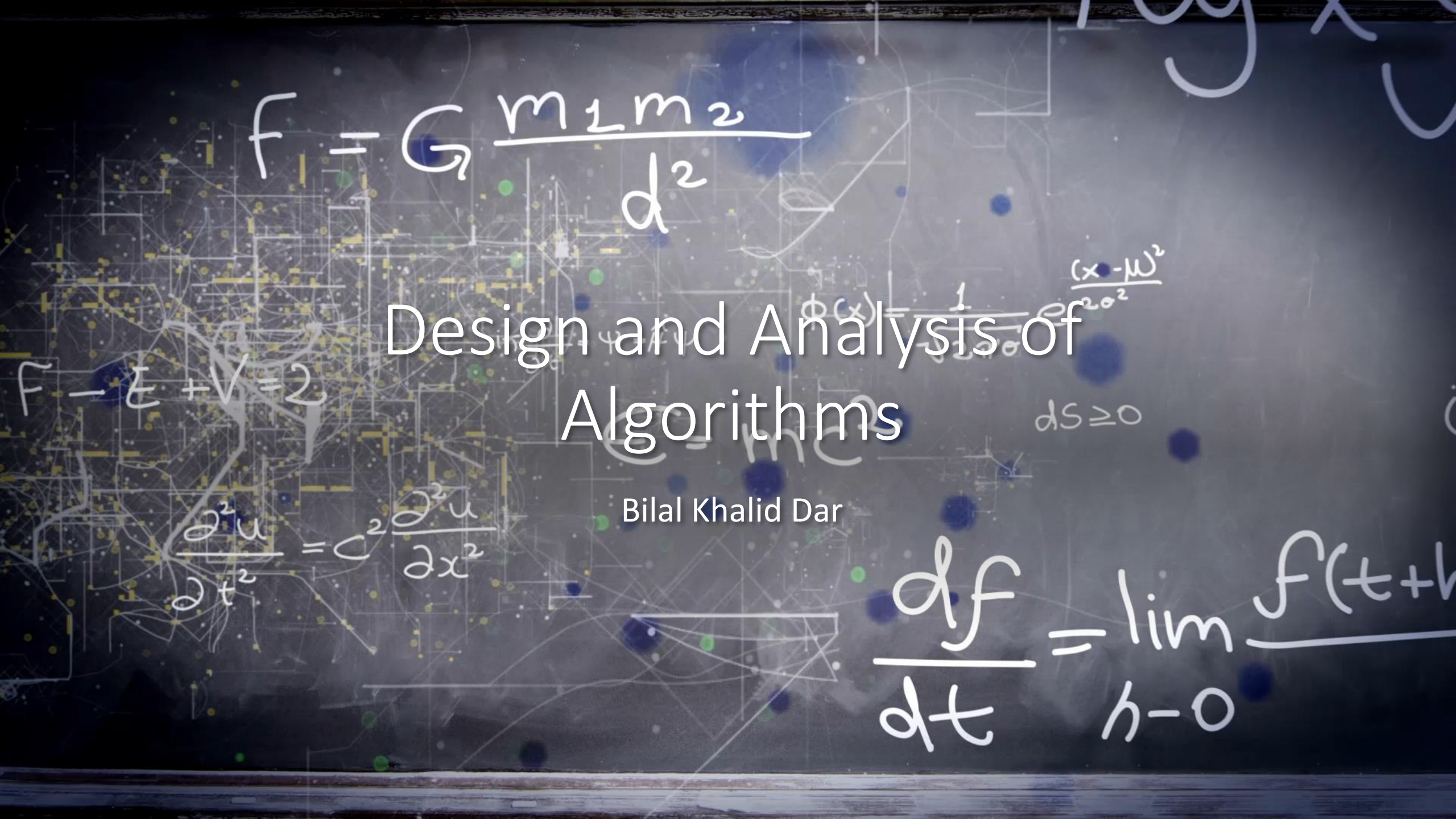



# Design and Analysis of Algorithms

Bilal Khalid Dar

# Let's start

# What is algorithm

- Algorithm
  - An algorithm is a well-defined and effective sequence of computation steps that takes some value, or set of values, as input and produces some value, or set of values, as output.

# Example

Featured
Price: Low to High
Price: High to Low
Avg. Customer Review
Newest Arrivals


ASUS TUF Dash 15 (2022) Gaming Laptop, 15.6” 144Hz FHD Display, Intel Core i7-12650H, GeForce RTX 3060, 16GB DDR5, 512GB SSD,...

4.5 out of 5 stars 1,307

$1,247.29

Ships to Pakistan

More Buying Choices
$1,230.08 (12 new offers)

Price may vary by color

SAMSUNG Galaxy Tab A7 Lite 8.7" 32GB Android Tablet w/ Compact, Slim Design, Sturdy Metal Frame, Long Lasting Battery, Gray

4.5 out of 5 stars 8,294

-25% $119.99 $159.99

Ships to Pakistan

More Buying Choices
$115.00 (22 used & new offers)

Price may vary by color

# Some questions

Why is the study of algorithms worthwhile?

What is the role of algorithms relative to other technologies used in computers?

# Example: sorting

- Input: A sequence of n numbers <a1,a2,a3…an>
- Output: A permutation (re-ordering) <b1,b2,b3…bn> of the input sequence such that b1<b2<b3…<bn

# Correctness of an algorithm

- An algorithm is said to be correct if, for every input instance, it halts with the correct output.
- An incorrect algorithm might not halt at all on some input instances, or It might halt with an answer other than desired on

# Problems solved by algorithms

- Sorting/searching are by no mean the only computational problem for which algorithms have been developed.
- Otherwise, we wouldn't have the whole course on this topic
- Practical application of algorithms are ubiquitous and include the following examples
  - Internet world
  - Electronic commerce
  - Manufacturing and other commercial settings
  - Shortest path
  - Matrices multiplication order
  - DNA sequence matching

# Common about algorithms

- There are many candidate solutions, most of which are not what we want, finding one that we do want can present quite a challenge.
- There are practical applications (its not just mathematical exercises to develop algorithms.)

