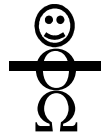


(Lecture 1) Introduction

CSE 548: Analysis of Algorithms

Rezaul A. Chowdhury
Department of Computer Science
SUNY Stony Brook
Fall 2017



Asymptotic Stickman
(by Aleksandra Patrzałek, SUNY Buffalo)

Some Mostly Useless Information

- **Lecture Time:** MW 7:00 pm - 8:20 pm
- **Location:** Javits Lecture Hall 102, West Campus
- **Instructor:** Rezaul A. Chowdhury
- **Office Hours:** MW 5:00 pm - 6:30 pm
239 Computer Science
- **Email:** rezaul@cs.stonybrook.edu
- **TA:** TBA
- **Class Webpage:**
<http://www3.cs.stonybrook.edu/~rezaul/CSE548-F17.html>

Prerequisites

- **Required:** Some background (undergrad level) in the design and analysis of algorithms and data structures
 - fundamental data structures (e.g., lists, stacks, queues and arrays)
 - discrete mathematical structures (e.g., graphs, trees, and their adjacency lists & adjacency matrix representations)
 - fundamental programming techniques (e.g., recursion, divide-and-conquer, and dynamic programming)
 - basic sorting and searching algorithms
 - fundamentals of asymptotic analysis (e.g., $O(\cdot)$, $\Omega(\cdot)$ and $\Theta(\cdot)$ notations)
- **Required:** Some background in programming languages (C / C++)

Topics to be Covered

- The following topics will be covered (hopefully)
- recurrence relations and divide-and-conquer algorithms
 - dynamic programming
 - graph algorithms (e.g., network flow)
 - amortized analysis
 - advanced data structures (e.g., Fibonacci heaps)
 - cache-efficient and external-memory algorithms
 - high probability bounds and randomized algorithms
 - parallel algorithms and multithreaded computations
 - NP-completeness and approximation algorithms
 - the alpha technique (e.g., disjoint sets, partial sums)
 - FFT (Fast Fourier Transforms)

(Lecture 1) Introduction

Grading Policy

- Four Homework Problem Sets
(highest score 15%, lowest score 5%, and others 10% each): 40%
- Two Exams (higher one 30%, lower one 15%): 45%
 - Midterm (in-class): Oct 11
 - Final (in-class): Nov 29
- Scribe note (one lecture): 10%
- Class participation & attendance: 5%

Textbooks

Required

- Thomas Cormen, Charles Leiserson, Ronald Rivest, and Clifford Stein.
Introduction to Algorithms (3rd Edition), MIT Press, 2009.

Recommended

- Sanjoy Dasgupta, Christos Papadimitriou, and Umesh Vazirani.
Algorithms (1st Edition), McGraw-Hill, 2006.
- Jon Kleinberg and Éva Tardos.
Algorithm Design (1st Edition), Addison Wesley, 2005.
- Rajeev Motwani and Prabhakar Raghavan.
Randomized Algorithms (1st Edition), Cambridge University Press, 1995.
- Vijay Vazirani.
Approximation Algorithms, Springer, 2010.
- Joseph JáJá.
An Introduction to Parallel Algorithms (1st Edition), Addison Wesley, 1992.

What is an Algorithm?

An algorithm is a **well-defined computational procedure** that solves a well-specified computational problem.

It accepts a value or set of values as **input**, and produces a value or set of values as **output**

Example: *mergesort* solves the **sorting problem** specified as a relationship between the input and the output as follows.

Input: A sequence of n numbers $\langle a_1, a_2, \dots, a_n \rangle$.

Output: A permutation $\langle a'_1, a'_2, \dots, a'_n \rangle$ of the input sequence such that $a'_1 \leq a'_2 \leq \dots \leq a'_n$.

Desirable Properties of an Algorithm

- ✓ Correctness
 - Designing an incorrect algorithm is straightforward
- ✓ Efficiency
 - Efficiency is easily achievable if we give up on correctness

Surprisingly, sometimes incorrect algorithms can also be useful!

- If you can control the error rate
- Tradeoff between correctness and efficiency:
 - Randomized algorithms
(Monte Carlo: always efficient but sometimes incorrect,
Las Vegas: always correct but sometimes inefficient)
 - Approximation algorithms
(always incorrect!)

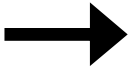
(Lecture 1) Introduction

How Do You Measure Efficiency?

We often want algorithms that can use the available resources efficiently.

Some measures of efficiency

- time complexity
- space complexity
- cache complexity
- I/O complexity
- energy usage
- number of processors/cores used
- network bandwidth



Goal of Algorithm Analysis

Goal is to predict the behavior of an algorithm without implementing it on a real machine.

But predicting the exact behavior is not always possible as there are too many influencing factors.

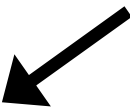
Runtime on a serial machine is the most commonly used measure.

We need to model the machine first in order to analyze runtimes.

But an exact model will make the analysis too complicated!
So we use an approximate model (e.g., assume unit-cost Random Access Machine model or RAM model).

We may need to approximate even further: e.g., for a sorting algorithm we may count the comparison operations only.

So the predicted running time will only be an approximation!



Performance Bounds

- **worst-case complexity:** maximum complexity over all inputs of a given size
- **average complexity:** average complexity over all inputs of a given size
- **amortized complexity:** worst-case bound on a sequence of operations
- **expected complexity:** for algorithms that make random choices during execution (randomized algorithms)
- **high-probability bound:** when the probability that the complexity holds is $\geq 1 - \frac{c}{n^\alpha}$ for input size n , positive constant c and some constant $\alpha \geq 1$



Searching in a Sorted Grid

$n = 2^m - 1$

$A[1:n, 1:n]$

2	5	10	11	20	22	26	30	31	31	34	34	37	40	45
4	6	15	18	27	30	31	38	39	40	42	42	44	48	48
7	9	16	21	27	31	39	41	41	41	48	50	55	55	59
8	13	22	22	27	34	40	45	48	50	50	50	58	58	65
11	14	29	31	35	36	41	49	55	55	58	59	61	62	67
15	20	30	32	39	42	42	50	59	60	60	60	65	68	71
16	21	35	41	41	43	44	58	62	69	69	70	70	70	75
20	22	36	41	42	50	50	61	65	70	75	75	76	78	78
21	25	37	44	44	59	60	62	70	72	75	76	78	78	80
22	28	39	48	50	61	62	66	71	75	75	76	78	81	85
26	31	40	56	65	65	65	69	75	78	78	80	82	82	88
29	34	41	61	66	69	72	72	78	80	80	81	82	84	88
31	41	45	66	67	70	72	72	78	82	84	84	85	85	91
32	45	49	67	67	72	78	80	81	86	85	86	86	88	95
40	55	56	70	71	75	81	81	81	86	86	88	91	93	98

You are given an $n \times n$ grid $A[1:n, 1:n]$, where $n = 2^m - 1$ for some integer $m > 0$.

Each grid cell contains a number.

The numbers in each row are sorted in non-decreasing order from left to right.

The numbers in each column are sorted in non-decreasing order from top to bottom.

(Lecture 1) Introduction

Searching in a Sorted Grid (Algorithm 1)

$n = 2^m - 1$

$A[1:n, 1:n]$

2	5	10	11	20	22	26	30	31	31	34	34	37	40	45
4	6	15	18	27	30	31	38	39	40	42	42	44	48	48
7	9	16	21	27	31	39	41	41	41	48	50	55	55	59
8	13	22	22	27	34	40	45	48	50	50	50	58	58	65
11	14	29	31	35	36	41	49	55	55	58	59	61	62	67
15	20	30	32	39	42	42	50	59	60	60	60	65	68	71
16	21	35	41	41	43	44	58	62	69	69	70	70	70	75
20	22	36	41	42	50	50	61	65	70	75	75	76	78	78
21	25	37	44	44	59	60	62	70	72	75	76	78	78	80
22	28	39	48	50	61	62	66	71	75	75	76	78	81	85
26	31	40	56	65	65	65	69	75	78	78	80	82	82	88
29	34	41	61	66	69	72	72	78	80	80	81	82	84	88
31	41	45	66	67	70	72	72	78	82	84	84	85	85	91
32	45	49	67	67	72	78	80	81	86	85	86	86	88	95
40	55	56	70	71	75	81	81	81	86	86	88	91	93	98

ALGORITHM 1 (SEARCH FOR x):

Scan the entire grid row by row until either x is found, or you are done scanning the entire grid.

Let $Q_1(n)$ = number of comparisons performed on an $n \times n$ grid.

Then $Q_1(n) \leq n^2$

Searching in a Sorted Grid (Algorithm 2)

$n = 2^m - 1$

$A[1:n, 1:n]$

2	5	10	11	20	22	26	30	31	31	34	34	37	40	45
4	6	15	18	27	30	31	38	39	40	42	42	44	48	48
7	9	16	21	27	31	39	41	41	41	48	50	55	55	59
8	13	22	22	27	34	40	45	48	50	50	50	58	58	65
11	14	29	31	35	36	41	49	55	55	58	59	61	62	67
15	20	30	32	39	42	42	50	59	60	60	60	65	68	71
16	21	35	41	41	43	44	58	62	69	69	70	70	70	75
20	22	36	41	42	50	50	61	65	70	75	75	76	78	78
21	25	37	44	44	59	60	62	70	72	75	76	78	78	80
22	28	39	48	50	61	62	66	71	75	75	76	78	81	85
26	31	40	56	65	65	65	69	75	78	78	80	82	82	88
29	34	41	61	66	69	72	72	78	80	80	81	82	84	88
31	41	45	66	67	70	72	72	78	82	84	84	85	85	91
32	45	49	67	67	72	78	80	81	86	85	86	86	88	95
40	55	56	70	71	75	81	81	81	86	86	88	91	93	98

ALGORITHM 2 (SEARCH FOR x):

Let y be the number at the center of the grid (i.e., at the intersection of the mid row and mid column).

