ANALYSIS AND DESIGN OF ALGORITHMS

INTRODUCTION

ALGORITHM

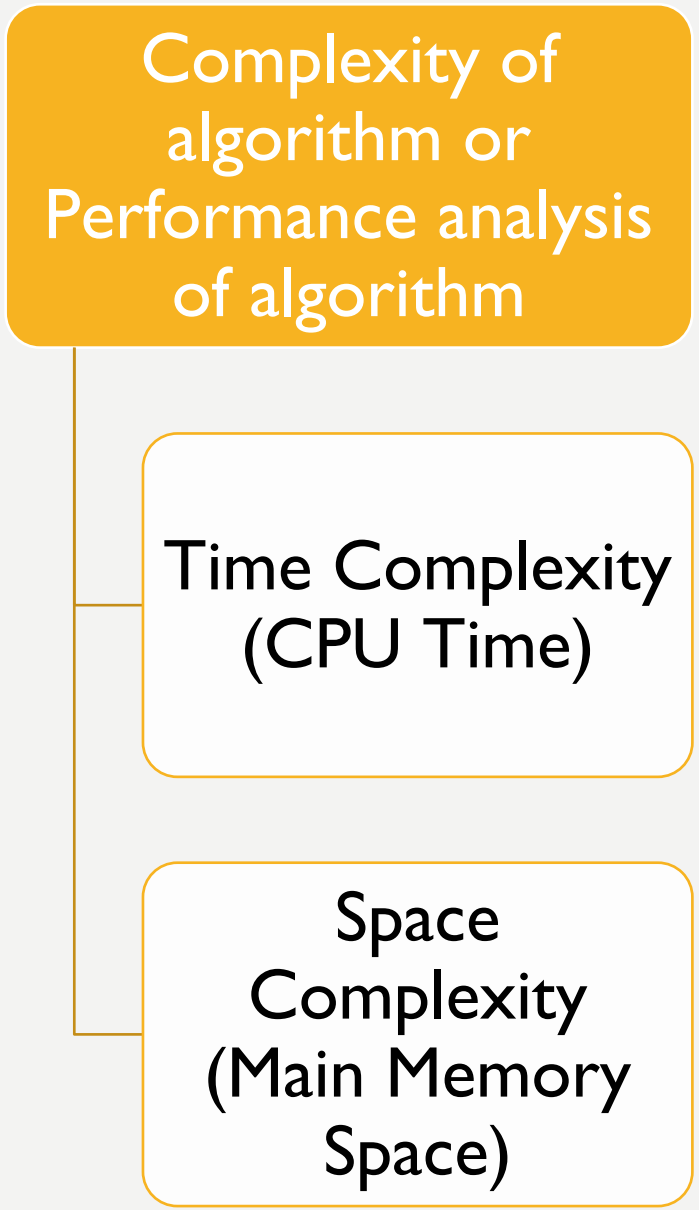
- It is a step by step process to solve any problem.
- An algorithm is a sequence of instructions that act on some input idea to produce some output in a finite number of steps.
- Algorithm is independent of any programming language.

PROPERTIES OF ALGORITHM

- Input: An algorithm must receive data supply externally.
- Output: An algorithm must produce at least 1 output as a result.
- Finiteness: An algorithm must terminate after a finite number of steps.
- Definiteness: The steps of the algorithm must be clear and unambiguous.
- Effectiveness: Without applying any intelligence, all can solve algorithm using pen and paper.

WHY WE ANALYZE THE ALGORITHM?

- The analysis of an algorithm can help us understand it better, and can suggest informed improvements. Algorithms tend to become shorter, simpler, and more elegant during the analysis process.
- For a given problem, there are many ways to design algorithms for it.
- Analysis of algorithm help us to determine which algorithm should be chosen to solve the problem.


Complexity of algorithm or Performance analysis of algorithm
Time Complexity (CPU Time)
Space Complexity (Main Memory Space)

- Time Complexity: Time complexity of an algorithm is the total time required by the program to run till it's completion.
- Space complexity: Space complexity is the total space required by an algorithm to run till it's completion.
- Time & space complexity depends on hardware, O.S, processor etc. But we don't consider any of these factors while analyzing the algorithm.

