GRADING POLICY

- Mid-term Exam 20%
- Quizzes 20%
- Final-Exam (Written Exam) 60 %

CS316: ALGORITHMS
LECTURE 1: INTRODUCTION

Assoc. Prof. Ensaf Hussein

Dr. Salwa Osama

GRADING POLICY

- Mid-term Exam 20%
- Quizzes 20%
- Final-Exam (Written Exam) 60 %

RESOURCES

Textbook:

Thomas Cormen, Charles Leierson, Ronald Rivest, and Clifford Stein. *Introduction to Algorithms*. 3rd ed. MIT Press, 2009.

Anany Levitin, Introduction to the design and analysis of algorithms 3rd Edition, 2012.

Click to add text

Handouts

https://teams.microsoft.com/l/team/19%3aTwB4GPgelOjjTwYSk6a0ncxP9iKsRUYJWIY-896ltGM1%40thread.tacv2/conversations?groupId=196a9aa4-9ea3-4ca0-a8f7-8117115c5dad&tenantId=aadc0e0a-65ee-471a-99a1-9f86faecbaed

COURSE SYLLABUS

- Algorithms Design and Analysis
- Asymptotic Notations
- Compute Complexity for :
  - Non-recursive algorithms
  - Recursive Algorithms
    - ~~Substitution Method~~
    - Iteration Tree
    - Master Method
- Divide and Conquer Algorithms (Merge Sort- Quick Sort)
- Linear time Sorting (count – bucket – radix)
- Graph
  - BFS - DFS
  - MST – Prim - Kruskal
  - Shortest path – Dijkstra – Bellman Ford
- Dynamic Programming – Matrix Chain Multiplication - Longest Common Subsequence
- Greedy Approach – Knapsack [0-1 / Fractional]

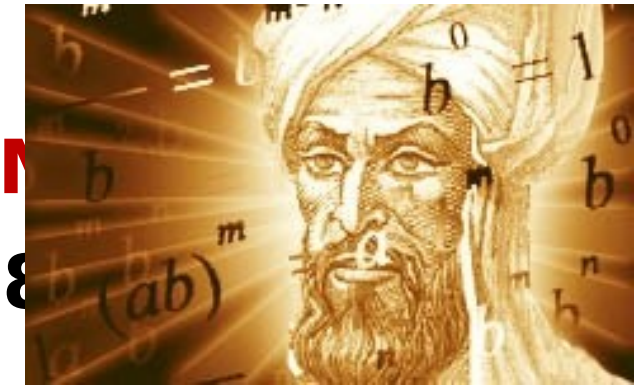
COURSE OBJECTIVES

- Design algorithms using Pseudocode.
- Demonstrate and study asymptotic notations in best, worst and average case.
- Define and analyze the complexity of recursive and non-recursive algorithms.
- Understand, analyze and apply standard algorithms involving searching, sorting, tree, graph, greedy, backtracking and dynamic programming algorithms.

ALGORITHMS

ABU JA‘FAR MOHAMMED IBN M

AL-KHOWARIZMI (C. 780 – C. 8



Al-Khowarizmi, an astronomer and mathematician, was a member of the House of Wisdom, an academy of scientists in Baghdad. The name al-Khowarizmi means “from Kowarzizm,” which was then part of Persia, but is now called *Khiwa* and is part of Uzbekistan. Al-Khowarizmi wrote books on mathematics, astronomy, and geography.

Western Europeans first learned about algebra from his works. The word *algebra* comes from al-jabr, part of the title of his book *Kitab al-jabr w’al muquabala*. This book was translated into Latin and was a widely used textbook. His book on the

WHY STUDYING ALGORITHMS?

Donald Knuth, one of the most prominent computer scientists in the history of algorithmics, put it as follows:

> A person well-trained in computer science knows how to deal with algorithms: how to construct them, manipulate them, understand them, analyze them. This knowledge is preparation for much more than writing good computer programs; it is a general-purpose mental tool that will be a definite aid to the understanding of other subjects, whether they be chemistry, linguistics, or music, etc. The reason for this may be understood in the following way: It has often been said that a person does not really understand something until after teaching it to someone else. Actually, a person does not *really* understand something until after teaching it to a *computer*, i.e., expressing it as an algorithm . . . An attempt to formalize things as algorithms leads to a much deeper understanding than if we simply try to comprehend things in the traditional way. [Knu96, p. 9]

WHY STUDYING ALGORITHMS?

