Lecture 3

Recurrence Relations

RECAP QN1:

f(n)	g(n)
$n^3 + 2n^2$	$100n^2 + 1000$
$n^{0.1}$	$\log n$
$n + 100n^{0.1}$	$2n + 10 \log n$
$5n^5$	$n!$
$n^{-15}2^n/100$	$1000n^{15}$
$8^{2\log n}$	$3n^7 + 7n$

Which function is greater??

f(n)	g(n)	larger
$n^3 + 2n^2$	$100n^2 + 1000$	(n^3)
$n^{0.1}$	$\log n$	$(n^{0.1})$
$n + 100n^{0.1}$	$2n + 10 \log n$	(n)
$5n^5$	$n!$	$(n!)$
$n^{-15}2^n/100$	$1000n^{15}$	(n^{-15})
$8^{2\log n}$	$3n^7 + 7n$	(n^7)

QN:2

For the following pairs of functions, indicate whether the ?? could be replaced with O, Ω or Θ.

Pr.	$f(n)$	$g(n)$	$f(n) \in ??(g(n))$
a)	$48n$	$2n^2 + 1$	
b)	$0.00001n^2$	$2^{2^{10}}(n \log n)$	
c)	$5n^{20} + n^2 \log_2 n$	$n!$	
d)	$4n$	$4 \log_2(2^n)$	
e)	$5n \log n$	$2n^{1.5}$	
f)	$7n^{1024}$	1.000001^n	
g)	$\frac{2n^2}{\log n}$	$3n$	

For the following pairs of functions, indicate whether the ?? could be replaced with O, Ω or Θ.

Pr.	$f(n)$	$g(n)$	$f(n) \in ??(g(n))$
a)	$48n$	$2n^2 + 1$	O
b)	$0.00001n^2$	$2^{2^{10}}(n \log n)$	Ω
c)	$5n^{20} + n^2 \log_2 n$	$n!$	O
d)	$4n$	$4 \log_2(2^n)$	Θ
e)	$5n \log n$	$2n^{1.5}$	O
f)	$7n^{1024}$	1.000001^n	O
g)	$\frac{2n^2}{\log n}$	$3n$	Ω

Introduction to Recurrence Relation

As many algorithms are recursive in nature, it is natural to analyze algorithms based on recurrence relations.

Definition:

Recurrence relation is a mathematical model that captures the underlying time-complexity of an algorithm.

Deriving a recurrence relation Ex 1

```
FOO1(A, left, right)
  if left < right
    mid = floor((left+right)/2)
    FOO1(A, left, mid)
    FOO1(a, mid+1, right)
    FOO2(A, left, mid, right)
```

```
FOO1(A, left, right)
  if left < right                  → Constant time
    mid = floor((left+right)/2)    → Constant time
    FOO1(A, left, mid)             → T(n/2)
    FOO1(a, mid+1, right)          → T(n/2)
    FOO2(A, left, mid, right)      → θ(n)
```

Thus, the total time taken by the function for the case left<right can be written as:

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

And for the case left >= right , only the condition check will occur i.e., if left < right and thus the function will complete in time.

So, we can write as:

$$T(n) = \begin{cases} \Theta(1), & n = 1 \text{ (left} = \text{right)} \\ 2T\left(\frac{n}{2}\right) + \Theta(n) + \Theta(1), & \text{if } n > 1 \end{cases}$$

Ex 2

```
FOO(A, low, high, x)
 if (low > high)
  return False
 mid = floor((high+low)/2)        ⟶  1
 if (x == A[mid])
  return True                     ⟶  1
 if (x < A[mid])
  return FOO(A, low, mid-1, x)    ⟶  T(n/2)
 if (x > A[mid])
   return FOO(A, mid+1, high,x)   ⟶  T(n/2)
```

Thus, the running time of this algorithm can be written as:

$$T(n) = \begin{cases} \Theta(1), & n = 1 \text{ (low} = \text{high)} \\ T\left(\frac{n}{2}\right) + \Theta(1), & \text{if } n > 1 \end{cases}$$

Solving Recurrences

→ To solve a recurrence relation means to find a function defined on the collection of indices (i.e. subscripts, usually the natural numbers) that satisfies the recurrence.