- $y = x$: you found the item

Searching in a Sorted Grid (Algorithm 2)

$n = 2^m - 1$

$A[1:n, 1:n]$

2	5	10	11	20	22	26	30	31	31	34	34	37	40	45
4	6	15	18	27	30	31	38	39	40	42	42	44	48	48
7	9	16	21	27	31	39	41	41	41	48	50	55	55	59
8	13	22	22	27	34	40	45	48	50	50	50	58	58	65
11	14	29	31	35	36	41	49	55	55	58	59	61	62	67
15	20	30	32	39	42	42	50	59	60	60	60	65	68	71
16	21	35	41	41	43	44	58	62	69	69	70	70	70	75
20	22	36	41	42	50	50	61	65	70	75	75	76	78	78
21	25	37	44	44	59	60	62	70	72	75	76	78	78	80
22	28	39	48	50	61	62	66	71	75	75	76	78	81	85
26	31	40	56	65	65	65	69	75	78	78	80	82	82	88
29	34	41	61	66	69	72	72	78	80	80	81	82	84	88
31	41	45	66	67	70	72	72	78	82	84	84	85	85	91
32	45	49	67	67	72	78	80	81	86	85	86	86	88	95
40	55	56	70	71	75	81	81	81	86	86	88	91	93	98

ALGORITHM 2 (SEARCH FOR x):

Let y be the number at the center of the grid (i.e., at the intersection of the mid row and mid column).

- $y = x$: you found the item

Searching in a Sorted Grid (Algorithm 2)

$n = 2^m - 1$

$A[1:n, 1:n]$

2	5	10	11	20	22	26	30	31	31	34	34	37	40	45
4	6	15	18	27	30	31	38	39	40	42	42	44	48	48
7	9	16	21	27	31	39	41	41	41	48	50	55	55	59
8	13	22	22	27	34	40	45	48	50	50	50	58	58	65
11	14	29	31	35	36	41	49	55	55	58	59	61	62	67
15	20	30	32	39	42	42	50	59	60	60	60	65	68	71
16	21	35	41	41	43	44	58	62	69	69	70	70	70	75
20	22	36	41	42	50	50	61	65	70	75	75	76	78	78
21	25	37	44	44	59	60	62	70	72	75	76	78	78	80
22	28	39	48	50	61	62	66	71	75	75	76	78	81	85
26	31	40	56	65	65	65	69	75	78	78	80	82	82	88
29	34	41	61	66	69	72	72	78	80	80	81	82	84	88
31	41	45	66	67	70	72	72	78	82	84	84	85	85	91
32	45	49	67	67	72	78	80	81	86	85	86	86	88	95
40	55	56	70	71	75	81	81	81	86	86	88	91	93	98

ALGORITHM 2 (SEARCH FOR x):

Let y be the number at the center of the grid (i.e., at the intersection of the mid row and mid column).

- $y = x$: you found the item
- $y > x$: the item cannot be in A_{22} , R_2 and C_2 . Search for x in R_1 and C_1 , and recursively in A_{11} , A_{12} & A_{21} .

4

(Lecture 1) Introduction

Searching in a Sorted Grid (Algorithm 2)

ALGORITHM 2 (SEARCH FOR x):

Let y be the number at the center of the grid (i.e., at the intersection of the mid row and mid column).

- $y = x$: you found the item
- $y > x$: the item cannot be in A_{22}, R_2 and C_2 .
Search for x in R_1 and C_1 , and recursively in A_{11}, A_{12} & A_{21} .
- $y < x$: the item cannot be in A_{11}, R_1 and C_1 .
Search for x in R_2 and C_2 , and recursively in A_{12}, A_{21} & A_{22} .

Searching in a Sorted Grid (Algorithm 2)

Let $Q_2(n)$ = number of comparisons performed on an $n \times n$ grid.

$$\text{Then } Q_2(n) \leq 1 + 2\left(\frac{n+1}{2} - 1\right) + 3Q_2\left(\frac{n+1}{2} - 1\right)$$

$$\Rightarrow Q_2(n) \leq 3Q_2\left(\frac{n+1}{2} - 1\right) + n$$

Solving: $Q_2(n) \leq 2(n+1)^{\log_2 3}$
 $\leq 2(n+1)^{1.6}$

Searching in a Sorted Grid (Algorithm 3)

ALGORITHM 3 (SEARCH FOR x):

Starting from the top row
perform a *binary search* for x in
each row until x is found.

Binary search in row i of A :

```

left ← 1
right ← n
while left ≤ right do
    mid ←  $\frac{\text{left} + \text{right}}{2}$ 
    if A[i, mid] = x then
        return "item found"
    else if A[i, mid] < x then
        left ← mid + 1
    else right ← mid - 1
end while
return "item not found"

```

Searching in a Sorted Grid (Algorithm 3)

ALGORITHM 3 (SEARCH FOR x):

Starting from the top row
perform a *binary search* for x in
each row until x is found.

Binary search in row i of A :

```

1 left ← 1
2 right ← n
3 while left ≤ right do
4     mid ←  $\frac{\text{left} + \text{right}}{2}$ 
5     if A[i, mid] = x then
6         return "item found"
7     else if A[i, mid] < x then
8         left ← mid + 1
9     else right ← mid - 1
10 end while
11 return "item not found"

```


Diagram illustrating the search process for $x = 35$ in a sorted matrix A of size $n = 2^m - 1$. The matrix is shown with rows and columns indexed from 1 to n .

2	5	10	11	20	22	26	30	31	31	34	34	37	40	45
4	6	15	18	27	30	31	38	39	40	42	42	44	48	48
7	9	16	21	27	31	39	41	41	41	48	50	55	55	59
8	13	22	22	27	34	40	45	48	50	50	50	58	58	65
11	14	29	31	35	36	41	49	55	55	58	59	61	62	67
15	20	30	32	39	42	42	50	59	60	60	60	65	68	71
16	21	35	41	41	43	44	58	62	69	69	70	70	70	75
20	22	36	41	42	50	50	61	65	70	75	75	76	78	78
21	25	37	44	44	59	60	62	70	72	75	76	78	78	80
22	28	39	48	50	61	62	66	71	75	75	76	78	81	85
26	31	40	56	65	65	65	69	75	78	78	80	82	82	88
29	34	41	61	66	69	72	72	78	80	80	81	82	84	88
31	41	45	66	67	70	72	72	78	82	84	84	85	85	91
32	45	49	67	67	72	78	80	81	86	85	86	86	88	95
40	55	56	70	71	75	81	81	81	86	86	88	91	93	98

The search process is shown with arrows indicating the path taken to find $x = 35$. The path starts at the top row (row 1) and moves down to row 11, then right to column 11, and finally down to row 21, where the value 35 is found.

Algorithm 3 (SEARCH FOR x):

```

Starting from the top row
perform a binary search for  $x$  in
each row until  $x$  is found.

Binary search in row  $i$  of  $A$ :
  left  $\leftarrow 1$ 
  right  $\leftarrow n$ 
  while left  $\leq$  right do
    mid  $\leftarrow \frac{\text{left} + \text{right}}{2}$ 
    if  $A[i, \text{mid}] = x$  then
      return "item found"
    else if  $A[i, \text{mid}] < x$  then
      left  $\leftarrow \text{mid} + 1$ 
    else right  $\leftarrow \text{mid} - 1$ 
  end while
  return "item not found"
  
```



$n = 2^m - 1$

$A[1:n, 1:n]$

2	5	10	11	20	22	26	30	31	31	34	34	37	40	45
4	6	15	18	27	30	31	38	39	40	42	42	44	48	48
7	9	16	21	27	31	39	41	41	41	48	50	55	55	59
8	13	22	22	27	34	40	45	48	50	50	50	58	58	65
11	14	29	31	35	36	41	49	55	55	58	59	61	62	67
15	20	30	32	39	42	42	50	59	60	60	60	65	68	71
16	21	35	41	41	43	44	58	62	69	69	70	70	70	75
20	22	36	41	42	50	50	61	65	70	75	75	76	78	78
21	25	37	44	44	59	60	62	70	72	75	76	78	78	80
22	28	39	48	50	61	62	66	71	75	75	76	78	81	85
26	31	40	56	65	65	65	69	75	78	78	80	82	82	88
29	34	41	61	66	69	72	72	78	80	80	81	82	84	88
31	41	45	66	67	70	72	72	78	82	84	84	85	85	91
32	45	49	67	67	72	78	80	81	86	85	86	86	88	95
40	55	56	70	71	75	81	81	81	86	86	88	91	93	98

ALGORITHM 3 (SEARCH FOR x):

Starting from the top row
perform a *binary search* for x in
each row until x is found.

