562.613 APPLIED DATA STRUCTURES

Lecture 06

Dr. M. G. Abbas Malik
muhammad.malik@manukau.ac.nz

Lecture 06

- Algorithmic Complexity Analysis
- Binary Search Complexity
- Sequential Search Complexity
- Bubble Sort Complexity
- Insertion Sort Complexity
- LinkedList<T>

Algorithmic Complexity

- **Time Complexity** – running time
- **Space Complexity** – memory required to work
- Whenever we talk about the Algorithmic Complexity, it means **Time Complexity**

Time Complexity

- $T(n)\ --\ where\ n\ is\ size\ of\ the\ problem$
- Define time taken without depending on the implementation details
- An algorithm will take different amounts of time on the same inputs depending on:
  - Processor speed, Instruction set, Disk speed, Brand of compiler, …

Estimate efficiency of each algorithm Asymptotically

Time Complexity

- $T(n)\ --where\ n\ is\ size\ of\ the\ problem$
- Define in terms of number of steps - each step take constant time to complete
- Example: Addition of two Integers
  - Add two integers digit by digit (or bit by bit) - a step in our computational model
  - Adding of two $n-digit/bit$ integers takes $n-steps$

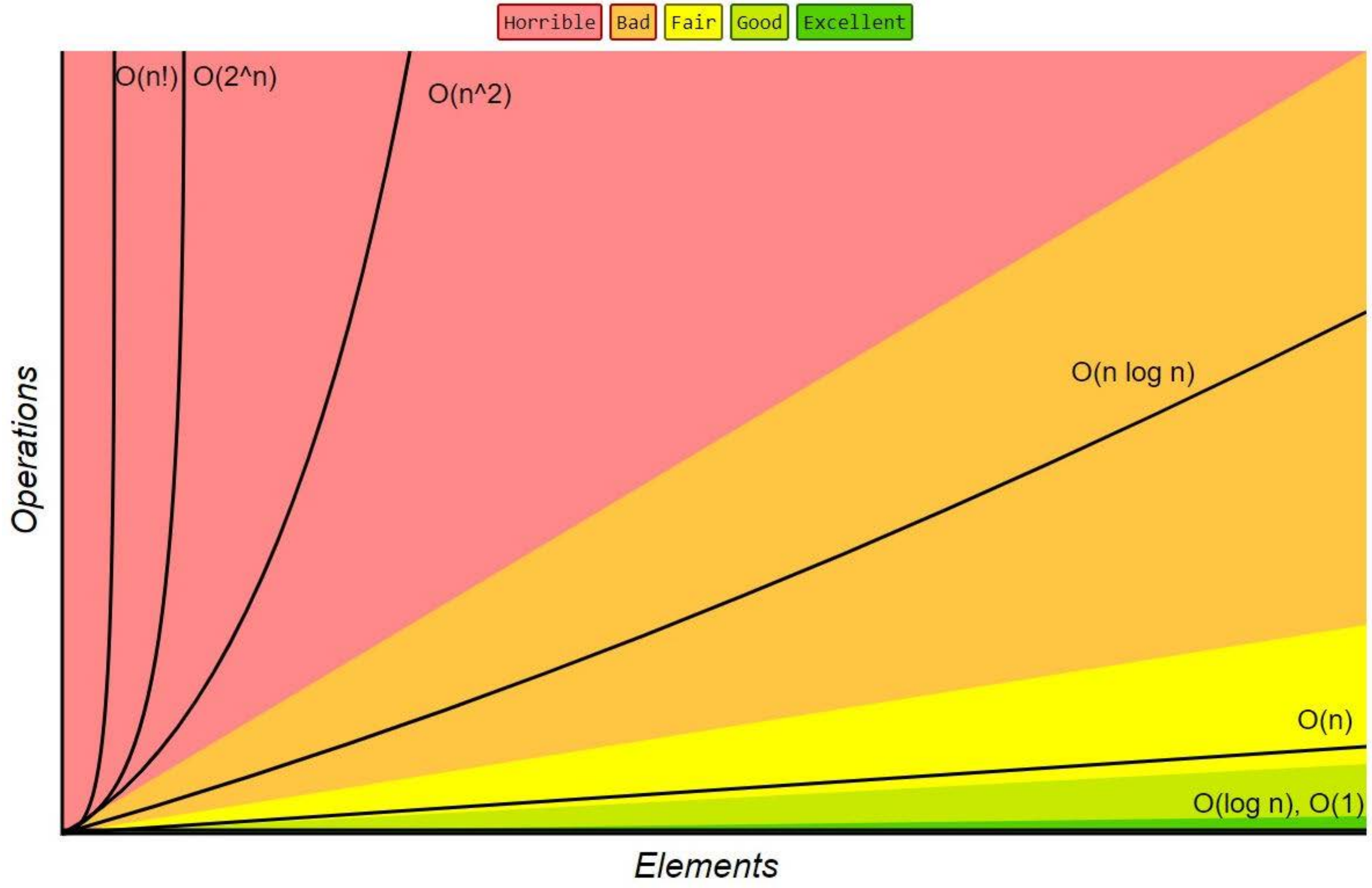
Time Complexity

- $T(n) = c \times n$
  - $where\ c\ is\ time\ taken\ to\ add\ two\ digits\ or\ bits$
- On different computers, c can be different
- Computer 1: $T(n) = c_1 \times n$
- Computer 1: $T(n) = c_2 \times n$

Asymptotic Notation

- Goal of **computational complexity** is to classify algorithms according to their performances
- Big-O notation – upper bound
- Big-Ω notation – lower bound
- Big-θ notation – upper and lower bound

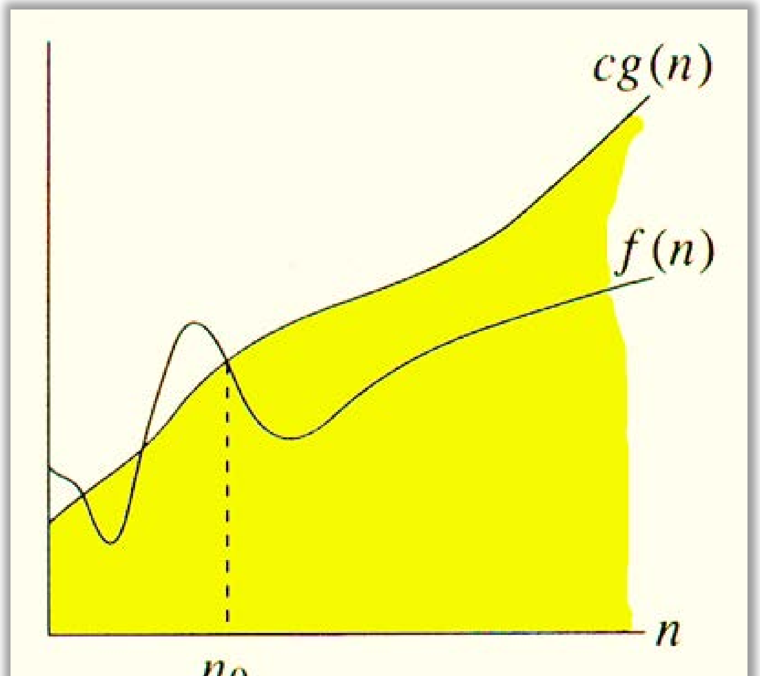
Order of growth in Big-O notation



Big - O Notation

- An **upper bound** of complexity of running time

For $f(n)$ and $g(n)$,
$f(n) = O(g(n))$, if there exists c and n_0 such that
$f(n) \leq c \times g(n)\ \forall\, n \geq n_0$