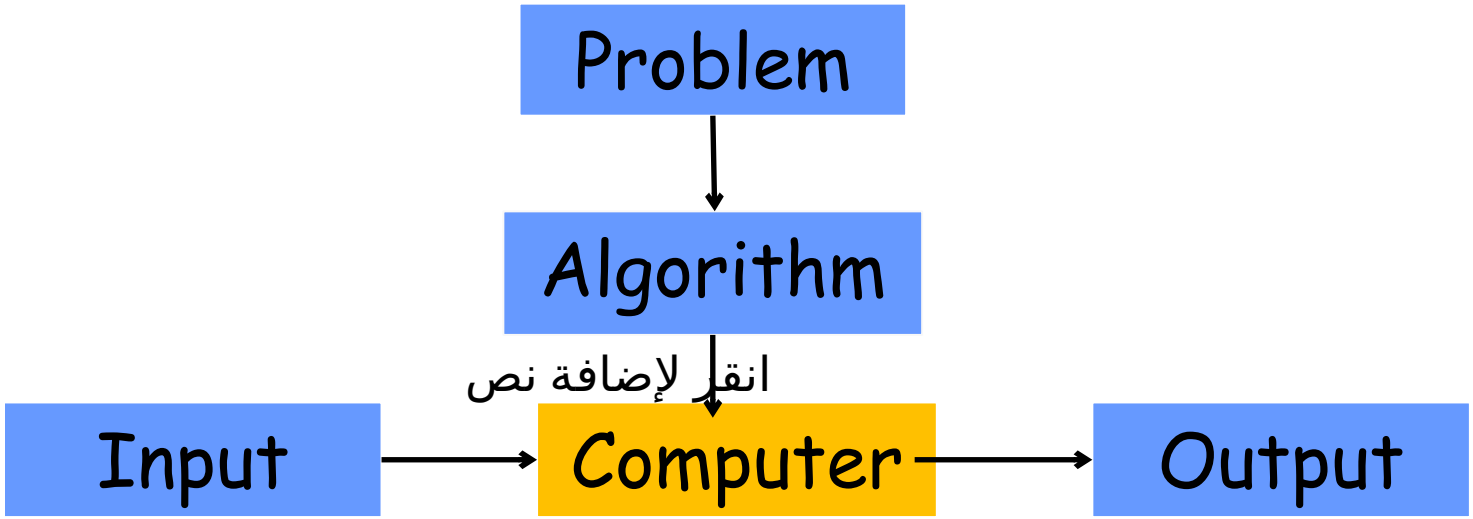
Click to add text

- The study of algorithms is the cornerstone of computer science.
- You have to know a standard set of important algorithms from different areas of computing; in addition, you should be able to design new algorithms and analyze their efficiency.
- Algorithms can be seen as special kinds of solutions to problems— not just answers but precisely defined procedures for getting answers

WHAT IS ALGORITHMS ?!