# Why Study Algorithms and Performance of Algorithms

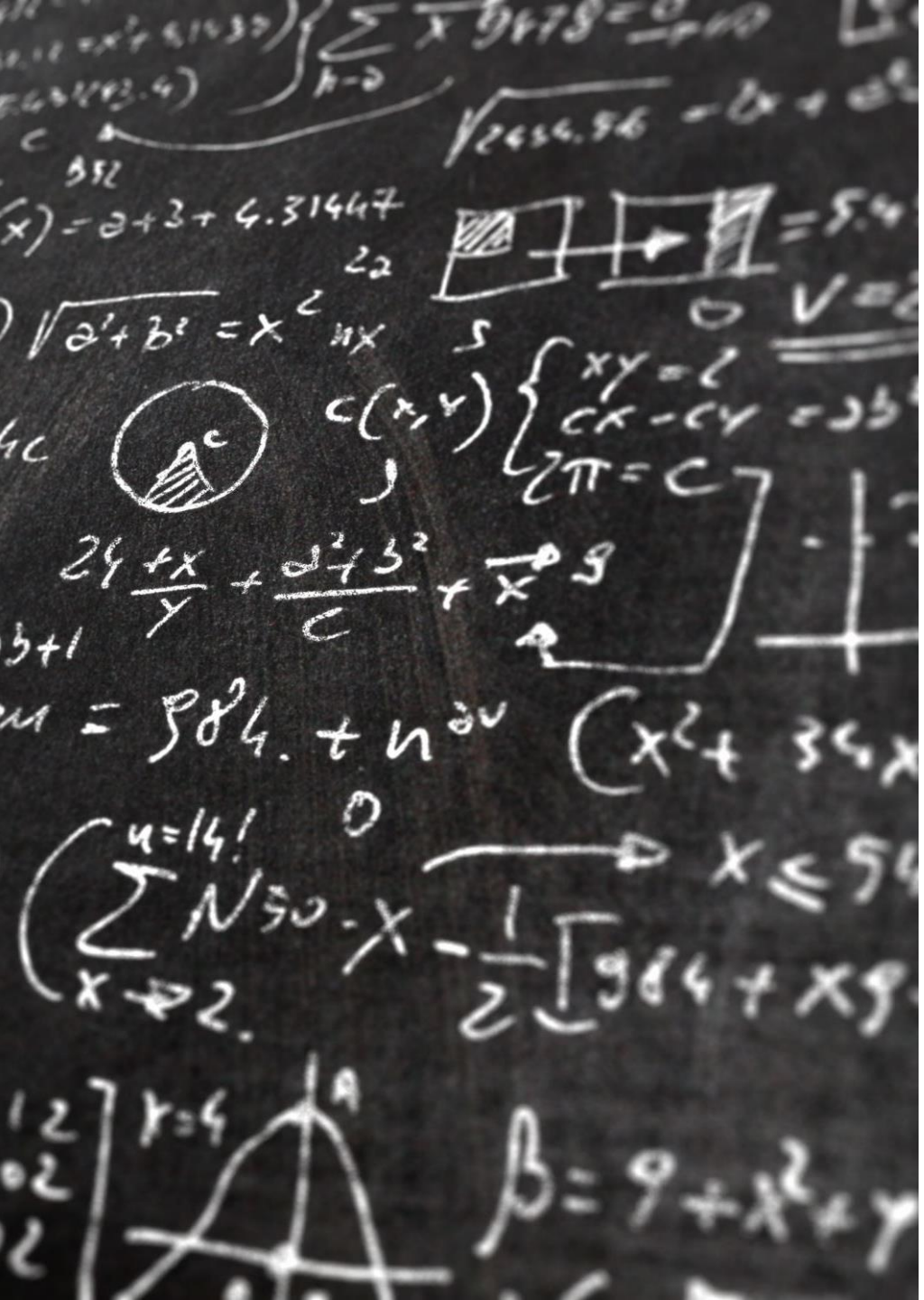
- Algorithms help us to understand scalability.
- Performance often draws the line between what is feasible and what is impossible.
- Algorithmic mathematics provides a language for talking about program behavior.
- Performance is the currency of computing.
- The lessons of program performance generalize to other computing resources.
- Speed is fun!

# What is Abstract Data Type

- A definition for a data type solely in terms of
  - a set of values and a
  - set of operations on that data type.
- The definition consists of:
- storage structures (data structures) to store the data items and algorithms for the basic operations