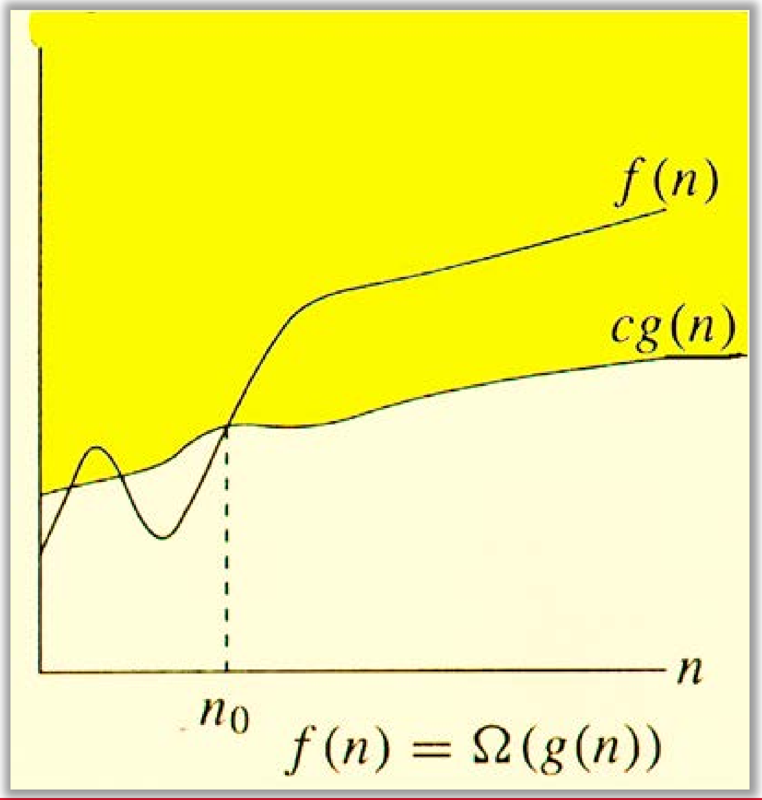


$f(n) = O(g(n))$

Big - Ω Notation

- An **Lower bound** of complexity of running time

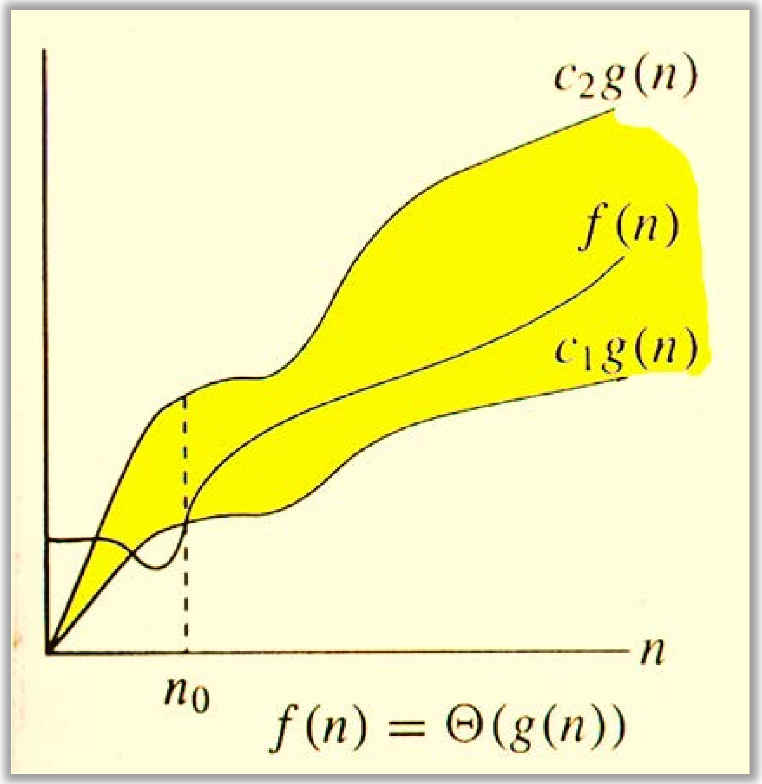
$For\ f(n)\ and\ g(n),$
$f(n) = \Omega(g(n)), if\ there$
$exists\ c\ and\ n_0\ such\ that$
$f(n) \geq c \times g(n)\ \forall\ n \geq n_0$



Big - Θ Notation

- An **upper** and **Lower bound** of complexity of running time

$For\ f(n)\ and\ g(n),$
$f(n) = \Theta(g(n)), if\ there$
$exists\ c_1, c_2\ and\ n_0\ such\ that$
$c_1 \times g(n) \leq f(n) \leq c_2 \times g(n)\ \forall\ n \geq n_0$



Time Complexity

- Best Case
- Worst Case
- Average Case

Binary Search Analysis

$O(\log_2 n)$

1. $lower_{Index} \leftarrow 0;\ upper_{index} = array.Length - 1;$ 1 time
2. $\text{while}\ (\boldsymbol{lower_{index} < upper_{index}})$ $\log_2$ n times
3. $\quad middle = \left\lfloor \frac{(lower_{index}+upper_{index})}{2} \right\rfloor$ $\log_2$ n times
4. $\quad if\,(array[middle] > element)\ then \quad \boldsymbol{upper_{index} = middle - 1}$ $\log_2$ n times
5. $\quad else\ if\,(array[middle] < element) \quad \boldsymbol{lower_{index} = middle + 1}$ $\log_2$ n times
6. $\quad else \quad \boldsymbol{return\ middle}$ $\log_2$ n times
7. $return\ -1$ 1 time

? time

$Time\ Analysis = 1 + \log_2 n + \log_2 n + \log_2 n + \log_2 n + \log_2 n + 1$

$Time\ Analysis = 2 + 5 \times \log_2 n$

Sequential Search

- Start searching from the first element and go till the end of list or array one by one

