c_2}{c_1} \log_2 n$

2
1
0

Ch 1 Cormen :

Lecture 02

algorithm : Set of well defined computational procedure that takes some value or set of values as input and produces some value or set of values as output in finite amount of time.

used to solve ↓ computational problems

instance of a problem

Ex : Sorting problem.

Input : A sequence of n number $\{a_1, a_2 \dots a_n\}$

Output : A permutation of input sequence $a'_1 \leq a'_2 \leq \dots \leq a'_n$

- if algorithm is correct it must halt with correct output.
- if algorithm is incorrect
  - ↳ it will not halt
  - ↳ halt for incorrect output

**Efficiency** : time and Space.

Insertion sort $c_1 n^2$ vs Merge sort $c_2 n \log n$

$c_1 n \cdot (n)$ ↓ $c_2 n (\log n)$

factor in time.

$c_1 \ll c_2$

mergesort will beat insertion sort

if $n = 1000$

$n^2 = 1000000$ $\log n = 10$.

$n = 1000000$

$n^2 = (10)^{12}$ $\log n = 20$

Ch# 1
1.2 - 1
- 2
- 3
1-1

In terms of number of instructions

A : 10 billion instructions per second

A if 1000 times faster than B

B : 10 million instructions per second.

Ins. Sort → Best specified $2n^2$

Mergesort → Worst specified $50 n \log n$.

Computer A

Computer B

I $\frac{2 \cdot (10^7)^2 \text{ ins}}{10^{10} \text{ ins/s}} = 20,000 \text{ s}$

M $\frac{50 \cdot 10^7 \log 10^7}{10^7} = 1163 \text{ se.}$

## Decrease and Conquer:-

4.1 AL

1) decrease by cons
2) decrease by cons factor
3) Variable size decrease

relationship b/w an instance soln and soln to smaller instance of that problem.
↳ top down (most recursive)
↳ bottom up (iterative)
↳ start with a soln to smaller problem.

Lecture 12

AL
Q 10
Q 11
Q 12
Q 2
Q 7

on each iteration of algorithm
- mostly const = 1 reduction.

say $a^n = a^{n-1} \cdot a$.

① Top down rec defn

$$f(n) = \begin{cases} f(n-1) \cdot a & n > 0 \\ 1 & n = 0 \end{cases}$$

Bottom up multiply a n times. (n times)

problem size n
↓
Sub problem size n-1
↓
Soln. to subprob
↓
Soln. to orignal problem

② other way

$$a^n = \begin{cases} (a^{n/2})^2 & n \text{ is even and +ve} \\ (a^{(n-1)/2})^2 \cdot a & n \text{ is odd} \\ 1 & n = 0. \end{cases}$$

⇒ (log n) times
Size reduction is n/2.

③ Variable size reduction

$$gcd(m, n) = gcd(n, m \bmod n)$$

Cormen

Cormen 2.1-1 will 5

## Insertion Sort (Straight insertion sort)

Array (In place)

Loop Invariant: $A[0] \leq \dots \leq A[j] \leq A[j+1] \leq A[i-1]$

$A[i] \dots A[n-1]$

Initialization: Its true prior to the first iteration of loop.
Maintainence: If it is true before an iteration of the loop, it remains true before next iteration.
Termination: The loop terminates when it terminates invariant gives the useful property that shows algorithm is correct.

Mathematical Induction.
→ Base Step
→ Inductive Step. (infinite)
→ I H
→ Prove I.H. for ~~k~~ k+1

I: Sub array consists of single element.
M: Sub Array of size n is sorted before entering into the loop
T: loop terminates with sub array sorted n≤(n+1) size.

loop invar. break the loop.

Correct O/P A[1:n] → Algo is correct.

- RAM modelling.
  * one processor
  * no. concurrent op

## Analysing Algorithms:

Resources Prediction for algo to run
↳ Computational time

one processor RAM Machine
↳ inst. executes one after other
↳ no concurrent operations

→ Basic _ Arithmetic Op
- Logical
- Control
- Data Store load

Constant time Ins.

RAM model has SORT ins.
don't write code !! abuse RAM model.

* Exponentiation assumed as
* Multiplication
due to bit shifts.

Run time } with RAM Model
↳ independant of particular computer.

Best case : $(c_1+c_2+c_4+c_5+c_8)n + (c_2+c_4+c_5+c_8)$ an+b linear.

Line 5) $\sum_{i=2}^{n} t_i$

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

Worst Case
Line calculation.

6/7) $\sum_{i=2}^{n} t_i - 1$

$$\sum_{i=2}^{n} i - 1 = \sum_{i=1}^{n-1} i = \frac{(n-1)(n-1+1)}{2} = \frac{n(n-1)}{2} = \frac{n^2-n}{2}.$$

Arithmetic Series

$$\sum_{k=1}^{n} k = 1+2+\cdots+n$$
$$= \frac{n(n+1)}{2}$$
$$= \Theta(n^2)$$

$$\sum_{k=0}^{n} k^2 = \frac{n(n+1)(2n+1)}{6}$$

$$\sum_{k=0}^{n} k^3 = \frac{n^2(n+1)^2}{4}$$

## Order of growth:

$an^2 + bn - c$ → Quadratic.

