Module-2: Analysis of Algorithms

Dr. P. Gayathri
Associate Professor
SCOPE, VIT University

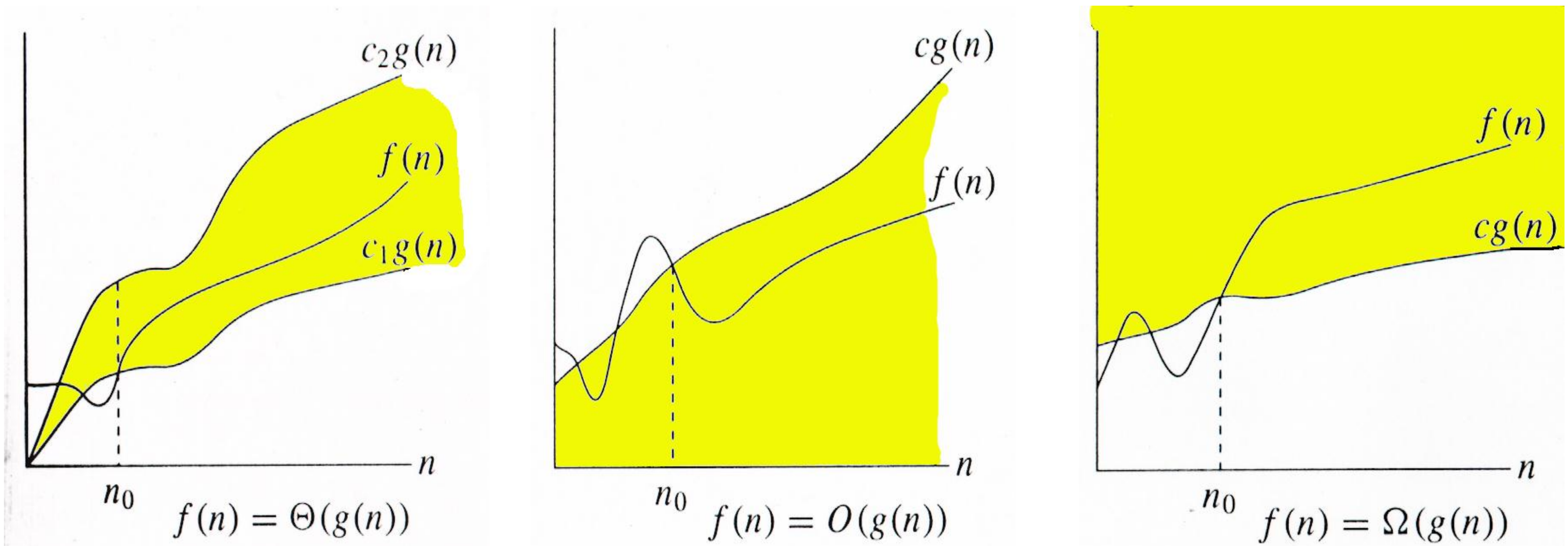
Algorithm

- Set of simple instructions to be followed to solve a problem
- Once an algorithm is given for a problem, important step is to determine the time and space requirement of algorithm

Asymptotic notations and their significance

- Estimation of resource usage of an algorithm involves following definition / assymptotic notations / growth functions
  - Big O (O)
  - Big omega (Ω)
  - Theta (Θ)
  - Little O (o)
  - Little omega (ω)

Relations Between Θ, *O*, Ω



Observations

- For any 2 functions f(n) and g(n), $f(n)=\Theta(g(n))$ iff $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$
- If a function $f(n) = o(g(n))$ then $f(n) = O(g(n))$ but the converse is not true
- If a function $f(n) = \omega(g(n))$ then $f(n)=\Omega(g(n))$ but converse is not true

Note

- Main idea behind these definitions is to establish relative order among functions.
- Problems related to notations – Refer notebook

Time Complexity of an Algorithm

- Best case analysis – usage of algorithm is atleast
- Worst case analysis – usage of algorithm is atmost
- Average case analysis – usage of algorithm is average
- Example: linear search
  - Best – searching for first element
  - Worst – searching for last element or the element which is not present in the list
  - Average – searching for other element which lies between first and last element

Various Computing Times

- C – constant
- Log n – logarithmic
- $\text{Log}^2 n$ – log squared
- N – linear
- N^2 – quadratic
- N^3 – cubic
- 2^n – exponential

Running Time of an Algorithm

- Example : $\sum_{i=1}^{n} i^3$

```
Sum (int n)
{
    int ps;
    ps = 0;
    for (i = 1; i<=n; i++)
            ps = ps + i * i * i
    return ps
}
```

Total RT = 1+1+n+1+n+4n+1 = 6n+4 = O(n)

Note

- Lower order terms and constants can be ignored in O notation

  Example 1: $T(n) = O(2n^2)$ – wrong

  Correct form is: $T(n) = O(n^2)$

  Example 2: $T(n) = O(n + n^2)$ – wrong

  Correct form is: $T(n) = O(n^2)$

RT Calculation - General Rules

- Refer notebook

Running Time of an Algorithm using General Rules

- Example : $\sum_{i=1}^{n} i^3$

```
Sum (int n)
{
    int ps;
    ps = 0;
    for (i = 1; i<=n; i++)
            ps = ps + i * i * i
    return ps
}
```

Total RT = max(O(1), O(n), O(1)) = O(n)

Performance Analysis of an Algorithm

- Analysis of iterative algorithms
- Analysis of recursive algorithms

Analysis of iterative algorithms

- Detailed analysis of Insertion sort

```
InsertionSort (A)
  for j = 2 to length[A)
     key = A[j]
     //insert A[j] into sorted sequence A[1..j-1]
     i = j - 1;
     while (i > 0) and (A[i] > key)
         A[i+1] = A[i]
         i = i - 1
     A[i+1] = key
```