ALGORITHMS

p= algorithms + data structures



Data structures: Methods of organizing data

You have to design your algorithm before coding

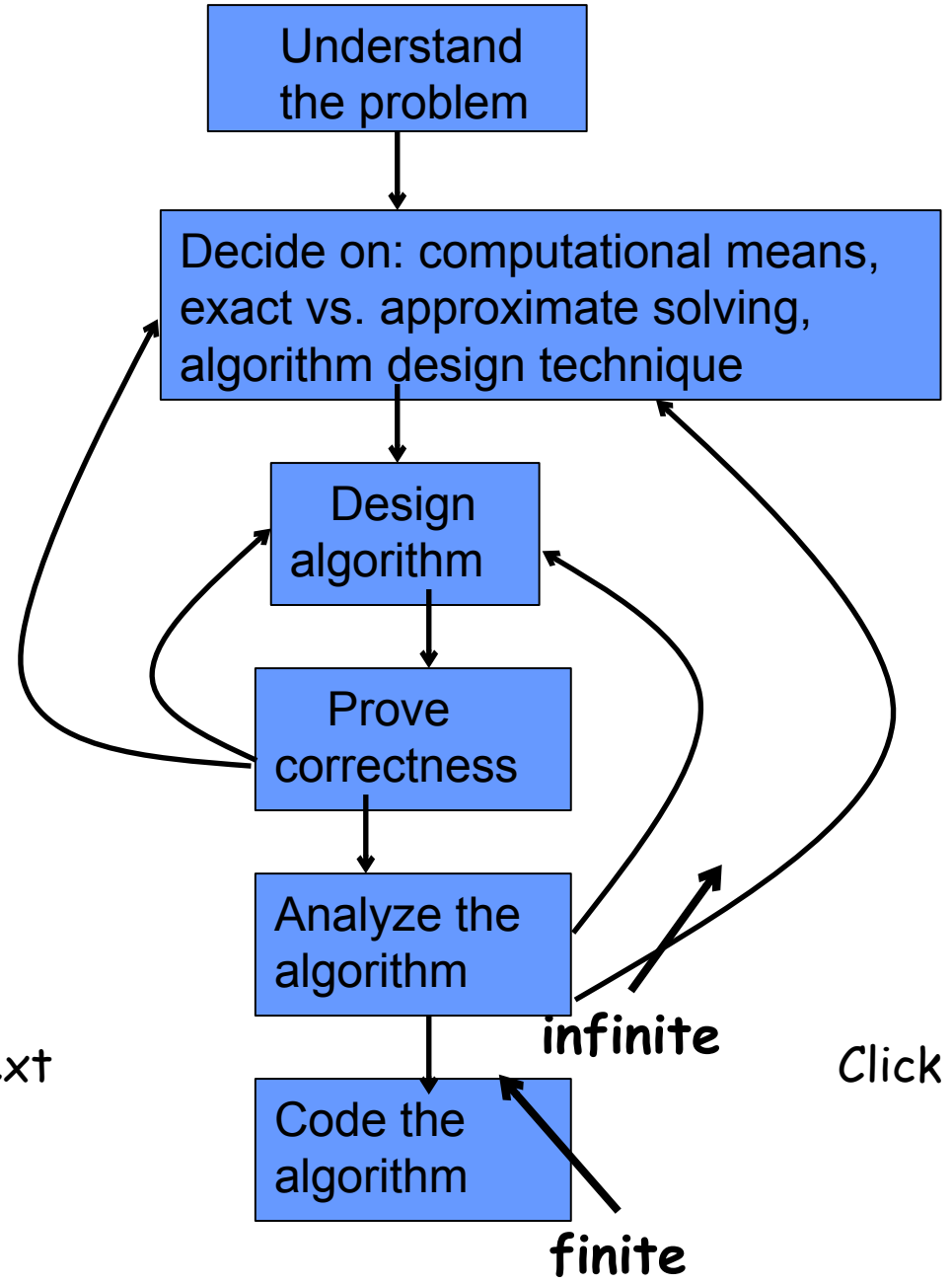
ALGORITHMS

An algorithm: sequence of **unambiguous** instructions for solving a problem.

- Obtaining a required output for any allowable input in a finite amount of time.

- Example: how do you make a cup of tea?

1. Boil kettle
2. Put tea in cup
3. Pour boiling water into cup



Click to add text

Click to add text

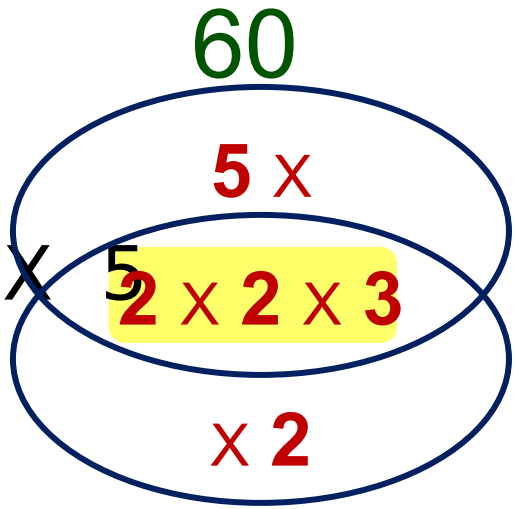
EXAMPLE:

The greatest common divisor of two nonnegative, not-both-zero integers m and n, denoted gcd(m, n), is defined as the largest integer that divides both m and n.

