Design & Analysis of Algorithms

Lecture # 02

Muhammad Waqas Sheikh

Today’s Topic

- Problem :- Given number M and N, Find the Greatest Common Divisor.

- Algorithm 1: Simple school level algorithm.
- Euclid Algorithm.

Middle School Algorithm for GCD

Middle-school procedure

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*)

Is this an algorithm?

Euclid's Algorithm

Problem: Find gcd(m,n), the greatest common divisor of two nonnegative

Examples: gcd(60,24) = 12, gcd(60,0) = 60, gcd(0,0) = ?

Euclid (m,n){
while *m does not divides n*
r ← *n mod m*
n← *m*
m ← *r*
end while
return *m*
}

- m=434   n=966
- #######first Iteration#######
- 434 divides 966 (No)
- r=966 mod 434 = 98
- n=434
- m=98
- ############# 2nd Iteration########
- r= 434 mod 98= 98x4+42
- n=98
- m=42
- #############3Rd Iteration ########
- r= 98 mod 42 =14
- n=42
- m=14
- `

Proof of Correctness

Examples: gcd(60,24) = 12, gcd(60
gcd(0,0) = ?

Euclid (m,n){
while *m does not divides n*
r ← *n mod m*
n← *m*
m ← *r*
end while
return *m*
}

- If m divides n then GCD (m,n) =m
- Otherwise GCD(m,n)=GCD(n mod m, m)
- The value of m and n changing in every iteration.
- If you want to calculate GCD(m,n) you have to calculate GCD(n mod m, m)
- We will maintain such integers m and n in each iteration whose GCD will be the GCD of original m and n
- Loop Invariant

Proof of Termination

- Why and when we will exit from the loop?
- Compare value after one Iteration.
- Is n mod m will be smaller then m?
- Always decreasing at least by one
- How much ? Will it become zoro?
- Loop terminates.

Types of Analysis

- Priory Analysis
  - Algorithms
  - Independent of Language
  - Hardware Independence
  - Time and Space as function
- Posteriori Analysis
  - Program
  - Language dependent
  - Machine oriented (Hardware dependent)
  - Calculate time and space by program execution.

Problem and Instance

- **Specification of valid input and what are the acceptable outputs for each valid input**
  - **Computing GCD of two numbers.**
    - **GCD(48,36)**
  - **Finding shortest path in map.**
    - **Karachi to Quetta**
  - **Meaning of word in dictionary**
    - **“Evolution”**
  - **Given an image and determine any disease**
    - **“X-Ray”**

Problem and Instance

- A value X is an input instance for problem P, if X is a valid input as per specification.
- Can be single value or set of value.
- Size if instance number of bits required to store instance

Mathematical Model

- **Mathematical Model of Computer (Generic)**
  - **Executer Algorithm on that model (Mentally)**
- **Time required to execute instruction.**
- **What is input data? (Execution time depends on data)**
- **how does this model relate to real computer?**
  - **If Model terribly different, conclusions will not be applicable on real computers.**

Mathematical Model

- Random Access Machine (RAM)
  - Processor +Memory
- Memory is Collection of elements, accessible Randomly.
- Assigning operation.
- Arithmetic operations.
- Jumps and conditional jumps
- Pointer instruction
- Array operations
- Functional calls

Complex Instruction

- We assume that each basic operation takes the same constant time to execute.
- The RAM model does a good job of describing the computational power of most modern (nonparallel) machines.
- x=y;
- Z=a +b;
- Z=w[10]
- X=a+b*c-d

Relation to Reality

- Real computers are complicated
  - Different kinds of memories
- Pipeline
- Memory to register copy instruction
- Computation is done only in register
- Intelligent Compilers

Relation to Reality

- Idealized model of computer
- It differs from real computer
- We are not talking abut compiler and what actually executed on machine
- What ever apply on our idealized model can be mimic by real computer.

