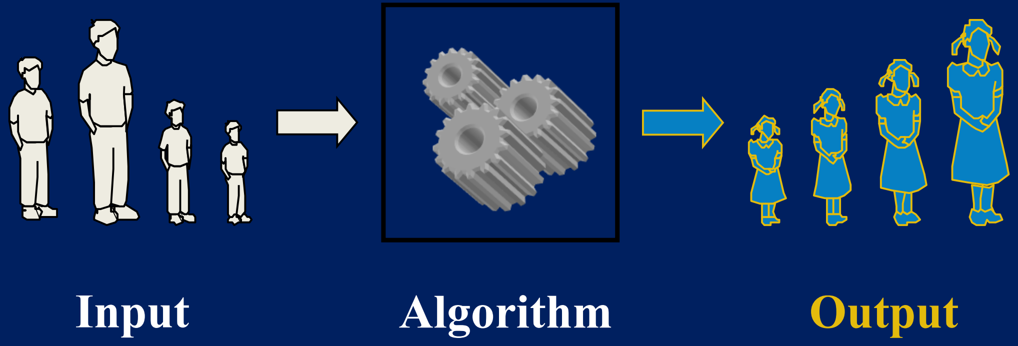
DESIGN & ANALYSIS OF ALGORITHM (BCSC0012)

Chapter 1: Introduction



Prof. Anand Singh Jalal

Department of Computer Engineering & Applications

Course Objectives

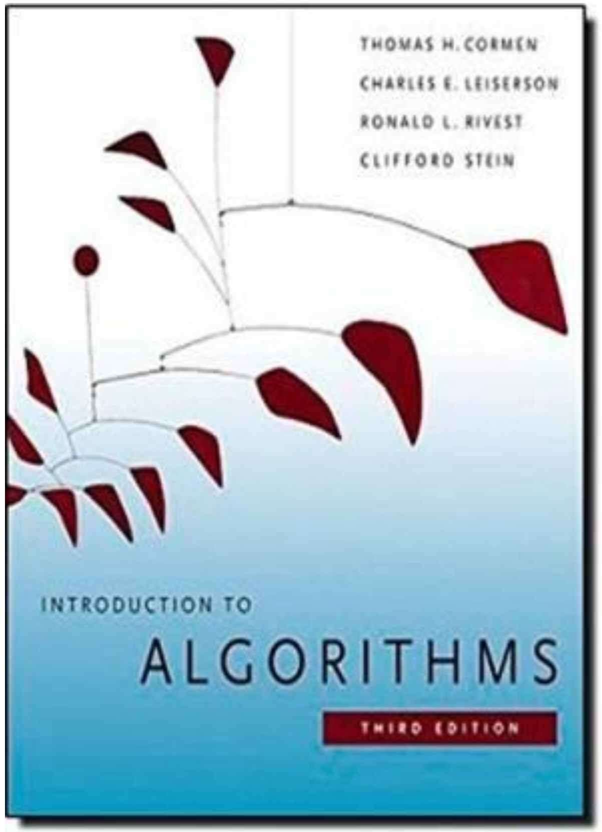
1. Reinforce basic design concepts (e.g., pseudocode, specifications, top-down design)

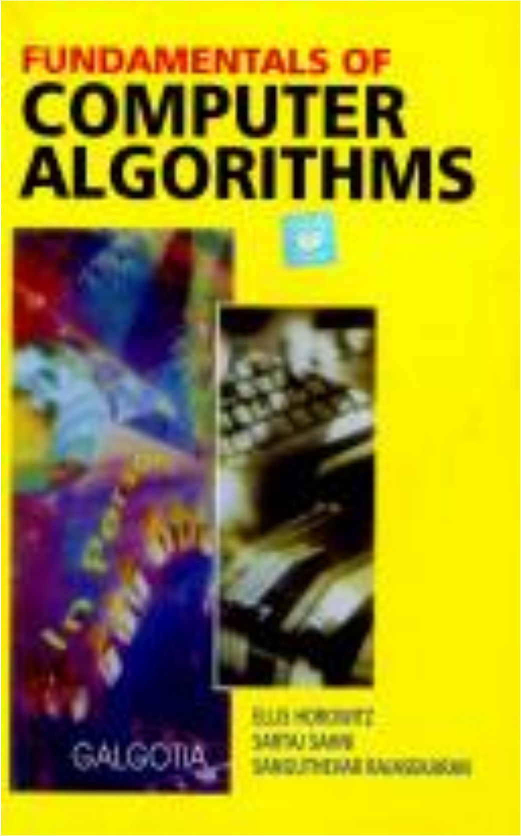
2. Knowledge of algorithm design strategies

3. Familiarity with an assortment of important algorithms

4. Ability to analyze time and space complexity

Algorithm: Books





T. H. Cormen, C. E. Leiserson, R. L. Rivest and C. Stein, “Introduction to Algorithms”

E.Horowitz and Shani “Fundamentals of Computer Algorithms”

Algorithm: Introduction

An **algorithm** is a step-by-step procedure for solving a problem in a finite amount of time.

https://study.com/academy/lesson/what-is-an-algorithm-definition-examples.html

Algorithm: Introduction ...

- An algorithm is a sequence of computational steps that transform input to desired output.

(Thomas H. Coreman)

- An algorithm is a finite set of instructions that, if followed, accomplishes a particular task

(Sartaj Sahni)

Algorithm: Introduction …

Difference between Algorithm and Program

Algorithm	Program
Written at Design stage	Written at implementation stage
Need domain expert	Need programmer
Written in any language	Written in programming language
H/W or OS independent	H/W or OS dependent
Can be Analyze	Can be tested

Algorithm: Introduction ...

Criteria (properties) of an algorithm

- **All algorithms must satisfy the following five criteria:**
    - **Input** – zero or more quantities are externally supplied
    - **Output** – Atleast one quantity is produced
    - **Definiteness** – Each instruction should be clear and unambiguous
    - **Finiteness** – The algorithm should terminate after finite number of steps
    - **Effectiveness** – Every instruction must be very basic. It is also always feasible.

Pseudo-Code

- Pseudo-code is a description of an algorithm that is more structured than usual prose but less formal than a programming language.
- **Example: swapping two numbers.**

```
Algorithm SWAP (a, b)
Begin
    Temp :=a;
     a:=b;
     b:=temp;
End
```

- Pseudo-code is our preferred notation for describing algorithms.
- However, pseudo-code hides program design issues.

What is Pseudo-Code ?

- A mixture of natural language and high-level programming concepts that describes the main ideas behind a generic implementation of a data structure or algorithm.