Binary search in row i of A :

```

left ← 1
right ← n
while left ≤ right do
    mid ←  $\frac{\text{left} + \text{right}}{2}$ 
    if  $A[i, \text{mid}] = x$  then
        return "item found"
    else if  $A[i, \text{mid}] < x$  then
        left ← mid + 1
    else right ← mid - 1
end while
return "item not found"

```

Search for $x = 35$

$n = 2^m - 1$

A[1:n, 1:n]

2	5	10	11	20	22	26	30	31	31	34	34	37	40	45
4	6	15	18	27	30	31	38	39	40	42	42	44	48	48
7	9	16	21	27	31	39	41	41	41	48	50	55	55	59
8	13	22	22	27	34	40	45	48	50	50	50	58	58	65
11	14	29	31	35	36	41	49	55	55	58	59	61	62	67
15	20	30	32	39	42	42	50	59	60	60	60	65	68	71
16	21	35	41	41	43	44	58	62	69	69	70	70	70	75
20	22	36	41	42	50	50	61	65	70	75	75	76	78	78
21	25	37	44	44	59	60	62	70	72	75	76	78	78	80
22	28	39	48	50	61	62	66	71	75	75	76	78	81	85
26	31	40	56	65	65	65	69	75	78	78	80	82	82	88
29	34	41	61	66	69	72	72	78	80	80	81	82	84	88
31	41	45	66	67	70	72	72	78	82	84	84	85	85	91
32	45	49	67	67	72	78	80	81	86	85	86	86	88	95
40	55	56	70	71	75	81	81	81	86	86	88	91	93	98

$n = 2^m - 1$

Search for $x = 35$



Diagram illustrating the search process for $x = 35$ in a 2D array A of size $n \times n$. The array is shown with rows and columns indexed from 1 to n . The search path is highlighted in blue, starting from the top row and moving down to the row containing x .

The array A is defined as:

2	5	10	11	20	22	26	30	31	31	34	34	37	40	45
4	6	15	18	27	30	31	38	39	40	42	42	44	48	48
7	9	16	21	27	31	39	41	41	41	48	50	55	55	59
8	13	22	22	27	34	40	45	48	50	50	50	58	58	65
11	14	29	31	35	36	41	49	55	55	58	59	61	62	67
15	20	30	32	39	42	42	50	59	60	60	60	65	68	71
16	21	35	41	41	43	44	58	62	69	69	70	70	70	75
20	22	36	41	42	50	50	61	65	70	75	75	76	78	78
21	25	37	44	44	59	60	62	70	72	75	76	78	78	80
22	28	39	48	50	61	62	66	71	75	75	76	78	81	85
26	31	40	56	65	65	65	69	75	78	78	80	82	82	88
29	34	41	61	66	69	72	72	78	80	80	81	82	84	88
31	41	45	66	67	70	72	72	78	82	84	84	85	85	91
32	45	49	67	67	72	78	80	81	86	85	86	86	88	95
40	55	56	70	71	75	81	81	81	86	86	88	91	93	98

The search path for $x = 35$ is highlighted in blue, starting from the top row and moving down to the row containing x .

Search for $x = 35$

Diagram illustrating the search for $x = 35$ in a sorted matrix A of size $n \times n$. The matrix is partitioned into four quadrants by a horizontal line at row $n/2$ and a vertical line at column $n/2$.

The matrix A is shown with rows and columns indexed from 1 to n . The search for $x = 35$ is performed by comparing x with the element at $A[n/2, n/2]$ (which is 30 in the diagram). Since $x > A[n/2, n/2]$, the search continues in the bottom-right quadrant.

The search process is summarized by the following steps:

- Start at the top row (row 1).
- Perform a binary search for x in each row until x is found.
- Binary search in row i of A :
 - Initialize $left \leftarrow 1$ and $right \leftarrow n$.
 - While $left \leq right$ do:
 - Calculate $mid \leftarrow \frac{left + right}{2}$.
 - If $A[i, mid] = x$ then return "item found".
 - Else if $A[i, mid] < x$ then $left \leftarrow mid + 1$.
 - Else $right \leftarrow mid - 1$.
 - End while.
 - Return "item not found".



Diagram illustrating the search for $x = 35$ in a sorted matrix A of size $n \times n$. The matrix is shown with rows and columns indexed from 1 to n . The search process is as follows:

- Start at the top row (row 1).
- Perform a binary search for x in each row until x is found.
- Binary search in row i of A :
 - left $\leftarrow 1$
 - right $\leftarrow n$
 - while left \leq right do
 - mid $\leftarrow \frac{\text{left} + \text{right}}{2}$
 - if $A[i, \text{mid}] = x$ then
 - return "item found"
 - else if $A[i, \text{mid}] < x$ then
 - left $\leftarrow \text{mid} + 1$
 - else right $\leftarrow \text{mid} - 1$
 - end while
 - return "item not found"

The matrix A is shown with the following values (rows 1 to 16):

2	5	10	11	20	22	26	30	31	31	34	34	37	40	45
4	6	15	18	27	30	31	38	39	40	42	42	44	48	48
7	9	16	21	27	31	39	41	41	41	48	50	55	55	59
8	13	22	22	27	34	40	45	48	50	50	50	58	58	65
11	14	29	31	35	36	41	49	55	55	58	59	61	62	67
15	20	30	32	39	42	42	50	59	60	60	60	65	68	71
16	21	35	41	41	43	44	58	62	69	69	70	70	70	75
20	22	36	41	42	50	50	61	65	70	75	75	76	78	78
21	25	37	44	44	59	60	62	70	72	75	76	78	81	85
22	28	39	48	50	61	62	66	71	75	75	76	78	81	80
26	31	40	56	65	65	65	69	75	78	78	80	82	82	88
29	34	41	61	66	69	72	72	78	80	80	81	82	84	88
31	41	45	66	67	70	72	72	78	82	84	84	85	85	91
32	45	49	67	67	72	78	80	81	86	85	86	86	88	95
40	55	56	70	71	75	81	81	81	86	86	88	91	93	98

Diagram illustrating the search for $x = 35$ in a sorted matrix A of size $n \times n$. The matrix is partitioned into four quadrants by a horizontal line at row $n/2 - 1$ and a vertical line at column $n/2 - 1$.

The matrix A is shown with rows and columns indexed from 1 to n . The search for $x = 35$ is performed by comparing x with the element at $A[n/2, n/2]$ (which is 30 in the diagram). Since $x > A[n/2, n/2]$, the search continues in the bottom-right quadrant.

The search process is summarized by the following steps:

- Start at the top row (row 1).
- Perform a binary search for x in each row until x is found.
- Binary search in row i of A :
 - Initialize $left \leftarrow 1$ and $right \leftarrow n$.
 - While $left \leq right$ do:
 - Calculate $mid \leftarrow \frac{left + right}{2}$.
 - If $A[i, mid] = x$ then return "item found".
 - Else if $A[i, mid] < x$ then $left \leftarrow mid + 1$.
 - Else $right \leftarrow mid - 1$.
 - End while.
 - Return "item not found".



$n = 2^m - 1$

$n = 2^m - 1$

2	5	10	11	20	22	26	30	31	31	34	34	37	40	45
4	6	15	18	27	30	31	38	39	40	42	42	44	48	48
7	9	16	21	27	31	39	41	41	41	48	50	55	55	59
8	13	22	22	27	34	40	45	48	50	50	50	58	58	65
11	14	29	31	35	36	41	49	55	55	58	59	61	62	67
15	20	30	32	39	42	42	50	59	60	60	60	65	68	71
16	21	35	41	41	43	44	58	62	69	69	70	70	70	75
20	22	36	41	42	50	50	61	65	70	75	75	76	78	78
21	25	37	44	44	59	60	62	70	72	75	76	78	78	80
22	28	39	48	50	61	62	66	71	75	75	76	78	81	85
26	31	40	56	65	65	65	69	75	78	78	80	82	82	88
29	34	41	61	66	69	72	72	78	80	80	81	82	84	88
31	41	45	66	67	70	72	72	78	82	84	84	85	85	91
32	45	49	67	67	72	78	80	81	86	85	86	86	88	95
40	55	56	70	71	75	81	81	81	86	86	88	91	93	98

Search for $x = 35$

ALGORITHM 3 (SEARCH FOR x):

Starting from the top row
perform a *binary search* for x in
each row until x is found.

Binary search in row i of A :

```

left ← 1
right ← n
while left ≤ right do
    mid ←  $\frac{\text{left} + \text{right}}{2}$ 
    if  $A[i, \text{mid}] = x$  then
        return "item found"
    else if  $A[i, \text{mid}] < x$  then
        left ← mid + 1
    else right ← mid - 1
end while
return "item not found"

```


Diagram illustrating the search process for $x = 35$ in a sorted matrix A of size $n = 2^m - 1$.

The matrix A is shown with rows indexed from 1 to n and columns indexed from 1 to n . The matrix is sorted row-wise and column-wise.