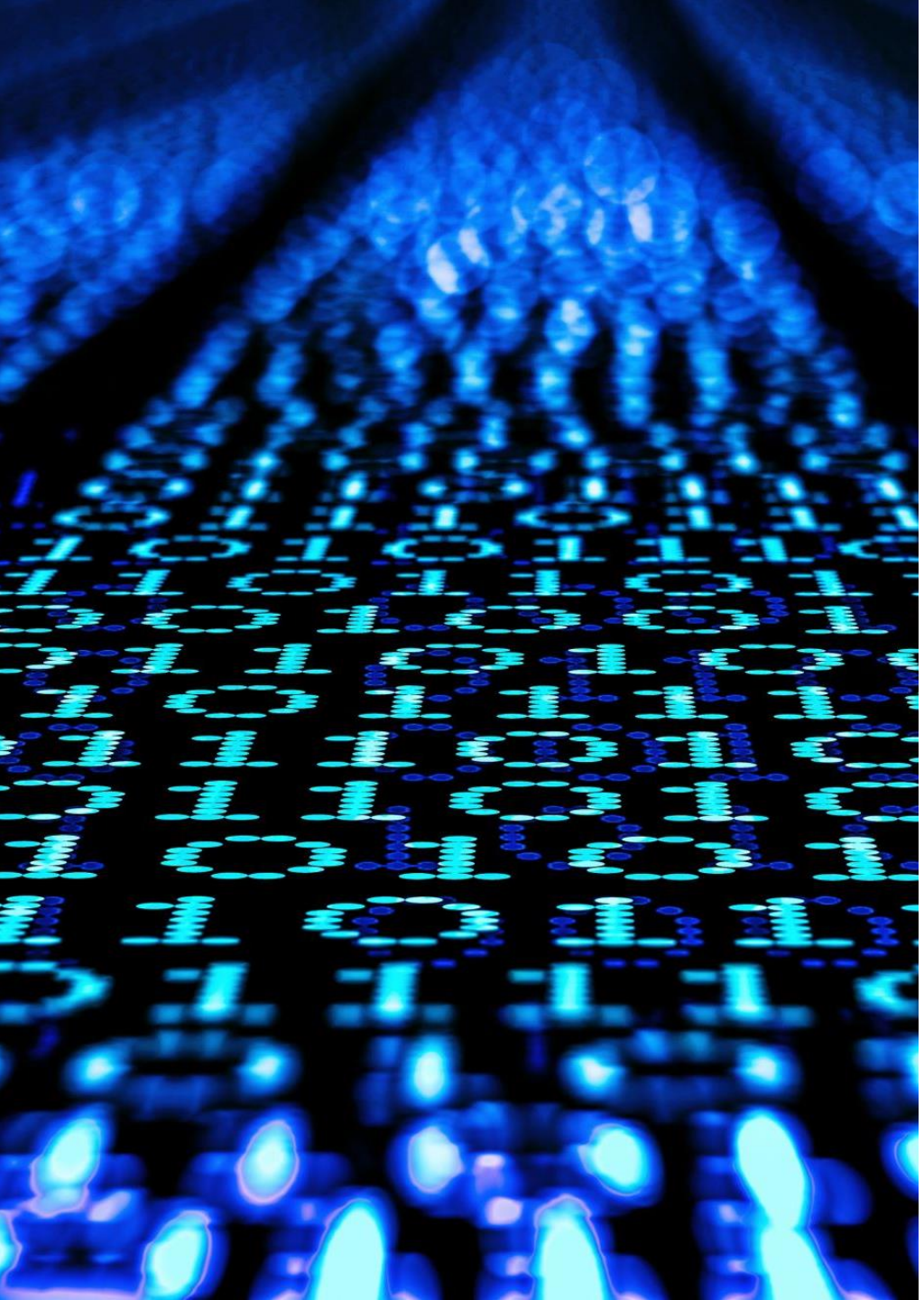
# What is a Data Structure

- A data structure is a way to store and organize data in order to facilitate access and modifications.
- No single data structure works well for all purposes
- Need to know the strengths and limitations of several of them.



# Technique

- Can’t get a “cookbook” for algorithms?
- Many problems you will encounter don’t have any published algorithm.
- So need to learn “techniques” of algorithms design and analysis
- So you develop algorithms in your own, show that they give correct answer and understand their efficiency.
- We will learn several such techniques in later part of this course.

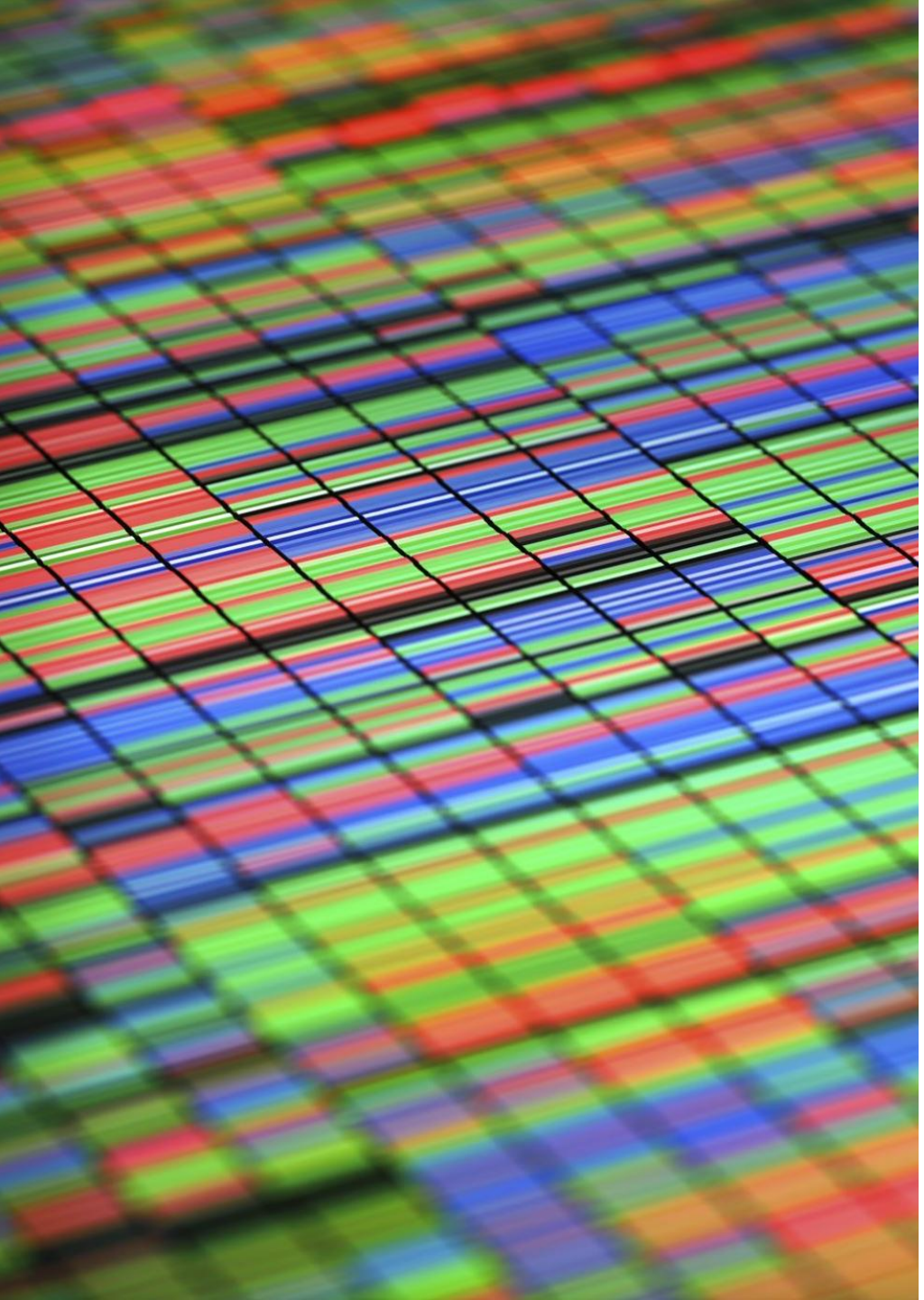


# Algorithms and other technologies

- Is an algorithm a technology like hardware, etc?
- Total system performance depends on choosing “efficient” algorithms as much as choosing fast hardware.

