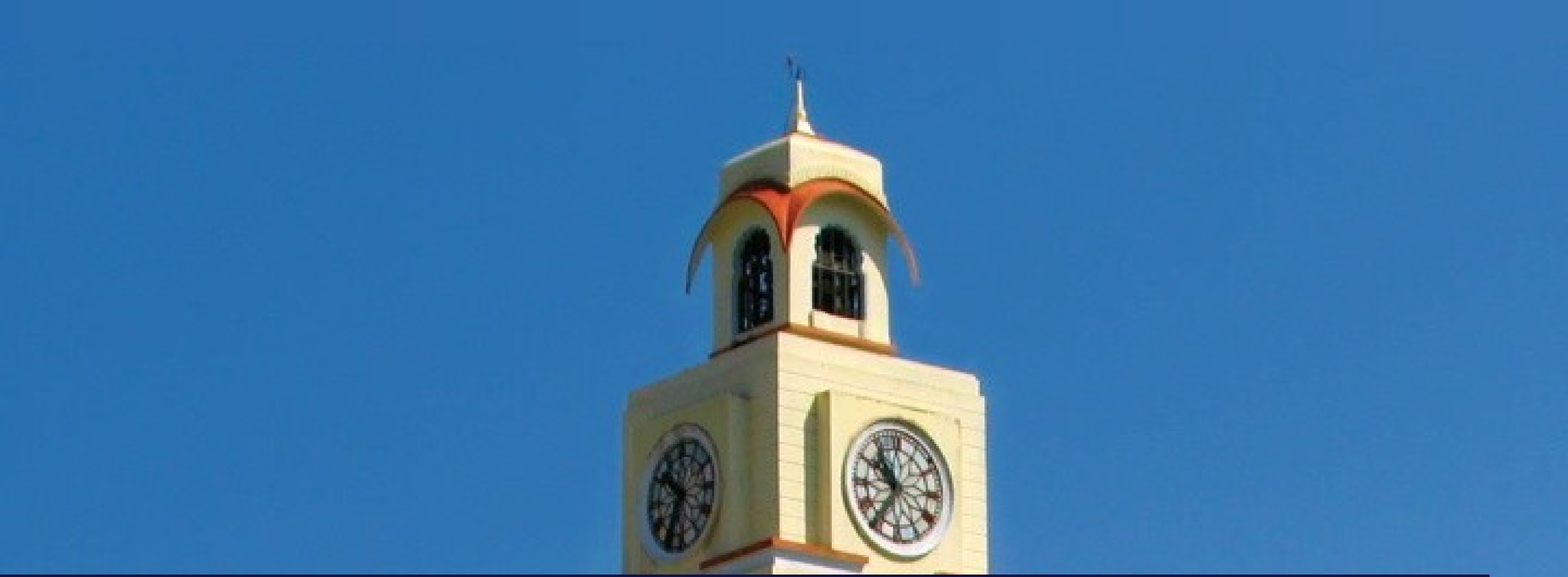

BITS Pilani
Pilani Campus

Course Name : Data Structures & Algorithms Design

A Baskar
Computer Science & Information Systems

Outline

- Recursion and Recurrence relations
- Abstract Data Types
- Stacks
- Queues

Recap of Lecture 5

- Asymptotic Analysis
- Small-oh o, small-omega ω
- Correctness of Algorithms
- Recursion

A query from a student

- If there are two for loops running n*n times in a program. I would say it is $O(n^2)$ but how one can say it is $o(n^3)$?
- It is not quite convincing because it is clear that even in worst case the program will run for n^2 units.
- Why do we have to say n^2 is $o(n^3)$ when Big Oh is giving precise results. I understand that you proved n^2 is $o(n^3)$ but apart from this, it is not clear to me. Please explain.

We don't use o much.

Algorithm A

Algorithm B

$O(n^3)$

$O(n^4)$

$O(n^3)$

big O

Small o.

$O(n^2)$

More information

$O(n^3)$

$o(n^2)$

n log log n

$O(n^2)$

$o(n^3) \Longrightarrow O(n^3)$

Recursion

- Define a procedure P that is allowed to make calls to itself
- Provided those calls to P on are for solving subproblems of smaller size
- The calls to P on a smaller instances are called recursive calls
- It should define a base case, which can be solved without using recursion

Examples

- Factorial

Algorithm fact(n)

Input: a positive integer n

base case

output: n!

if n=1 then return 1

return fact(n-1)*n

recursive call

- Product of two integers by only using addition

Algorithm prod(a,b)

Input: Two positive integers a and b

Output: a*b

if b= 1 then return a

return prod(a,b-1)+a

Recursive ArrayMax

- Algorithm recursiveArrayMax(A,n)

Input: an integer n, array A of n integer

Output: Maximum element in the array

if n=1 then return A[0]

return max(recursiveArrayMax(A,n-1),A[n-1])

Illustration

$T(n) = T(n-1) + 7$ if $n \neq 1$.