Insertion sort RT

- Best case – O(n)
- Worst case – O(n^2)
- Average case – as bad as worst case – depends on how much the elements are already sorted – So, O(n^2)

Supplementary exercise

- Analysis of selection sort

  Best – $O(n^2)$

  Worst - $O(n^2)$
- Analysis of bubble sort

  Best – $O(n^2)$

  Worst - $O(n^2)$
- Improved bubble sort

  Best – $O(n)$

  Worst - $O(n^2)$

Analysis of Recursive Algorithms

- Each recursive algorithm has one or more recursive cases and one or more base cases
- Example: Recurrence relation T(n)

$$T(n) = \begin{cases} a & \text{if } n=1 \\ 2T(n/2)+b(n)+c & \text{if } n>1 \end{cases}$$

- The portion of definition that does not contain T is called base case
- The portion that contains T is called recursive case

Forming Recurrence Relations

```
Void f (int n)
{
        if n > 0 {
                Print n;
                f(n-1); }
}
```

$$T(n) = \begin{cases} a & \text{if } n=0 \\ b+T(n-1) & \text{if } n>0 \end{cases}$$

Reason:

If n = 0, one comparison operation

If n > 0, 2 basic operations (comparison & print), one recursive call with parameter n-1

Exercise Problem - 1

```
Int g (int n)
{
	if(n==1)
		return 2;
	else
		return 3*g(n/2)+g(n/2)+5
}
```

Exercise Problem - 2

```
Fib (int n)
{
        if (n==1 || n==2)
                return 1
        else
                return fib(n-1) + fib(n-2);
}
```

Exercise Problem - 3

```
Power (int x, int n)
{
        if(n==0)
                return 1;
        else if (n==1)
                return x;
        else if (n%2 == 0)
                return power(x,n/2) * power(x,n/2);
        else
                return x*power(x,n/2) * power(x,n/2);
```

Methods to solve Recurrence Relations

There are many methods to solve recurrence relations. Few are listed below:

- Master Method
- Iterative Method
- Recursion Tree method
- Substitution method
  - Can be applied only in cases when it is easy to guess the solution
  - 2 steps
    - Guess the solution
    - Show that the solution works

Substitution Method – Example

Merge sort:

$$T(n) = \begin{cases} c & \text{if } n=1 \\ 2T(n/2) + n & \text{if } n>1 \end{cases}$$

Let solution be $T(n) = O(n \log n)$

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

Our aim is to prove $T(n) <= c\, n \log n$ for $c > 0$

Let $n = n/2$

$T(n/2) <= c\,(n/2 \log n/2)$ -------- (2)

Apply 2 in 1

$$\begin{aligned} T(n) &= 2\,(c\, n/2 \log n/2) + n \\ &= c\, n \log n/2 + n \\ &= c\, n \log n - c\, n \log 2 + n \\ &= c\, n \log n - c\, n + n \quad \text{(b'coz } \log 2 = 1) \\ &= c\, n \log n \quad (\text{as } c>=1) \\ &= O(n \log n) \end{aligned}$$

$T(n) = O(n \log n)$

Master Method

- The Master theorem allows us to easily calculate the running time of recursive algorithm in Θ notation
- Not all recurrence relations can be solved with the use of the master theorem
- General syntax of recurrence relation

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

Case 1

Generic form: $f(n) = O(n^c)$ where $c<\log_b a$
Then, $T(n) = \Theta(n^{\log_b a})$

Example: $T(n) = 8T(n/2) + 1000n^2$
$a = 8$, $b = 2$, $f(n) = 1000n^2$
Next, we see if we satisfy the case 1 condition:
$f(n) = O(n^c)$ where $c =2$
$\text{Log}_b a = \text{Log}_2 8 = 3$. So $c<\log_b a$ is true
Therefore, $T(n) = \Theta(n^{\log_b a}) = \Theta(n^{\log_2 8}) = \Theta(n^3)$

Case 2

Generic form: $f(n) = \Theta\ (n^c \log^k n)$ where $c=\log_b a$ and $k>=0$

Then, $T(n) = \Theta(n^c \log^{k+1} n)$

Example: $T(n) = 2T(n/2) + 10n$

$a = 2$, $b = 2$, $f(n) = 10n$

Next, we see if we satisfy the case 2 condition:

$f(n) = \Theta\ (n^c \log^k n)$ where $c =1$ and $k=0$

$\text{Log}_b a = \text{Log}_2 2 = 1$. So $c = \log_b a$ is true

Therefore, $T(n) = \Theta(n^c \log^{k+1} n) = \Theta(n^1 \log^1 n) = \Theta(n \log n)$

Case 3

Generic form: $f(n) = \Omega\ (n^c)$ where $c>\log_b a$

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

Example: $T(n) = 2T(n/2) + n^2$

$a = 2$, $b = 2$, $f(n) = n^2$

Next, we see if we satisfy the case 3 condition:

$f(n) = \Omega\ (n^c)$ where $c =2$

$\text{Log}_b a = \text{Log}_2 2 = 1$. So $c > \log_b a$ is true

Therefore, $T(n) = \Theta(f(n)) = \Theta(n^2)$

Iterative Method

- Factorial Example

```
Int fact (int n)
{
If (n==0)
Return 1
Else
Return n*fact(n-1);
}
```

Recurrence relation and solution

$$T(n) = \begin{cases} C & n=0 \\ b + T(n-1) & n>=1 \end{cases}$$

$T(0) = C$

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

Let $n=5$, $T(5) = b+T(4)$

$T(4) = b+T(3)$

$T(3) = b+T(2)$

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

$T(1) = b+T(0)$

$T(n) = b+b+b+b+b+C$

$= nb+C$

$=O(n)$