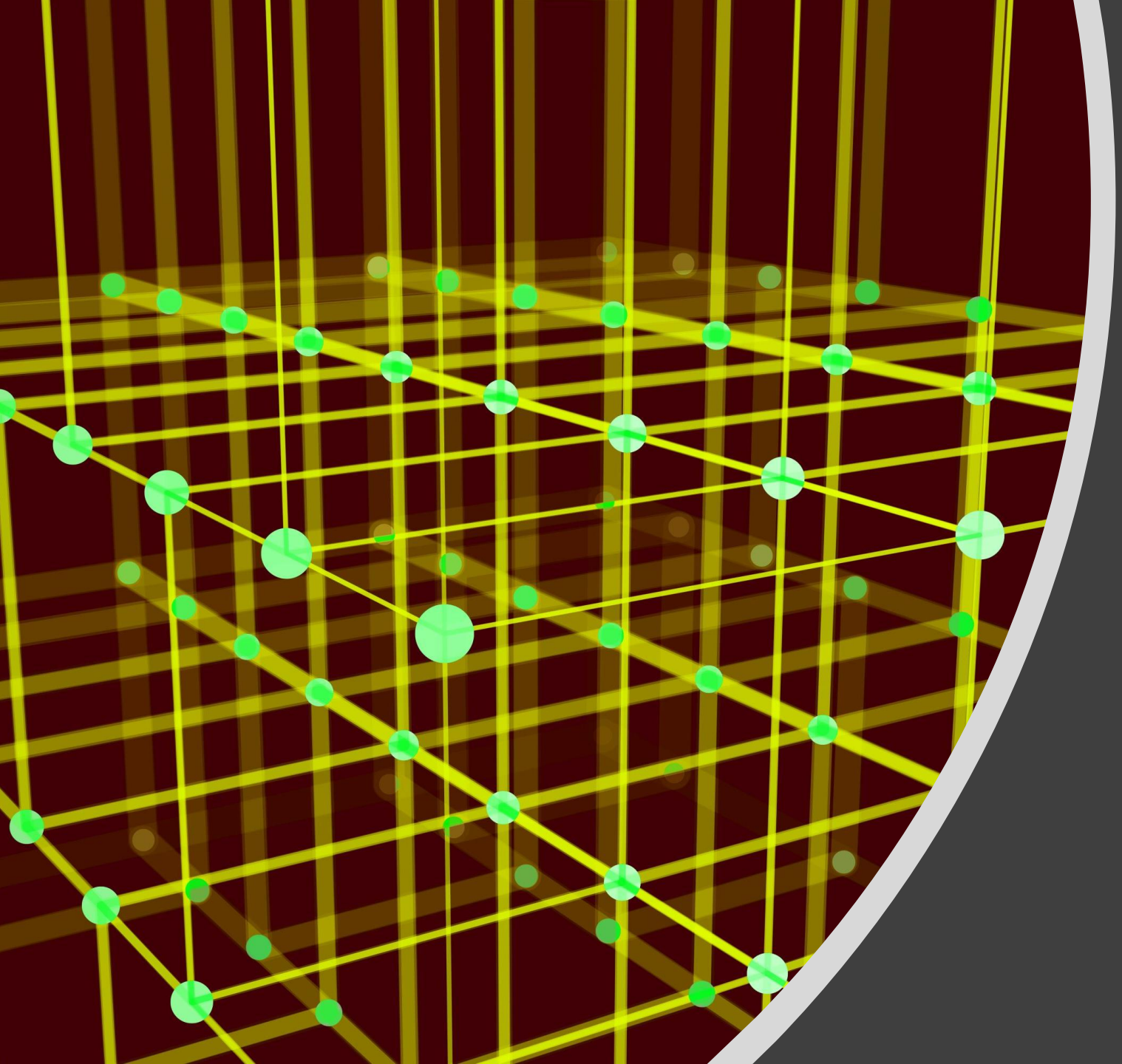
# Algorithms and other advanced technologies

- Hardware with high clock rates, pipelining and superscalar architecture.
- Easy to use graphical user interface (GUI's)
- Object oriented systems.
- Local-area and wide-area networking.
- Are algorithms as important as above technologies?