-Expressions: use standard mathematical symbols to describe numeric and boolean expressions -use ← for assignment (“=” in Java)

-use **=** for the equality relationship (“==” in Java)

-Method Declarations: -**Algorithm** name(***param1, param2***)

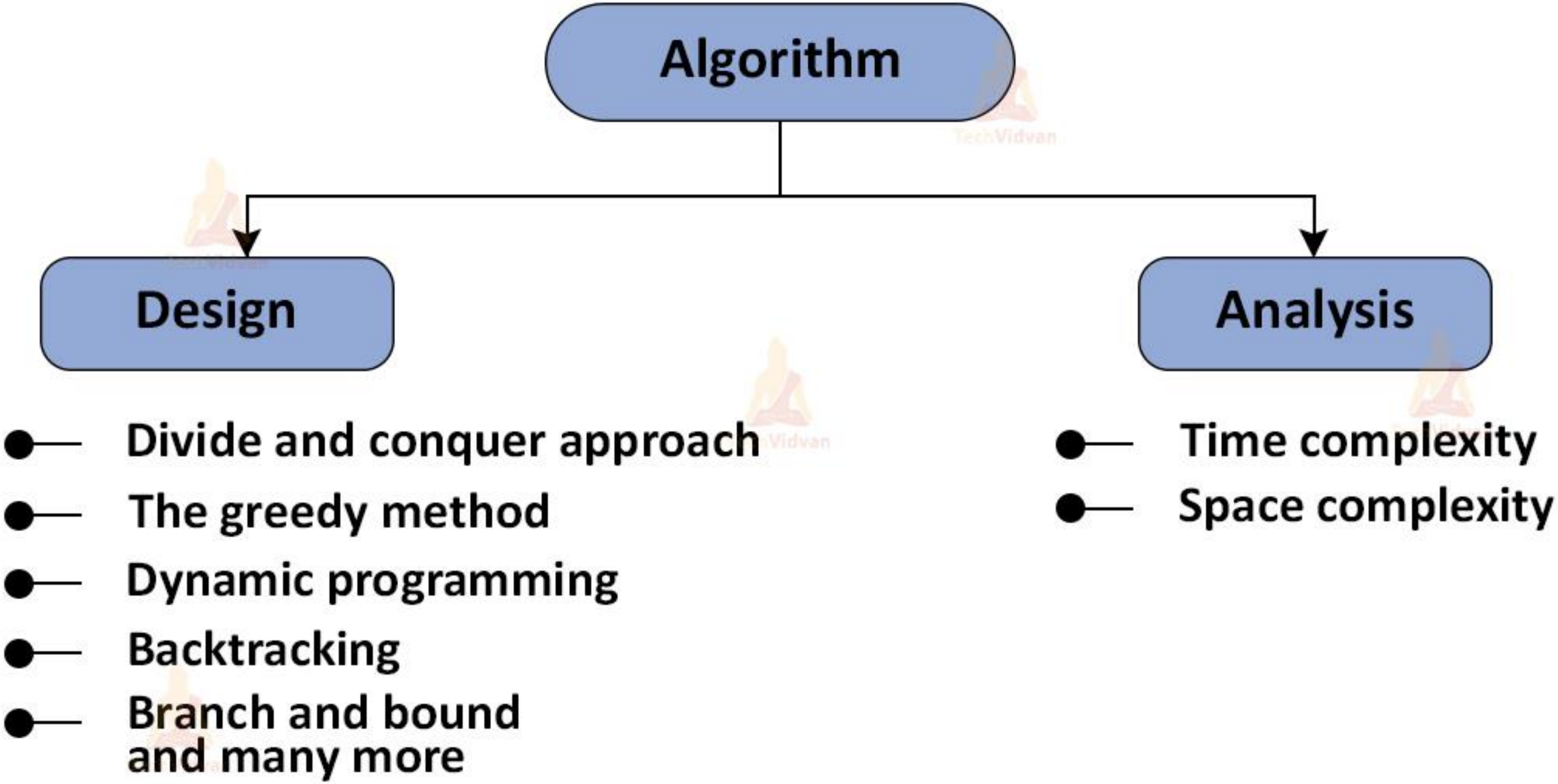
-Programming Constructs: - decision structures: **if ... then ... [else ...]**

- while-loops: **while ... do**
- repeat-loops: **repeat ... until ...**
- for-loop: **for ... do**
- array indexing: **A[i]**

-Methods: - calls: object method(args)

- returns: **return** value

Two Aspects of Algorithm



Algorithm

Design

- Divide and conquer approach
- The greedy method
- Dynamic programming
- Backtracking
- Branch and bound and many more

Analysis

- Time complexity
- Space complexity

https://techvidvan.com/tutorials/data-structure-and-algorithm/

Analysis of Algorithms

- **Primitive Operations:** Low-level computations independent from the programming language can be identified in pseudocode.
- Examples:
  - calling a method and returning from a method
  - arithmetic operations (e.g. addition)
  - comparing two numbers, etc.
- **By inspecting the pseudo-code, we can count the number of primitive operations executed by an algorithm.**

Algorithm: Introduction ...

How to Analyze an Algorithm

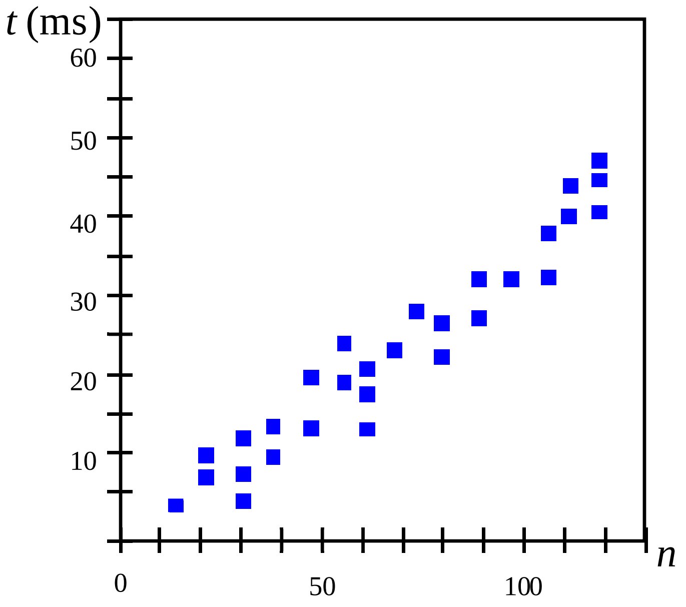
1. Time
2. Space
3. N/W
4. Power
5. CPU

Time complexity of an algorithm is the amount of computer time it needs to run the completion.

