CSE 2011 Data structures and Algorithms

Algorithm Analysis: Order of Growth, Asymptotic Notations

Recap -- Complexity of the Algorithm

- Time and space complexity
- Time Complexity:- Number of computational steps/operations to be performed. Obtained as time function
- Check whether it depends on input size (Very Important!!)
- Space complexity:- The memory space occupied by the data processed by the algorithm

Analysis by counting frequency

```
Algorithm_product(x,y)
{
   return x*y
}
```

- Will execute in constant time

$$f(n) = O(1)$$

Algorithm analysis by counting frequency

```
Algorithm_sum(A, n)
{
    s=0
    for ( i=0; i<n; i++)
    {
        s = s + A[ i ]
    }
    return s
}
```

- $s = 0 \rightarrow 1$
- $i = 0 \rightarrow 1,\ i < n \rightarrow n+1,\ i++ \rightarrow n$
- $s = s + A[i] \rightarrow n$
- return $s \rightarrow 1$
- $f(n) = 1 + 1 + (n+1) + n + n$

  $= 3n + 3$
- It's a first degree polynomial

  $\therefore f(n) = O(n)$

Another Example

```
Algorithm_add(A,B,n)
{
   for(i=0;i<n;i++)
   {
       for{j=0;j<n;j++}
       {
           C[i,j]=A[i,j]+B[i,j]
        }
    }
}
```

- 1st for loop $\rightarrow n+1$ time
- 2nd for loop $\rightarrow n+1$ time
- Addition $\rightarrow n$ time
- For each iteration of first for loop, second for loop will execute $n+1$ times and addition n times

$$\therefore f(n) = (n+1)(n+1) + (n+1)n$$

$$= 2n^2 + 3n + 1$$

- Second degree polynomial
- $\therefore f(n) = O(n^2)$

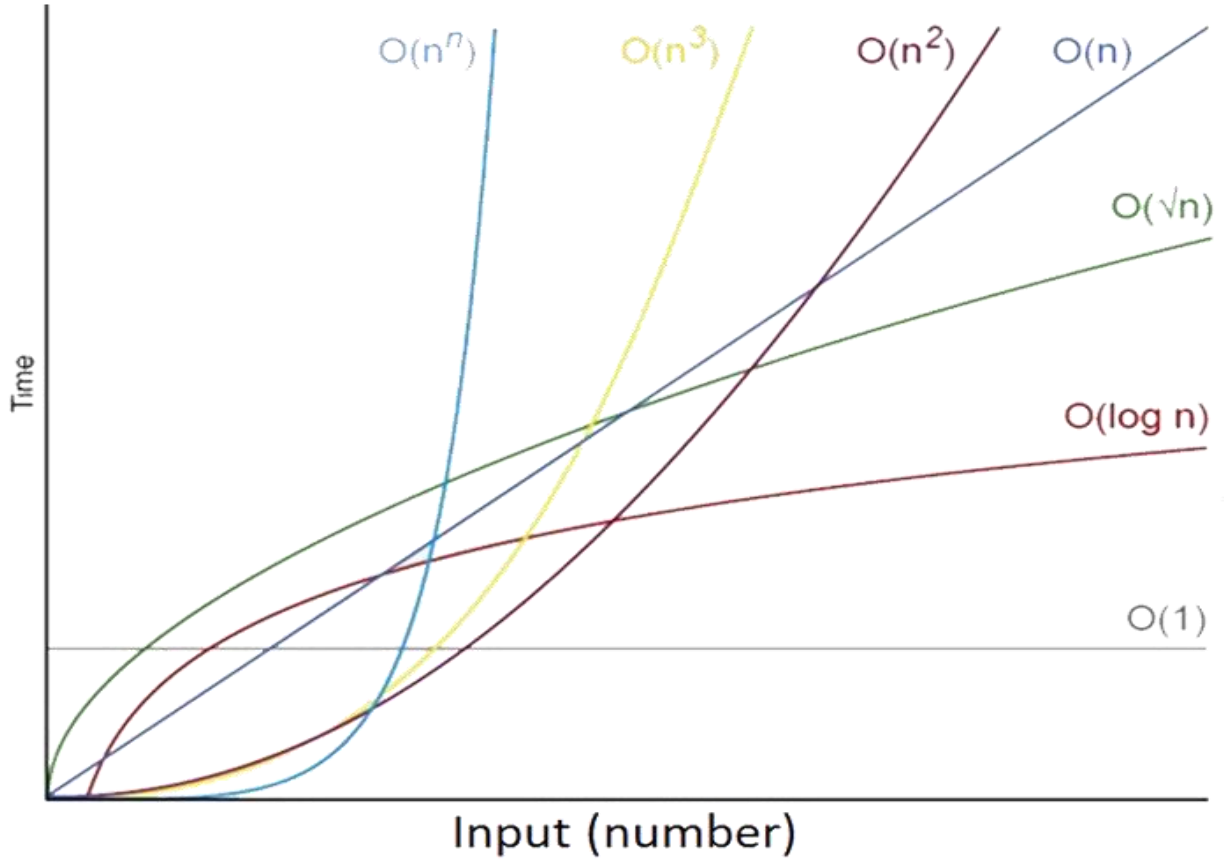
Classes of Time Functions and their Variation