## Divide and Conquer:

5.1 AL. chap5

- Sub problems of same type (many)
- Sub problems are solved rec. mostly or any other algo is used if sub-prob is small.
- Soln of sub problems are combined to get a soln of original Problem.

Sum of 'n' no. if $n = 1$ return a0 else add $\lfloor n/2 \rfloor$ and $\lceil n/2 \rceil$

```
Merge Sort (A, p, r)
if p ≥ r
    return
q = ⌊(p+r)/2⌋
Merge_Sort (A, p, q)
Merge Sort (A, q+1, r)
Merge (Ap, q, r)
```

Analyzing Rec. (Divide and Conquer) Algorithms

$T(n) = \Theta(1)$ if $n < n_0$
↳ problem size is so small.

Divide into 'a' subproblems of $1/b$ size of original problem

MergeSort $a=2$ $b=2$.

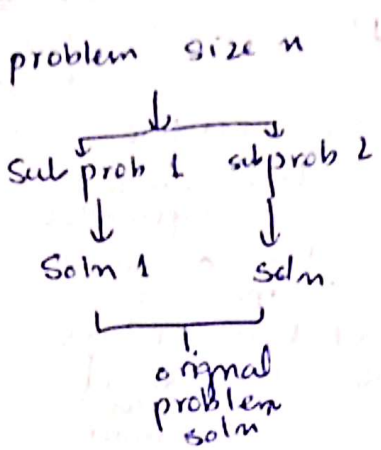
$T(n)$ → time to solve problem of size 'n'

$T(n/b)$ → time to solve one subproblem of size $n/b$.

$a\,T(n/b)$ → time to solve 'a' sub-problems of size $n/b$.

Sometime division results in non-integer value so we don't bother we squint the (1) as $n/2$ & $n/2$

Lecture 3




Cormen 2,3
recursive
↳ base step
↳ recursive step

Divide
Conquer (rec)
Combine

Divide: $A[p:r]$ $A[p:q]$ $A[q+1:r]$

Conquer: sort $A[p:q]$ and $A[q+1:r]$ recursively

Combine: Merge two sorted sub arrays
↓ key operation $A[p:q]$ $A[q+1:r] \Rightarrow A[p:r]$

Merge Procedure $\Theta(n)$ $(n = r-p+1)$

Constant time 1-3, 8-10

for loop $\Theta(n_L + n_R) = \Theta(n)$

while loop (all 3) ⇒ $\Theta(n)$ iteration

$1/2$
$2n$
$c_2 \frac{n}{4}$
$2n$

Division Step: $D(n)$ to divide problem into sub problems.

Combine Step: $C(n)$ to combine the solutions to sub-problems into solution of original problem.

Recurrence Equation

$$T(n) = \begin{cases} \Theta(1) & n < n_0 \\ D(n) + aT(n/b) + C(n) & \text{otherwise} \end{cases}$$

Analysis M-S:

Divide: $D(n) = \Theta(1)$

Conquer (Rec. solving subproblems): $a\,T(n/b) = 2T(n/2)$

Combine = Merge: $C(n) = \Theta(n)$

$$T(n) = D(n) + a\,T(n/b) + C(n)$$
$$= \Theta(1) + 2T(n/2) + \Theta(n)$$
$$\cong 2T(n/2) + \Theta(n)$$

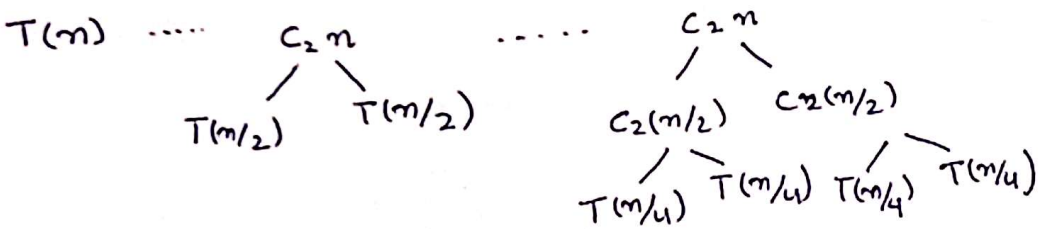
from Master theorem $\Theta(n \log_2 n)$