Ex: **gcd (60,24) ??**

60 = 2 X 2 X 3 X 5

24 = 2 X 2 X 2 X 3



gcd (60,24) = 2 X 2 X 3 = 12

A SOLUTION FOR GCD(M,N):

Step 1 Find the prime factorization of *m*

Step 2 Find the prime factorization of *n*

Step 3 Find all the common prime factors

Step 4 Compute the product of all the common prime factors and return it as gcd(*m*,*n*)

Middle-school procedure

Is this an algorithm???

The algorithm should has no ambiguous instructions

ANOTHER SOLUTION FOR GCD(M,N)

Consecutive integer checking algorithm

Step 1 Assign the value of min {*m,n* } to *t*

Step 2 Divide *m* by *t.* If the remainder is 0, go to Step 3 otherwise, go to Step 4

Step 3 Divide *n* by *t.* If the remainder is 0, return *t* and stop; otherwise, go to Step 4

Step 4 Decrease *t* by 1 and go to Step 2

Is this an algorithm???

**It is so important to specify the sa
algorithm's inputs explicitly**

- How it will work when one of its inp
numbers is zero?

EUCLID'S SOLUTION FOR GCD

Euclid's algorithm is based on:

Repeated application of equality

gcd(m, n) = gcd(n , m mod n),

until the second number becomes 0

Ex: **gcd(60, 24)**

= gcd(**24**, **60** mod **24**)

= gcd (**24**, **12**)

= gcd (**12**, **24** mod **12**)

= gcd (**12**, **0**)

= 12

PROOF: GCD(A, B) = GCD(B, A MOD B)

If (a mod b) = c, then there is a y such that $a-c=by$, i.e., $c=a-by$

If d divides both a and b, then it also divides $a-by$.

Therefore any common divisor of a and b is also a divisor of c.

Similarly, if d divides both c and b, then it also divides $c+by=a$, so any common divisor of c and b is a common divisor of a and b.

This shows that the common divisors of a and b are exactly the common divisors of c and b, so, in particular, they have the same greatest common divisor.

EUCLID'S METHOD

Step 1 If n = 0, return m and stop; otherwise go to Step 2

Step 2 Divide m by n and assign the value of the remainder to r

Step 3 Assign the value of n to m and the value of r to n. Go to Step 1.

English like

Could be writen as:

while $n \neq 0$ do

$r \leftarrow m$ mod n

$m \leftarrow n$

$n \leftarrow r$

return m

Pseudo-code like Algorithm

ALGORITHM DESIGN

ALGORITHM DESIGN:

Algorithm can be described in three ways:

1- Natural language like English:

When this way is chosen care should be taken, we should ensure that each & every statement is definite.

2- Graphic representation called flowchart:

This method will work well when the algorithm is small& simple.

3- Pseudo-code Method:

In this method, we should typically describe algorithms as program, which resembles language like Pascal & algol.

PSEUDO-CODE CONVENTIONS:

1- Comments begin with // and continue until the end of line.

2- Blocks are indicated with matching braces {and}.

3- An identifier begins with a letter. The data types of variables are not explicitly declared.

4- Assignment of values to variables is done using the assignment statement.

<Variable>:= <expression>; Or <Variable> ← <expression>;

5- There are two Boolean values TRUE and FALSE.

- Logical Operators AND, OR, NOT
- Relational Operators <, <=,>,>=, =, !=

6- The following looping statements are employed.

For, while and repeat-until

While Loop:

```
While < condition > do
{
        <statement-1>
        <statement-n>
}
```

For Loop:

For variable: = to1 value-2 step vlaue-3 dovalue-

{

<statement-1>

<statement-n>

}

repeat-until:

repeat

<statement-1>

<statement-n>

until<condition>

7- A conditional statement has the following forms.

- If <condition> then <statement>
- If <condition> then <statement-1>
  Else <statement-1>

Case statement:

select case(expression)

{

case 1 **:** <statement-1>

case n **:** <statement-n>

default **:** <statement-n+1>

}

8- Input and output are done using the instructions read & write.

9- There is only one type of procedure:

Algorithm, the heading takes the form,

Algorithm Name (Parameter lists)

- **As an example, the following algorithm finds & returns the maximum of ‘n’ given numbers:**

EXAMPLE

```
1. algorithm Max( A, n)
2. // A is an array of
   size n
3. {
4. Result := A[1]
5. for I ← 2 to n do
6.     if A[I] > Result
   then
7.           Result ← A[I]
8.   return Result
9. }
```

This algorithm:

- named Max
- A & n are procedure parameters.
- Result & I are Local variables.