The search process is shown with the following steps:

- Start at the top row (row 1).
- Compare x with the first element of the row ($A[1,1] = 2$).
- Since $x > 2$, move to the next row (row 2).
- Compare x with the first element of the row ($A[2,1] = 4$).
- Since $x > 4$, move to the next row (row 3).
- Compare x with the first element of the row ($A[3,1] = 7$).
- Since $x > 7$, move to the next row (row 4).
- Compare x with the first element of the row ($A[4,1] = 8$).
- Since $x > 8$, move to the next row (row 5).
- Compare x with the first element of the row ($A[5,1] = 11$).
- Since $x > 11$, move to the next row (row 6).
- Compare x with the first element of the row ($A[6,1] = 15$).
- Since $x > 15$, move to the next row (row 7).
- Compare x with the first element of the row ($A[7,1] = 16$).
- Since $x > 16$, move to the next row (row 8).
- Compare x with the first element of the row ($A[8,1] = 20$).
- Since $x > 20$, move to the next row (row 9).
- Compare x with the first element of the row ($A[9,1] = 22$).
- Since $x > 22$, move to the next row (row 10).
- Compare x with the first element of the row ($A[10,1] = 26$).
- Since $x > 26$, move to the next row (row 11).
- Compare x with the first element of the row ($A[11,1] = 30$).
- Since $x > 30$, move to the next row (row 12).
- Compare x with the first element of the row ($A[12,1] = 34$).
- Since $x > 34$, move to the next row (row 13).
- Compare x with the first element of the row ($A[13,1] = 38$).
- Since $x > 38$, move to the next row (row 14).
- Compare x with the first element of the row ($A[14,1] = 42$).
- Since $x > 42$, move to the next row (row 15).
- Compare x with the first element of the row ($A[15,1] = 46$).
- Since $x > 46$, move to the next row (row 16).
- Compare x with the first element of the row ($A[16,1] = 50$).
- Since $x > 50$, move to the next row (row 17).
- Compare x with the first element of the row ($A[17,1] = 54$).
- Since $x > 54$, move to the next row (row 18).
- Compare x with the first element of the row ($A[18,1] = 58$).
- Since $x > 58$, move to the next row (row 19).
- Compare x with the first element of the row ($A[19,1] = 62$).
- Since $x > 62$, move to the next row (row 20).
- Compare x with the first element of the row ($A[20,1] = 66$).
- Since $x > 66$, move to the next row (row 21).
- Compare x with the first element of the row ($A[21,1] = 70$).
- Since $x > 70$, move to the next row (row 22).
- Compare x with the first element of the row ($A[22,1] = 74$).
- Since $x > 74$, move to the next row (row 23).
- Compare x with the first element of the row ($A[23,1] = 78$).
- Since $x > 78$, move to the next row (row 24).
- Compare x with the first element of the row ($A[24,1] = 82$).
- Since $x > 82$, move to the next row (row 25).
- Compare x with the first element of the row ($A[25,1] = 86$).
- Since $x > 86$, move to the next row (row 26).
- Compare x with the first element of the row ($A[26,1] = 90$).
- Since $x > 90$, move to the next row (row 27).
- Compare x with the first element of the row ($A[27,1] = 94$).
- Since $x > 94$, move to the next row (row 28).
- Compare x with the first element of the row ($A[28,1] = 98$).
- Since $x > 98$, move to the next row (row 29).
- Compare x with the first element of the row ($A[29,1] = 102$).
- Since $x > 102$, move to the next row (row 30).
- Compare x with the first element of the row ($A[30,1] = 106$).
- Since $x > 106$, move to the next row (row 31).
- Compare x with the first element of the row ($A[31,1] = 110$).
- Since $x > 110$, move to the next row (row 32).
- Compare x with the first element of the row ($A[32,1] = 114$).
- Since $x > 114$, move to the next row (row 33).
- Compare x with the first element of the row ($A[33,1] = 118$).
- Since $x > 118$, move to the next row (row 34).
- Compare x with the first element of the row ($A[34,1] = 122$).
- Since $x > 122$, move to the next row (row 35).
- Compare x with the first element of the row ($A[35,1] = 126$).
- Since $x > 126$, move to the next row (row 36).
- Compare x with the first element of the row ($A[36,1] = 130$).
- Since $x > 130$, move to the next row (row 37).
- Compare x with the first element of the row ($A[37,1] = 134$).
- Since $x > 134$, move to the next row (row 38).
- Compare x with the first element of the row ($A[38,1] = 138$).
- Since $x > 138$, move to the next row (row 39).
- Compare x with the first element of the row ($A[39,1] = 142$).
- Since $x > 142$, move to the next row (row 40).
- Compare x with the first element of the row ($A[40,1] = 146$).
- Since $x > 146$, move to the next row (row 41).
- Compare x with the first element of the row ($A[41,1] = 150$).
- Since $x > 150$, move to the next row (row 42).
- Compare x with the first element of the row ($A[42,1] = 154$).
- Since $x > 154$, move to the next row (row 43).
- Compare x with the first element of the row ($A[43,1] = 158$).
- Since $x > 158$, move to the next row (row 44).
- Compare x with the first element of the row ($A[44,1] = 162$).
- Since $x > 162$, move to the next row (row 45).
- Compare x with the first element of the row ($A[45,1] = 166$).
- Since $x > 166$, move to the next row (row 46).
- Compare x with the first element of the row ($A[46,1] = 170$).
- Since $x > 170$, move to the next row (row 47).
- Compare x with the first element of the row ($A[47,1] = 174$).
- Since $x > 174$, move to the next row (row 48).
- Compare x with the first element of the row ($A[48,1] = 178$).
- Since $x > 178$, move to the next row (row 49).
- Compare x with the first element of the row ($A[49,1] = 182$).
- Since $x > 182$, move to the next row (row 50).
- Compare x with the first element of the row ($A[50,1] = 186$).
- Since $x > 186$, move to the next row (row 51).
- Compare x with the first element of the row ($A[51,1] = 190$).
- Since $x > 190$, move to the next row (row 52).
- Compare x with the first element of the row ($A[52,1] = 194$).
- Since $x > 194$, move to the next row (row 53).
- Compare x with the first element of the row ($A[53,1] = 198$).
- Since $x > 198$, move to the next row (row 54).
- Compare x with the first element of the row ($A[54,1] = 202$).
- Since $x > 202$, move to the next row (row 55).
- Compare x with the first element of the row ($A[55,1] = 206$).
- Since $x > 206$, move to the next row (row 56).
- Compare x with the first element of the row ($A[56,1] = 210$).
- Since $x > 210$, move to the next row (row 57).
- Compare x with the first element of the row ($A[57,1] = 214$).
- Since $x > 214$, move to the next row (row 58).
- Compare x with the first element of the row ($A[58,1] = 218$).
- Since $x > 218$, move to the next row (row 59).
- Compare x with the first element of the row ($A[59,1] = 222$).
- Since $x > 222$, move to the next row (row 60).
- Compare x with the first element of the row ($A[60,1] = 226$).
- Since $x > 226$, move to the next row (row 61).

Searching in a Sorted Grid (Algorithm 3)

$A[1:n, 1:n]$

$n = 2^m - 1$

2	5	10	11	20	22	26	30	31	31	34	34	37	40	45
4	6	15	18	27	30	31	38	39	40	42	42	44	48	48
7	9	16	21	27	31	39	41	41	41	48	50	55	55	59
8	13	22	22	27	34	40	45	48	50	50	50	58	58	65
11	14	29	31	35	36	41	49	55	55	58	59	61	62	67
15	20	30	32	39	42	42	50	59	60	60	60	65	68	71
16	21	35	41	41	43	44	58	62	69	69	70	70	75	75
20	22	36	41	42	50	50	61	65	70	75	75	76	78	78
21	25	37	44	44	50	60	62	70	72	75	76	78	80	80
22	28	39	48	50	61	62	66	71	75	75	76	78	81	85
26	31	40	56	65	65	65	69	75	78	78	80	82	82	88
29	34	41	61	66	69	72	72	78	80	80	81	82	84	88
31	41	45	66	67	70	72	72	78	82	84	84	85	85	91
32	45	49	67	67	72	78	80	81	86	85	86	86	88	95
40	55	56	70	71	75	81	81	81	86	86	88	91	93	98

Search for $x = 35$

ALGORITHM 3 (SEARCH FOR x):

Starting from the top row
perform a *binary search* for x in
each row until x is found.

Binary search in row i of A :

```

left ← 1
right ← n
while left ≤ right do
    mid ←  $\frac{\text{left} + \text{right}}{2}$ 
    if  $A[i, \text{mid}] = x$  then
        return "item found"
    else if  $A[i, \text{mid}] < x$  then
        left ← mid + 1
    else right ← mid - 1
end while
return "item not found"

```

Searching in a Sorted Grid (Algorithm 3)

$n = 2^m - 1$

$A[1:n, 1:n]$

2	5	10	11	20	22	26	30	31	31	34	34	37	40	45
4	6	15	18	27	30	31	38	39	40	42	42	44	48	48
7	9	16	21	27	31	39	41	41	41	48	50	55	55	59
8	13	22	22	27	34	40	45	48	50	50	50	58	58	65
11	14	29	31	35	36	41	49	55	55	58	59	61	62	67
15	20	30	32	39	42	42	50	59	60	60	60	65	68	71
16	21	35	41	41	43	44	58	62	69	69	70	70	70	75
20	22	36	41	42	50	50	61	65	70	75	75	76	78	78
21	25	37	44	44	59	60	62	70	72	75	76	78	78	80
22	28	39	48	50	61	62	66	71	75	75	76	78	81	85
26	31	40	56	65	65	65	69	75	78	78	80	82	82	88
29	34	41	61	66	69	72	72	78	80	80	81	82	84	88
31	41	45	66	67	70	72	72	78	82	84	84	85	85	91
32	45	49	67	67	72	78	80	81	86	85	86	86	88	95
40	55	56	70	71	75	81	81	81	86	86	88	91	93	98

$n = 2^m - 1$

Search for $x = 35$

ALGORITHM 3 (SEARCH FOR x):

Starting from the top row perform a *binary search* for x in each row until x is found.