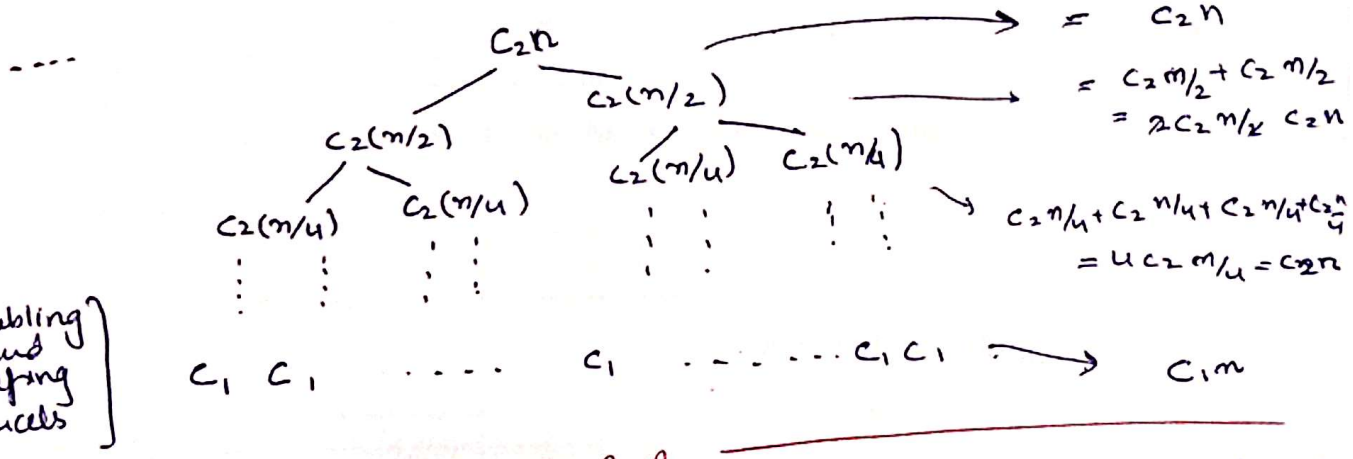
Assume $n$ is exact power of '2'.

So Base Case $\Rightarrow n = 1$.

$$T(n) = \begin{cases} c_1 & n = 1 \\ 2T(n/2) + c_2 n & n > 1 \end{cases} \quad \begin{array}{l} c_2 > 0 \\ c_1 > 0 \\ \text{Provided} \end{array}$$

$c_2 n$ → cost of divide step and combine step




Level $\Rightarrow i^{th}$

Total nodes $\Rightarrow 2^i$

Cost by one node $= c_2(n/2^i)$

over all cost all level $i$ $= 2^i\, c_2(n/2^i)$

$= c_2 n$

Bottom level = $n$ nodes, cost = $c_1$ $= c_1 n$ total cost

Proof of correctness

Levels in rec tree $\lg n + 1$

number of leaves $= n$

Inductive Argument Proof

Test rec for base step:

when $n = 1$.

# of levels $= 1 \cong \lg n + 1$

$= \lg(1) + 1$

$= 0 + 1 = 1$

IH: no. of levels of rec tree with $2^i$ leaves $= \lg 2^i + 1$

$= i + 1$

Assumption: Input size is exact power of 2.

next input size $n = 2^{i+1}$ (a tree with $2^{i+1}$ leaves will have one more level than a tree with $2^i$ leaves)

$$\lg(i+1) + 1 = \lg 2^{i+1} + 1$$

$$= (i+1) + 1$$

$\lg 2^i \cdot \lg 2 + 1$

$i \cdot 1 + 1$

$i + 1$

## Total cost:

cost of all levels ⇒ Rec. Tree has $\lg n + 1$ levels.

level above leaves cost $c_2 n$

leaf level cost $c_1 n$

$$c_2 n(\lg n) + 1$$

$$= c_2 n \lg n + c_2 n = \Theta(n \lg n)$$

**Why Theta?** When ⊛ big O / big Ω bounds is 'X' apply Θ

$$\text{big}O(n) \\ \text{big}\Omega(n) \Rightarrow \Theta(n)$$

$$\Theta(n^2) \\ \Omega(n) \not\Rightarrow \Theta(n) \,||\, \Theta(n^2).$$

MergeSort is stable as compared to Quick Sort and HeapSort

Shortcoming:- linear amount of extra storage (can be done in-place)

## Analysis framework

→ Time (how fast algorithms) → large inputs — Algo run longer
→ Space (memory units required) ↳ function of n (one para: array)
(two para: Graphs)

### Algo running time:

* Count the no. of times basic op. is executed.
* Identify basic op.
↳ inner most loop

ex Sorting key comparison.

$C_{op}$ — execution time of basic operation

$C(n)$ — times operation needs to be executed

$$T(n) \approx C_{op} C(n)$$

if $C(n) = \frac{1}{2} n(n-1)$
how long will it take for an algo to run if input is doubled.

$$T(2n) = C_{op} \frac{1}{2} 2n(2n-1)$$
$$= n(2n-1)$$
$$= 2n^2 - n \approx 2n^2$$

$$T(n) = C_{op} \frac{1}{2} n^2 - \frac{1}{2} n \approx \frac{1}{2} n^2$$

$$\frac{T(2n)}{T(n)} = \frac{2n^2}{\frac{1}{2} n^2} = 4 \text{ times}$$

So: order of growth is important.

- log base is imp. for count

$$\log_2 100 = \log_2 10 \log_{10} 100$$

logarithmic : grow so slow
factorial and exponential. very large.

use for very small input.

Check the increment in functions by increasing n two folds formula.

Challenge
avg* classes for instances *
Check basic op.