ANALYSIS OF ALGORITHMS

ANALYSIS OF ALGORITHMS

- Term “analysis of algorithms” means an investigation of an algorithm’s efficiency
- The main goal is to determine the cost of running an algorithm and how to reduce that cost.
- Cost is expressed as Complexity
  - Time Complexity
  - Space Complexity

TIME COMPLEXITY

It may be useful to time how long an algorithm takes to run

- In some cases it may be *essential* to know how long an algorithm takes on some system
  - e.g. air traffic control systems

But, **is this a good general comparison method?**

Running time is affected by a number of factors other than algorithm efficiency



RUNNING TIME IS AFFECTED BY

CPU speed

Amount of main memory

Specialized hardware (e.g. graphics card)

Operating system

System configuration (e.g. virtual memory)

Programming language

Algorithm implementation

Other programs

System tasks (e.g. memory management)



TIME COMPLEXITY

Instead of *timing* an algorithm, *count* the number of instructions that it performs

The number of instructions performed may vary based on

- The size of the input– *ex, multiply two matrixes, though not always* – *ex, find binary representation of a decimal number.* Almost all algorithms run longer on larger inputs.
- The organization of the input – *ex, consider searching a large array, If the target is the first item in the array the search will be very quick*



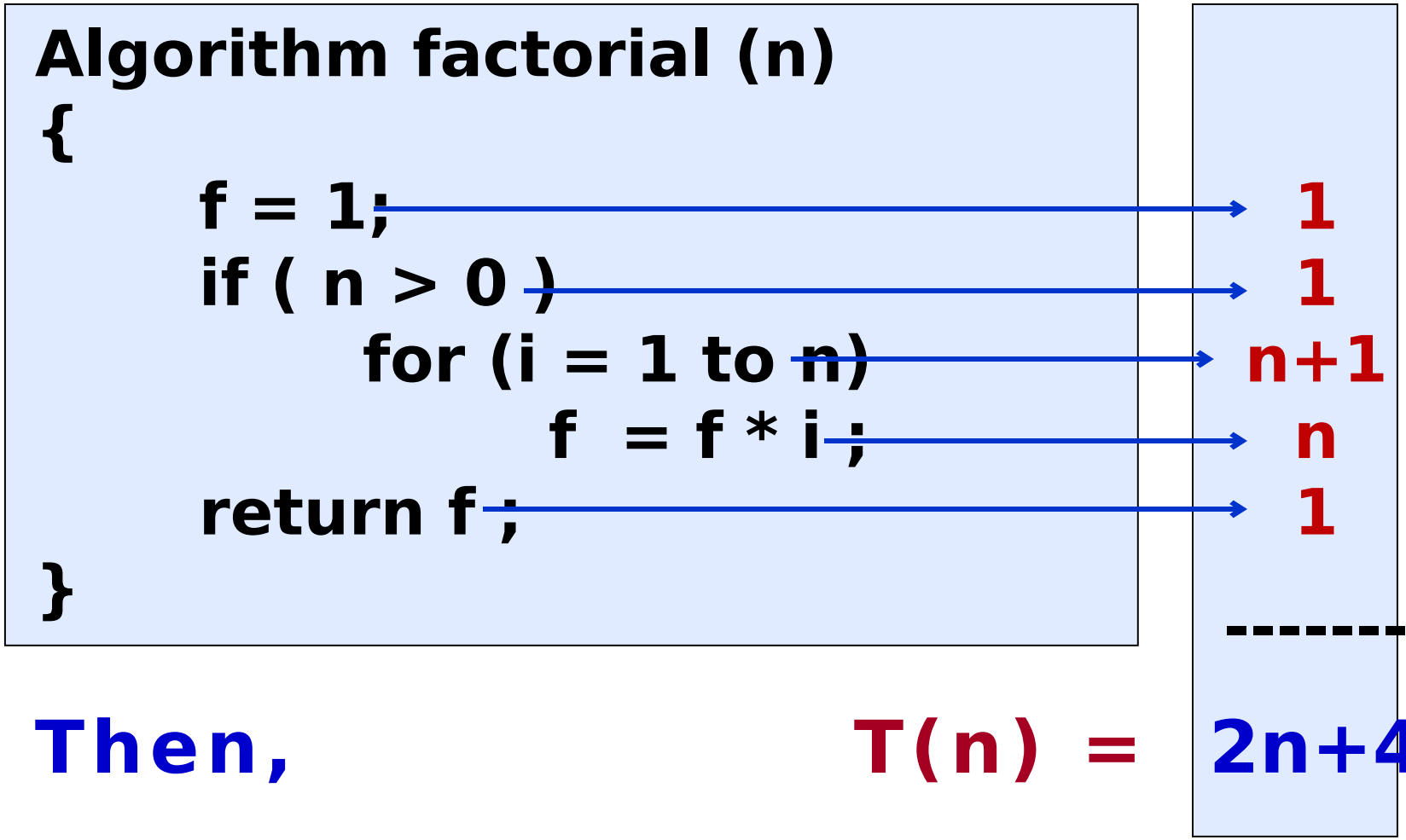
NUMBER OF OPERATIONS