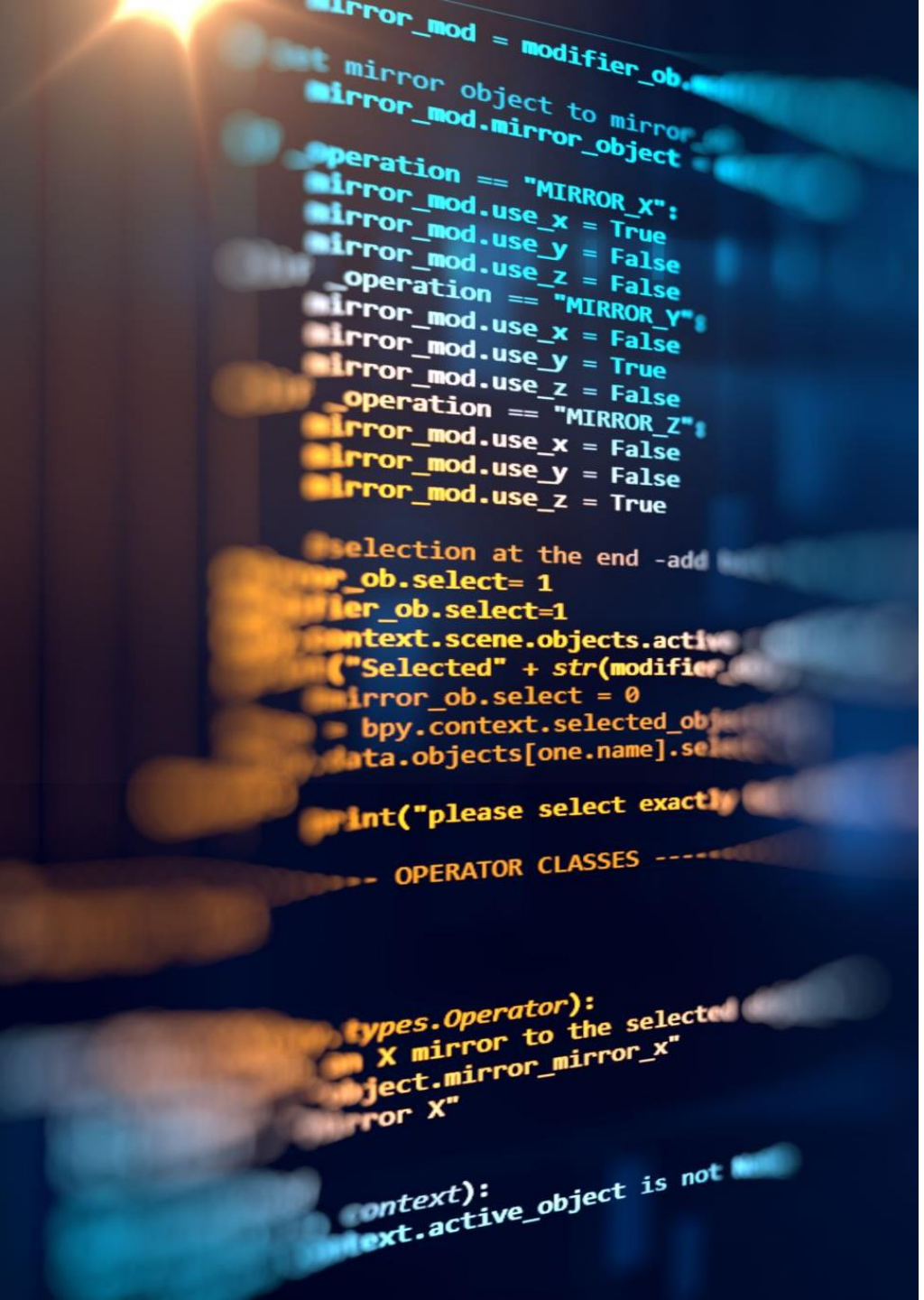


# Course-Overview

- Basics of Algorithms
- Asymptotic Notations
- Sorting Algorithms
- Divide and Conquer
- String Matching Algorithms
- Graph Theory
- Dynamic Programming



# Pseudocode



# Pseudocode

- Pseudocode is a compact and informal high-level description of a program using the conventions of a programming language, but intended more for humans.
- Pseudocode is not an actual programming language.
- So it cannot be compiled into an executable program.

# Pseudocode

- **INPUT** – indicates a user will be inputting something
- **OUTPUT** – indicates that an output will appear on the screen
- **WHILE** – a loop (iteration that has a condition at the beginning)
- **FOR** – a counting loop (iteration)
- **REPEAT** – **UNTIL** – a loop (iteration) that has a condition at the end
- **IF – THEN – ELSE** – a decision (selection) in which a choice is made any instructions that occur inside a
- <u>selection or iteration are usually indented</u>

# Pseudocode Example

- Calculate Area and Perimeter of Rectangle

```
BEGIN
NUMBER b1,b2,area,perimeter
INPUT b1
INPUT b2
area=b1*b2
perimeter=2*(b1+b2)
OUTPUT area
OUTPUT perimeter
END
```

# Pseudocode If Else Example

```
1
2  BEGIN
3  NUMBER age
4
5  INPUT "Enter your age for driving licence"
6  OUTPUT age
7
8  IF age>=18  THEN
9      OUTPUT "You can take driving licence"
10 ELSE
11   OUTPUT "You can't take driving licence"
12 ENDIF
13 END
```

# Pseudocode For Loop Example

```
BEGIN
NUMBER counter

FOR  counter = 1 TO  100 STEP 1 DO
    OUTPUT counter
ENDFOR

END
```

Read 10 numbers and find sum of even numbers

```
1
2  BEGIN
3  NUMBER counter, sum=0, num
4  FOR counter=1 TO 10 STEP 1 DO
5      OUTPUT "Enter a Number"
6      INPUT num
7      IF num % 2 == 0 THEN
8          sum=sum+num
9      ENDIF
10 ENDFOR
11 OUTPUT sum
12
13 END
```

Find the sum of all elements of array

```
PROCEDURE SUM_Array
BEGIN
Input:List numbers[1..n], n
Output: Sum of an array
sum=0
FOR i=0 to n-1
   sum = sum + numbers[i]
ENDFOR

OUTPUT  "Sum of numbers in the array"+sum

END
```

# Reference

Chapter#1 (The Role of Algorithms in Computing)
Introduction to Algorithms Thomas H.Corman

# Efficiency

- Computer Scientists don’t just write programs.
- They also analyze them.
- How efficient is a program?
  - How much time does it take program to complete?
  - How much memory does a program use?
  - How do these change as the amount  of data changes?
  - What is the difference between the average case and worst case efficiency if any?

# Algorithm Characteristics

The necessary features of an algorithm:

- Definiteness
  - The steps of an algorithm must be precisely defined.
- Effectiveness
  - Individual steps are all do-able.
- Finiteness
  - It won’t take forever to produce the desired output for any input in the specified domain.
- Output
  - Information or data that goes out.

# Algorithm Characteristics

Other important features of an algorithm:

- Input.
  - Information or data that comes in.
- Correctness.
  - Outputs correctly relate to inputs.
- Generality.
  - Works for many possible inputs.
- Efficiency.
  - Takes little time & memory to run.

# Let’s refresh your mind – Task 1

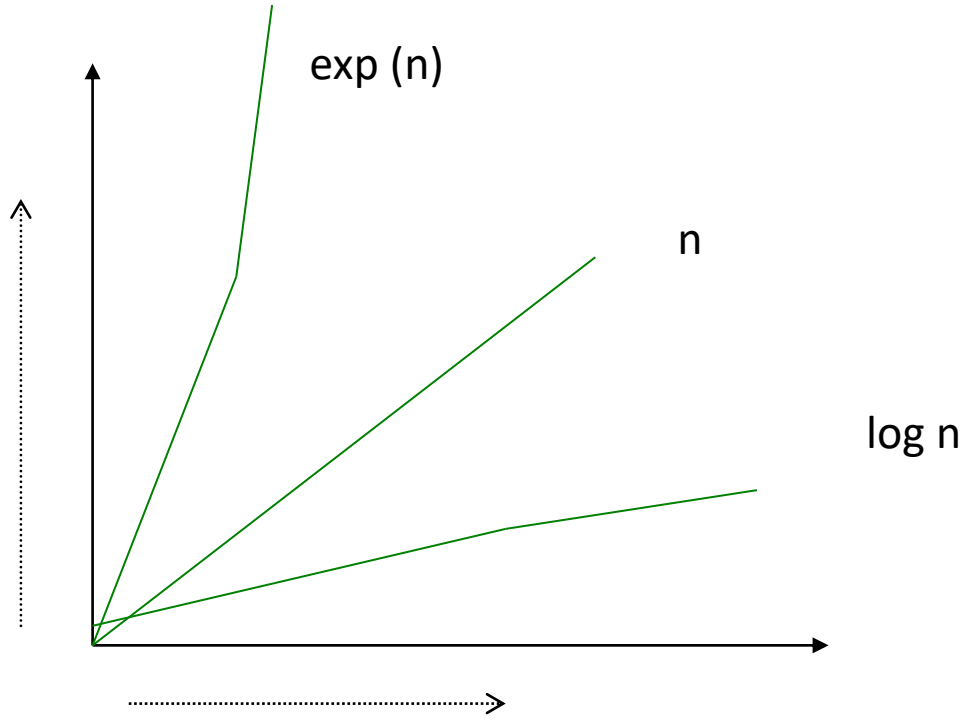
- Write pseudocode to multiply two positive integers a and b (without using multiplication operation)

# Complexity Analysis

# Complexity

- In general, we are not so much interested in the time and space complexity for small inputs.