92

ELEMENT FOUND

12	23	37	48	52	64	77	88	92	123	137	148
0	1	2	3	4	5	6	7	8	9	10	11

Sequential Search

$\textbf{Input}: an\ integer\ array$
$search\ integer\ element$
$\textbf{Output}: Index\ of\ \ element$

1. $for(int\ i = 0; i < array.Length; i++)$ n times
2. $if(array[i] == element)$ n times
3. $return\ i;$ 0 or 1 times
4. $return\ -1$ 0 or 1 times

92

ELEMENT FOUND

12	23	37	48	52	64	77	88	92	123	137	148
0	1	2	3	4	5	6	7	8	9	10	11

Sequential Search Analysis

$$Time\ Analysis = n + n + 1$$
$$Time\ Analysis = 2n + 1$$

$$O(n)$$

Bubble Sort Analysis

$\boldsymbol{Input}: an\ integer\ array$

$\boldsymbol{Output}: Sorted\ array$

1. $temp \leftarrow 0$
2. $for\ (i \leftarrow array.Length\ - 1; \boldsymbol{i > 0}; i - -)$ $(n-1)\ times$
3. $for\ (j \leftarrow 0;\ \boldsymbol{j < i};\ j + +)$ $(n-1)+(n-2)+(n-3)+\cdots+3+2+1\ times$
4. $if\,(\boldsymbol{array}[\boldsymbol{j}] > \boldsymbol{array}[\boldsymbol{j+1}])$ $(n-1)+(n-2)+(n-3)+\cdots+3+2+1\ times$
5. $temp \leftarrow array[j+1]$ $(n-1)+(n-2)+(n-3)+\cdots+3+2+1\ times$
6. $array[j+1] \leftarrow array[j]$ $(n-1)+(n-2)+(n-3)+\cdots+3+2+1\ times$
7. $array[j] \leftarrow temp$ $(n-1)+(n-2)+(n-3)+\cdots+3+2+1\ times$

Bubble Sort Analysis

$O(n^2)$

$$1 + 2 + 3 + \cdots + n = \frac{n(n+1)}{2}$$

$$1 + 2 + 3 + \cdots + (n-1) = \frac{(n-1)(n-1+1)}{2} = \frac{n(n-1)}{2}$$

$$Time\ Analysis = 1 + (n-1) + 5 \times \frac{n(n-1)}{2} = n + \frac{5}{2}n^2 - \frac{5}{2}n$$

$$Time\ Analysis = \frac{5}{2}n^2 - \frac{3}{2}n$$

Insertion Sort Analysis

$\boldsymbol{Input}$: $an\ integer\ array$
$\boldsymbol{Output}$: $Sorted\ array$

1. $i \leftarrow 1$
2. $while\ (\boldsymbol{i < array.Length})$ $(n-1)\ times$
3. $\quad temp \leftarrow array[i]$ $(n-1)\ times$
4. $\quad j \leftarrow i - 1$ $(n-1)\ times$
5. $\quad while\ (\boldsymbol{j \geq 0\ and\ array[j] > temp})$ $1 + 2 + \cdots + (n-1)\ times$
6. $\qquad array[j+1] \leftarrow array[j]$ $1 + 2 + \cdots + (n-1)\ times$
7. $\qquad j \leftarrow j - 1$ $1 + 2 + \cdots + (n-1)\ times$
8. $\quad array[j+1] \leftarrow temp$ $(n-1)\ times$
9. $\quad i \leftarrow i + 1$ $(n-1)\ times$

Insertion Sort Analysis

$O(n^2)$

$$1 + 2 + 3 + \cdots + (n-1) = \frac{n(n-1)}{2}$$

$$Time\ Analysis = 1 + 5 \times (n-1) + 3 \times \frac{n(n-1)}{2}$$

$$Time\ Analysis = 5n - 4 + \frac{3}{2}n^2 - \frac{3}{2}n$$

$$Time\ Analysis = \frac{3}{2}n^2 + \frac{7}{2}n - 4$$

LinkedList<T>

- A general-purpose generic linked list
- Nodes of type LinkedListNode<T>
  - **Properties: Previous, Next and Value**
- Properties: **count, first, last**

LinkedList<T>

Methods

- AddAfter
- AddBefore
- AddFirst
- AddLast
- Clear
- Contains
- Find
- FindLast
- Remove
- RemoveFirst
- RemoveLast

Reference and Reading Material

- Algorithmic Complexity – Link, Link
- LinkedList<T>: Link, Link