- Constant $f(n) = O(1)$
- Logarithmic $f(n) = O(logn)$
- Linear $f(n) = O(n)$
- Quadratic $f(n) = O(n^2)$
- Cubic $f(n) = O(n^3)$
- Exponential $f(n) = O(2^n)$

Comparison of Time Functions (Order of Growth)

$$1 < logn < \sqrt{n} < n < nlogn < n^2 < n^3 < \cdots < 2^n < 3^n < \cdots < n^n$$

$logn$	n	n^2	2^n
0	1	1	2
1	2	4	4
2	4	16	16
3	8	64	256
3.16	9	81	512
3.22	10	100	1024



Asymptotic Analysis

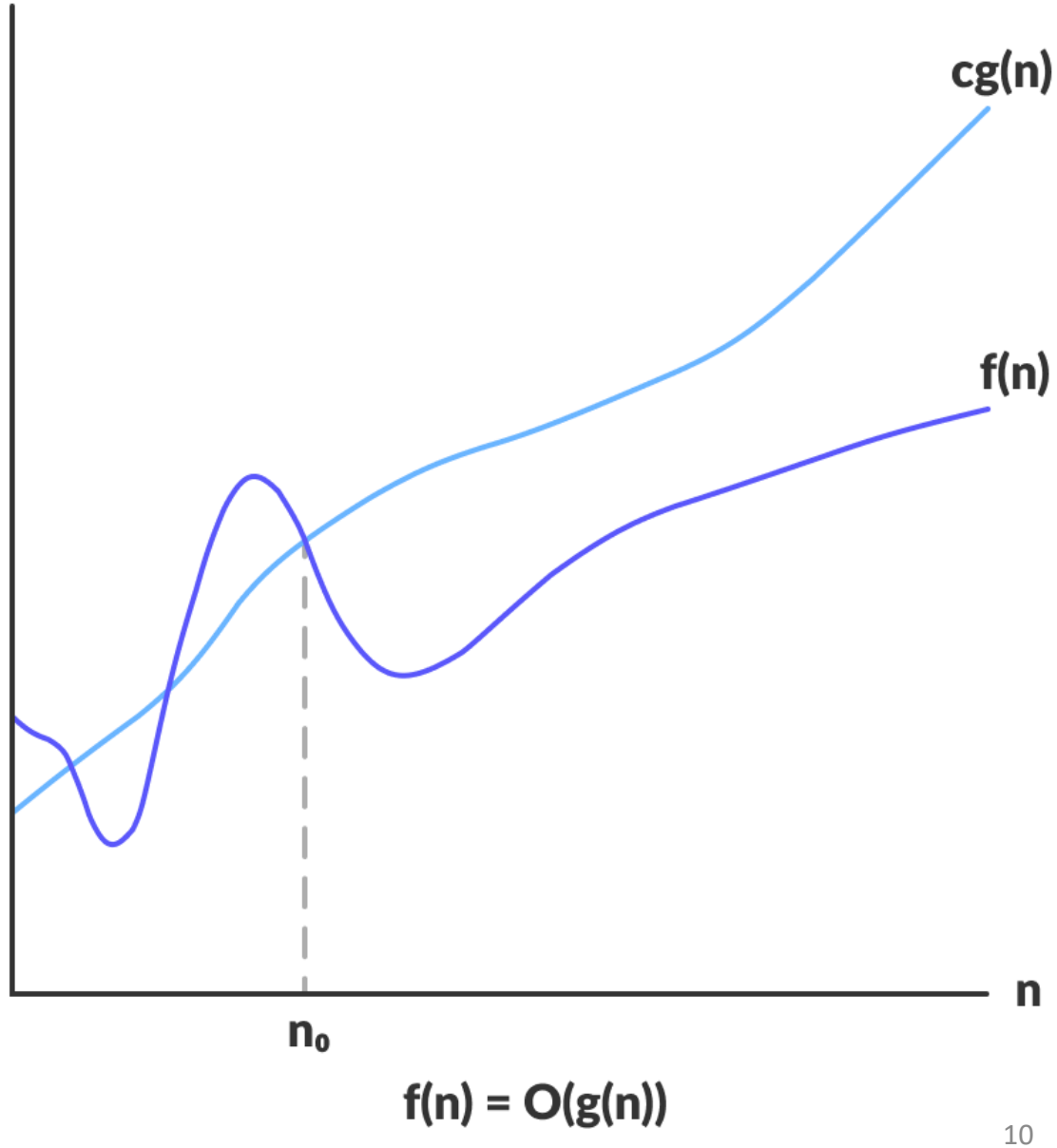
- Running times of algorithms are not clock times, but time functions
- They are functions of input size n
- For smaller values of n, the differences in running times (efficiency) is small.
- So algorithm analyses are performed for very large values of n, typically when $n \to \infty$
- Such analyses are called asymptotic analyses