* polynomial : coeff 3
-degree
* Matrices : order
* primality of positive no.
- input is single no.
- magnitude of no. measure the size (b bits).

$$b = \lfloor \log_2 n \rfloor + 1$$

### Algo. Seq Search : Basic Op key comp.

Cases ↳ match found first element.
↳ match not found. / last element

* Worst case efficiency $C_{worst}(n) = n$
* Best case efficiency $C_{best}(n) = 1$
↳ not small size
↳ first element equal to search key
* Best case efficiency is not satisfactory
discard algo
* not much info
* Wost case give pessimist analysis

Avg case efficiency
↳ make assumption on "possible inputs" of size n

* Succ. Search $p$ $(0 \le p \le 1)$
→ prob. of first match occurs at i pos is $\frac{p}{n}$ (same for every i)
* UnSucc Search $(1-p)$

$$C_{avg}(n) = [1 \cdot p/n + 2 \cdot p/n + \dots + n \cdot p/n] + n(1-p)$$
$$= \frac{p}{n} [1 + 2 + \dots + n] + n(1-p)$$
$$= \frac{p}{n} \frac{n(n+1)}{2} + n(1-p)$$
$$= p\frac{(n+1)}{2} + n(1-p)$$

if $p = 1$ $C_{avg}(n) = \frac{n+1}{2}$
if $p = 0$ $C_{avg}(n) = n$

## Amortized efficiency.

* not on single run
* Sequence of operation performed on same DS
  Single op. efficiency $\times n \geq$ Entire Seq. of n oper. (better)
  Total time of an

* $100n + 5 \in O(n^2)$

  $100n + 5 \leq 100n + n = 101n \leq 101n^2$

  $n_0 \geq 5$ | $c = 101$

  $100n + 5 \leq 100n + 5n = 105n$ | $c = 105$ | $n_0 = 1$

* $n^3 \in \Omega(n^2)$

  $n^3 \geq n^2 \quad \forall n \geq 0$ | $c = 1$ $n_0 = 0$

* $\frac{1}{2}n(n-1) \in \Theta(n^2)$

  $\frac{1}{2}n(n-1) \leq \frac{1}{2}n^2 - \frac{1}{2}n \leq \frac{1}{2}n^2 \quad n \geq 0$ upper bound.

  $\frac{1}{2}n(n-1) = \frac{1}{2}n^2 - \frac{1}{2}n \geq \frac{1}{2}n^2 - \frac{1}{2}n \cdot \frac{1}{2}n \quad n \geq 2$

  $= \frac{1}{4}n^2$ | $c_2 = \frac{1}{4}$ | $c_1 = \frac{1}{2}$ | $n_0 = 2$

**Max Element:** Two ops in inner loop — assignment, comparison ✓ B/Op

* Comparison will occur for array of size 'n'
* no Best/Avg/Worst

Comparisons $= C(n) = \sum_{i=1}^{n-1} 1$

$= n - 1 = \Theta(n)$

**Unique Element:** inner most loop

* key comparison B/Op

dependance = Equality of key
- size of arrays for comparing.
- which position is occupied.

* Worst: largest comparisons. Largest amongst array of size n.
  - ↳ array with no equal elements
  - ↳ array with last pair equal.

One comparision

$$\sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} 1 = \sum_{i=0}^{n-2} n - 1 - (i+1) + 1$$

$$= \sum_{i=0}^{n-2} (n - i - 1)$$

$$= \sum_{i=0}^{n-2} (n-1) - \sum_{i=0}^{n-2} i$$

$$= (n-1) \sum_{i=0}^{n-2} 1 - \frac{(n-2)(n-1)}{2}$$

$$= (n-1)(n-1) - \frac{(n-2)(n-1)}{2}$$

$$= n(n-1)/2 \cong \frac{1}{2}n^2 \in \Theta(n^2)$$

Recursive Algorithm. (Mathematical Analysis)

$F(n) = n!$ (non neg integer $-n$)

$n! = 1 \cdot (n-1) \cdot n = (n-1)!\, n \quad n \geq 1$

$0! = 1$

$1! = 1$

Input size : 'n'

Basic op : Multiplication

$M(n) = M(n-1) + 1$

$F(n-1)$ — To multiply $F(n-1)$ with $n$.

* find M of n in terms of 'n' only

* M of n is not a function of 'n' exactly but a function dep at any point (n-1)

initial condition : value with which sequence starts

$n = 0 \rightarrow 1$ } for smallest value of 'n' which is 0

$M(n) = 0$

$\Rightarrow$ no multiplications performed

$M(n) = M(n-1) + 1$ for $n > 0$ Rec Rel.

$M(0) = 0$ InCond

$F(n) = F(n-1) \cdot n \quad n > 0$

$F(0) = 1$

Backward Substitution: Soln? Guess starts with 0, increase by 1 on each step.

$M(n) = M(n-1) + 1$

$= [M(n-2) + 1] + 1$

$= M(n-2) + 2$

$= [M(n-3) + 1] + 2$

$= M(n-3) + 3$

$\Rightarrow M(n) = M(n-i) + i$ $\boxed{i = n}$

$M(n) = M(n-n) + n$

$= M(0) + n$

$M(n) = n$

$M(n-1) = M(n-2) + 1$

$M(n-2) = M(n-3) + 1$

iterative algo that accumulates the product of n consecutive integers require the same no. of multiplication with out overhead of time and space for maintaining recursive stack of problem size of F(n)

## Tower of Hanoi: 'n' disks different sizes, '3' pegs

move $n \geq 1$ disks from Peg 1 to peg 3 (peg 2 Aux)
move $(n-1)$ disks recursively from peg 1 to peg 2 (peg 3 Aux)
move $n^{th}$ disk (largest) from peg 1 to peg 3.
move $(n-1)$ disks recursively from peg 2 to peg 3 (peg 1 Aux)

if $n = 1$ : peg S to peg D (move the disk)

input size 'n'
Basic op: moving one disk.
$M(n)$ — (no of moves) depend on 'n'