Binary search in row i of A :

$left \leftarrow 1$

$right \leftarrow n$

while $left \leq right$ do

$mid \leftarrow \frac{left+right}{2}$

if $A[i, mid] = x$ then

return "item found"

else if $A[i, mid] < x$ then

$left \leftarrow mid + 1$

else $right \leftarrow mid - 1$

end while

return "item not found"

Searching in a Sorted Grid (Algorithm 3)

$A[1:n, 1:m]$

$n = 2^m - 1$

2	5	10	11	20	22	26	30	31	31	34	34	37	40	45
4	6	15	18	27	30	31	38	39	40	42	42	44	48	48
7	9	16	21	27	31	39	41	41	41	48	50	55	55	59
8	13	22	22	27	34	40	45	48	50	50	50	58	58	65
11	14	29	31	35	36	41	49	55	55	58	59	61	62	67
15	20	30	32	39	42	42	50	59	60	60	60	65	68	71
16	21	35	41	41	43	44	58	62	69	69	70	70	75	75
20	22	36	41	42	50	50	61	65	70	75	75	76	78	78
21	25	37	44	44	59	60	62	70	72	75	76	78	78	80
22	28	39	48	50	61	62	66	71	75	75	76	78	81	85
26	31	40	56	65	65	65	69	75	78	78	80	82	82	88
29	34	41	61	66	69	72	72	78	80	80	81	82	84	88
31	41	45	66	67	70	72	72	78	82	84	84	85	85	91
32	45	49	67	67	72	78	80	81	86	85	86	86	88	95
40	55	56	70	71	75	81	81	81	86	86	88	91	93	98

$n = 2^m - 1$

Search for $x = 35$

ALGORITHM 3 (SEARCH FOR x):

Starting from the top row perform a *binary search* for x in each row until x is found.

Binary search in row i of A :

```

left ← 1
right ← n
while left ≤ right do
    mid ←  $\frac{\text{left} + \text{right}}{2}$ 
    if  $A[i, \text{mid}] = x$  then
        return "item found"
    else if  $A[i, \text{mid}] < x$  then
        left ← mid + 1
    else right ← mid - 1
end while
return "item not found"
  
```


(Lecture 1) Introduction

Searching in a Sorted Grid (Algorithm 3)

$n = 2^m - 1$

$A[1:n, 1:n]$

2	5	10	11	20	22	26	30	31	31	34	34	37	40	45
4	6	15	18	27	30	31	38	39	40	42	42	44	48	48
7	9	16	21	27	31	39	41	41	41	48	50	55	55	59
8	13	22	22	27	34	40	45	48	50	50	50	58	58	65
11	14	29	31	35	36	41	49	55	55	58	59	61	62	67
15	20	30	32	39	42	42	50	59	60	60	60	65	68	71
16	21	35	41	41	43	44	58	62	69	69	70	70	70	75
20	22	36	41	42	50	50	61	65	70	75	75	76	78	78
21	25	37	44	44	59	60	62	70	72	75	76	78	78	80
22	28	39	48	50	61	62	66	71	75	75	76	78	81	85
26	31	40	56	65	65	69	75	78	78	80	82	82	88	
29	34	41	61	66	69	72	72	78	80	80	81	82	84	88
31	41	45	66	67	70	72	72	78	82	84	84	85	85	91
32	45	49	67	67	72	78	80	81	86	85	86	86	88	95
40	55	56	70	71	75	81	81	81	86	86	88	91	93	98

Search for $x = 35$

ALGORITHM 3 (SEARCH FOR x):

Starting from the top row
perform a *binary search* for x in
each row until x is found.

Binary search in row i of A :

$left \leftarrow 1$
 $right \leftarrow n$
while $left \leq right$ do
 $mid \leftarrow \frac{left+right}{2}$
 if $A[i, mid] = x$ then
 return "item found"
 else if $A[i, mid] < x$ then
 $left \leftarrow mid + 1$
 else $right \leftarrow mid - 1$
end while
return "item not found"

Searching in a Sorted Grid (Algorithm 3)

$n = 2^m - 1$

$A[1:n, 1:n]$

2	5	10	11	20	22	26	30	31	31	34	34	37	40	45
4	6	15	18	27	30	31	38	39	40	42	42	44	48	48
7	9	16	21	27	31	39	41	41	41	48	50	55	55	59
8	13	22	22	27	34	40	45	48	50	50	50	58	58	65
11	14	29	31	35	36	41	49	55	55	58	59	61	62	67
15	20	30	32	39	42	42	50	59	60	60	60	65	68	71
16	21	35	41	41	43	44	58	62	69	69	70	70	70	75
20	22	36	41	42	50	50	61	65	70	75	75	76	78	78
21	25	37	44	44	59	60	62	70	72	75	76	78	78	80
22	28	39	48	50	61	62	66	71	75	75	76	78	81	85
26	31	40	56	65	65	69	75	78	78	80	82	82	88	
29	34	41	61	66	69	72	72	78	80	80	81	82	84	88
31	41	45	66	67	70	72	72	78	82	84	84	85	85	91
32	45	49	67	67	72	78	80	81	86	85	86	86	88	95
40	55	56	70	71	75	81	81	81	86	86	88	91	93	98

Search for $x = 35$

ALGORITHM 3 (SEARCH FOR x):

Starting from the top row
perform a *binary search* for x in
each row until x is found.

Binary search in row i of A :

$left \leftarrow 1$
 $right \leftarrow n$
while $left \leq right$ do
 $mid \leftarrow \frac{left+right}{2}$
 if $A[i, mid] = x$ then
 return "item found"
 else if $A[i, mid] < x$ then
 $left \leftarrow mid + 1$
 else $right \leftarrow mid - 1$
end while
return "item not found"

Searching in a Sorted Grid (Algorithm 3)

$n = 2^m - 1$

$A[1:n, 1:n]$

2	5	10	11	20	22	26	30	31	31	34	34	37	40	45
4	6	15	18	27	30	31	38	39	40	42	42	44	48	48
7	9	16	21	27	31	39	41	41	41	48	50	55	55	59
8	13	22	22	27	34	40	45	48	50	50	50	58	58	65
11	14	29	31	35	36	41	49	55	55	58	59	61	62	67
15	20	30	32	39	42	42	50	59	60	60	60	65	68	71
16	21	35	41	41	43	44	58	62	69	69	70	70	70	75
20	22	36	41	42	50	50	61	65	70	75	75	76	78	78
21	25	37	44	44	59	60	62	70	72	75	76	78	78	80
22	28	39	48	50	61	62	66	71	75	75	76	78	81	85
26	31	40	56	65	65	69	75	78	78	80	82	82	88	
29	34	41	61	66	69	72	72	78	80	80	81	82	84	88
31	41	45	66	67	70	72	72	78	82	84	84	85	85	91
32	45	49	67	67	72	78	80	81	86	85	86	86	88	95
40	55	56	70	71	75	81	81	81	86	86	88	91	93	98

Search for $x = 35$

Let $Q_3(n)$ = number of
comparisons performed on an
 $n \times n$ grid.

Binary search on each row
performs m comparisons.

So, $Q_3(n) \leq nm = n \log_2(n + 1)$

Searching in a Sorted Grid (Algorithm 4)

$n = 2^m - 1$

$A[1:n, 1:n]$

2	5	10	11	20	22	26	30	31	31	34	34	37	40	45
4	6	15	18	27	30	31	38	39	40	42	42	44	48	48
7	9	16	21	27	31	39	41	41	41	48	50	55	55	59
8	13	22	22	27	34	40	45	48	50	50	50	58	58	65
11	14	29	31	35	36	41	49	55	55	58	59	61	62	67
15	20	30	32	39	42	42	50	59	60	60	60	65	68	71
16	21	35	41	41	43	44	58	62	69	69	70	70	70	75
20	22	36	41	42	50	50	61	65	70	75	75	76	78	78
21	25	37	44	44	59	60	62	70	72	75	76	78	78	80
22	28	39	48	50	61	62	66	71	75	75	76	78	81	85
26	31	40	56	65	65	69	75	78	78	80	82	82	88	
29	34	41	61	66	69	72	72	78	80	80	81	82	84	88
31	41	45	66	67	70	72	72	78	82	84	84	85	85	91
32	45	49	67	67	72	78	80	81	86	85	86	86	88	95
40	55	56	70	71	75	81	81	81	86	86	88	91	93	98

Search for $x = 35$

ALGORITHM 4 (SEARCH FOR x):

Start the search for x from the
bottom-left corner.