Asymptotic Notations

- Mathematical notations used for asymptotic analyses are called asymptotic notations
- There are five different asymptotic notations
  - Big-Oh denoted as O
  - Big-Omega denoted as Ω
  - Big-Theta denoted as Θ
  - Small-Oh denoted as o
  - Small-Omega denoted as ω

Big-Oh Notation

- Big-Oh Notation (Asymptotic Upper bound)
- The function $f(n) = O(g(n))$ if ∃ +ve constants c and n_0 such that $f(n) \leq cg(n)\ \forall\ n \geq n_0$



Big-oh notation example

- $f(n) = 100n + 5$ what is $g(n)$, c, n_0?
  - $100n + 5 \leq 101n$ for $n \geq 6$
  - Thus, if $g(n) = n$, c $= 101, n_0 = 6$,
  - We can write $f(n) = O(g(n))$
- Alternatively,
  - $100n + 5 \leq 105n$ for $n \geq 1$
  - Again $f(n) = O(g(n))$, for $g(n) = n, c = 105\ , n_0 = 1$
- Thus c and n_0 are not unique

Big –oh notation

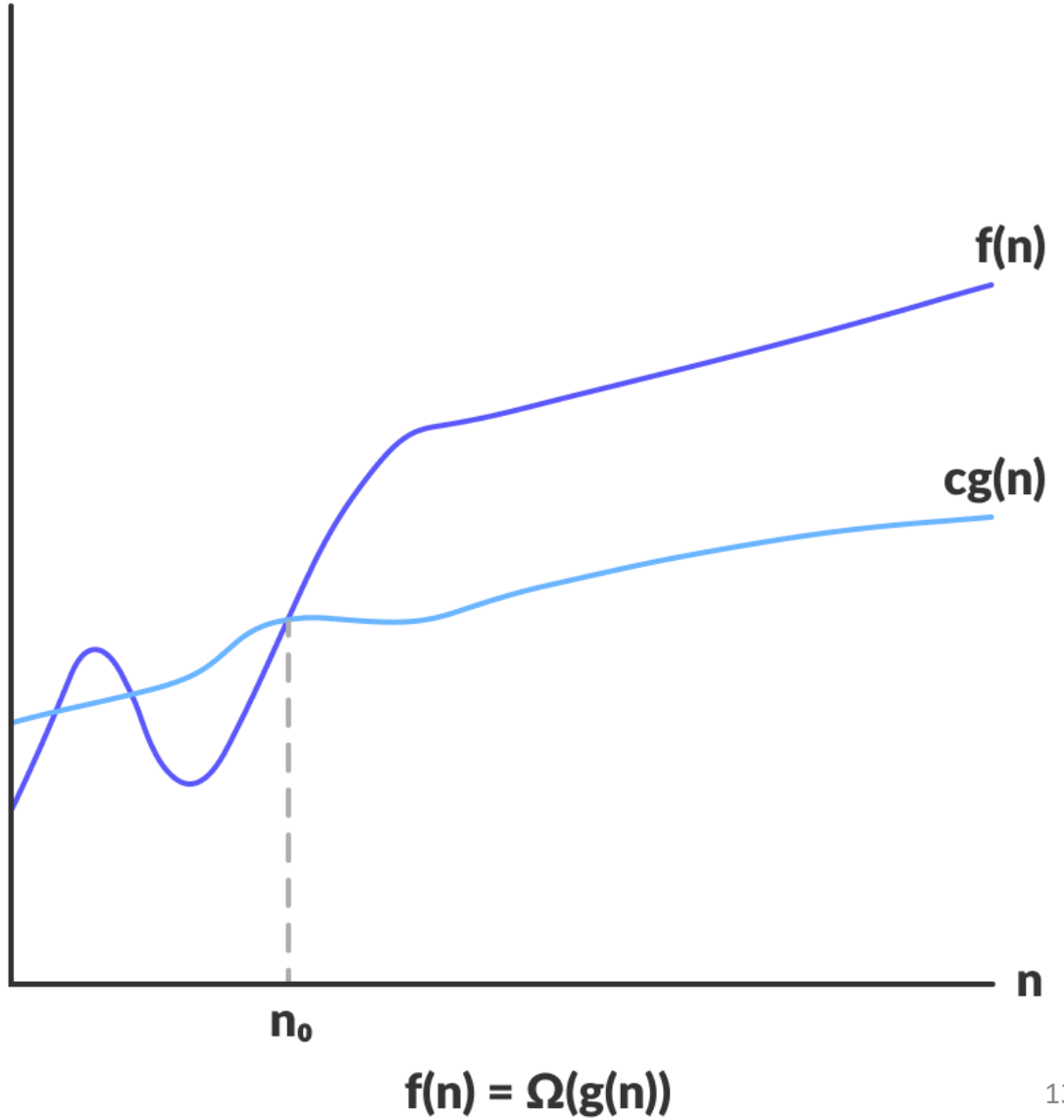
- With the same argument it can be shown that:
- $f(n) = 100n + 5 = O(n^2)$ also
- Remember

$$1 < logn < \sqrt{n} < n < nlogn < n^2 < n^3 < \cdots < 2^n < 3^n < \cdots < n^n$$

- So if any function is $O(n)$ it will be $O(n \log n\ ), O(n^2), O(n^3)$ … (all greater than $O(n)$.
- But never $O(\log n)\ , O(1)$ (smaller bounds)
- Conventionally take the closest one as the upper bound

Big-Omega Notation

- Big-Omega Notation (Asymptotic Lower bound)
- The function $f(n) = \Omega(g(n))$ if ∃ +ve constants c and n_0 such that $f(n) \geq cg(n)\ \forall\ n \geq n_0$