$M(n) = M(n-1) + 1 + M(n-1)$ for $n > 1$

Initial condition:
$M(1) = 1 \quad n = 1$

Rec. Relation: $M(n) = 2M(n-1)+1 \quad n \geq 1$
$M(1) = 1$

$M(n) = 2M(n-1) + 1$

$*M(n-1) = 2M(n-2) + 1$

$= 2(2M(n-2)+1)+1$

$= 4M(n-2) + 2 + 1$

$* M(n-2) = 2M(n-3) + 1$

$= 4(2M(n-3)+1) + 2 + 1$

$= 8M(n-3) + 4 + 2 + 1$

or

$2 \cdot 2 \cdot 2\, M(n-3) + 4 + 2 + 1$

$2^3 M(n-3) + 2^2 + 2^1 + 2^0$

$2^i M(n-i) + 2^{i-1} + 2^{i-2} + 2^{i-3}$

$2^i M(n-i)\, 2^i - 1 = 2^i M(n-i) + 2^i - 1$

Initial condition of $n = 1$

$n - i = 1$

$i = n - 1$

$n - (n+1) = 1$
$n - n - 1 = 1$

$2^{n-1} M(n-(n-1)) + 2^{n-1} - 1$
$= 2^{n-1} M(1) + 2^{n-1} - 1$
$= 2^{n-1} + 2^{n-1} - 1$
$= 2 \cdot 2^{n-1} - 1$
$= 2^n - 1$

Exponential.
problems intrinsic difficulty makes it computationally hard.

Total no. of calls in ToH

$$C(n) = \sum_{l=0}^{n-1} 2^l = 2^n - 1$$

Rec. Tree internal nodes
rec. calls

* No of binary digits in binary rep of +ve decimal no

Basic Op:

* Comparison in loop Condition

Choice not imp cz loop execution +1 is the rep.

'n' is halved to its initial value (div by 2) no of times loop executed is $\log_2 n$

exact formula: for comparisons to be executed $\lfloor \log_2 n \rfloor + 1$ for $n > 1$

i-e the no of bits in binary rep of 'n'

Iterative $\Rightarrow (\log_2 n)$

no. of addition $A(n)$ in algo is $A(\lfloor n/2 \rfloor)$, plus '1' made by algo to increase value by 1

$$A(n) = A(\lfloor n/2 \rfloor) + 1 \quad n > 1 \quad \text{GR}$$

Initial condition: when $n = 1$

$$A(1) = 0 \quad \text{IC}$$

Smoothness rule for n not power of 2 for btwd subs. $n = 2^k$

After smoothness formula can be fine tuned for arbitrary n.

Now $n = 2^k$

$$A(2^k) = A(2^{k-1}) + 1 \quad k > 0$$

$$A(2^0) = 0 \quad k = 0$$

$$A(2^k) = A(2^{k-1}) + 1$$

$$= [A(2^{k-2}) + 1] + 1$$

$$= [A(2^{k-3}) + 1] + 2$$

$$= [A(2^{k-4}) + 1] + 3$$

$$= A(2^{k-4}) + 4$$

$$= A(2^{k-i}) + i$$

from IC $A(2^0) = 0$

$k - i = 0$

$k = i$

$i = k$

$$= A(2^{k-k}) + k$$

$$= A(2^0) + k$$

$$= A(1) + k$$

$$A(n^k) = 0 + k = k.$$

$$A(2^{\log_2 n}) = \log_2 n$$

$$A(n) = \log_2 n.$$

Since

$n = 2^k$

~~$\log_2 n = \log_2 2^k$~~

~~$\log_2 n = k$~~

$\Rightarrow \log_2 n = k$

$b^x = a$

$\log_b a = x$

$T(n) = 2T(\lfloor n/2 \rfloor) + \Theta(n)$ Substitution (Induction)

Guess: $T(n) = O(n \lg n)$

I.H. $T(n) \le cn \lg n$ $n \ge n_0$, $n_0 > 0$, $c > 0$

IND hold for all no. greater no and less n.

So $n \ge 2n_0$

$$T(n) \le cn \lg n \quad \text{---} (1)$$

$$T(n/2) \le c\lfloor n/2 \rfloor \lg \lfloor n/2 \rfloor$$

$$\Rightarrow T(n) = 2c\lfloor n/2 \rfloor \log \lfloor n/2 \rfloor + \Theta(n)$$

$$\le 2cn \lg_2 n/2 + \Theta(n)$$

$$= cn \lg(n/2) + \Theta(n)$$

$$= cn \lg n - cn \lg 2 + \Theta(n)$$

$$= cn \lg n - cn + \Theta(n)$$

This is less than $cn \lg n$ due to $-cn$

$\le cn \lg n$. Hence (1) is proved

$\Theta(n)$ is $cn$ not same as $cn$

$cn > \Theta(n)$

$n_0$ and $c$ are large then '$cn$' dominates.