Click to add text

We measure **T(n)** of an algorithm by counting the number of operations.

- Each "simple" operation (+, -, <=, , >=) is one operation.
- Loops and function calls are ***not*** simple operations, but depend upon the size of the data and the contents of a function. We do not want ***”sort"*** to be a single step operation.
- Each memory access is one operation.

Example



Then, $T(n) = 2n+4$

TIME COMPLEXITY

It can be difficult to determine the exact number of operations performed by an algorithm, *Though it is often still useful to do so*

An alternative to counting all instructions is to focus on an algorithm's *basic operation*

- The *basic operation* is the instruction that is executed the most number of times in an algorithm
- The number of times that the *basic operation* is executed is usually

Example

```
Algorithm factorial (n)
{
                for (i = 1 to n)
                        f  = f * i ;
        return f ;
}
```

The basic operation

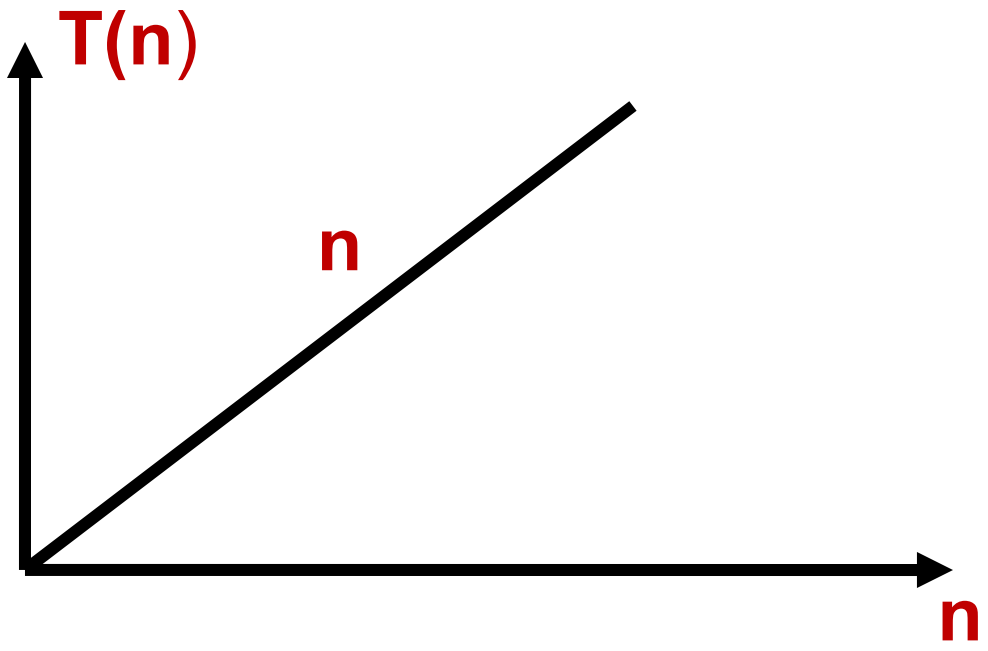
n

Then, approximately T(n) n



COMPLEXITY OF THE FACTORIAL ALGORITHM

$T(n) = n$



BEST-CASE, AVERAGE-CASE, WORST-CASE

- The efficiencies of some algorithms may differ significantly for inputs of the same size.
- For such algorithms, we need to distinguish between the worst-case, average-case, and best-case efficiencies.
- Some algorithms are same for all three cases – *ex, find the maximum value in an unsorted array.*