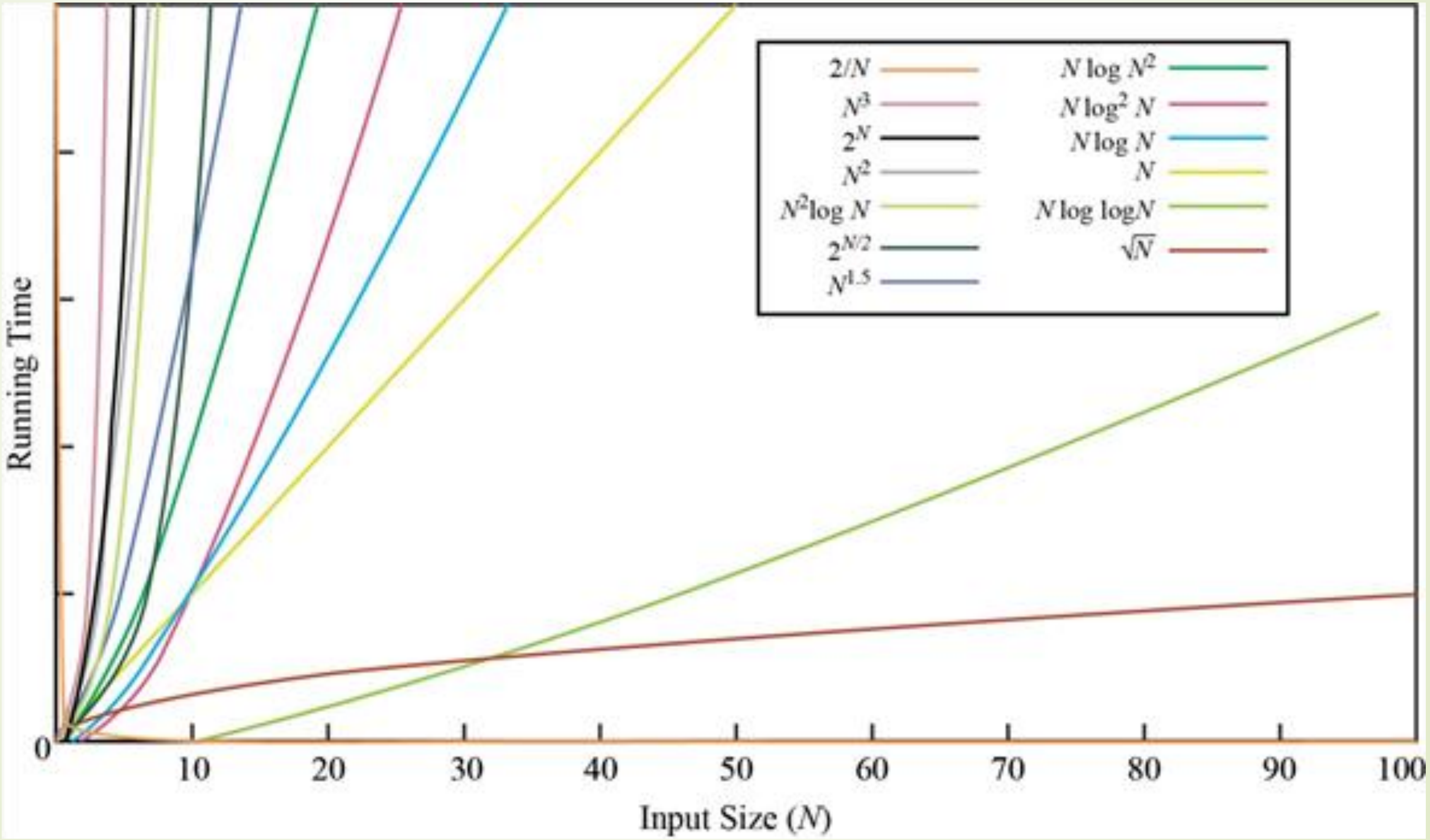
General Analysis Strategy

- $T(n)$: Maximum time taken by algorithm to solve any instance of size n.
- $T(n)$: Measure of goodness
  - How good or bad indicated by function
- The function will indicate how good is algorithm.
- Conservative Definition "Worst Case"

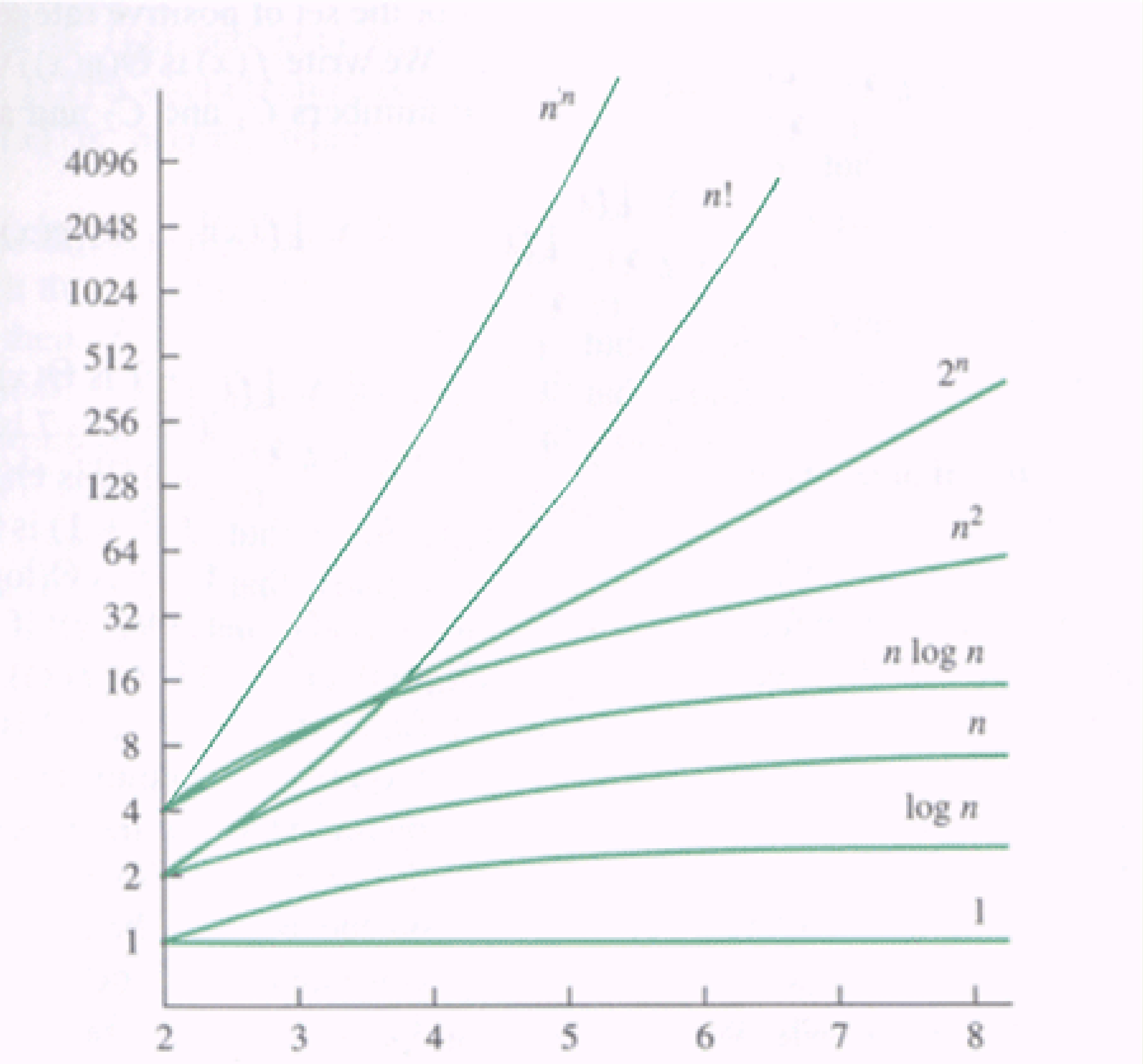
General Analysis Strategy

- Form of $T(n)$ (independent from machine)
  - "Linear", "Cubic", "Quadratic" etc.
- Bounds of $T(n)$:
  - Upper bound, Lower bound
- Large n is Important
  - As n become larger and larger which algorithm is better

Growth of Function $T(n)$



Growth of Function $T(n)$



Running Time Analysis

- The running time depends upon the input size, e.g. n
- Different inputs of the same size may result in different running time.
- Criteria for measuring running time.
  - Worst-Case Time (Maximum running time over legal input of size n)