$T(n) \le cn \lg n$ $\quad n_0 \le n \le 2n_0$, $n_0 > 1$, $\lg n > 0$

put $n_0 = 2$

$T(2) \le c < 2 \lg 2\, c$

$T(3) \le c < 3 \lg 3\, c$

Wrong Guess:

If Guess is $O(n)$ for Rec.

$$T(n) = 2T(\lfloor n/2 \rfloor) + \Theta(n) \quad \text{— (R)}$$

Guess: $T(n) = O(n)$

$T(n) \leq cn$ — (1) $n \geq n_0$, $n_0 > 0$, $c > 0$

IND hold for all number greater than $n_0$.

(i) if $T(n) \leq cn$

then $T(n/2) \leq c\,n/2$

(i+1) Proof

$\Rightarrow T(n) \leq 2\,c\,n/2 + \Theta(n)$

$= \boxed{cn + \Theta(n)}$

This is not less than $cn$ (our guess) which we have to prove

* We have to prove our guess

$T(n) = O(n) \Rightarrow T(n) \leq cn$ — (G)

So the Rec equation (R) given must conclude to $T(n) \leq cn$.

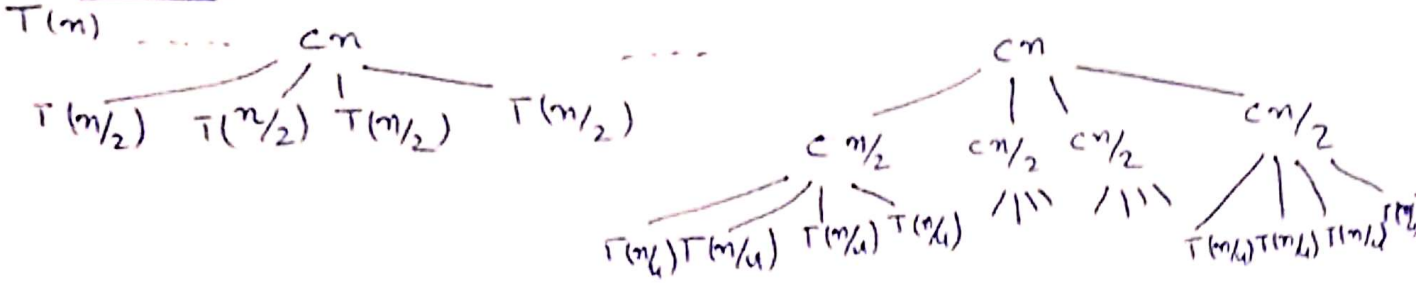
We saw in proof $T(n) \leq cn + \Theta(n)$ which is not same as the equation to be proved. (G)

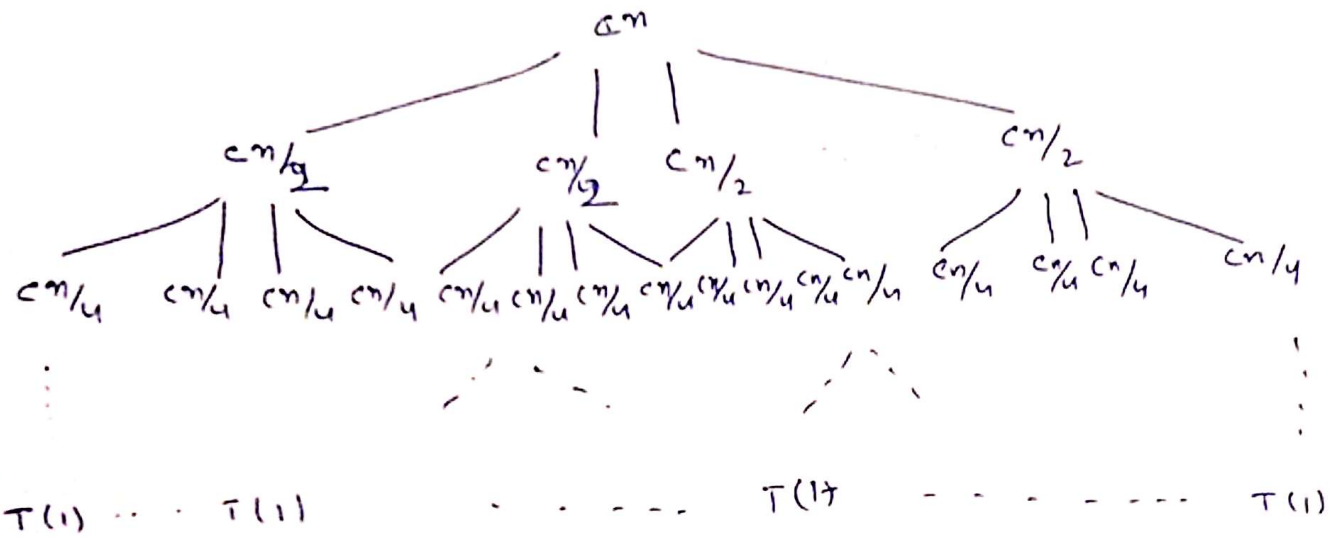
Imp In proof we have to prove exactly, we can't conclude it as big O.