Space Complexity is an amount of memory requirement for an algorithm to completion of its execution.

Measuring the Running Time

- How should we measure the running time of an algorithm?
- **Approach 1: Experimental Study**
    - Write a program that implements the algorithm
    - Run the program with data sets of varying size and composition.
    - Use a method like System.currentTimeMillis() to get an accurate measure of the actual running time.



Limitations of Experimental Studies

- Experimental studies have several limitations:
  - It is necessary to implement and test the algorithm in order to determine its running time.
  - Experiments can be done only on a limited set of inputs, and may not be indicative of the running time on other inputs not included in the experiment.
  - In order to compare two algorithms, the same hardware and software environments should be used.

Beyond Experimental Studies

- We will now develop a general methodology for analyzing the running time of algorithms. In contrast to the "experimental approach", this methodology:
  - Uses a high-level description of the algorithm instead of testing one of its implementations.
  - Takes into account all possible inputs.
  - Allows one to evaluate the efficiency of any algorithm in a way that is independent from the hardware and software environment.

GLA UNIVERSITY MATHURA
Recognised by UGC Under Section 2(f)
Accredited with A Grade by NAAC

Algorithm: Introduction ...

Frequency Count Method

- **Ex 1) Algorithm for sum of elements of an array**

Algorithm_Sum(A, n)

```
//A is an array of size n
1.      {
2.          S=0;
3.    for (i=0; i<n; i++)
4.    {
5.          S=S+A[i]
6.    }
7.    return S;
8. }
```

Time Analysis	
Instruction	**Time Unit**
2	1
3	n+1
5	n
7	1
Total	**2n+3**
Degree	**O(n)**

Space Analysis	
Variable	**Space Unit**
A	n
n	1
S	1
i	1
Total	**n+3**
Degree	**O(n)**

Algorithm: Introduction ...

Frequency Count Method

- **Ex 2) Algorithm for sum of two matrix**

Algorithm_Sum_MAT(A, B, n)

```
//A, B is an array of size
1. {
2. for (i=0; i<n; i++)
3. {
4.  for (j=0; j<n; j++)
5.  {
6. C[i,j]=A[i,j]+B[i,j]
7.    }
8. }
```

Time Analysis	
Instruction	**Time Unit**
2	n+1
4	n(n+1)
6	n*n
Total	**$2n^2+2n+1$**
Degree	**$O(n^2)$**

Space Analysis	
Variable	**Space Unit**
A	n^2
B	n^2
C	n^2
n	1
i	1
J	1
Total	**$3n^2+3$**
Degree	**$O(n^2)$**

Algorithm: Introduction ...

Frequency Count Method

Ex 3) Algorithm for multiplication of two matrix

```
Algorithm_Mul_MAT(A, B, n)
//A is an array of size n
1.  {
2.   for (i=0; i<n; i++)
3.  {
4.    for (j=0; j<n; j++)
5.    C[I,j]=0;
6.     for (k=0; k<n; k++)
7.     {
8.    C[i,j]=C[i,j] + A[i,k]*B[k,j]
9.       }
10. }
11. }
```

Time Analysis

Instruction	Time Unit
2	$n+1$
4	$n(n+1)$
5	$n*n$
6	$(n+1)n*n$
8	$n*n*n$
Total	**$2n^3+3n^2+2n$**
Degree	**$O(n^3)$**

Space Analysis

Variable	Space Unit
A	n^2
B	n^2
C	n^2
n	1
i	1
j	1
k	1
Total	**$3n^2+4$**
Degree	**$O(n^2)$**

Algorithm: Introduction ...

Frequency Count Method

Ex 4)

```
1. for (i=0; i<n; i++)
2.  {
3.          Stat;
4.         }
```

Time Analysis	
Instruction	**Time Unit**
1	n+1
3	n
Total	**2n+1**
Degree	**O(n)**

Algorithm: Introduction ...

Frequency Count Method

Ex 5)

```
1. for (i=n; i>0; i--)
2.  {
3.      Stat;
4.     }
```

Time Analysis	
Instruction	**Time Unit**
1	n+1
3	n
Total	**2n+1**
Degree	**O(n)**

Algorithm: Introduction ...

Frequency Count Method

Ex 6)

```
1. for (i=0; i<n; i=i+2)
2.  {
3.    Statement;
4.       }
```

Time Analysis	
Instruction	**Time Unit**
1	n/2+1
3	n/2
Total	**n+1**
Degree	**O(n)**

Algorithm: Introduction …

Ex 7)

```
for (i=0; i<n; i++)
 {
     for (j=0; j<i; j++)
     {
   Statement;
       }
   }
```

$$\text{Total Time}=1+2+3+\ldots\ldots.+n=\frac{n(n+1)}{n}$$

$$f(n)=O(n^2)$$

Time Analysis		
i	**j**	**No. of Time**
0	0	0
1	0	1
	1	
2	0	2
	1	
	2	
-		
-		
n		

Algorithm: Introduction ...

Ex 8)

```
P=0;
for (i=1; P<=n; i++)
 {
         P=P+i;
  }
```

Time Analysis	
i	**P**
1	0+1=1
2	1+2=3
3	1+2+3
4	1+2+3+4
.	
.	
.	
k	1+2+3…..+k

Assume that the loop will terminate when P>n

$$P=\frac{k(k+1)}{2} \quad => \quad \frac{k(k+1)}{2} > n \quad => k^2 > n$$

$$k > \sqrt{n} \qquad => \quad f(n) = O(\sqrt{n})$$

Algorithm: Introduction ...

Ex 9)

```
for (i=1; i<n; i=i*2)
 {
           Statement;
 }
```

Assume that the loop will terminate when i>=n

$$2^k >= n \quad => \quad 2^k = n \quad => \quad \mathrm{k{=}log}_2 n$$

$$f(n) = O(\log_2 n)$$

Time Analysis

iteration	i
1	1
2	2
3	2^2
4	2^3
.	
.	
.	
k-1	2^k

Algorithm: Introduction ...

Ex 10)

```
for (i=n; i>=1; i=i/2)
 {
        Statement;
      }
```

Assume that the loop will terminate when i<1

$$n/2^k < 1 \quad => \quad n/2^k = 1 \quad => \quad n = 2^k$$

$$k = \log_2 n \quad => \quad f(n) = O(\log_2 n)$$

Time Analysis

iteration	i
1	n
2	n/2
3	$n/2^2$
4	$n/2^3$
.	
.	
.	
k-1	$n/2^k$

Algorithm: Introduction ...