→ There are usually many such functions.
→ If initial conditions are given, we will want to chose the one function that gives the correct initial values.
→ To analyze recurrence relations:

  * Substitution method,
  * Recurrence tree method
  * Master theorem.

→ Solutions to recurrence relations yield the time-complexity of underlying algorithms.

Evaluating Recurrence:

How to think about $T(n) = T(n-1) + 1$

How to find the value of a $T(k)$ for a particular k: Substitute up from $T(1)$ to $T(k)$

Substitute down from $T(k)$ to $T(1)$

Solving the recurrence and evaluate the resulting expression

All three methods require having the **initial conditions** for the recurrence

Initial Conditions

The initial conditions are the values of the recurrence for small values of n For example, the values of $T(0)$, $T(1)$, $T(2)$

We will see that the initial conditions are determined by the specific problem being solved

A Recurrence Equation has multiple solutions.

The initial conditions determines which of those solutions applies.

Substitution Method

The most general method:

1. ***Guess*** the form of the solution.
2. ***Verify*** by induction.
3. ***Solve*** for constants.

Eg:

$\mathbf{T(n) = T(n/2) + c}$ (1)

but $T(n/2) = T(n/4) + c$,

So $T(n) = T(n/4) + c + c$

$T(n) = T(n/4) + 2c$ (2)

$T(n/4) = T(n/8) + c$

$T(n) = T(n/8) + c + 2c$

$T(n) = T(n/8) + 3c$ (3)

Result at i^{th} unwinding	**i**
$T(n) = T(n/2) + c$	1
$T(n) = T(n/4) + 2c$	2
$T(n) = T(n/8) + 3c$	3
$T(n) = T(n/16) + 4c$	4

We need to write an expression for the kth unwinding (in n & k)

Must find patterns, changes, as i=1, 2, …, k

We will then need to relate n and k

Result at i^{th} unwinding			i
$T(n)$	$= T(n/2) + c$	$=T(n/2^1) + 1c$	1
$T(n)$	$= T(n/4) + 2c$	$=T(n/2^2) + 2c$	2
$T(n)$	$= T(n/8) + 3c$	$=T(n/2^3) + 3c$	3
$T(n)$	$= T(n/16) + 4c$	$=T(n/2^4) + 4c$	4

After k unwindings:

$T(n) = T(n/2^k) + kc$

Need a convenient place to stop unwinding – need to relate k & n

Let's pick $T(0) = c_0$ So,

$n/2^k = 0$ =>

$n=0$

let's consider $T(1) = c_0$

So, let:

$n/2^k = 1$ =>

$n = 2^k$ =>

$k = \log_2 n = \lg n$

Substituting back in (getting rid of k):

$$T(n) = T(1) + c\ \lg(n)$$

$$= c\ \lg(n) + c_0$$

$$= \mathbf{O(\ lg(n)\ )}$$

Eg 2: Solve the recurrence relation using Substitution Method:

$$T(n) = 2T(n/2) + n - 1, \ T(1) = 0$$

$$T(n/2) = 2[2T(n/2^2) + (n/2) - 1] + n - 1 \rightarrow (2)$$

$$T(n/k) = 2^k T(n/2^k) + n - 2^k + n - 2^{k-1} + \cdots + n - 1$$

When $n = 2^k$, $T(n) = 2^k T(1) + n + \cdots + n - [2^k - 1 + \cdots + 2^0]$

$$[2^k - 1 + \cdots + 2^0] = 2^k - 1 = n - 1$$

Given $T(1) = 0$,

$$T(n) = n \log_2(n) - n + 1 = \Theta(n \log_2(n))$$

Homework

Show that the solution of $T(n) = T(n - 1) + n$ is $O(n^2)$ using Substitution Method.