11

(Lecture 1) Introduction

Searching in a Sorted Grid (Algorithm 4)

$n = 2^m - 1$

$A[1:n, 1:n]$

2	5	10	11	20	22	26	30	31	31	34	34	37	40	45
4	6	15	18	27	30	31	38	39	40	42	42	44	48	48
7	9	16	21	27	31	39	41	41	41	48	50	55	55	59
8	13	22	22	27	34	40	45	48	50	50	50	58	58	65
11	14	29	31	35	36	41	49	55	55	58	59	61	62	67
15	20	30	32	39	42	42	50	59	60	60	60	65	68	71
16	21	35	41	41	43	44	58	62	69	69	70	70	70	75
20	22	36	41	42	50	50	61	65	70	75	75	76	78	78
21	25	37	44	44	59	60	62	70	72	75	76	78	78	80
22	28	39	48	50	61	62	66	71	75	75	76	78	81	85
26	31	40	56	65	65	65	69	75	78	78	80	82	82	88
29	34	41	61	66	69	72	72	78	80	80	81	82	84	88
31	41	45	66	67	70	72	72	78	82	84	84	85	85	91
32	45	49	67	67	72	78	80	81	86	85	86	86	88	95
40	55	56	70	71	75	81	81	81	86	86	88	91	93	98

ALGORITHM 4 (SEARCH FOR x):

Start the search for x from the bottom-left corner.

Keep performing the steps below until either you find x or you fall off the grid:

Let y be the number at current location.

- $y = x$: you found the item
- $y < x$: move to the right
- $y > x$: move to the cell above

Searching in a Sorted Grid (Algorithm 4)

$n = 2^m - 1$

$A[1:n, 1:n]$

2	5	10	11	20	22	26	30	31	31	34	34	37	40	45
4	6	15	18	27	30	31	38	39	40	42	42	44	48	48
7	9	16	21	27	31	39	41	41	41	48	50	55	55	59
8	13	22	22	27	34	40	45	48	50	50	50	58	58	65
11	14	29	31	35	36	41	49	55	55	58	59	61	62	67
15	20	30	32	39	42	42	50	59	60	60	60	65	68	71
16	21	35	41	41	43	44	58	62	69	69	70	70	70	75
20	22	36	41	42	50	50	61	65	70	75	75	76	78	78
21	25	37	44	44	59	60	62	70	72	75	76	78	78	80
22	28	39	48	50	61	62	66	71	75	75	76	78	81	85
26	31	40	56	65	65	65	69	75	78	78	80	82	82	88
29	34	41	61	66	69	72	72	78	80	80	81	82	84	88
31	41	45	66	67	70	72	72	78	82	84	84	85	85	91
32	45	49	67	67	72	78	80	81	86	85	86	86	88	95
40	55	56	70	71	75	81	81	81	86	86	88	91	93	98

ALGORITHM 4 (SEARCH FOR x):

Start the search for x from the bottom-left corner.

Keep performing the steps below until either you find x or you fall off the grid:

Let y be the number at current location.

- $y = x$: you found the item
- $y < x$: move to the right
- $y > x$: move to the cell above

Searching in a Sorted Grid (Algorithm 4)

$n = 2^m - 1$

$A[1:n, 1:n]$

2	5	10	11	20	22	26	30	31	31	34	34	37	40	45
4	6	15	18	27	30	31	38	39	40	42	42	44	48	48
7	9	16	21	27	31	39	41	41	41	48	50	55	55	59
8	13	22	22	27	34	40	45	48	50	50	50	58	58	65
11	14	29	31	35	36	41	49	55	55	58	59	61	62	67
15	20	30	32	39	42	42	50	59	60	60	60	65	68	71
16	21	35	41	41	43	44	58	62	69	69	70	70	70	75
20	22	36	41	42	50	50	61	65	70	75	75	76	78	78
21	25	37	44	44	59	60	62	70	72	75	76	78	78	80
22	28	39	48	50	61	62	66	71	75	75	76	78	81	85
26	31	40	56	65	65	65	69	75	78	78	80	82	82	88
29	34	41	61	66	69	72	72	78	80	80	81	82	84	88
31	41	45	66	67	70	72	72	78	82	84	84	85	85	91
32	45	49	67	67	72	78	80	81	86	85	86	86	88	95
40	55	56	70	71	75	81	81	81	86	86	88	91	93	98

ALGORITHM 4 (SEARCH FOR x):

Start the search for x from the bottom-left corner.

Keep performing the steps below until either you find x or you fall off the grid:

Let y be the number at current location.

- $y = x$: you found the item
- $y < x$: move to the right
- $y > x$: move to the cell above

Searching in a Sorted Grid (Algorithm 4)

$n = 2^m - 1$

$A[1:n, 1:n]$

2	5	10	11	20	22	26	30	31	31	34	34	37	40	45
4	6	15	18	27	30	31	38	39	40	42	42	44	48	48
7	9	16	21	27	31	39	41	41	41	48	50	55	55	59
8	13	22	22	27	34	40	45	48	50	50	50	58	58	65
11	14	29	31	35	36	41	49	55	55	58	59	61	62	67
15	20	30	32	39	42	42	50	59	60	60	60	65	68	71
16	21	35	41	41	43	44	58	62	69	69	70	70	70	75
20	22	36	41	42	50	50	61	65	70	75	75	76	78	78
21	25	37	44	44	59	60	62	70	72	75	76	78	78	80
22	28	39	48	50	61	62	66	71	75	75	76	78	81	85
26	31	40	56	65	65	65	69	75	78	78	80	82	82	88
29	34	41	61	66	69	72	72	78	80	80	81	82	84	88
31	41	45	66	67	70	72	72	78	82	84	84	85	85	91
32	45	49	67	67	72	78	80	81	86	85	86	86	88	95
40	55	56	70	71	75	81	81	81	86	86	88	91	93	98

ALGORITHM 4 (SEARCH FOR x):

Start the search for x from the bottom-left corner.

Keep performing the steps below until either you find x or you fall off the grid:

Let y be the number at current location.

- $y = x$: you found the item
- $y < x$: move to the right
- $y > x$: move to the cell above

12

(Lecture 1) Introduction

Searching in a Sorted Grid (Algorithm 4)

Figure 1 shows a 16x16 grid of numbers, arranged in a 4x4 block structure. The grid is labeled with $n = 2^m - 1$ at the top and $n = 2^m - 1$ on the left. The top row is labeled $A[1:n, 1:n]$. The numbers in the grid are as follows:

2	5	10	11	20	22	26	30	31	31	34	34	37	40	45	
4	6	15	18	27	30	31	38	39	40	42	42	44	48	48	
7	9	16	21	27	31	39	41	41	41	48	50	55	55	59	
8	13	22	22	27	34	40	45	48	50	50	50	58	58	65	
11	14	29	31	35	36	41	49	55	55	58	59	61	62	67	
15	20	30	32	39	42	42	50	59	60	60	65	65	68	71	
16	21	35	41	41	43	44	58	62	69	69	70	70	70	75	
20	22	36	41	42	50	50	61	65	70	75	75	76	78	78	
21	25	37	44	44	59	60	62	70	72	75	76	78	78	80	
22	28	39	48	50	61	62	66	71	75	75	76	78	81	85	
26	31	40	56	65	65	65	69	75	78	78	80	82	82	88	
29	34	41	61	66	69	72	72	78	80	80	81	82	84	88	
31	41	45	66	67	70	72	72	78	82	84	84	85	85	91	
32	45	49	67	67	72	78	80	81	86	85	86	86	88	95	
40	55	56	70	71	75	81	81	81	86	86	88	91	93	98	

Search for $x = 68$

ALGORITHM 4 (SEARCH FOR x):

Start the search for x from the bottom-left corner.

Keep performing the steps below until either you find x or you fall off the grid:

Let y be the number at current location.

- $y = x$: you found the item
- $y < x$: move to the right
- $y > x$: move to the cell above



Searching in a Sorted Grid (Algorithm 4)

$A[1:n, 1:n]$					$n = 2^m - 1$										
					2	5	10	11	20	22	26	30	31	31	34
$n = 2^m - 1$	4	6	15	18	27	30	31	38	39	40	42	42	44	48	48
	7	9	16	21	27	31	39	41	41	41	48	50	55	55	59
	8	13	22	22	27	34	40	45	48	50	50	58	58	58	65
	11	14	29	31	35	36	41	49	55	55	58	59	61	62	67
	15	20	30	32	39	42	43	50	59	60	60	65	68	71	75
	16	21	35	41	41	44	44	58	62	69	69	70	70	70	75
	20	22	36	41	42	50	50	61	65	70	75	75	76	78	78
	21	25	37	44	44	59	60	62	70	72	75	76	78	78	80
	22	28	39	48	50	61	62	66	71	75	75	76	78	81	85
	26	31	40	56	65	65	65	69	75	78	78	80	82	82	88
	29	34	41	61	66	69	72	72	78	80	80	81	82	84	88
	31	41	45	66	67	70	72	72	78	82	84	84	85	85	91
	32	45	49	67	67	72	78	80	81	86	85	86	86	88	95
	40	55	56	70	71	75	81	81	81	86	86	88	91	93	98

Search for $x = 68$

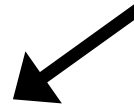
ALGORITHM 4 (SEARCH FOR x):

Start the search for x from the bottom-left corner.

Keep performing the steps below until either you find x or you fall off the grid:

Let y be the number at current location.