Ex 11)

```
1.for (i=0; i<n; i++)
2.   {
3.          Statement;
4.        }
5.for (j=0; j<n; j++)
6.   {
7.          Statement;
8.        }
```

Time Analysis	
Instruction	**Time Unit**
1	n+1
3	n
5	n+1
7	n
Total	**4n+2**
Degree f(n)	**O(n)**

Algorithm: Introduction ...

Ex12)

```
P=0;
for (i=1; i<n; i=i*2)
  {
      P++;
    }
for (j=1; j<P; j=j*2)
  {
      Statement;
    }
```

$P=\log_2 n$

$\log_2 P$

$$f(n)=O(\log\log_2 n)$$

Algorithm: Introduction ...

Ex 13)

```
for (i=1; i<n; i++)
 {
      for (j=1; j<n; j=j*2)
   {
        Statement;
       }
```

$n+1$

$n*\log_2 n$

$n*\log_2 n$

$$\text{Total Time}=2n\log_2 n+n$$

$$f(n)=O(n\log_2 n)$$

Algorithm: Introduction ...

Ex 14) While loop

```
i=0;                          1
while (i<n)                   n
 {
   Statement;                 n
   i++;                       n
 }
```

$$\text{Total Time} = 3n + 2$$

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

While loop and for loop are similar.

Do while is different from these two. As it execute at lest one time even the condition is false.

Algorithm: Introduction ...

Ex 15) While loop

```
a=1;
while (a<b)
 {
   Statement;
   a=a*2
   }
```

Value of a
1
1*2=2
$2*2=2^2$
$2^2*2=2^3$
2^k

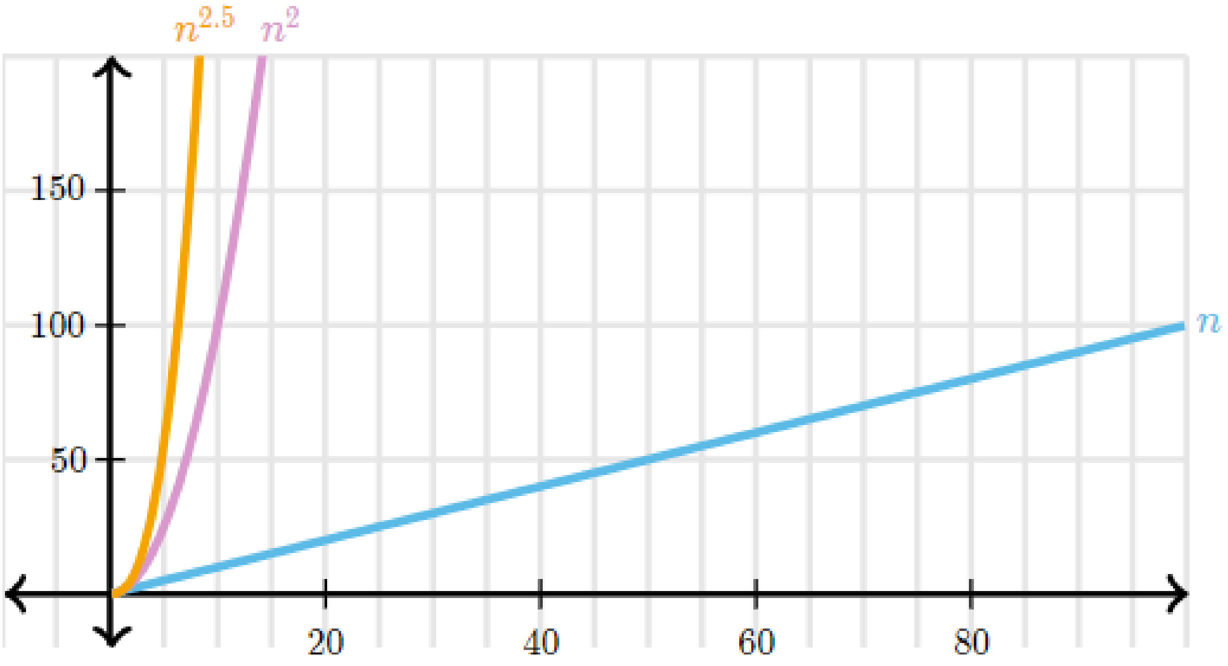
the loop will terminate when a ≥ b

$$\because a=2^k \quad \Rightarrow \quad \therefore 2^k \geq b \quad \Rightarrow$$

$$2^k = b \quad \Rightarrow \quad k=\log_2 b \quad \Rightarrow \quad f(n) = O(\log_2 n)$$

Algorithm: Growth of a Function

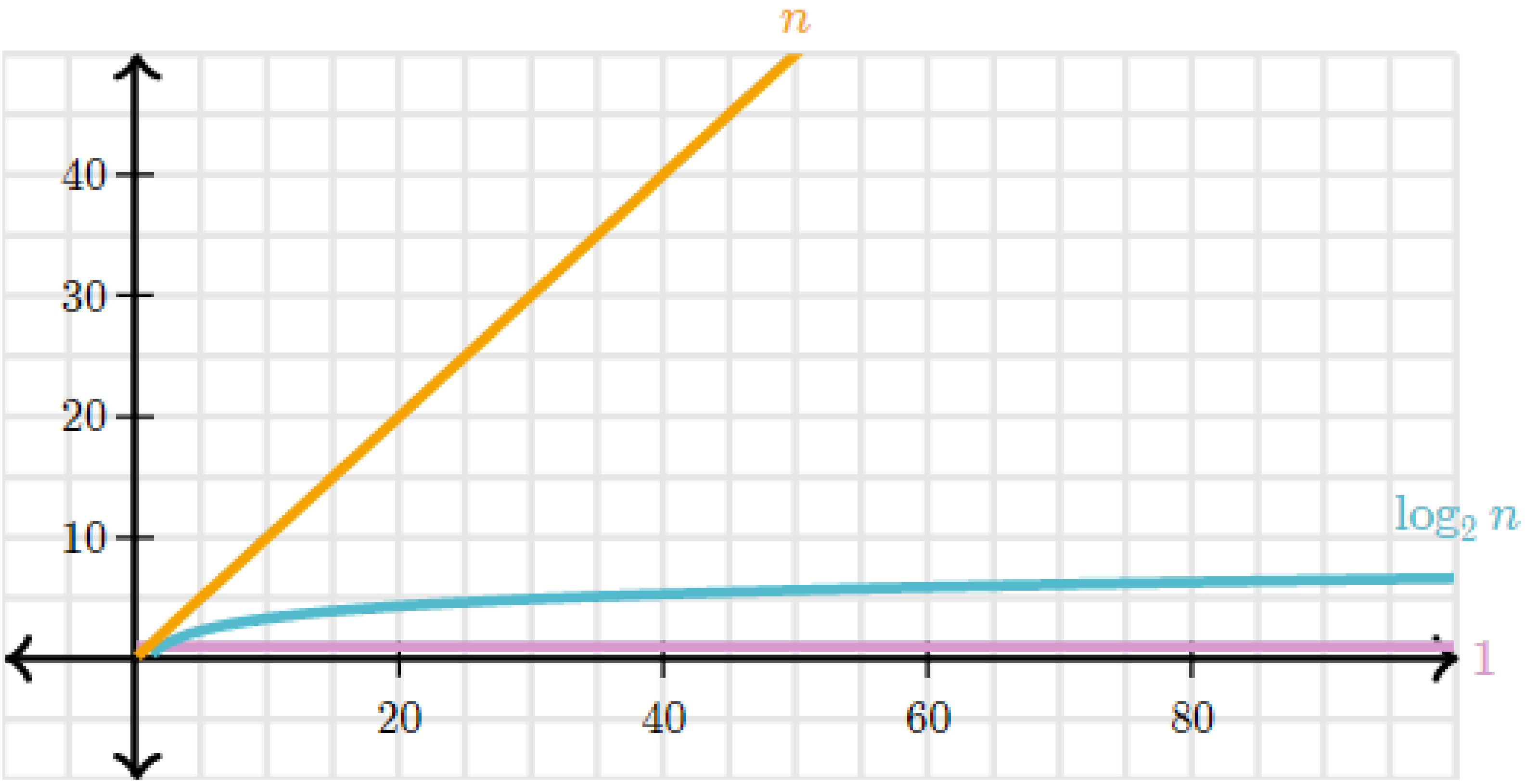
The following graph compares the growth of n,n^2, and $n^{2.5}$:



Logarithms grow more slowly than polynomials. That is, $\Theta(\log_2 n)$ grows more slowly than $\Theta(n^a)$ for *any* positive constant a. But since the value of $\log_2 n$ increases as n increases, $\Theta(\log_2 n)$ grows faster than $\Theta(1)$.

Algorithm: Growth of a Function …

The following graph compares the growth of 1, n, and $\log_2 n$:



Algorithm: Growth of a Function ...

$$1 < \log_2 n < \sqrt{n} < n < n\log_2 n < n^2 < n^3 < \ldots\ldots.. < 2^n < 3^n \ldots < n^n$$

$O(1)$	Constant
$O(\log_2 n)$	Logrithemic
$O(n)$	Linear
$O(n^2)$	Quadratic
$O(n^3)$	Cubic
$O(2^n)$	Exponential

n	$\log_2 n$	n^2	2^n
1	0	1	2
2	1	4	4
4	2	16	16
8	3	64	256

Algorithm: Introduction …

General steps to develop an Algorithm

There are four steps:

- To devise an algorithm
- Validate an algorithm
- Analyze an algorithm
- Test an algorithm
  - ✓ Debugging
  - ✓ Profiling

Review on Summations

Why do we need summation formulas?

- For computing the running times of iterative constructs

- **Constant Series:** For integers *a* and *b*, *a* ≤ *b*,

$$\sum_{i=a}^{b} 1 = b - a + 1$$

- **Linear Series (Arithmetic Series):** For *n* ≥ 0,

$$\sum_{i=1}^{n} i = 1 + 2 + \cdots + n = \frac{n(n+1)}{2}$$

- **Quadratic Series:** For *n* ≥ 0,

$$\sum_{i=1}^{n} i^2 = 1^2 + 2^2 + \cdots + n^2 = \frac{n(n+1)(2n+1)}{6}$$

Review on Summations ...

- **Cubic Series:** For $n \geq 0$,

$$\sum_{i=1}^{n} i^3 = 1^3 + 2^3 + \cdots + n^3 = \frac{n^2(n+1)^2}{4}$$

- **Geometric Series:** For real $x \neq 1$,

$$\sum_{k=0}^{n} x^k = 1 + x + x^2 + \cdots + x^n = \frac{x^{n+1} - 1}{x - 1}$$

For $|x| < 1$,

$$\sum_{k=0}^{\infty} x^k = \frac{1}{1-x}$$

Review on Summations ...

- **Linear-Geometric Series:** For $n \geq 0$, real $c \neq 1$,

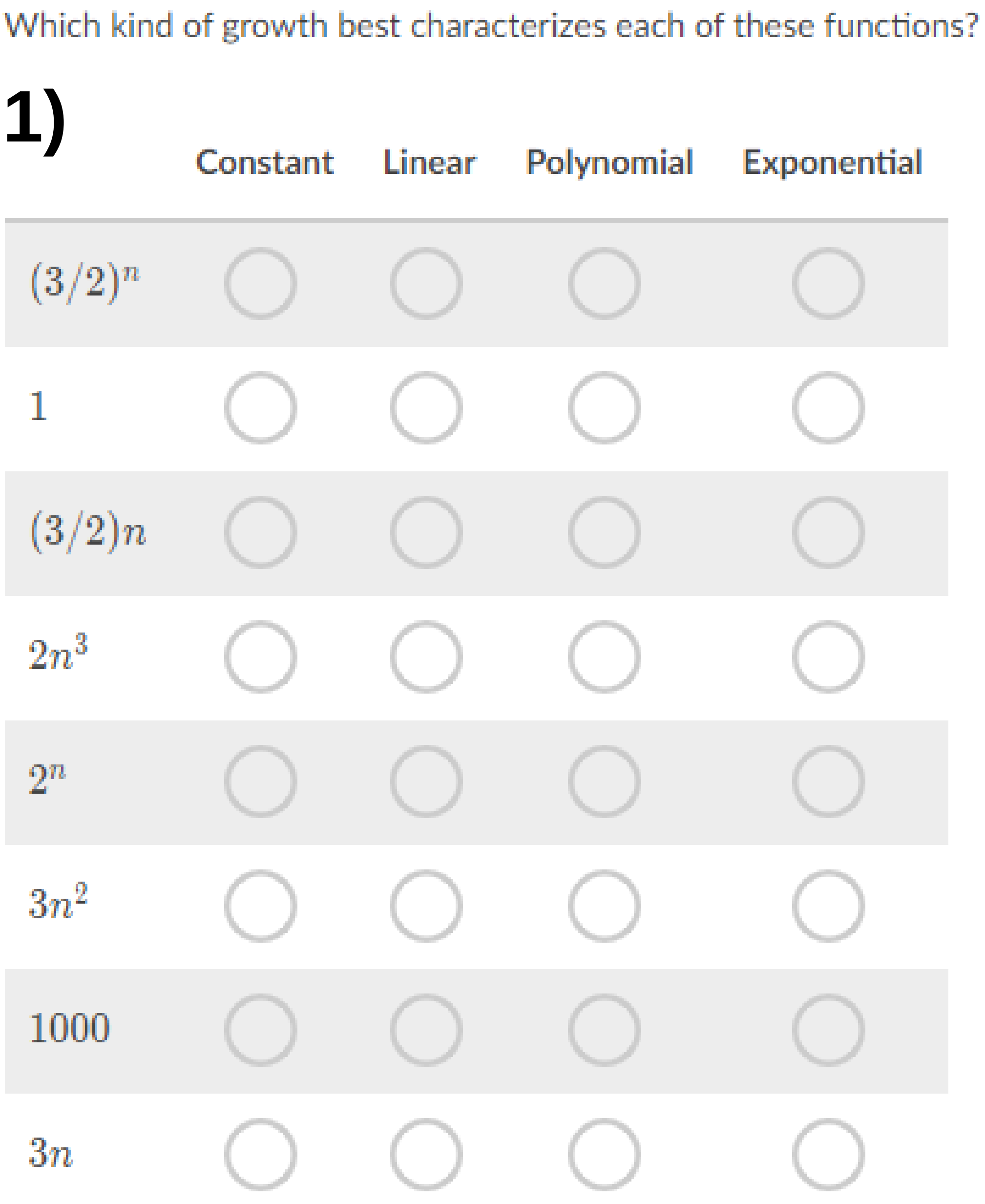
$$\sum_{i=1}^{n} ic^{i} = c + 2c^{2} + \cdots + nc^{n} = \frac{-(n+1)c^{n+1} + nc^{n+2} + c}{(c-1)^{2}}$$