~~Step~~ line 1 — 1 unit of time

line 2 — n-1, 1 Subtraction, 1 function call | n-1, 1 subtraction, indexing

6+ T(n-1)

max

return n.

Running Time

$$T(n) = \begin{cases} 3 & \text{if } n=1 \\ T(n-1)+7 & \text{o.w} \end{cases}$$

Recurrence relation!

not a closed form

$T(n)$ is defined using T again

We need a soln $T(n)$ without using T.

Iterative-Substitution Method

$$T(n) = T(n-1) + 7$$

$$= T(n-2) + 7 + 7$$

$$= T(n-3) + 7 + 7 + 7$$

$$\vdots$$

$$= T(1) + \underbrace{7 + 7 + \cdots + 7}_{(n-1)}$$

$$= 3 + 7(n-1)$$

$$= 7n - 4$$

innovate achieve lead

Example 2

$$T(n) = \begin{cases} 1 & \text{if } n=1 \\ T(n-1)+n & \text{o.w} \end{cases}$$

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

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

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

$= T(n-3) + n + n + n \ldots$

$\vdots$

$= T(1) + \underbrace{n + n + \cdots + n}_{(n-1)}$

$= 1 + n(n-1)$

$$T(n) = \begin{cases} 1 & \text{if } n=1 \\ 2T(n-1) & \text{o.w} \end{cases}$$

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

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

$\vdots$

$= T(1) + 2 + 3 + \cdots + n$

$= 1 + 2 + 3 + \cdots + n$

$= \dfrac{n(n+1)}{2}$

Why (n-1) should be added for T(n-2)

$T(n) = T(n-1) + n$ if $n \neq 1$

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

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

$$T(n) = \begin{cases} 1 & \text{if } \cancel{n \leq 2} \;\; n = 1 \\ 2T(n-1) & \text{o.w} \end{cases}$$

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

$$= 2 \cdot 2\, T(n-2)$$

$$= 2^k\, T(n-k)$$

$$= 2^{n-1}\, T(1)$$

$$= 2^{n-1}(1) = 2^{n-1}$$

Guess-and-Test Method

$$T(n) = \begin{cases} b & \text{if } \cancel{n < 2} \quad n = 1 \\ 2T(n/2) + bn & \text{o w} \end{cases}$$

Guess 1 : $T(n) \le cn \log n$ for some $c > 0$

Test — Induction method

- Guess is correct for base case?
- Assume the guess for all $k \le n$ and prove it for $k = n$.

$1, \quad n \to n+1$

Base Case

$T(1) \leq c \log \frac{1}{0}$ ✗

$T(2) \leq c\, 2 \log 2 = c2$

$2T(1) + b\cdot 2$

$2b + 2b$

$4b$

Take $c = 3b$ ✓

$c = 2b$

$T(2) \leq 2c$

$6b$

$4b$

1

2

3

$\vdots$

$\forall n$

$T(2) = 2T(1) + 2b$

$= 2b + 2b$

$= 4b$

$T(2) \leq 2c \log 2$

$T(2) \leq 2c$

$\boxed{4b \leq 2c}$ Want to prove

~~Ass~~

$4b \le 2c.$

Take cony c st. $4b \le 2c.$

$c = 2b$ | $c = 3b$ |

Assume $\forall k < n \quad T(k) \le ck\log k$.

$T(n)$

$= 2\,\underline{\underline{T(n/2)}} + bn$

Prove $T(n) \le cn\log n$.

$\frac{n}{2} < n$, by assumption

$T(n/2) \le c\, n/2 \log n/2$

$T(n) \le \cancel{2}\left(c\frac{n}{\cancel{2}}\log(n/2)\right) + bn$

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

$\le cn\log n \quad (\because c = 2b)$

defn of T

$$T(n) \le 2T(n/2) + bn$$

from assumption

$$\le \cancel{2}\left(\frac{cn}{\cancel{2}}\log(n/2)\right) + bn$$

$$\le cn\left(\log(n/2)\right) + bn$$

$$\le cn(\log n - \log 2) + bn$$

$$\le cn\log n - cn\log 2 + bn$$

$$\le cn\log n - cn(1) + bn$$

$$\le cn\log n - bn$$

$$T(n) \leq Cn\log n$$

$$T(n) \text{ is } O(n\log n)$$

innovate achieve lead

$T(n) \leq cn\log n - cn + bn$ $\quad T(n)$ is $O(n\log n)$

we have ~~assumed~~ taken

$c = 2b$

$\therefore T(n) \leq (2b)n\log n - 2bn + bn = 2bn\log n - bn$

$T(n) \leq (2b)n\log n - bn \leq 2bn\log n$

$T(n) \leq \underbrace{(2b)}_{c} n\log n$

$T(n) \leq cn\log n$