A Baskar

Course Name :

Data Structures & Algorithms Design

BITS Pilani
Pilani Campus

Computer Science & Information Systems

Recap of Lecture 4

- Asymptotic Analysis
- Big-Oh Notation & Examples
- Big-theta Θ, Big-omega Ω
- Small-oh o, small-omega ω

Big-theta Θ, Big-omega Ω

* $f(n)$ is $\Omega(g(n))$
  iff $g(n)$ is $O(f(n))$

* $f(n)$ is $\Theta(g(n))$
  iff $f(n)$ is $O(g(n))$
  and $g(n)$ is $O(f(n))$

It is possible that $f(n)$ is $O(g(n))$
$g(n)$ is $O(f(n))$

Small-oh o, small-omega ω

* $f(n)$ is $o(g(n))$ iff

~~$\forall c > 0 \quad \exists n_0 > 0 \quad \forall n \geq n_0$~~

~~for~~

"$f(n)$ is strictly better than $g(n)$"

for all $c > 0$ ~~$\exists n_0$~~ there exists $n_0 > 0$

s.t. for all $n \geq n_0$ $f(n) \leq c\,g(n)$

$12n^2 + 6n$ is $O(n^3)$

Suppose $c > 0$. we have to find n_0 s.t.

$12n^2 + 6n < cn^3$ for all $n \geq n_0$

$12n^2 + 6n$

$\leq 12n^2 + 6n^2$

$= 18n^2 < cn^3$ for all $n > n_0 = \left(\frac{18}{c}\right)$

$12n^2 + 6n < cn^3 \; \forall n > \left(\frac{18}{c}\right)$

$f(n)$ is $\omega(g(n))$

iff $g(n)$ is $O(f(n))$

* n^3 is $\omega(12n^2+6n)$

* $12n^2+6n$ is $\omega(n)$

$f(n)$ is $o(g(n))$ if $\lim_{n \to \infty} \frac{f(n)}{g(n)} = 0$

$\lim_{n \to \infty} \left(\frac{12n^2 + 6n}{n^3} \right) \Rightarrow \lim_{n \to \infty} 1/n = 0$

n^2 is $o(n^3)$

$\frac{n^2}{n^3} = 1/n$

$\lim_{n \to \infty} \frac{1}{n} = 0$

As n becomes larger $1/n$ will go closer to 0

n	1000	10^5
$1/n$	$\frac{1}{1000}$	.00001

Big-oh and small-oh: Comparison

$f(n)$ is $O(g(n))$

$\exists$ - there exists

$\forall$ - for every

$\exists c > 0$ & $n_0 > 0$

s.t. $\forall n \geq n_0$ $f(n) \leq c \cdot g(n)$

Just one pair (c, n_0)

$f(n)$ is $o(g(n))$

$\forall c > 0$ $\exists n_0 > 0$

s.t. $\forall n \geq n_0$ $f(n) \leq c \cdot g(n)$

Here for every c we have to find n_0

Correctness of Algorithms

* prove our algorithm is Correct.
* proving Correctness ~~is difficult~~.
* Using pseudocode will provide Some help

Algorithm *arrayMax(A, n)*

```
put     :    n, an array A of n integers
utput :    The  maximum element of A
 currentMax ←A[0]
 for i ← 1 to n - 1 do
     if A[i] > currentMax then currentMax ← A[
return currentMax
```

Loop invariant

$S_0, S_1, S_2, \ldots S_n \rightsquigarrow$ Statements

S_0 is correct before the loop starts

S_i is correct after i^{th} iteration

S_n — will be the final statement

After the algorithm terminates

Current Max is the maximum of all elements in the array

loop invariant S_i – Current Max is equal to maximum first "i+1" elements

✓ S_0 – Current max is equal to max of first 1 element

i^{th} iteration

Assume S_{i-1} is Correct

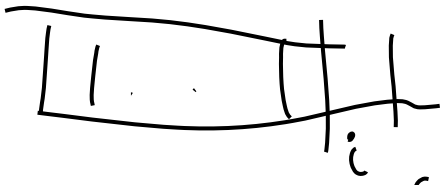
CurrentMax is max of first i elements

Check S_{i+1} is Correct

CurrentMax $\geq A[j]$ $j \leq i$ ~~After~~

if $A[i] >$ CurrentMax

Then CurrentMax $\leftarrow A[i]$



From S_i is CurrentMax is the maximum all elements in A

$A[i] >$ CurrentMax

CurrentMax is max first $i+1$ elements

$A[i] \leq$ CurrentMax

CurrentMax is Max first "i+1"

Algorithm *arrayFind(A, n, x)*

Input : an element x, n, an array A of n integers
Output : The index i such that A[i]=x or -1 if no element in A is equal to x

```
i ← 0
while i <n do
    if x = A[i] then
        return i
    else
        i ←i+1
return -1
```

Recursive algorithm

* Write a procedure which can call itself
* those calls is for smaller instance

$n! = 1 \times 2 \times 3 \times \ldots$

n factorial

base case can be solved without recursive

Input : n

Output : n!

if n=1 then return 1

return (Fact(n-1) * n)

Smaller instance

Find Recursive Array Max (A, n)

input : n, array A

Output : Maximum of Array

if n = 1 then return A[0]

base case
no recursive calls

else

return Max (RecursiveArrayMax (A, n-1), A(n))

recursive call

Smaller input

Find ~~product~~ of two integers a, b ~~with~~

only using addition.

Product (a, b)

Input: a, b

Output: a * b

if b = 1 then return a

return (Product (a, b-1)) + a

Time Complexity for Array Max

Recurrence equation

$$T(n) = \begin{cases} 3 & \text{when } n = 1 \\ T(n-1) + 5 & \text{o.w} \end{cases}$$

"Closed form" Solving recurrence equation

$$T(n) = T(n-1) + 5 = T(n-2) + 5 + 5$$

$$\vdots$$

$$= T(1) + \underbrace{5 + 5 + \cdots + 5}_{(n-1)}$$

$$= 5n$$

Product (a,b)

if b=1 then return a

return (Product (a, b-1)) + a

Product(3, 2) 3+3 = 6

3

Product (3,1) →