- **Harmonic Series:** *n*th harmonic number, $n \in \mathrm{I}^{+}$,

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

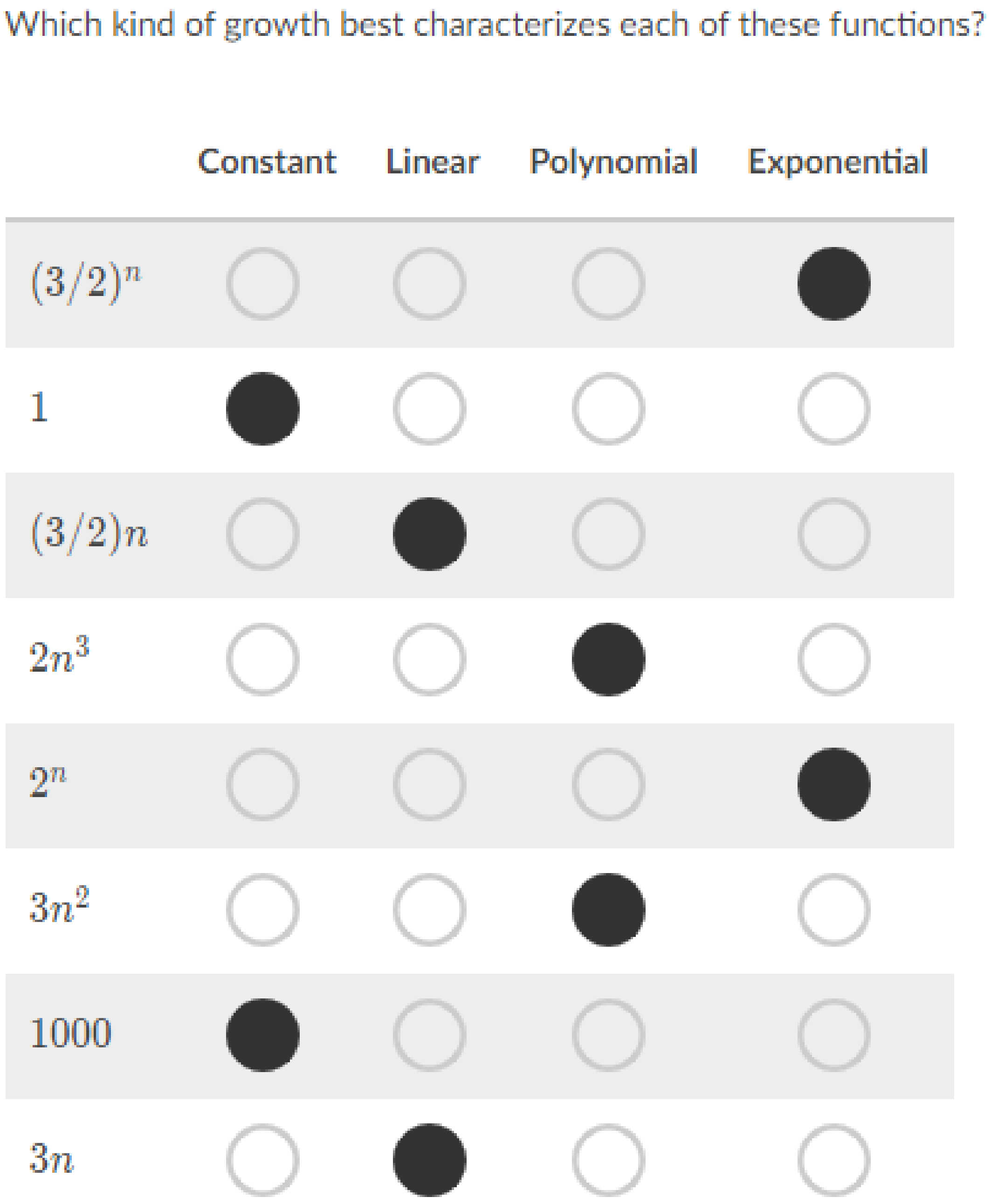
$$= \sum_{k=1}^{n} \frac{1}{k} = \ln(n) + O(1)$$

Q1)