$T(n) = 4T(n/2) + \Theta n$ $\quad T(n) = 4T(n/2) + cn$



Rec. Tree:




| Level | Total cost at each level | Size of subproblem | # of nodes |
|---|---|---|---|
| 0 | $cn = 2^0 cn$ | $n = n/2^0$ | $4^0$ |
| 1 | $4c(n/2) = 2cn = 2^1 cn$ | $n/2 = n/2^1$ | $4^1$ |
| 2 | $16c(n/4) = 4cn = 2^2 cn$ | $n/4 = n/2^2$ | $4^2$ |
| 3 | $64c(n/8) = 8cn = 2^3 cn$ | $n/8 = n/2^3$ | $4^3$ |
| $i^{th}$ | $2^i cn$ | $n/2^i$ | $4^i$ |

Last level: $n = 1 \Rightarrow n/2^i = 1 \quad n = 2^i \quad i = \log_2 n$

Size of sub-problem is $T(1) = \Theta(1)$

- no. of nodes $= 4^i = 4^{\log_2 n}$
- Cost = no. of leaves $\times \Theta(1)$ $= 4^{\log_2 n} = n^{\log_2 4} = n^{\log_2 2^2} = n^2$

# of levels = ~~depth~~ $\log_2 n + 1$

Cost of all levels: $c=1$

$$T(n) = \{ n + 2n + 4n + 8n + \cdots \} + \Theta(n^2)$$

$$= \sum_{i=0}^{\log_2 n} 2^i n + \Theta(n^2)$$

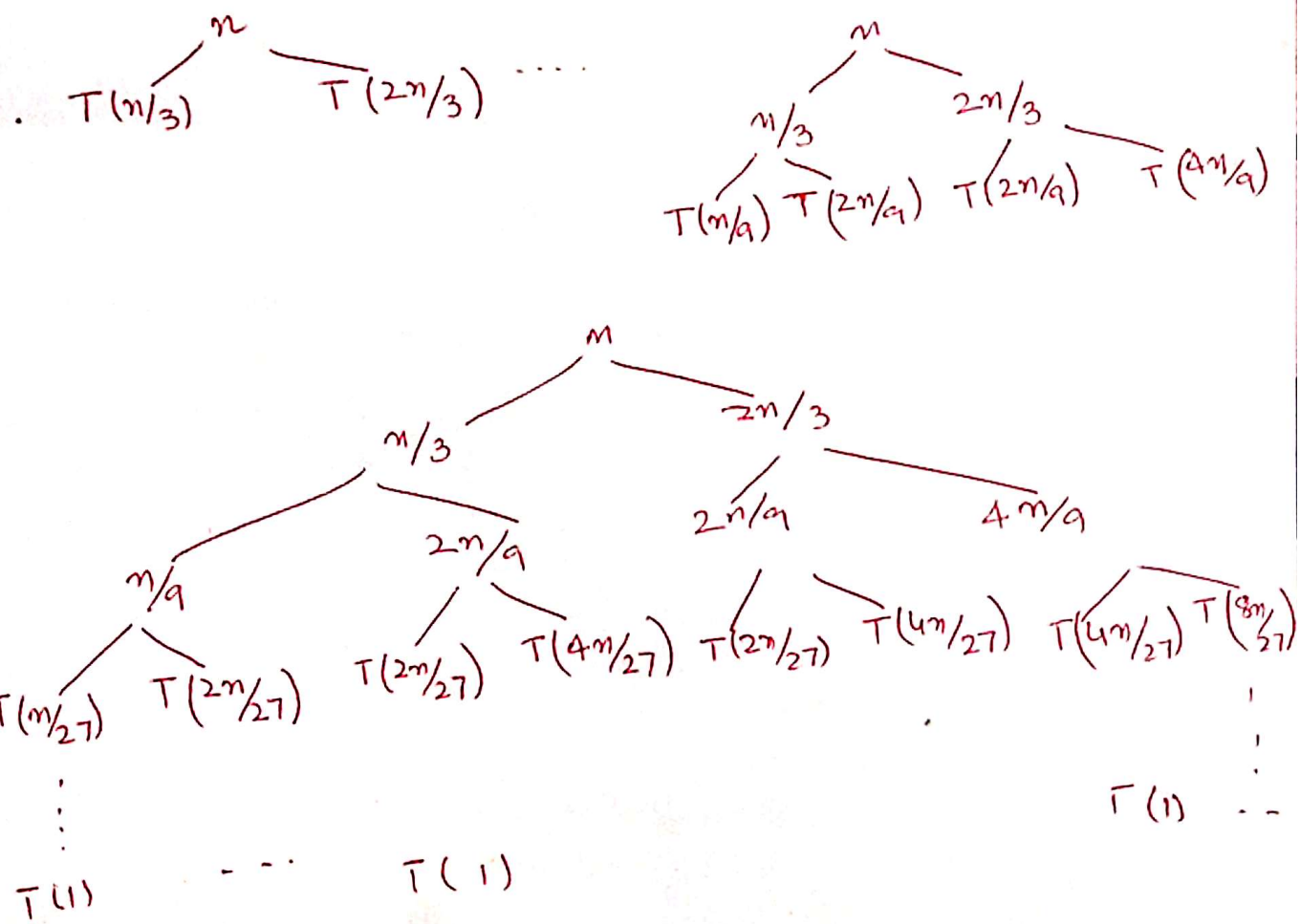
$$= n \sum_{i=0}^{\log_2 n} 2^i + \Theta(n^2)$$