Running Time analysis

- Criteria Worst-case time:
- Let I denote an input instance
- let |I| denote its length, and
- let T(I) denote the running time of algorithm on input I

$$T_{worst}(n) = \max_{|I|=n} T(I)$$

Running Time analysis

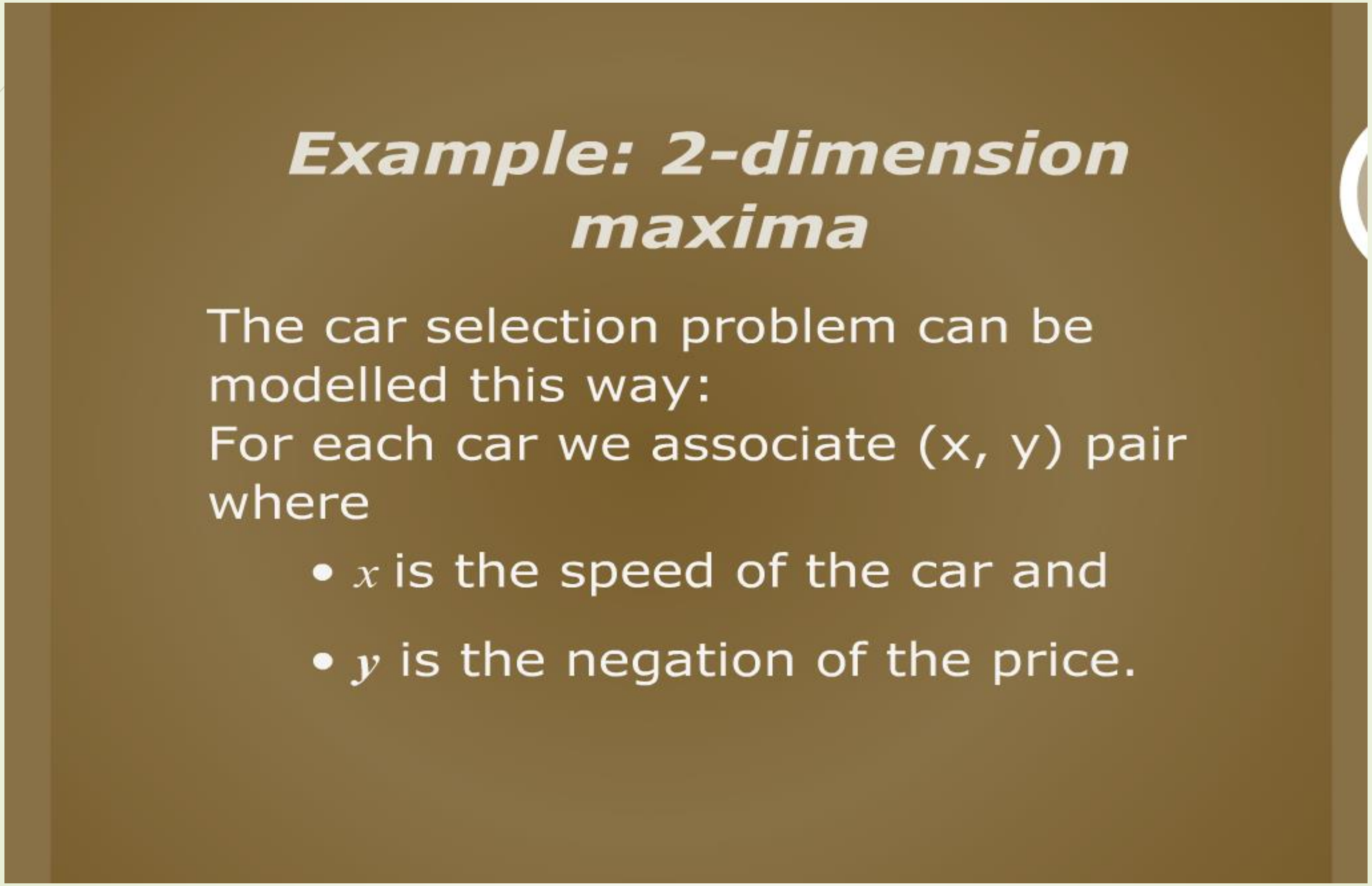
- Average- Case time is the average running time over all inputs of size n.
- Let p(I) denote the probability of seeing this input.
- Average case time is the weighted sum of running times with weights being the probabilities.

$$T_{avg}(n) = \sum_{|I|=n} p(I)T(I)$$

Running Time Analysis

- We will almost always work with worst-case time
- Average-case time is more difficult to calculate; it is difficult to specify probability distribution on inputs
- Worst-case time will specify an upper limit on the running time.

Example: 2-dimension maxima



Example: 2-dimension maxima

Example: 2-dimension maxima

- High y value means a cheap car and low y means expensive car.
- Think of y as the money left in your pocket after you have paid for the car.
- Maximal points correspond to the fastest and cheapest cars.

Example: 2-dimension maxima

Example: 2-dimension maxima

2-dimensional Maxima:

- Given a set of points $P = \{p_1, p_2, \ldots, p_n\}$ in 2-space,
- output the set of maximal points of P,
- i.e.,those points p_i such that p_i is not dominated by any other point of P.

Example: 2-dimension maxima

Example: 2-dimension maxima

Here is how we might model this as a formal problem.

- Let a point p in 2-dimensional space be given by its integer coordinates, $p = (p.x, p.y)$.
- A point p is said to be *dominated* by point q if $p.x \le q.x$ and $p.y \le q.y$.

Example: 2-dimension maxima

Example: 2-dimension maxima

- Given a set of n points, P = $\{p1, p2, \ldots, pn\}$ in 2-space a point is said to be *maximal* if it is not dominated by any other point in P.

Example: 2-dimension maxima

Example: 2-dimension maxima

The car selection problem can be modelled this way:
For each car we associate (x, y) pair where

- x is the speed of the car and
- y is the negation of the price.