Example:

Swapping of two numbers using third variable.

set temp to 0

temp ← a

a ← b // a holds value of b

b ← temp // b holds value of a stored in temp

Swapping of two numbers without using third variable

X = X + Y

Y = X - Y

X = X -Y

TIME COMPLEXITY OF AN ALGORITHM CAN BE CALCULATED BY USING TWO METHODS:

- Posteriori Analysis: In Posteriori analysis, algorithm is implemented and executed on certain fixed hardware and software then we select which algorithm will take less time to execute. Here time used is given in time units.
- Priori Analysis: In Priori analysis, the time of algorithm is found prior to implementing in any programming language. Here time used is not in terms of seconds or any such time units. Instead it represents the number of operation that are carried out while executing the algorithm.

- In Prior analysis, we built an equation that relates number of operations (steps) that a particular algorithm does to the size of input. Once the equation for two algorithms are formed, we can then compare rate at which equation grows.

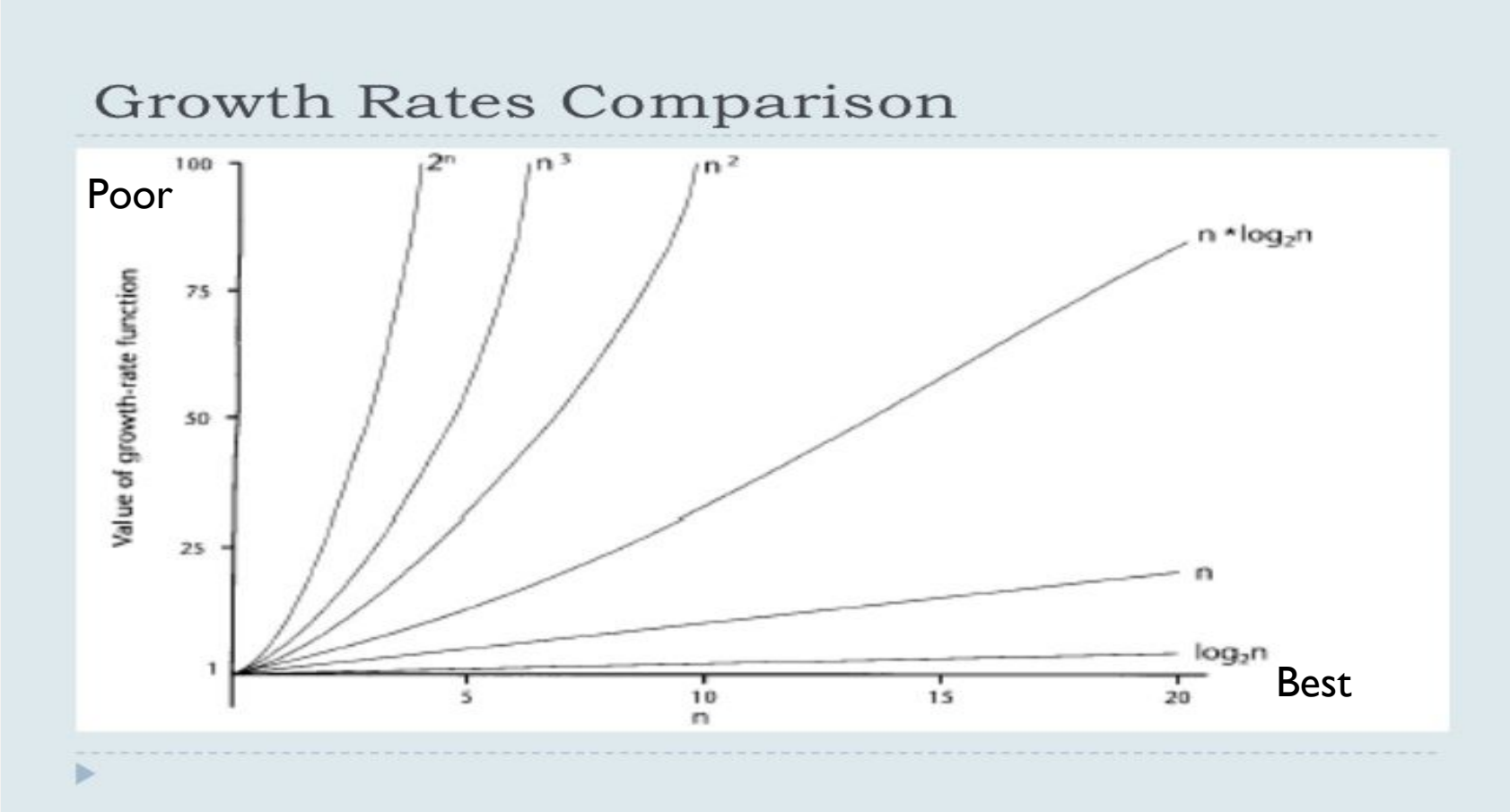
Example:

F(n) = n^2 + 100n + 5 ------- n^2

F(n) = n^3 + 5n ------- n^3 (Slow)

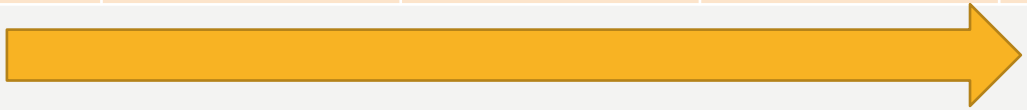
In above equation we consider only the leading terms.

RATE OF GROWTH OF ALGORITHM



- If rate of growth is high ☐ algorithm slows
- The growth rate of an algorithm is the rate of running time (Cost of algorithm grows as the size of the input grows).

N!	2^n	n^3	n^2	nlog2 N	N	Log2 N
64!	2^64	64^3	64^2	64 log2 64	64	Log2 64 = 8



Rate of growth

ALGORITHM DESIGN TECHNIQUES

- Divide and Conquer Technique
  - Binary Search
  - Quick Sort
  - Merge Sort
  - Heap Sort
  - V. Stressens Matrix Multiplication
- Greedy Method
  - Knapsack Problem
  - Fractional Knapsack Problem
  - Minimum Cost Spanning Tree
    - Kruskal Algorithm
    - Prim's Algorithm
  - Single Source Shortest Path
    - Dijkstra's Algorithm
    - Bellmonford Algorithm

- Dynamic Programing
  - Longest Common Subsequence
  - 0/1 Knapsack Problem
  - All pair shortest path algorithm
    - Floyd Warshell's Algorithm
  - Optimal Binary Search Tree
- Backtracking
  - N-Queen's Problem
  - Graph Coloring Problem
  - Travelling Salesman Problem
- Branch and Bound
  - Assignment Problem
  - Travelling Salesman Problem

ASYMPTOTIC NOTATIONS

- Asymptotic Notations are used to represent the complexity of an algorithm.
- With the help of asymptotic notations we can analyze runtime performance of algorithm.
- Asymptotic notations are used to describe the asymptotic (approximate) running time of an algorithm by defining a time function f(n) where n is the input size.
- Usually, the time required by an an algorithm falls under 3 types:
  - Best case: Minimum time required for program execution
  - Average case: Average time required for program execution
  - Worst case: Worst time required for program execution
- With the help of asymptotic notation, we can compare the performance of various algorithms. And such an analysis is independent of machine time, programing style etc.

ASYMPTOTIC NOTATIONS

- Asymptotic notations are the mathematical notations used to describe the running time of an algorithm when the input tends towards a particular value or a limiting value.
- For example: In bubble sort, when the input array is already sorted, the time taken by the algorithm is linear i.e. the **best case**.
- But, when the input array is in reverse condition, the algorithm takes the maximum time (quadratic) to sort the elements i.e. the **worst case**.
- When the input array is neither sorted nor in reverse order, then it takes **average time**. These durations are denoted using asymptotic notations.