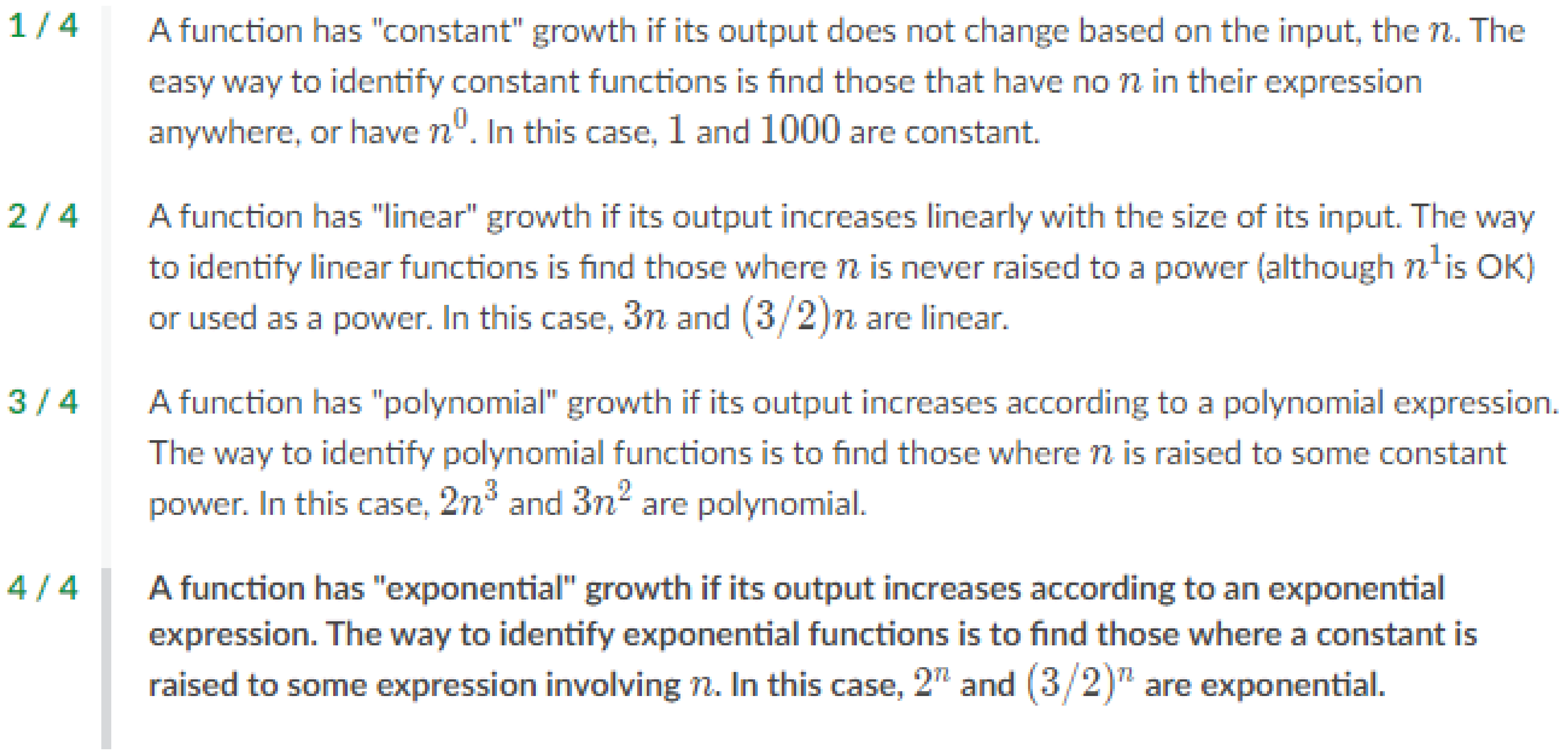
Which kind of growth best characterizes each of these functions?

	Constant	Linear	Polynomial	Exponential
$(3/2)^n$	○	○	○	○
1	○	○	○	○
$(3/2)n$	○	○	○	○
$2n^3$	○	○	○	○
2^n	○	○	○	○
$3n^2$	○	○	○	○
1000	○	○	○	○
$3n$	○	○	○	○



Which kind of growth best characterizes each of these functions?

	Constant	Linear	Polynomial	Exponential
$(3/2)^n$	○	○	○	●
1	●	○	○	○
$(3/2)n$	○	●	○	○
$2n^3$	○	○	●	○
2^n	○	○	○	●
$3n^2$	○	○	●	○
1000	●	○	○	○
$3n$	○	●	○	○



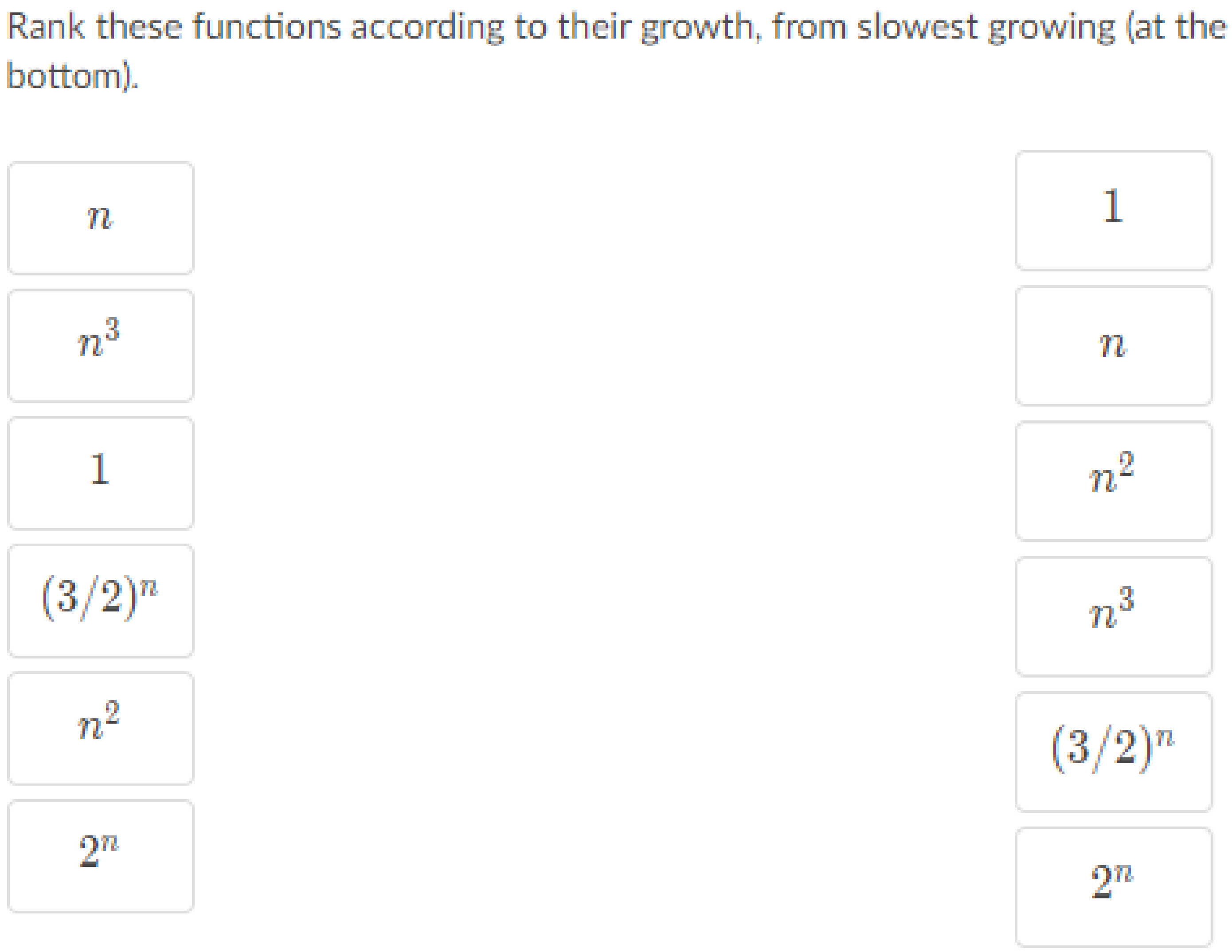
1 / 4 A function has "constant" growth if its output does not change based on the input, the n. The easy way to identify constant functions is find those that have no n in their expression anywhere, or have n^0. In this case, 1 and 1000 are constant.