Click to add text

EXAMPLE (2):

```
Algorithm linearSearch (a, key, n)
{
   for (i = 0 to n-1)
        if (a[i ]== key) return  i;
   return -1;
}
```

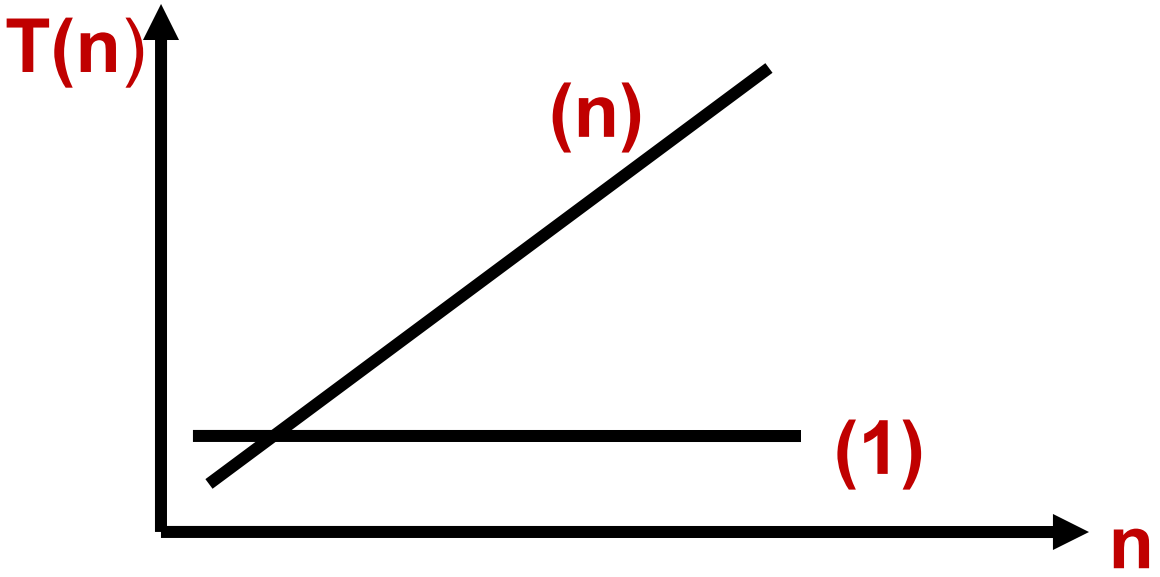
T(n) = number of array element comparisons.

Best case: T(n) = 1

Worst case: T(n) = n

COMPLEXITY OF THE LINEAR SEARCH ALGORITHM

T(n) = 1 in the best case. T(n) = n in the worst case



BEST-CASE, AVERAGE-CASE, WORST-CASE

Average case not easy to be computed

- NOT the average of worst and best case
- Expected number of basic operations considered as a random variable under some assumption about the probability distribution of all possible inputs of size n
- There are many important algorithms for which the average case efficiency is much better than the worst-case efficiency



EIGHT GROWTH FUNCTIONS

Eight functions O(n) that occur frequently in the analysis of algorithms (in order of increasing rate of growth relative to n):

- Constant ≈ ***1***
- Logarithmic ≈ log ***n***
- Linear ≈ ***n***
- Log Linear ≈ ***n*** log ***n***
- Quadratic ≈ $\boldsymbol{n}^2$
- Cubic ≈ $\boldsymbol{n}^3$
- Exponential ≈ $\mathbf{2}^n$
- Factorial≈ ***n***!

GROWTH RATES COMPARED

	n=1	n=2	n=4	n=8	n=16	n=32
1	1	1	1	1	1	1
log n	0	1	2	3	4	5
n	1	2	4	8	16	32
nlogn	0	2	8	24	64	160
n^2	1	4	16	64	256	1024
n^3	1	8	64	512	4096	32768
2^n	2	4	16	256	65536	4294967296
n!	1	2	24	40320	20.9T	Don’t ask!

GROWTH RATES COMPARED

To appreciate the qualitative difference among the orders of growth functions, consider how they react to if the value of their argument *n* *is duplicated*.