- $y = x$: you found the item
- $y < x$: move to the right
- $y > x$: move to the cell above



Searching in a Sorted Grid (Algorithm 4)

	2	5	10	11	20	22	26	30	31	31	34	34	37	40	45
	4	6	15	18	27	30	31	38	39	40	42	42	44	48	48
	7	9	16	21	27	31	39	41	41	41	48	50	55	55	59
	8	13	22	22	27	34	40	45	48	50	50	50	58	58	65
	11	14	29	31	35	36	41	49	55	55	58	59	61	62	67
	15	20	30	32	39	42	42	50	59	60	60	60	65	68	71
	16	21	35	41	41	43	44	58	62	69	69	70	70	70	75
	20	22	36	41	42	50	50	61	65	70	75	75	76	78	78
	21	25	37	44	44	59	60	62	70	72	75	76	78	78	80
	22	28	39	48	50	61	62	66	71	75	75	76	78	81	85
	26	31	40	56	65	65	65	69	75	78	78	80	82	82	88
	29	34	41	61	66	69	72	72	78	80	80	81	82	84	88
	31	41	45	66	67	70	72	72	78	82	84	84	85	85	91
	32	45	49	67	67	72	78	80	81	86	85	86	86	88	95
	40	55	56	70	71	75	81	81	81	86	86	88	91	93	98

Search for $x = 68$

ALGORITHM 4 (SEARCH FOR x):

Start the search for x from the bottom-left corner.

Keep performing the steps below until either you find x or you fall off the grid:

Let y be the number at current location.

- $y = x$: you found the item
- $y < x$: move to the right
- $y > x$: move to the cell above



Searching in a Sorted Grid (Algorithm 4)

	2	5	10	11	20	22	26	30	31	31	34	34	37	40	45
	4	6	15	18	27	30	31	38	39	40	42	42	44	48	48
	7	9	16	21	27	31	39	41	41	41	48	50	55	55	59
	8	13	22	22	27	34	40	45	48	50	50	50	58	58	65
	11	14	29	31	35	36	41	49	55	55	58	59	61	62	67
	15	20	30	32	39	42	43	50	59	60	60	65	68	71	71
	16	21	35	41	41	43	44	58	62	69	69	70	70	70	75
	20	22	36	41	42	50	50	61	65	70	75	75	76	78	78
	21	25	37	44	44	59	60	62	70	72	75	76	78	78	80
	22	28	39	48	50	61	62	66	71	75	75	76	78	81	85
	26	31	40	56	65	65	65	69	75	78	78	80	82	82	88
	29	34	41	61	66	69	72	72	78	80	80	81	82	84	88
	31	41	45	66	67	70	72	72	78	82	84	84	85	85	91
	32	45	49	67	67	72	78	80	81	86	85	86	86	88	95
	40	55	56	70	71	75	81	81	81	86	86	88	91	93	98

Search for $x = 68$

ALGORITHM 4 (SEARCH FOR x):

Start the search for x from the bottom-left corner.

Keep performing the steps below until either you find x or you fall off the grid:

Let y be the number at current location.

- $y = x$: you found the item
- $y < x$: move to the right
- $y > x$: move to the cell above

(Lecture 1) Introduction

Searching in a Sorted Grid (Algorithm 4)

$n = 2^m - 1$

$A[1:n, 1:n]$

2	5	10	11	20	22	26	30	31	31	34	34	37	40	45
4	6	15	18	27	30	31	38	39	40	42	42	44	48	48
7	9	16	21	27	31	39	41	41	41	48	50	55	55	59
8	13	22	22	27	34	40	45	48	50	50	50	58	58	65
11	14	29	31	35	36	41	49	55	55	58	59	61	62	67
15	20	30	32	39	42	42	50	59	60	60	60	65	68	71
16	21	35	41	41	43	44	58	62	69	69	70	70	70	75
20	22	36	41	42	50	50	61	65	70	75	75	76	78	78
21	25	37	44	44	59	60	62	70	72	75	76	78	78	80
22	28	39	48	50	61	62	66	71	75	75	76	78	81	85
26	31	40	56	65	65	65	69	75	78	78	80	82	82	88
29	34	41	61	66	69	72	72	78	80	80	81	82	84	88
31	41	45	66	67	70	72	72	78	82	84	84	85	85	91
32	45	49	67	67	72	78	80	81	86	85	86	86	88	95
40	55	56	70	71	75	81	81	81	86	86	88	91	93	98

Search for $x = 68$

ALGORITHM 4 (SEARCH FOR x):

Start the search for x from the bottom-left corner.

Keep performing the steps below until either you find x or you fall off the grid:

Let y be the number at current location.

- $y = x$: you found the item
- $y < x$: move to the right
- $y > x$: move to the cell above

Searching in a Sorted Grid (Algorithm 4)

$n = 2^m - 1$

$A[1:n, 1:n]$

2	5	10	11	20	22	26	30	31	31	34	34	37	40	45
4	6	15	18	27	30	31	38	39	40	42	42	44	48	48
7	9	16	21	27	31	39	41	41	41	48	50	55	55	59
8	13	22	22	27	34	40	45	48	50	50	50	58	58	65
11	14	29	31	35	36	41	49	55	55	58	59	61	62	67
15	20	30	32	39	42	42	50	59	60	60	60	65	68	71
16	21	35	41	41	43	44	58	62	69	69	70	70	70	75
20	22	36	41	42	50	50	61	65	70	75	75	76	78	78
21	25	37	44	44	59	60	62	70	72	75	76	78	78	80
22	28	39	48	50	61	62	66	71	75	75	76	78	81	85
26	31	40	56	65	65	65	69	75	78	78	80	82	82	88
29	34	41	61	66	69	72	72	78	80	80	81	82	84	88
31	41	45	66	67	70	72	72	78	82	84	84	85	85	91
32	45	49	67	67	72	78	80	81	86	85	86	86	88	95
40	55	56	70	71	75	81	81	81	86	86	88	91	93	98

Search for $x = 68$

ALGORITHM 4 (SEARCH FOR x):

Start the search for x from the bottom-left corner.

Keep performing the steps below until either you find x or you fall off the grid:

Let y be the number at current location.

- $y = x$: you found the item
- $y < x$: move to the right
- $y > x$: move to the cell above

Searching in a Sorted Grid (Algorithm 4)

$n = 2^m - 1$

$A[1:n, 1:n]$

2	5	10	11	20	22	26	30	31	31	34	34	37	40	45
4	6	15	18	27	30	31	38	39	40	42	42	44	48	48
7	9	16	21	27	31	39	41	41	41	48	50	55	55	59
8	13	22	22	27	34	40	45	48	50	50	50	58	58	65
11	14	29	31	35	36	41	49	55	55	58	59	61	62	67
15	20	30	32	39	42	42	50	59	60	60	60	65	68	71
16	21	35	41	41	43	44	58	62	69	69	70	70	70	75
20	22	36	41	42	50	50	61	65	70	75	75	76	78	78
21	25	37	44	44	59	60	62	70	72	75	76	78	78	80
22	28	39	48	50	61	62	66	71	75	75	76	78	81	85
26	31	40	56	65	65	65	69	75	78	78	80	82	82	88
29	34	41	61	66	69	72	72	78	80	80	81	82	84	88
31	41	45	66	67	70	72	72	78	82	84	84	85	85	91
32	45	49	67	67	72	78	80	81	86	85	86	86	88	95
40	55	56	70	71	75	81	81	81	86	86	88	91	93	98

Search for $x = 68$

ALGORITHM 4 (SEARCH FOR x):

Start the search for x from the bottom-left corner.

Keep performing the steps below until either you find x or you fall off the grid:

Let y be the number at current location.

- $y = x$: you found the item
- $y < x$: move to the right
- $y > x$: move to the cell above

Searching in a Sorted Grid (Algorithm 4)

$n = 2^m - 1$

$A[1:n, 1:n]$

2	5	10	11	20	22	26	30	31	31	34	34	37	40	45
4	6	15	18	27	30	31	38	39	40	42	42	44	48	48
7	9	16	21	27	31	39	41	41	41	48	50	55	55	59
8	13	22	22	27	34	40	45	48	50	50	50	58	58	65
11	14	29	31	35	36	41	49	55	55	58	59	61	62	67
15	20	30	32	39	42	42	50	59	60	60	60	65	68	71
16	21	35	41	41	43	44	58	62	69	69	70	70	70	75
20	22	36	41	42	50	50	61	65	70	75	75	76	78	78
21	25	37	44	44	59	60	62	70	72	75	76	78	78	80
22	28	39	48	50	61	62	66	71	75	75	76	78	81	85
26	31	40	56	65	65	65	69	75	78	78	80	82	82	88
29	34	41	61	66	69	72	72	78	80	80	81	82	84	88
31	41	45	66	67	70	72	72	78	82	84	84	85	85	91
32	45	49	67	67	72	78	80	81	86	85	86	86	88	95
40	55	56	70	71	75	81	81	81	86	86	88	91	93	98

Search for $x = 68$

ALGORITHM 4 (SEARCH FOR x):

Start the search for x from the bottom-left corner.

Keep performing the steps below until either you find x or you fall off the grid:

Let y be the number at current location.