Example: 2-dimension maxima

Example: 2-dimension maxima

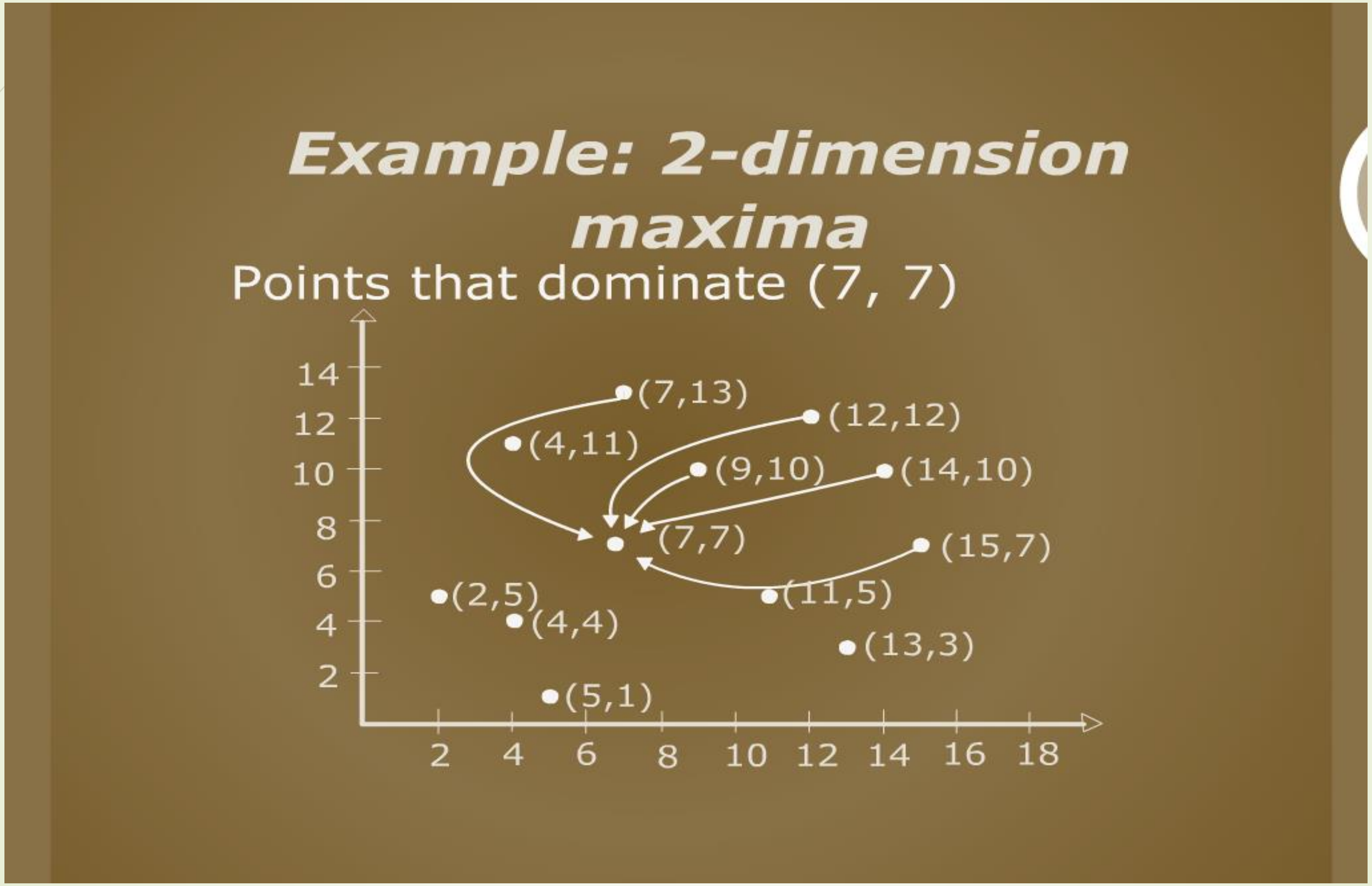
```
MAXIMA(int n, Point P[1 . . . n])
for i ← 1 to n
do maximal ← true
    for j ← 1 to n
    do
        if (i ≠ j) and (P[i].x ≤ P[j].x) and
        (P[i].y ≤ P[j].y)
          then maximal ← false; break
```

Example: 2-dimension maxima

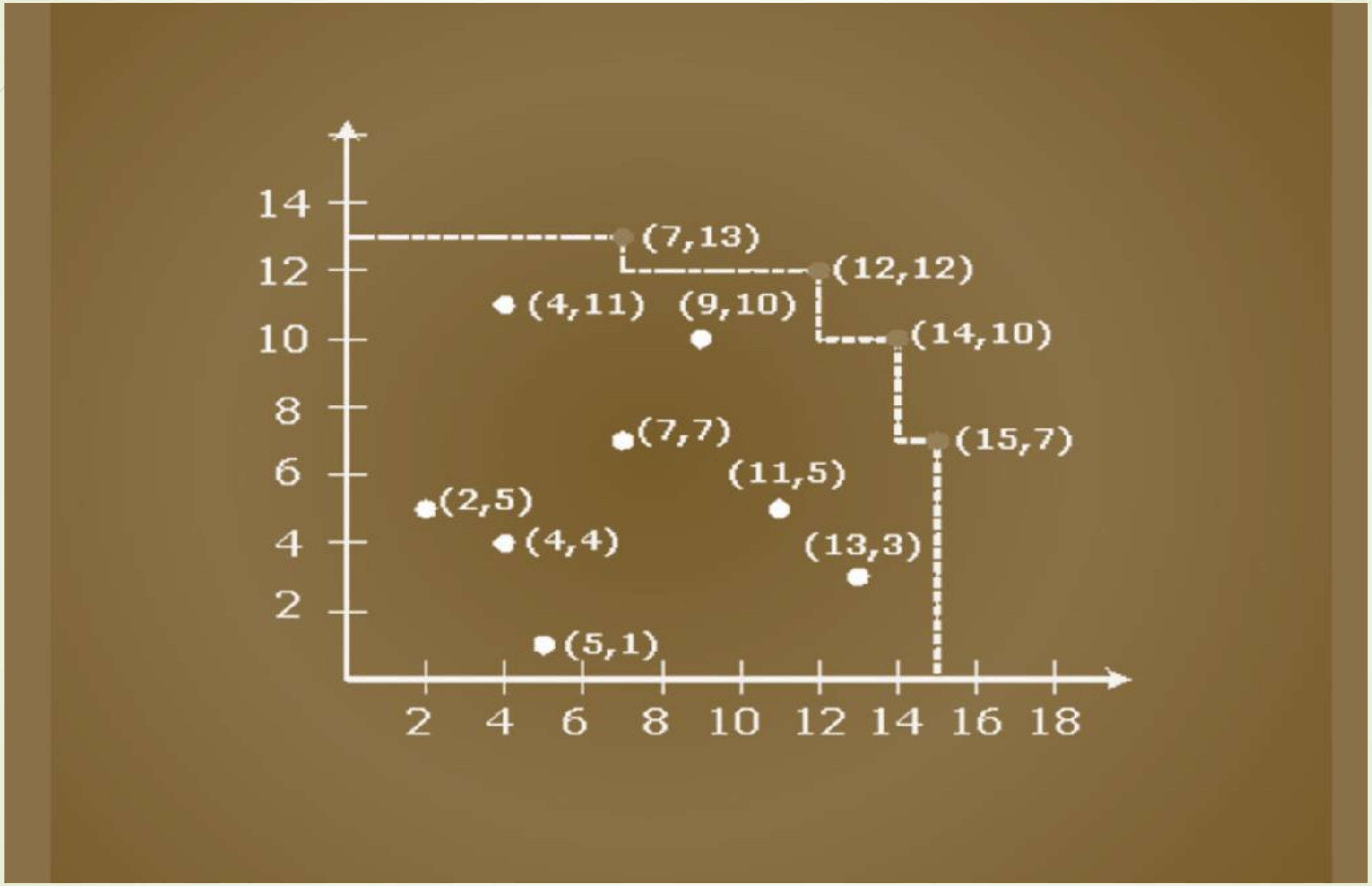
Example: 2-dimension maxima

```
for j ← 1 to n
do
    if (i ≠ j) and (P[i].x ≤ P[j].x) and
    (P[i].y ≤ P[j].y)
      then maximal ← false; break
  if (maximal = true)
      then output P[i]
```