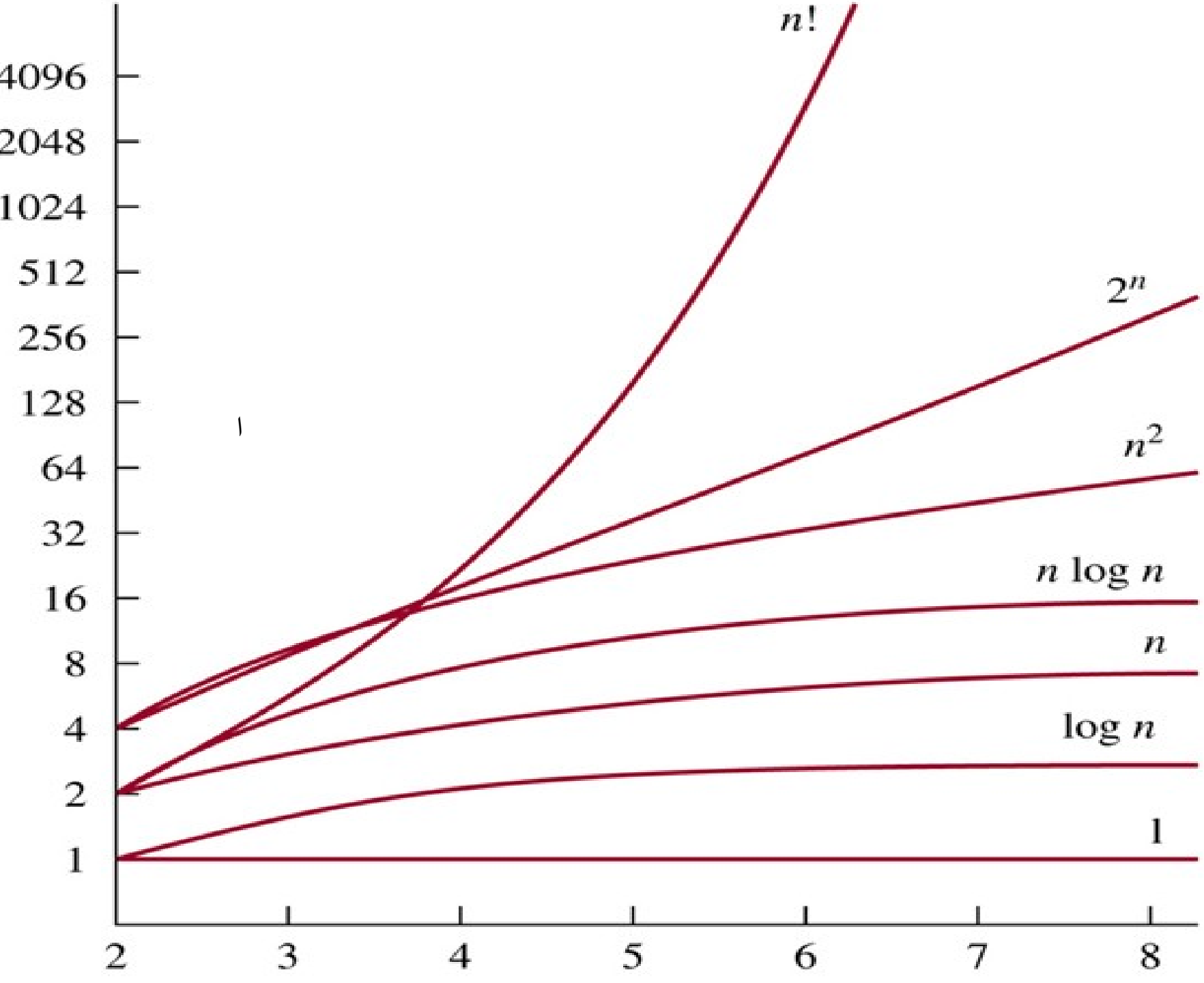
- ***log*$_2$** *n increases in* value by just ***1*** (because $log_2 2n = log_2 2 + log_2 n = 1 + log_2 n$)
- ***n*²** *increase* ***fourfold*** , because $(2n)^2 = 4n^2$
- ***2*^{n}** *gets* ***squared***, because $2^{2n} = (2^n)^2$

REFERENCES

Anany Levitin, Introduction to the design and analysis of algorithms, 2nd Edition.

Chapter 1, sections 1.1, 1.2

Chapter 2, Sections 2.1,2.2


n!
2^n
n^2
n log n
n
log n
1
4096
2048
1024
512
256
128
64
32
16
8
4
2
1
2
3
4
5
6
7
8