ASYMPTOTIC NOTATIONS TYPES

- Big-O notation: Worst Case Upper bound
- Omega notation: Best Case Lower bound
- Theta notation: Average Case Tight bound

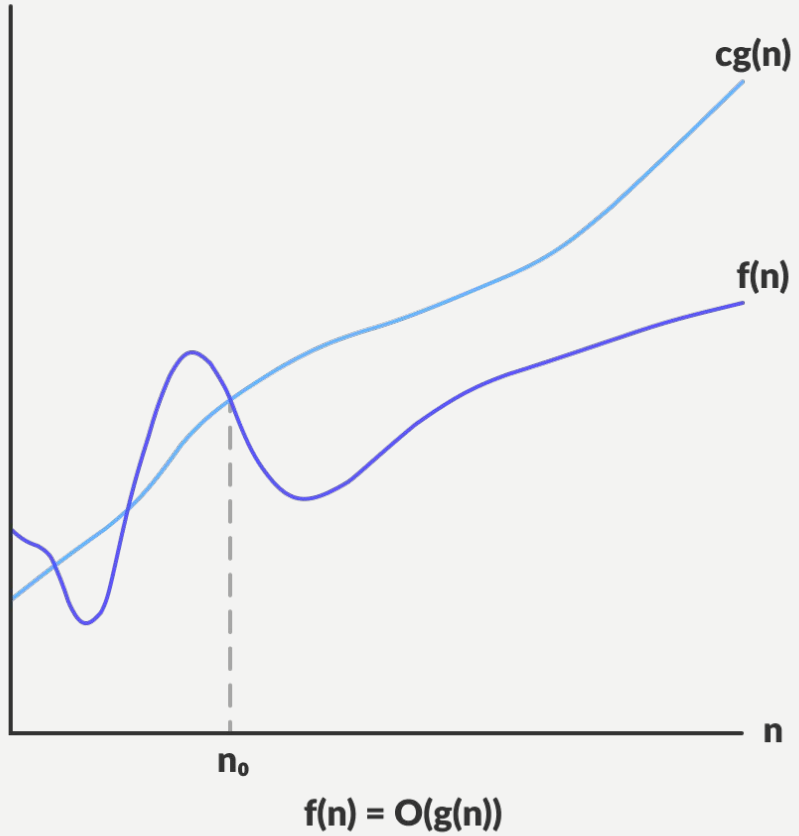
BIG O NOTATION

Big-O notation represents the upper bound of the running time of an algorithm i.e. longest amount of time. Thus, it gives the worst-case complexity of an algorithm.

A function f(n) = O g(n), if there exist a value of positive integers n & n_0 ,positive constant C such that

$$0 \leq f(n) \leq cg(n) \text{ for all } n \geq n_0$$

Here function g(n) is an upper bound for function f(n), as g(n) grows faster than f(n).



EXAMPLE

Example

$f(n) = 2n^2 + 5n + 1$

$g(n) = n^2$, $c = 8$

$\Rightarrow f(n) \leq c\, g(n)$

$2n^2 + 5n + 1 \leq 8n^2$ (if $n = 1$)

$8 \leq 8$

$\therefore f(n) = O(n^2)$ $\forall\, n \geq 1$

n^2 is upper bound of for function

Example

$f(n) = 2n^2 + 5n + 1$

$g(n) = n^2$ $c = 4$

$f(n) \leq c \cdot g(n)$

$2n^2 + 5n + 1 \leq 4 \cdot n^2$

$2 + 5 + 1 \leq 4$ ($n = 1$)