$$\leq n \sum_{i=0}^{\log_2 n} 2^i + \Theta(n^2) = \frac{2^{\log_2 n} - 1}{2-1} + \Theta(n^2)$$

$$= n - 1 + \Theta(n^2) = \Theta(n^2)$$

---

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



| Level | Total cost at each level | Size of subproblems | # of nodes |
|---|---|---|---|
| 0 | $n$ | $n$ | 1 |
| 1 | $n(\frac{1}{3}n+\frac{2}{3})$ | $\frac{2}{3}n = (\frac{2}{3})n$ | 2 |
| 2 | $n$ | $\frac{4}{9}n = (\frac{2}{3})^2 n$ | 4 |
| 3 | $n$ | $\frac{8}{27}n = (\frac{2}{3})^3 n$ | 8 |
| most shallow | $n$ | | |
| $i$ | $\leq n$ | $(\frac{2}{3})^i n$ | $2^i$ |
| deepest | $\leq n$ | | |

most shallow: in left most branch # length $\log_3 n$

deepest: in right most branch # length $\log_{3/2} n$

* problem size is 1 at most shallow level $\Rightarrow (\frac{n}{3^i}) = 1$

$n = 3^i \quad i = \log_3 n$

problem size is 1 at most deepest level

$\Rightarrow (\frac{2}{3})^d n = 1$

$n = (\frac{3}{2})^d \quad d = \log_{3/2} n$

at last level (deepest) no. of nodes $= 2^i = 2^{\log_{3/2} n}$

Cost at last level = no. of leaves $\times \Theta(1)$

$= 2^i \times \Theta(1) = 2^{\log_{3/2} n} \times 1 = n^{\log_{3/2} 2}$

Cost at all levels $= n^{\overbrace{\log_{3/2} n}^{\text{deepest level}}} + n^{\log_{3/2} 2}$

$= n \log_{3/2} n + n^{1.7}$

Cost at all levels below the shallow level has problem size $\leq n$ — (deepest level cost is minimum)

$T(n) \leq n \log_{3/2} n$

## Master Theorem:

$T(n) = a\,T(n/b) + f(n)$

$a > 0 \quad b > 1 \quad f(n) \Rightarrow +ve$

$f(n)$ 'driving function' (cost of dividing and rec.)

$a \rightarrow$ subproblems

Size $n/b$ < 'n'

a subproblems are solved recursively in $T(n/b)$ time.

$a\,T(n/b) \Rightarrow a'\,T(\lceil n/b \rceil)$ or $\rightarrow a''\,T(\lfloor n/b \rfloor)$

$a' \geq 0 \quad a'' \geq 0$

$a = a' + a''$

1) $\varepsilon > 0 \quad f(n) = O(n^{\log_b a - \varepsilon})$

$T(n) = \Theta(n^{\log_b a})$

2) $k \geq 0 \quad f(n) = \Theta(n^{\log_b a} \lg^k n)$

$T(n) = \Theta(n^{\log_b a} \lg^{k+1} n)$

3) $\varepsilon > 0 \quad f(n) = \Omega(n^{\log_b a + \varepsilon})$

+ Regularity condition hold

$a\,f(n/b) \leq c\,f(n) \quad c \leq 1$ for large $n$

$T(n) = \Theta\, f(n)$

---

$n^{\log_b a}$ is watershed function $\overset{\text{vs}}{=}$ $f(n)$ is compared

watershed func > driving func — Case 1 (polynomially faster) or $\Theta(n^{\varepsilon})$

Total cost of leaves dominates the total cost of internal nodes

Each level of rec. cost $\overset{\text{WSF}}{=} \Theta(n^{\log_b a} \lg^k n)$ $\Theta(\lg n)$ levels $k=0$ commonly — Case 2 DF faster than WSF by $\Theta(\lg^k n)$, $k \geq 0$

Cost per level drop geometrically from root to leaves $\overset{\text{WSF}}{<}$ DF — Case 3 DF must grow polynomially faster than WSF by factor $\Theta(n^{\varepsilon})$

$f(n) = n^{1.99}$ $\quad n^{\log_b a} = n^2$ $\quad n^{1.99} < n^2$ by $n^{0.01}$ | $\varepsilon = 0.01$ Case 1

$T(n) = 9\,T(n/3) + n$

$n^{\log_b a} = n^{\log_3 9} = \Theta(n^2)$

$n^2 > n$

$n = O(n^{2-\varepsilon}) \quad \varepsilon \leq 1$

Case 1

$\Theta(n^2)$

$T(n) = 2n/3 + 1$

$n^{\log_b a} = n^{\log_{3/2} 1} = n^0 = 1$

$f(n) = 1$

Case II

$\Theta(n^{\log_b a} \lg^0 n) = \Theta(\lg n)$

$T(n) = 3T(n/4) + n \lg n$

$n^{\log_b a} = n^{\log_4 3} = n^{0.79}$

$f(n) = n \lg n$

$= \Omega(n^{\log_4 3 + \varepsilon})$

$= \varepsilon = 0.2$

$a\,f(n/b) \leq c\,f(n)$

$3(n/4) \lg(n/4) \leq$

Case III $\quad \tfrac{3}{4} n \lg n$

$T(n) = \Theta(n \lg n)$