2 / 4 A function has "linear" growth if its output increases linearly with the size of its input. The way to identify linear functions is find those where n is never raised to a power (although $n^1$is OK) or used as a power. In this case, $3n$ and $(3/2)n$ are linear.

3 / 4 A function has "polynomial" growth if its output increases according to a polynomial expression. The way to identify polynomial functions is to find those where n is raised to some constant power. In this case, $2n^3$ and $3n^2$ are polynomial.

4 / 4 **A function has "exponential" growth if its output increases according to an exponential expression. The way to identify exponential functions is to find those where a constant is raised to some expression involving n. In this case, 2^n and $(3/2)^n$ are exponential.**

Q2)



Rank these functions according to their growth, from slowest growing (at the top) to fastest growing (at the bottom).

n

n^3

1

$(3/2)^n$

n^2

2^n

1

n

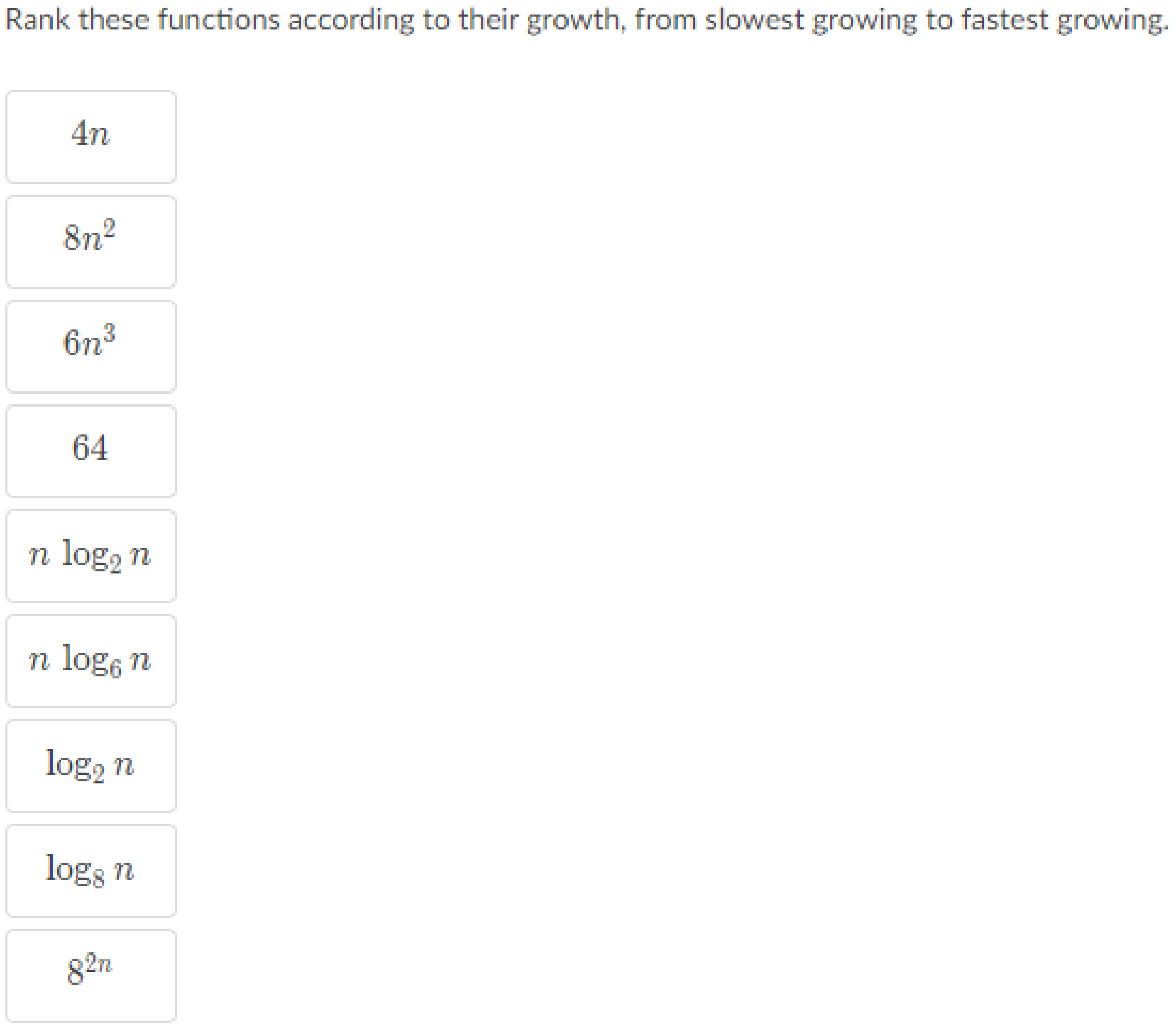
n^2

n^3

$(3/2)^n$

2^n

Q3)



Rank these functions according to their growth, from slowest growing to fastest growing.

- $4n$
- $8n^2$
- $6n^3$
- 64
- $n \log_2 n$
- $n \log_6 n$
- $\log_2 n$
- $\log_8 n$
- 8^{2n}

- 64
- $\log_8 n$
- $\log_2 n$
- $4n$
- $n \log_6 n$
- $n \log_2 n$
- $8n^2$
- $6n^3$
- 8^{2n}

Any Questions ?

Dr. Anand Singh Jalal
Professor
Email: asjalal@gla.ac.in