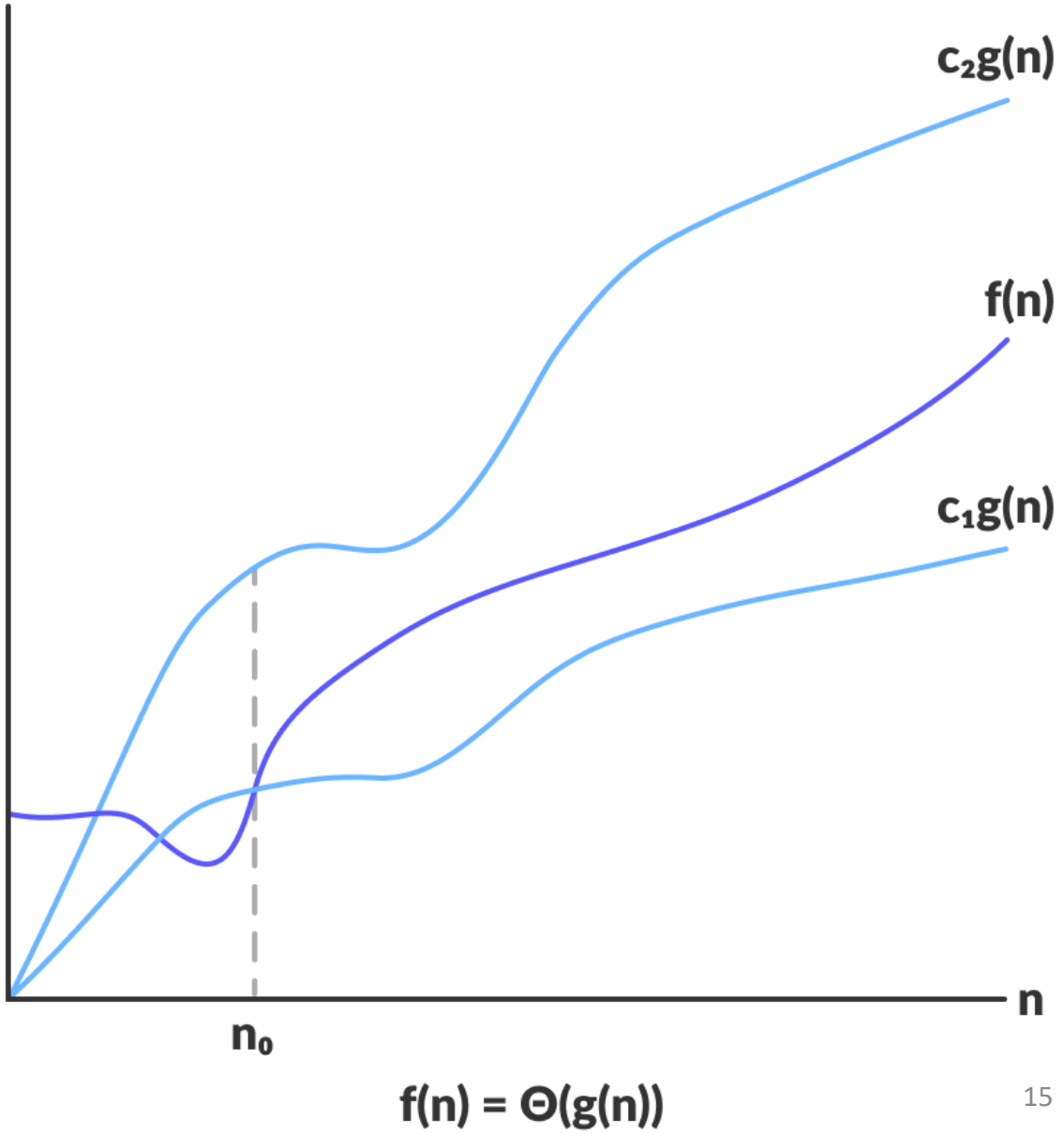


Big-omega notation example

- $f(n) = 100n + 5$ what is $g(n), c, \ n_0$?
  - $100n + 5 \geq 90n$ for $n \geq 1$
  - Thus, if $g(n) = n, \mathrm{c} = 90, n_0 = 1,$
  - We can write $f(n) = \Omega(g(n))$
- If $f(n) = \Omega(n) \Rightarrow f(n) = \Omega(\log n)$ or $\Omega(1)$ but never $\Omega(n \log n)$ or greater
- Use the closest possible always although many would be correct