- For example, while the difference in time complexity between linear and binary search is meaningless for a sequence with n = 10, it is gigantic for n = $2^{30}$.



# Complexity

- For example, let us assume two algorithms A and B that solve the same class of problems.
- The time complexity of A is 5,000n, the one for B is $\lceil 1.1^n \rceil$ for an input with n elements.
- For n = 10, A requires 50,000 steps, but B only 3, so B seems to be superior to A.
- For n = 1000, however, A requires 5,000,000 steps, while B requires $2.5 \cdot 10^{41}$ steps.

# Complexity

- Comparison: time complexity of algorithms A and B

| Input Size | Algorithm A | Algorithm B |
|---|---|---|
| n | 5,000n | $1.1^n$ |
| 10 | 50,000 | 3 |
| 100 | 500,000 | 13,781 |
| 1,000 | 5,000,000 | $2.5x10^{41}$ |
| 1,000,000 | $5\square10^9$ | $4.8x10^{41392}$ |

# Complexity

- This means that algorithm B cannot be used for large inputs, while algorithm A is still feasible.
- So what is important is the growth of the complexity functions.
- The growth of time and space complexity with increasing input size n is a suitable measure for the comparison of algorithms.

# Complexity classes
# f(n)

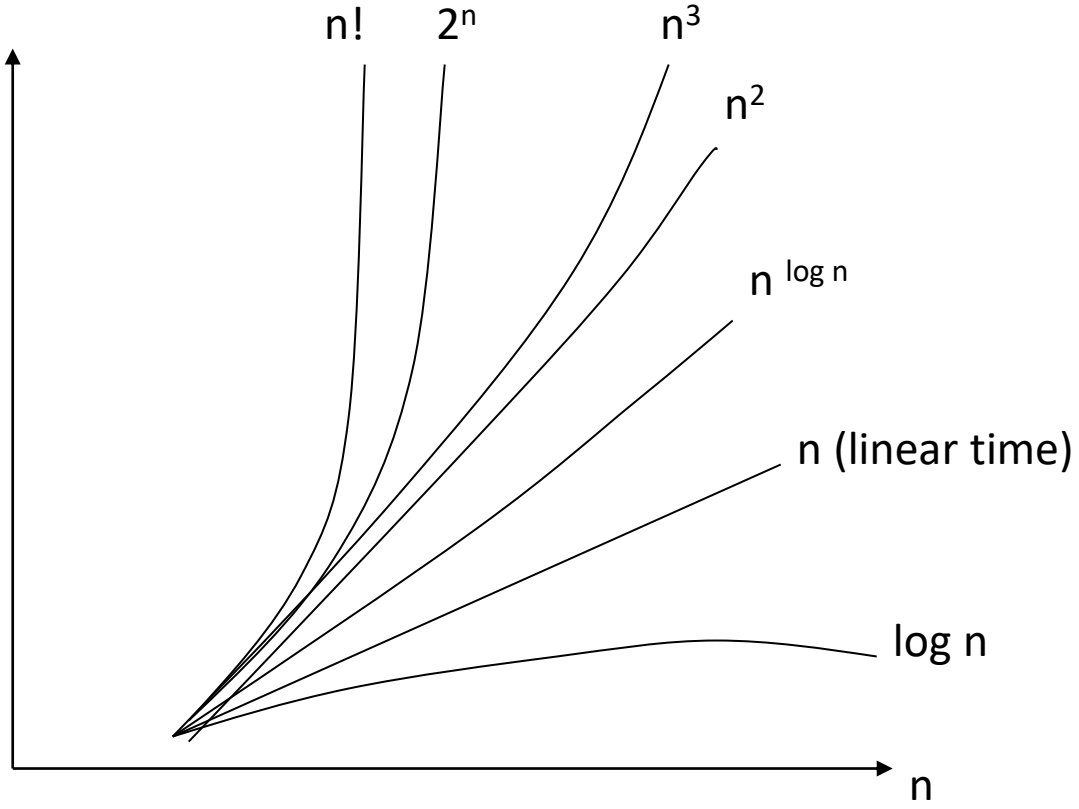



**Figure 2**. Growth rates of some important complexity classes

# Clicker Question 1

- “A program finds all the prime numbers between 2 and 1,000,000,000 from scratch
  in 0.37 seconds."
- Is this a fast solution?

A. no

B. yes

C. it depends

# Clicker Question 2

- What is output by the following code?

```
int total = 0;
for (int i = 0; i < 13; i++)
    for (int j = 0; j < 11; j++)
        total += 2;
System.out.println(total);
```

A. 24
B. 120
C. 143
D. 286
E. 338

# Standard Analysis Techniques

- Constant time statements
- Analyzing Loops
- Analyzing Nested Loops
- Analyzing Sequence of Statements
- Analyzing Conditional Statements

# Analysis Example

- How many statements are executed by method total as a function of values.length
- Let N = values.length
- N is commonly used as a variable that denotes the amount of data

```
public int total(int[] values) {
    int result = 0;
    for (int i = 0; i < values.length; i++)
        result += values[i];
    return result;
}
```

# Counting Up Statements

```
public int total(int[] values) {
    int result = 0;
    for (int i = 0; i < values.length; i++)
       result += values[i];
    return result;
}
```

- int result = 0; 1
- int i = 0; 1
- i < values.length; N + 1
- i++ N
- result += values[i]; 3N
- return total; 1
- T(N) = 5N + 4
- T(N) is the number of executable statements in method total as function of values.length