$8 \leq 4$ false

EXAMPLE

$2 \times 4 + 5 \times 2 + 1 \leq 4 \times 4 \quad (n = 2)$

$19 \leq 16 \Rightarrow$ false

$2 \times 9 + 5 \times 3 + 1 \leq 4 \times 9 \quad (n = 3)$

$18 + 15 + \leq 36$

$34 \leq 36 =$ True

$\therefore f(n) = O(n^2) \quad \forall \quad n \geq 3$

Q $f(n) = 2n^2 + 5n + 1$

$g(n) = n^3 \quad , c = 4$

$2n^2 + 5n + 1 \leq 4n^3$

$n = 1$

$2 + 5 + 1 \leq 4$

$8 \leq 4 \Rightarrow$ false

$n = 2$

$2 \times 4 + 10 + 1 \leq 4 \times 8$

$19 \leq 32$

$\Rightarrow$ True

$\therefore f(n) = O(n^3) \quad \forall \quad n \geq 2$

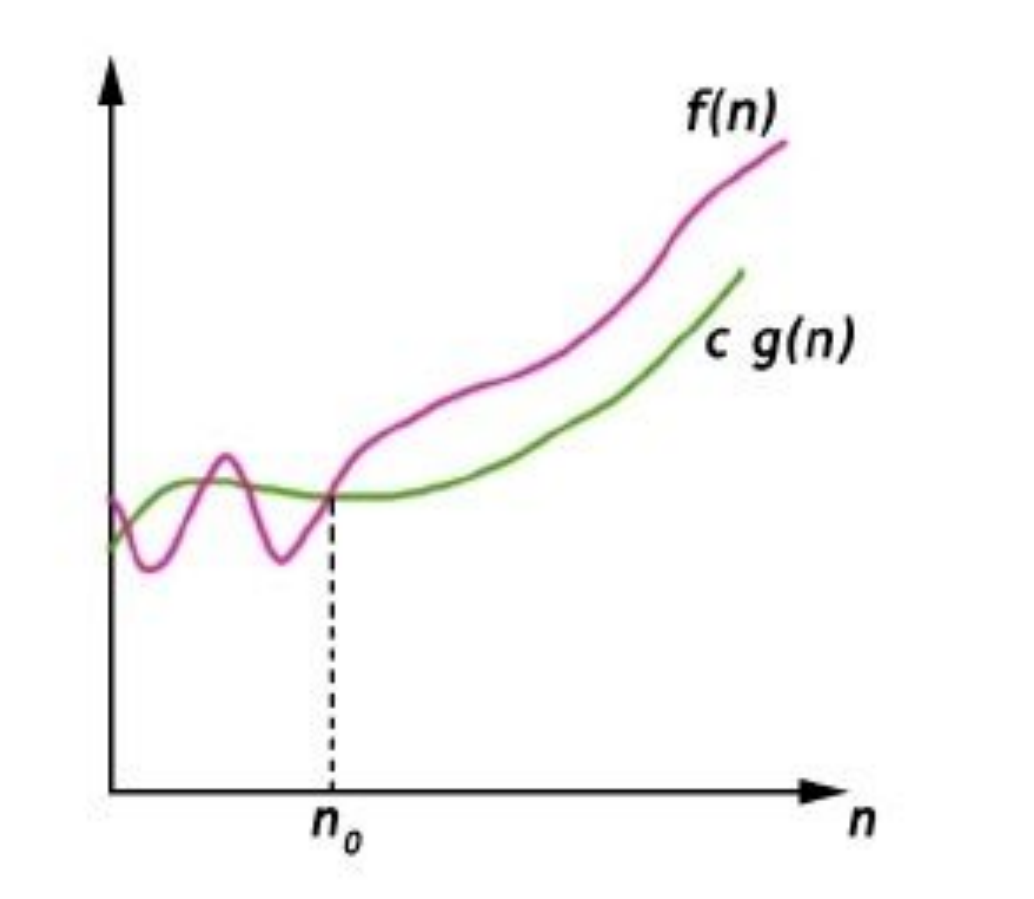
$\therefore f(n) = O(n^2)$ Sol

OMEGA NOTATION

- Big-Omega (Ω) notation gives a lower bound for a function f(n) to within a constant factor.
- We write f(n) = Ω(g(n)), If there are positive constants n0 and c such that, to the right of n_0 the f(n) always lies on or above c*g(n).
- A function f(n) = Ω(g(n), if there exist a function f(n) positive integers n & n_0 ,positive constant C such that

  c*g(n) <= f(n) for all n >= n0

Hence function g(n) is asymptotic lower bound of f(n)



EXAMPLE

Eg: $f(n) = 3n+2$
$g(n) = n$

$f(n) \geq c \cdot g(n)$
$3n+2 \geq n \qquad c=1 \quad n=1$
$5 \geq 1$
$\therefore f(n) = \Omega(n) \qquad \forall\ n \geq 1$