Big-Theta Notation

- Big-Theta Notation (Asymptotic Tight bound)
- The function $f(n) = \Theta(g(n))$ if ∃ +ve constants c_1, c_2 and n_0 such that $c_1 g(n) \leq f(n) \leq c_2 g(n)\ \forall\ n \geq n_0$



Big-theta notation example

- $f(n) = 100n + 5$ what is $g(n)$, c_1, c_2 n_0?
  - $100n + 5 \leq 101n$ for $n \geq 6$
  - $100n + 5 \geq 90n$ for $n \geq 1$
  - Thus, if $g(n) = n, c_1 = 101, c_2 = 90$ $n_0 = 6$,
  - We can write $f(n) = \Theta(g(n))$
- We can't write anything lesser or greater than $O(n)$ here

Small-O and Small-Omega Notations

- The $\leq$ and $\geq$ in big-oh and big-omega definitions are replaced by $<$ and $>$ respectively
- Small-Oh:
  The function $f(n) = o(g(n))$ if $\exists$ +ve constants c and n_0 such that $f(n) < cg(n)\ \forall\ n \geq n_0$
- Small-Omega:
  The function $f(n) = \omega(g(n))$ if $\exists$ +ve constants c and n_0 such that $f(n) > cg(n)\ \forall\ n \geq n_0$
- Also called little-o and little-omega

Asymptotic Notations- Problems

1. $f(n) = 2n^2 + 3n + 4$
   - $2n^2 + 3n + 4 \leq 2n^2 + 3n^2 + 4n^2 = 9n^2 \ \forall n \geq 1$
   - $c = 9, n_0 = 1$
   - $\therefore 2n^2 + 3n + 4 = O(n^2)$
   - Also $2n^2 + 3n + 4 \geq n^2 \ \forall n \geq 1$
   - $c = 1, n_0 = 1$
   - $\therefore 2n^2 + 3n + 4 = \Omega(n^2)$
   - $\because f(n) = O(n^2)$ and $f(n) = \Omega(n^2), f(n) = \Theta(n^2)$
2. $f(n) = n^2 \log n + n$
   - $n^2 \log n \leq n^2 \log n + n \ \leq 10n^2 \log n$ for $n \geq 2$ $(c_1 = 10, c_2 = 1, n_0 = 2)$
   - $\therefore f(n) = O(n^2 \log n), f(n) = \Omega(n^2 \log n)$ and $f(n) = \Theta(n^2 \log n)$

Asymptotic Notations- Problems

3. $n!$
   - $1 \times 1 \times \cdots n\ times \leq 1 \times 2 \times \cdots n\ times \leq n \times n \times n\ times$
   - $1 \leq n! \leq n^n$
   - $\therefore n! = \Omega(1)$ and $n! = O(n^n)$
   - No tight bound (Θ). Why??
4. $\log n!$ → Solve yourself
5. $3 \log n + 100$
   - $3 \log n + 100 \leq 150 \log n$ for $n \geq 3$
   - So $c = 150$ and $n_0 = 3$
   - $\therefore 3 \log n + 100 = O(\log n)$

Properties of Asymptotic Notations

1. Reflexivity: $f(n) = O(f(n))$
2. Symmetry: $f(n) = \Theta(g(n))$ if and only if $g(n) = \Theta(f(n))$
3. Transitivity: $f(n) = O\big(g(n)\big)$ and $g(n) = O\big(h(n)\big) \Rightarrow f(n) = O\big(h(n)\big)$
4. Transpose Symmetry: $f(n) = O(g(n))$ if and only if $g(n) = \Omega(f(n)$

Very Useful properties

$$1.\ \max\big(f(n), g(n)\big) = \Theta(f(n) + g(n))$$

$$2.\ O\big(f(n) + g(n)\big) = \max(f(n), g(n))$$

- Proofs: Self study
- Refer to the link: https://www.geeksforgeeks.org/properties-of-asymptotic-notations/

Thank you