Example: 2-dimension maxima



Graph



14
12
10
8
6
4
2
(7,13)
(12,12)
(4,11)
(9,10)
(14,10)
(7,7)
(15,7)
(11,5)
(2,5)
(4,4)
(13,3)
(5,1)
2 4 6 8 10 12 14 16 18

Analysis of 2-dimension maxima

Example: 2-dimension maxima

```
MAXIMA(int n, Point P[1 . . . n])
for i ← 1 to n                                  n times
do maximal ← true
    for j ← 1 to n                              n times
    do
        if (i ≠ j) & (P[i].x ≤ P[j].x) &
        (P[i].y ≤ P[j].y)                       4 accesses
          then maximal ← false; break
```

Example: 2-dimension maxima

```
do maximal ← true
    for j ← 1 to n                      n times
    do
        if (i ≠ j) & (P[i].x ≤ P[j].x) &
        (P[i].y ≤ P[j].y)               4 accesses
            then maximal ← false; break
    if maximal
        then output P[i].x,
                    P[i].y              2 accesses
```

Analysis of 2-dimension maxima

Analysis of 2-dimension maxima

Worst-case running time:

Pair of nested summations, one for i-loop and the other for the j-loop

$$T(n) = \sum_{i=1}^{n}\left(2 + \sum_{j=1}^{n} 4\right)$$

Analysis of 2-dimension maxima

Analysis of 2-dimension maxima

Worst-case running time:

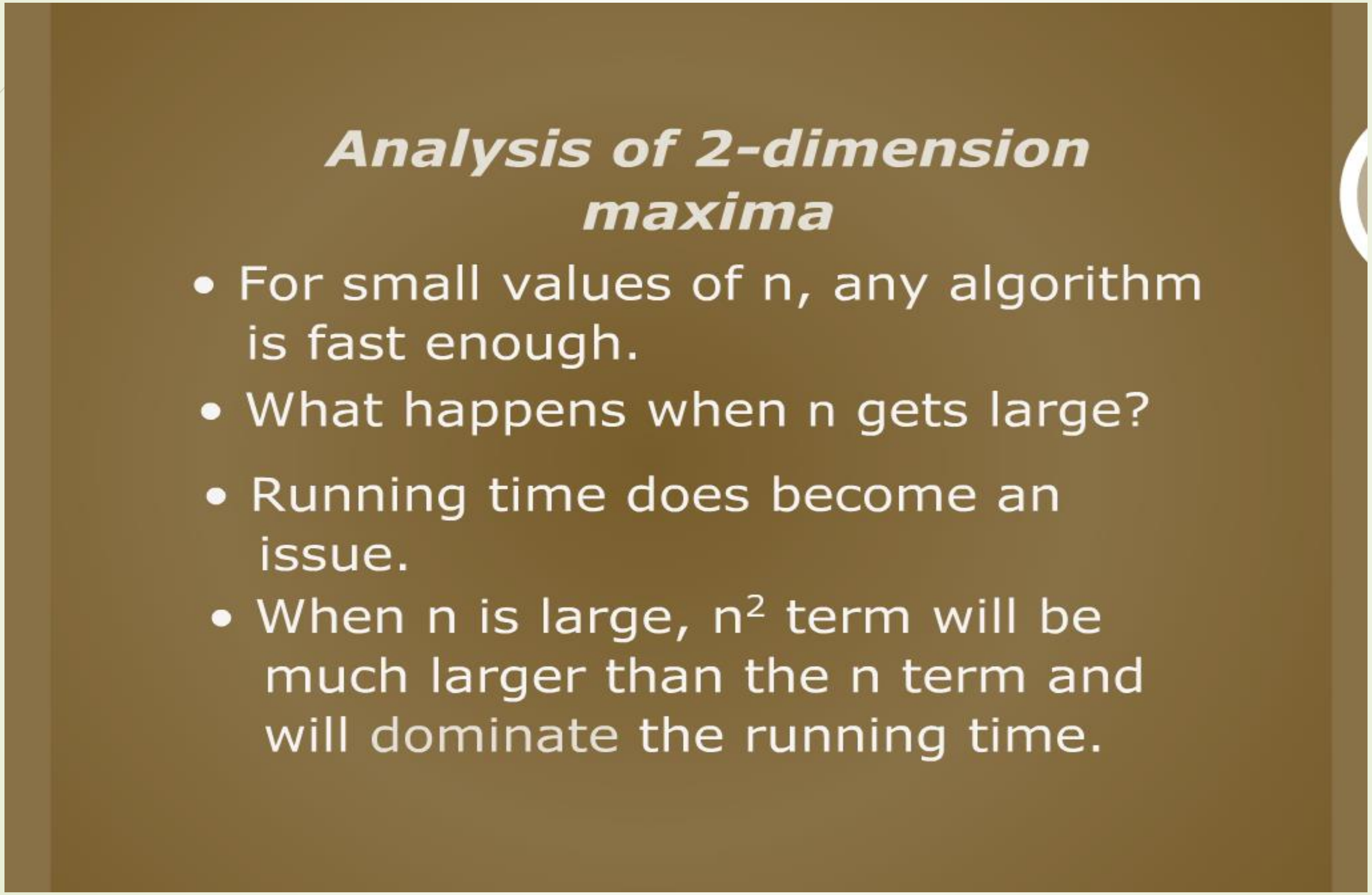
$$T(n) = \sum_{i=1}^{n} \left(2 + \sum_{j=1}^{n} 4 \right)$$