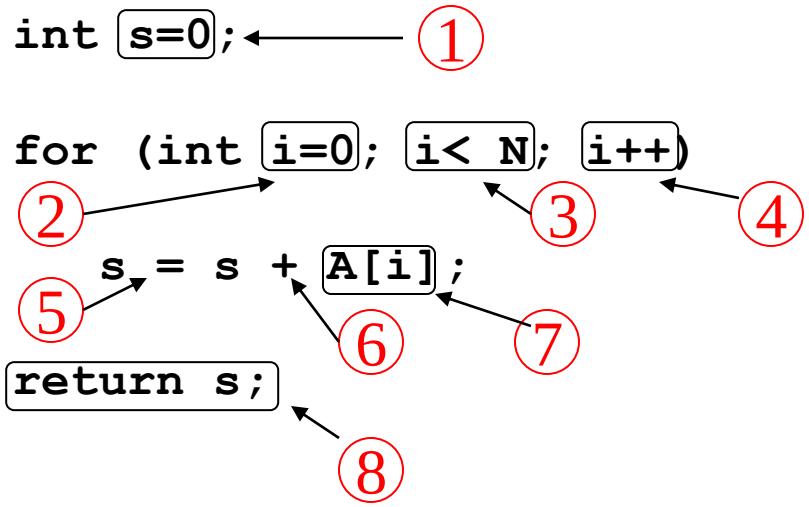
# Analyzing an Algorithm

```
// Input: int A[N], array of N integers
// Output: Sum of all numbers in array A

int Sum(int A[], int N)
{
  int s=0;
  for (int i=0; i< N; i++)
    s = s + A[i];
  return s;
}
```

How should we analyse this?

# Analyzing an Algorithm

```
// Input: int A[N], array of N integers
// Output: Sum of all numbers in array A

int Sum(int A[], int N){
    int s=0;
    for (int i=0; i< N; i++)
        s = s + A[i];
    return s;
}
```




1,2,8: Once

3,4,5,6,7: Once per each iteration of for loop, N iteration

Total: 5N + 4

The *complexity function* of the algorithm is : *f(N) = 5N +4*

# Another Simplification

- When determining complexity of an algorithm we want to simplify things
  - hide some details to make comparisons easier
- Like assigning your grade for course
  - At the end of of degree your transcript won't list all the details of your performance in the course
  - it won't list scores on all assignments, quizzes, and tests
  - simply a letter grade, B- or A or D+
- So we focus on the dominant term from the function and ignore the coefficient

# Constant time statements

- Simplest case: O(1) time statements

- Assignment statements of simple data types
  int x = y;

- Arithmetic operations:
  x = 5 * y + 4 - z;

- Array referencing:
  A[j] = 5;

- Most conditional tests:
  if (x < 12) ...

# Analyzing Loops

- Any loop has two parts:
- How many iterations are performed?
- How many steps per iteration?

```
int sum = 0,j;
for (j=0; j < N; j++)
sum = sum +j;
```

# Analyzing Loops

- Any loop has two parts:
  - How many iterations are performed?
  - How many steps per iteration?

```
int sum = 0,j;
for (j=0; j < N; j++)
  sum = sum +j;
```

  - Loop executes N times (0..N-1)
  - O(1) steps per iteration
- Total time is N * O(1) = O(N*1) = O(N)

# Analyzing Nested Loops

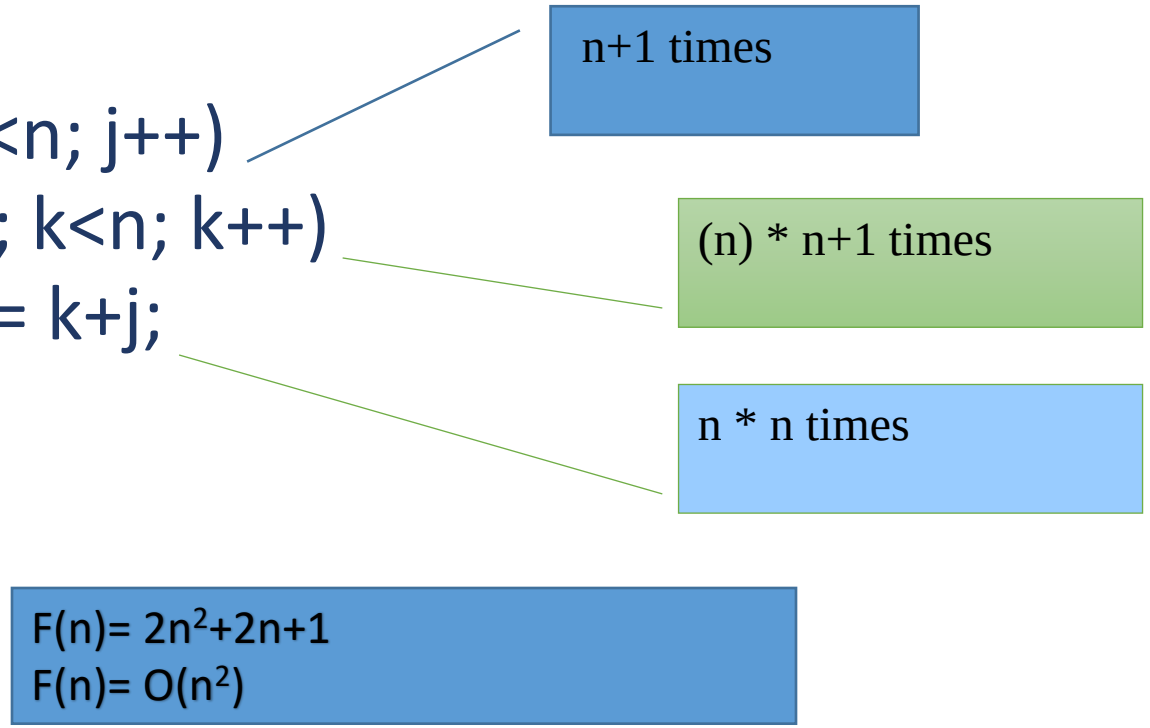
- Treat just like a single loop and evaluate each level of nesting as needed:

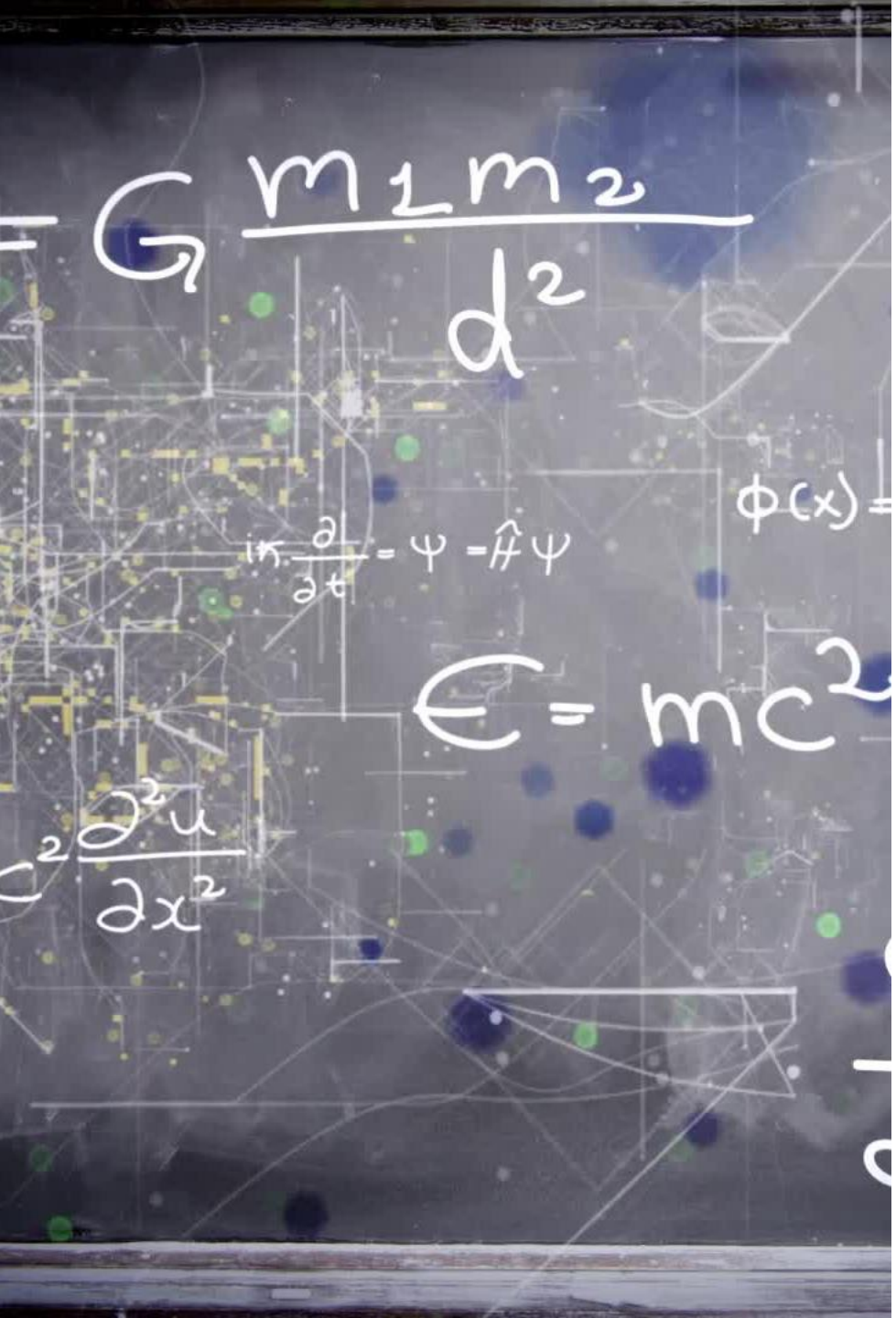
```
int j,k;
for (j=0; j<n; j++)
   for (k=0; k<n; k++)
      sum += k+j;
```

# Analyzing Nested Loops

- Treat just like a single loop and evaluate each level of nesting as needed:

```
int j,k;
for (j=0; j<n; j++)
  for (k=0; k<n; k++)
    sum += k+j;
```






# Clicker Question 3

- What is output when method `sample` is called?

```
// pre: n >= 0, m >= 0
public static void sample(int n,
int m) {
    int total = 0;
    for (int i = 0; i < n; i++)
        for (int j = 0; j < m;
j++)
            total += 5;
    System.out.println(total);
}
```

A. 5

B. n * m

C. n * m * 5

D. $n^m$

E. $(n * m)^5$

# Analyzing an Algorithm

- Simple statement sequence

  $s_1$; $s_2$; …. ; $s_k$

  - Basic Step = $1$ as long as $k$ is constant
- Simple loops

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

  where `s` is Basic Step = $1$
  - Basic Steps : $n$
- Nested loops

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

  - Basic Steps : $n^2$

# Analyzing Nested Loops

- Treat just like a single loop and evaluate each level of nesting as needed:

```
int j,k;
for (j=0; j<N; j++)
  for (k=N; k>0; k--)
    sum += k+j;
```

- Start with outer loop:
  - How many iterations?  N
  - How much time per iteration? Need to evaluate inner loop

- Inner loop uses O(N) time
- Total time is N * O(N) = O(N*N) = $O(N^2)$

# Class Activity

```
void add( int A[ ], int B[ ], int n)
{
   for (i=0; i<n; i++)
    {
       for (j=0; j< n; j++)
          {
                    c[i,j]  =A[i][k]  + B[k][j];

           }
       }
}
```

# Class Activity

```
void multiply( int A[ ], int B[ ], int n)
{
   for (i=0; i<n; i++)
    {
       for (j=0; j< n; j++)
          {
            c[i,j]= 0;
                 for (k=0; k< n; ++)
                    {
                      c[i,j]+=A[i][k]*b[k][j];
                              }
          }
       }
}
```

# Analyzing Sequence of Statements

```
for (j=0; j < N; j++)
   for (k =0; k < j; k++)
      sum = sum + j*k;
for (l=0; l < N; l++)
   sum = sum -l;
cout<<“Sum=”<<sum;
```

# Analyzing Sequence of Statements

- For a sequence of statements, compute their complexity functions individually and add them up

```
for (j=0; j < N; j++)
  for (k =0; k < j; k++)
    sum = sum + j*k;
for (l=0; l < N; l++)
  sum = sum -l;
cout<<“Sum=”<<sum;
```

$O(N^2)$

$O(N)$

$O(1)$

Total cost is $O(N^2) + O(N) + O(1) = O(N^2)$

SUM RULE

# Analysing an Algorithm

- Loop index doesn't vary linearly

```
i = 1;
while ( i < n ) {
      s;
      i = 2 * i;
      }
```

  - *i* takes values $1, 2, 4, \ldots$ until it exceeds $n$

# In Next Lecture

# Write Pseudocode and calculate complexity

- A code for finding the sum of the digits in the number.

- Example: 3554
- 3+5+5+4 = 17

# Thank you