Eg: $f(n) = 2n^2 + 5(n) + 1$
$g(n) = n^2$

$f(n) \geq c \cdot g(n)$
$2n^2 + 5n + 1 \geq cn^2$

Let $c=2$,
$2n^2 + 5n + 1 \geq 2n^2$

$n=1 \qquad 2+5+1 \geq 2$
$8 \geq 2 \Rightarrow$ True

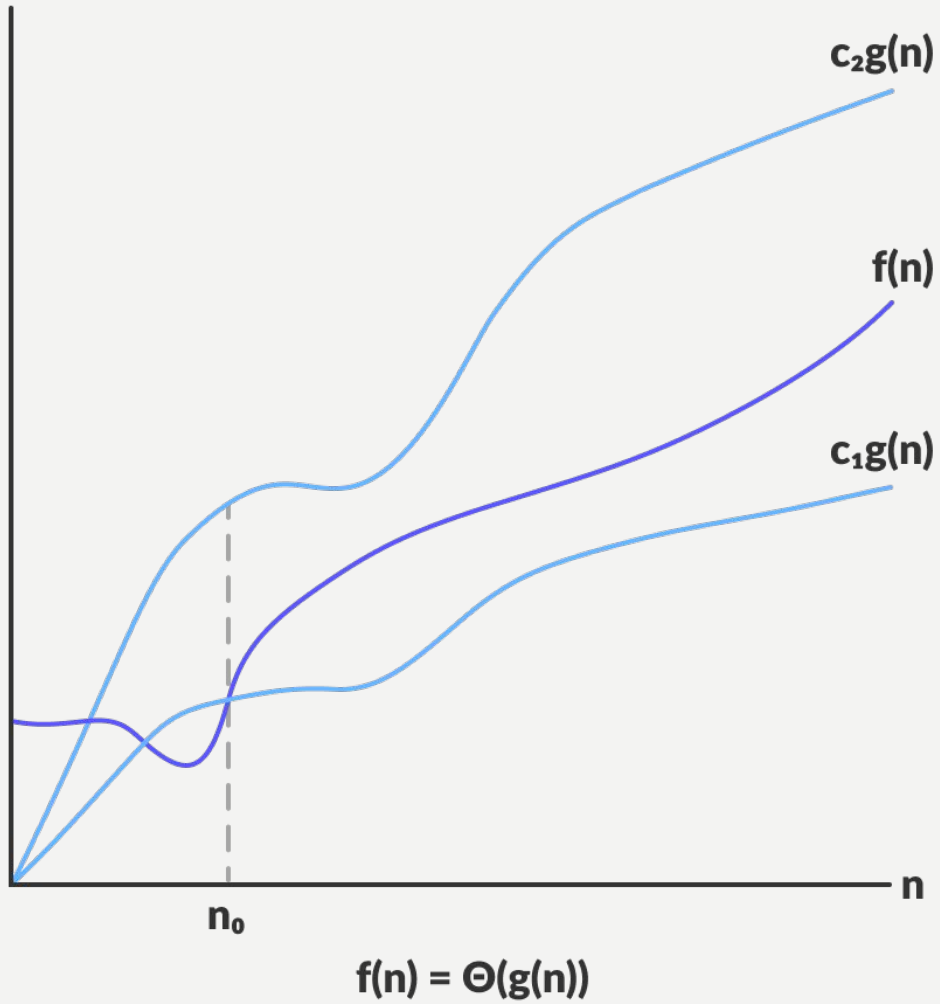
$\therefore f(n) = \Omega(g(n)) = \Omega(n^2)$
$\forall\ n \geq 1$

THETA NOTATION

- It represents the average case of an algorithm time complexity.
- It denote the asymptotically tight bound i.e. lower bound and upper bound, of an algorithm's running time.
- A function f(n) = O g(n), if there exist a value of positive integers n & n_0 ,positive constant C1 and C2 such that

$$c1g(n) \leq f(n) \leq c2g(n)$$

- Hence function g(n) is asymptotic lower bound of f(n)



EXAMPLE

Eg: $f(n) = 3n + 2 \qquad g(n) = n$

$c_1 g(n) \leq f(n) \leq c_2 g(n)$

Let $c_1 = 1, c_2 = 4$

$\Rightarrow 1 \cdot n \leq 3n + 2 \leq 4n$

$n = 1, \quad 1 \leq 5 \leq 4 \Rightarrow$ false

$n = 2, \quad 2 \leq 8 \leq 8 \Rightarrow$ True

So $f(n) = O(n)$ if $n \geq 2, c_1 = 1, c_2 = 4$

THANKS!