$\left(\sum_{j=1}^{n} 4 \right) = 4n$, and so

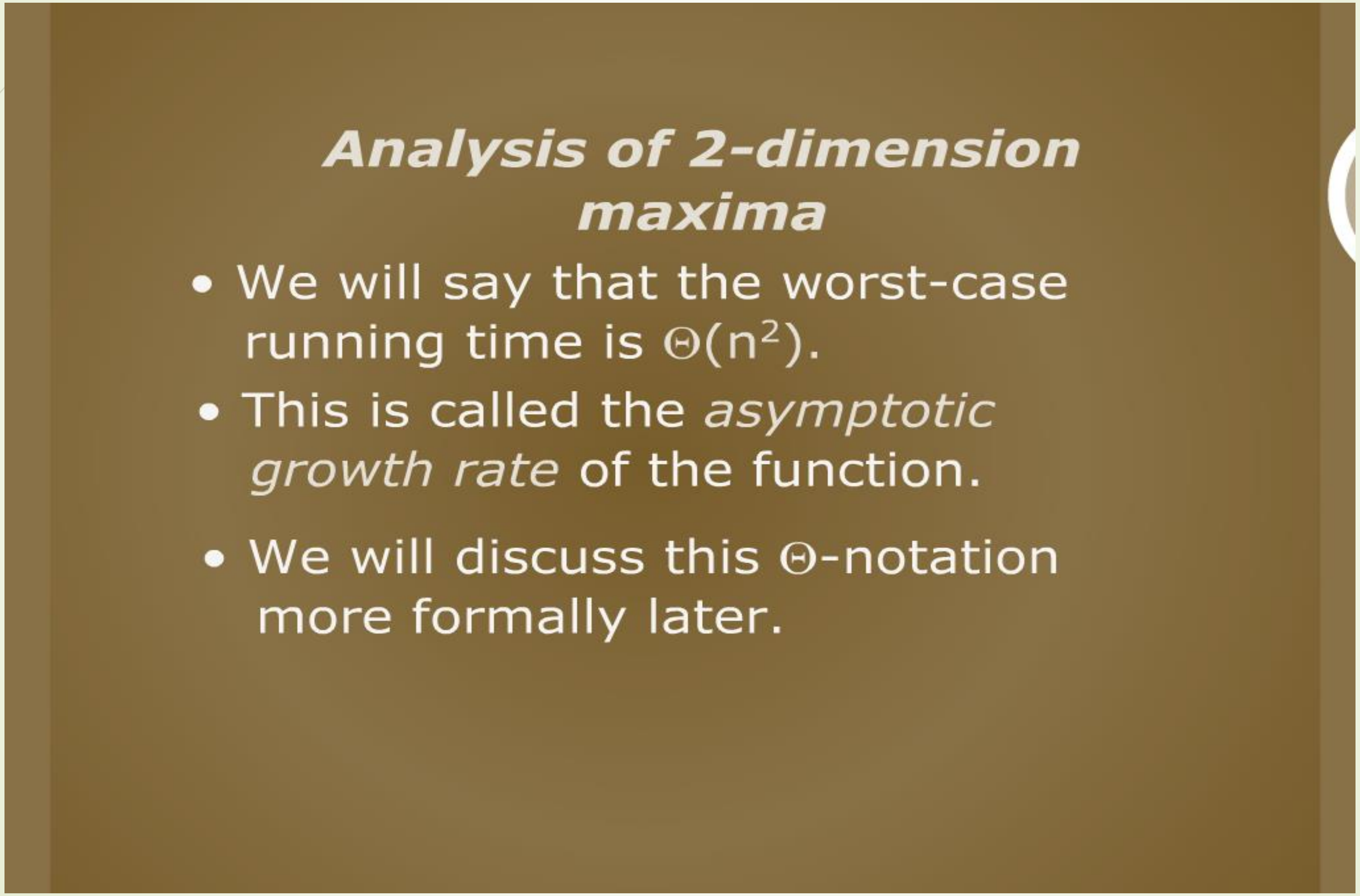
$$T(n) = \sum_{i=1}^{n} \left(4n + 2 \right)$$

$$= (4n + 2)n = 4n^2 + 2n$$

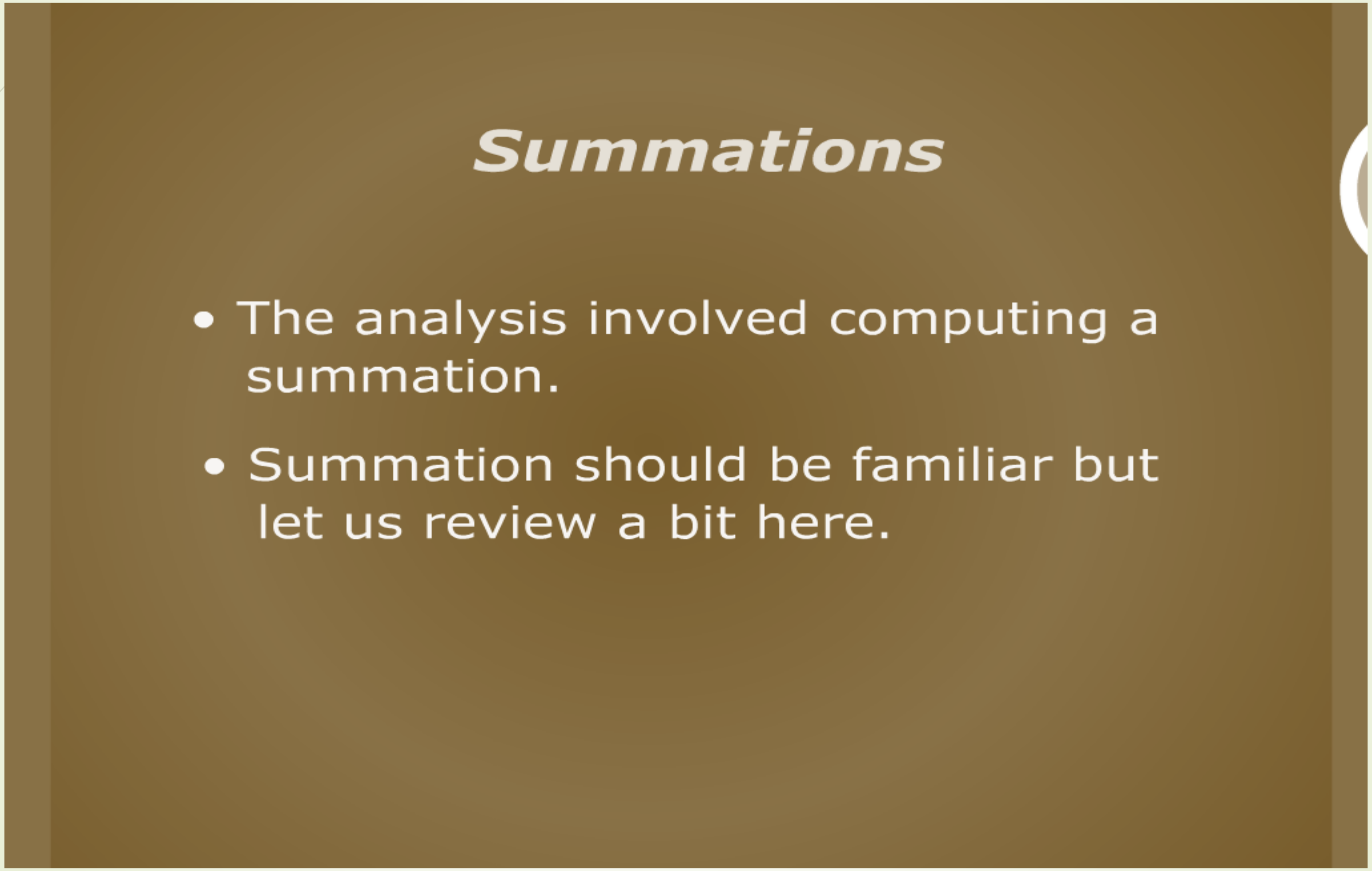
Analysis of 2-dimension maxima



Analysis of 2-dimension maxima



Summations


Summations
• The analysis involved computing a summation.
• Summation should be familiar but let us review a bit here.

Summations

Summations

- Given a finite sequence of values

  $a_1, a_2, \ldots, a_n,$

- their sum $a_1 + a_2 + \ldots + a_n$ is expressed in summation notation as

$$\sum_{i=1}^{n} a_i$$

Summations

Summations

Some facts about summation:

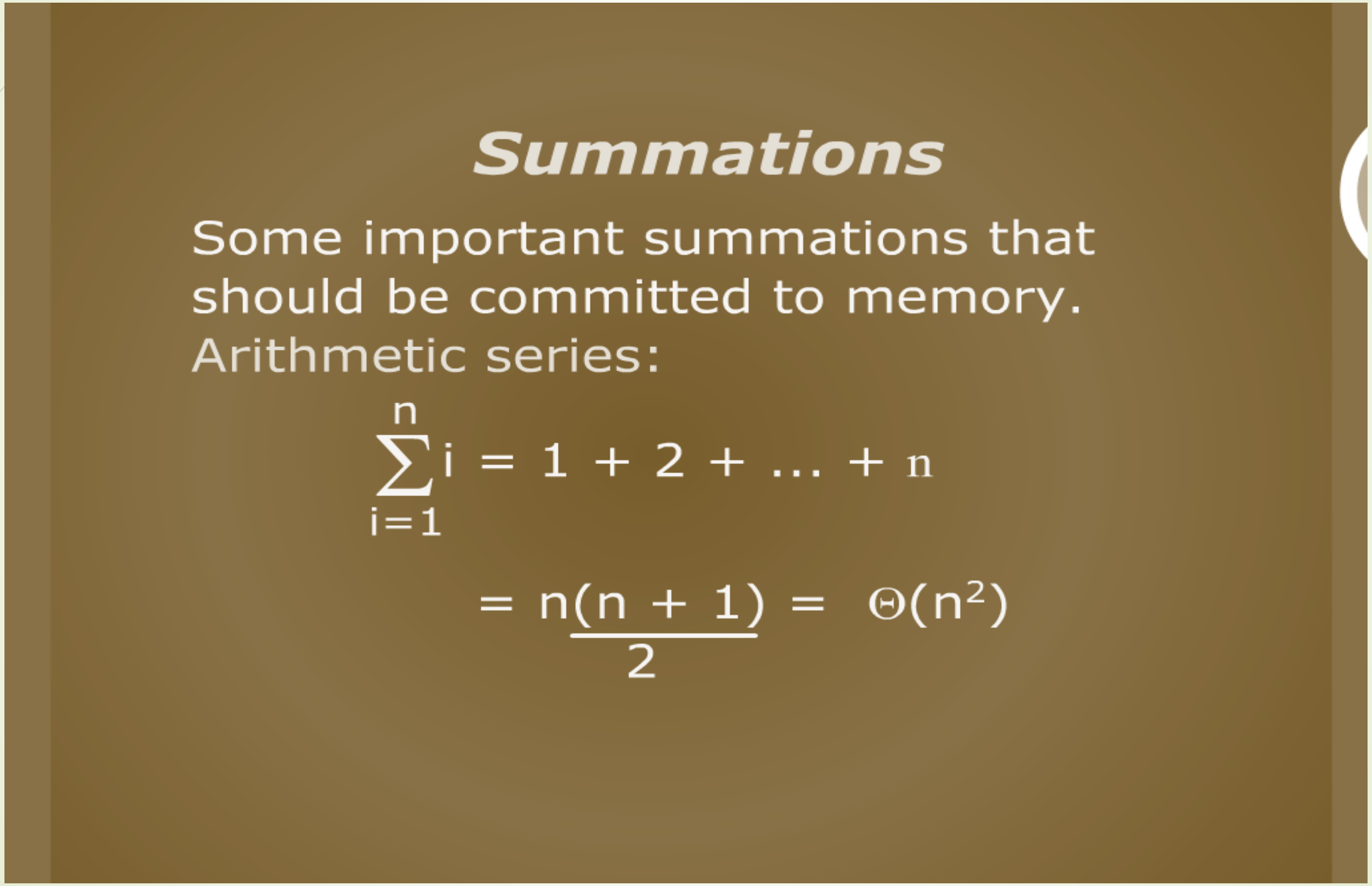
- If c is a constant

$$\sum_{i=1}^{n} ca_i = c\sum_{i=1}^{n} a_i$$

and

$$\sum_{i=1}^{n} (a_i + b_i) = \sum_{i=1}^{n} a_i + \sum_{i=1}^{n} b_i$$

Summations


Summations
Some important summations that should be committed to memory.
Arithmetic series:
$\sum_{i=1}^{n} i = 1 + 2 + \ldots + n$
$= \frac{n(n + 1)}{2} = \Theta(n^2)$

Summations

Summations

Quadratic series:

$$\sum_{i=1}^{n} i^2 = 1 + 4 + 9 + \ldots + n^2$$

$$= \frac{2n^3 + 3n^2 + n}{6} = \Theta(n^3)$$

Summations

Summations

Geometric series:

$$\sum_{i=1}^{n} x^i = 1 + x + x^2 + \ldots + x^n$$

$$= \frac{x^{(n+1)} - 1}{x - 1} = \Theta(n^2)$$

Summations

Summations

Harmonic series: For $n \geq 0$

$$H_n = \sum_{i=1}^{n} \frac{1}{i}$$

$$= 1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{n} \approx \ln n$$

$$= \Theta(\ln n)$$

A harder Example

A Harder Example

NESTED-LOOPS()

```
for i ←1 to n
do
    for j← 1 to 2i
    do k = j . . .
       while (k ≥0)
       do k = k - 1 . . .
```

How do we analyze the running time of an algorithm that has complex nested loop?

A harder Example

A Harder Example

NESTED-LOOPS()

```
for i ←1 to n
do
    for j← 1 to 2i
    do k = j . . .
       while (k ≥0)
       do k = k - 1 . . .
```

To convert loops into summations, we work from inside-out.

A harder Example

A Harder Example

NESTED-LOOPS()

```
for i ←1 to n
do
    for j← 1 to 2i
    do k = j . . .
        while (k ≥0)
        do k = k - 1 . . .
```

The answer is we write out the loops as summations and then solve the summations.

A harder Example

A Harder Example

NESTED-LOOPS()

```
for i ←1 to n
do
    for j← 1 to 2i
    do k = j . . .
        while (k ≥0)
        do k = k - 1 . . .
```

The answer is we write out the loops as summations and then solve the summations.

A harder Example

A Harder Example

NESTED-LOOPS()

```
for i ←1 to n
do    for j← 1 to 2i
         do k = j
            while (k ≥0) ◁
            do k = k - 1
```

- Consider the inner most *while* loop.
- It is executed for $k=j, j-1, j-2, \ldots, 0$.

A harder Example

A Harder Example

NESTED-LOOPS()

```
for i ← 1 to n
do    for j ← 1 to 2i
          do k = j
             while (k ≥ 0)
             do k = k - 1
```

- Time spent inside the while loop is constant.
- Let I() be the time spent in the while loop.

Thus:

$$I(j) = \sum_{k=0}^{j} 1 = j + 1$$

A harder Example

A Harder Example

NESTED-LOOPS()

```
for i ←1 to n
  do   for j← 1 to 2i ◁
         do k = j
            while (k ≥0)
            do k = k - 1
```

- Consider the *middle* for loop.
- It's running time is determined by i.

A harder Example

A Harder Example

NESTED-LOOPS()

```
for i ← 1 to n
do   for j ← 1 to 2i ◁
         do k = j
            while (k ≥ 0)
            do k = k - 1
```

- Let M() be the time spent in the for loop:

$$M(i) = \sum_{j=1}^{2i} I(j)$$

A harder Example

A Harder Example

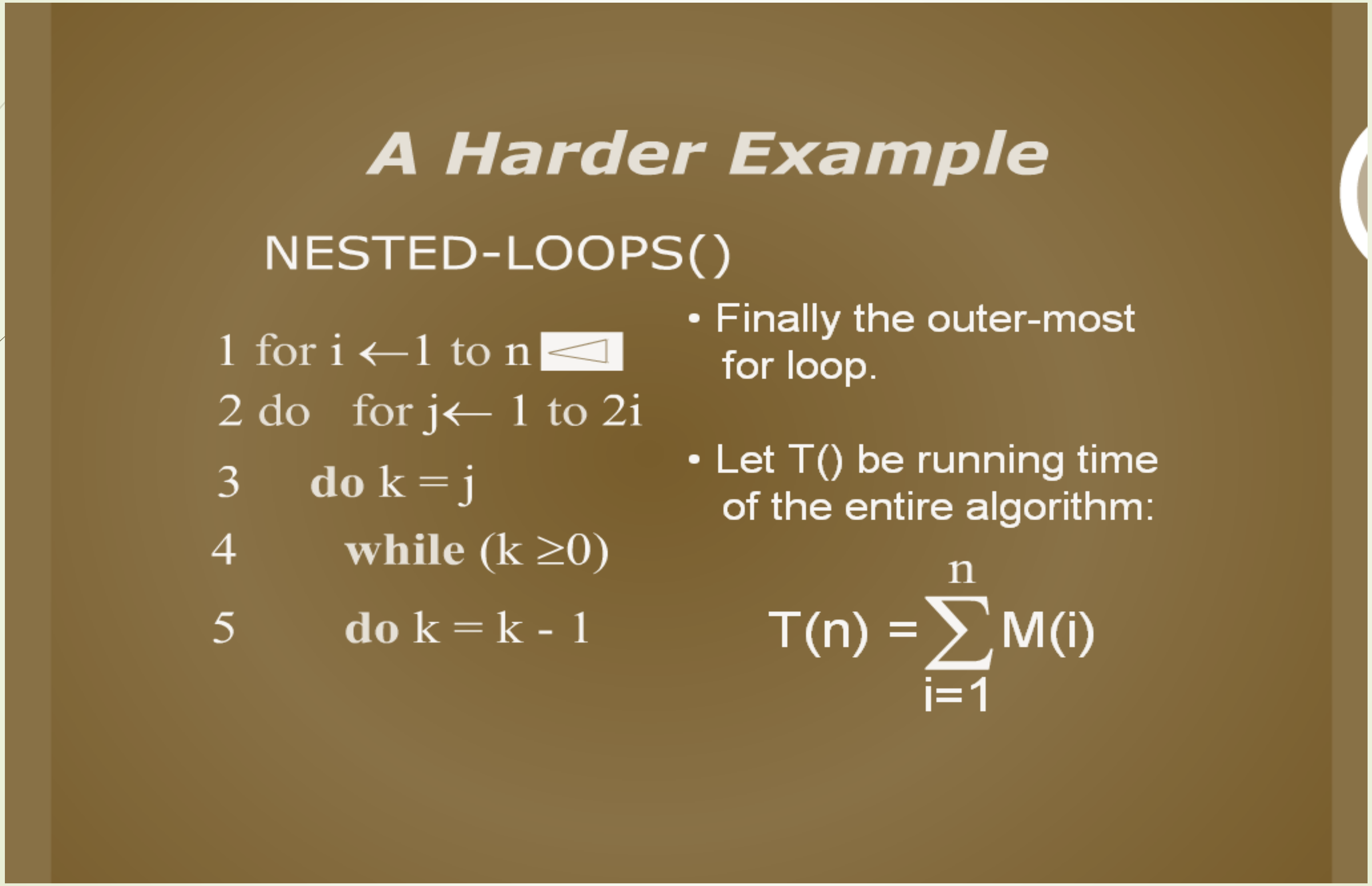
$$M(i) = \sum_{j=1}^{2i} I(j) = \sum_{j=1}^{2i} (j + 1)$$

$$= \sum_{j=1}^{2i} j + \sum_{j=1}^{2i} 1$$

$$= \frac{2i(2i + 1)}{2} + 2i$$

$$= 2i^2 + 3i$$

A harder Example



A harder Example

A Harder Example

$$T(n) = \sum_{i=1}^{n} M(i) = \sum_{i=1}^{n} (2i^2 + 3i)$$

$$= \sum_{i=1}^{n} 2i^2 + \sum_{i=1}^{n} 3i$$

$$= 2\frac{2n^3 + 3n^2 + n}{6} + 3\,\frac{n(n+1)}{2}$$

$$= \frac{4n^3 + 15n^2 + 11n}{6}$$

$$= \Theta(n^3)$$