- $y = x$: you found the item
- $y < x$: move to the right
- $y > x$: move to the cell above

15

(Lecture 1) Introduction

Searching in a Sorted Grid (Algorithm 4)

$n = 2^m - 1$

$A[1:n, 1:n]$

2	5	10	11	20	22	26	30	31	31	34	34	37	40	45
4	6	15	18	27	30	31	38	39	40	42	42	44	48	48
7	9	16	21	27	31	39	41	41	41	48	50	55	55	59
8	13	22	22	27	34	40	45	48	50	50	50	58	58	65
11	14	29	31	35	36	41	49	55	55	58	59	61	62	67
15	20	30	32	39	42	42	50	59	60	60	60	65	68	71
16	21	35	41	41	43	44	58	62	69	69	70	70	70	75
20	22	36	41	42	50	50	61	65	70	75	75	76	78	78
21	25	37	44	44	59	60	62	70	72	75	76	78	78	80
22	28	39	48	50	61	62	66	71	75	75	76	78	81	85
26	31	40	56	65	65	65	69	75	78	78	80	82	82	88
29	34	41	61	66	69	72	72	78	80	80	81	82	84	88
31	41	45	66	67	70	72	72	78	82	84	84	85	85	91
32	45	49	67	67	72	78	80	81	86	85	86	86	88	95
40	55	56	70	71	75	81	81	81	86	86	88	91	93	98

Search for $x = 68$

ALGORITHM 4 (SEARCH FOR x):

Start the search for x from the bottom-left corner.

Keep performing the steps below until either you find x or you fall off the grid:

Let y be the number at current location.

- $y = x$: you found the item
- $y < x$: move to the right
- $y > x$: move to the cell above

Searching in a Sorted Grid (Algorithm 4)

$n = 2^m - 1$

$A[1:n, 1:n]$

2	5	10	11	20	22	26	30	31	31	34	34	37	40	45
4	6	15	18	27	30	31	38	39	40	42	42	44	48	48
7	9	16	21	27	31	39	41	41	41	48	50	55	55	59
8	13	22	22	27	34	40	45	48	50	50	50	58	58	65
11	14	29	31	35	36	41	49	55	55	58	59	61	62	67
15	20	30	32	39	42	42	50	59	60	60	60	65	68	71
16	21	35	41	41	43	44	58	62	69	69	70	70	70	75
20	22	36	41	42	50	50	61	65	70	75	75	76	78	78
21	25	37	44	44	59	60	62	70	72	75	76	78	78	80
22	28	39	48	50	61	62	66	71	75	75	76	78	81	85
26	31	40	56	65	65	65	69	75	78	78	80	82	82	88
29	34	41	61	66	69	72	72	78	80	80	81	82	84	88
31	41	45	66	67	70	72	72	78	82	84	84	85	85	91
32	45	49	67	67	72	78	80	81	86	85	86	86	88	95
40	55	56	70	71	75	81	81	81	86	86	88	91	93	98

Search for $x = 68$

ALGORITHM 4 (SEARCH FOR x):

Start the search for x from the bottom-left corner.

Keep performing the steps below until either you find x or you fall off the grid:

Let y be the number at current location.

- $y = x$: you found the item
- $y < x$: move to the right
- $y > x$: move to the cell above

Searching in a Sorted Grid (Algorithm 4)

$n = 2^m - 1$

$A[1:n, 1:n]$

2	5	10	11	20	22	26	30	31	31	34	34	37	40	45
4	6	15	18	27	30	31	38	39	40	42	42	44	48	48
7	9	16	21	27	31	39	41	41	41	48	50	55	55	59
8	13	22	22	27	34	40	45	48	50	50	50	58	58	65
11	14	29	31	35	36	41	49	55	55	58	59	61	62	67
15	20	30	32	39	42	42	50	59	60	60	60	65	68	71
16	21	35	41	41	43	44	58	62	69	69	70	70	70	75
20	22	36	41	42	50	50	61	65	70	75	75	76	78	78
21	25	37	44	44	59	60	62	70	72	75	76	78	78	80
22	28	39	48	50	61	62	66	71	75	75	76	78	81	85
26	31	40	56	65	65	65	69	75	78	78	80	82	82	88
29	34	41	61	66	69	72	72	78	80	80	81	82	84	88
31	41	45	66	67	70	72	72	78	82	84	84	85	85	91
32	45	49	67	67	72	78	80	81	86	85	86	86	88	95
40	55	56	70	71	75	81	81	81	86	86	88	91	93	98

Search for $x = 68$

ALGORITHM 4 (SEARCH FOR x):

Start the search for x from the bottom-left corner.

Keep performing the steps below until either you find x or you fall off the grid:

Let y be the number at current location.

- $y = x$: you found the item
- $y < x$: move to the right
- $y > x$: move to the cell above

Searching in a Sorted Grid (Algorithm 4)

$n = 2^m - 1$

$A[1:n, 1:n]$

2	5	10	11	20	22	26	30	31	31	34	34	37	40	45
4	6	15	18	27	30	31	38	39	40	42	42	44	48	48
7	9	16	21	27	31	39	41	41	41	48	50	55	55	59
8	13	22	22	27	34	40	45	48	50	50	50	58	58	65
11	14	29	31	35	36	41	49	55	55	58	59	61	62	67
15	20	30	32	39	42	42	50	59	60	60	60	65	68	71
16	21	35	41	41	43	44	58	62	69	69	70	70	70	75
20	22	36	41	42	50	50	61	65	70	75	75	76	78	78
21	25	37	44	44	59	60	62	70	72	75	76	78	78	80
22	28	39	48	50	61	62	66	71	75	75	76	78	81	85
26	31	40	56	65	65	65	69	75	78	78	80	82	82	88
29	34	41	61	66	69	72	72	78	80	80	81	82	84	88
31	41	45	66	67	70	72	72	78	82	84	84	85	85	91
32	45	49	67	67	72	78	80	81	86	85	86	86	88	95
40	55	56	70	71	75	81	81	81	86	86	88	91	93	98

Search for $x = 68$

ALGORITHM 4 (SEARCH FOR x):

Start the search for x from the bottom-left corner.

Keep performing the steps below until either you find x or you fall off the grid:

Let y be the number at current location.

- $y = x$: you found the item
- $y < x$: move to the right
- $y > x$: move to the cell above

16

(Lecture 1) Introduction

Searching in a Sorted Grid (Algorithm 4)

$n = 2^m - 1$

$A[1:n, 1:n]$

2	5	10	11	20	22	26	30	31	31	34	34	37	40	45
4	6	15	18	27	30	31	38	39	40	42	42	44	48	48
7	9	16	21	27	31	39	41	41	41	48	50	55	55	59
8	13	22	22	27	34	40	45	48	50	50	50	58	58	65
11	14	29	31	35	36	41	49	55	55	58	59	61	62	67
15	20	30	32	39	42	42	50	59	60	60	65	68	71	
16	21	35	41	41	43	44	58	62	69	69	70	70	75	
20	22	36	41	42	50	50	61	65	70	75	75	76	78	78
21	25	37	44	44	59	60	62	70	72	75	76	78	80	
22	28	39	48	50	61	62	66	71	75	75	76	78	81	85
26	31	40	56	65	65	65	69	75	78	78	80	82	82	88
29	34	41	61	66	69	72	72	78	80	80	81	82	84	88
31	41	45	66	67	70	72	72	78	82	84	84	85	85	91
32	45	49	67	67	72	78	80	81	86	85	86	86	88	95
40	55	56	70	71	75	81	81	81	86	86	88	91	93	98

Search for $x = 68$

ALGORITHM 4 (SEARCH FOR x):

Start the search for x from the bottom-left corner.

Keep performing the steps below until either you find x or you fall off the grid:

Let y be the number at current location.

- $y = x$: you found the item
- $y < x$: move to the right
- $y > x$: move to the cell above

Searching in a Sorted Grid (Algorithm 4)

$n = 2^m - 1$

$A[1:n, 1:n]$

2	5	10	11	20	22	26	30	31	31	34	34	37	40	45
4	6	15	18	27	30	31	38	39	40	42	42	44	48	48
7	9	16	21	27	31	39	41	41	41	48	50	55	55	59
8	13	22	22	27	34	40	45	48	50	50	50	58	58	65
11	14	29	31	35	36	41	49	55	55	58	59	61	62	67
15	20	30	32	39	42	42	50	59	60	60	65	68	71	
16	21	35	41	41	43	44	58	62	69	69	70	70	75	
20	22	36	41	42	50	50	61	65	70	75	75	76	78	78
21	25	37	44	44	59	60	62	70	72	75	76	78	80	
22	28	39	48	50	61	62	66	71	75	75	76	78	81	85
26	31	40	56	65	65	65	69	75	78	78	80	82	82	88
29	34	41	61	66	69	72	72	78	80	80	81	82	84	88
31	41	45	66	67	70	72	72	78	82	84	84	85	85	91
32	45	49	67	67	72	78	80	81	86	85	86	86	88	95
40	55	56	70	71	75	81	81	81	86	86	88	91	93	98

Search for $x = 68$

ALGORITHM 4 (SEARCH FOR x):

Start the search for x from the bottom-left corner.

Keep performing the steps below until either you find x or you fall off the grid:

Let y be the number at current location.

- $y = x$: you found the item
- $y < x$: move to the right
- $y > x$: move to the cell above

Searching in a Sorted Grid (Algorithm 4)

$n = 2^m - 1$

$A[1:n, 1:n]$

2	5	10	11	20	22	26	30	31	31	34	34	37	40	45
4	6	15	18	27	30	31	38	39	40	42	42	44	48	48
7	9	16	21	27	31	39	41	41	41	48	50	55	55	59
8	13	22	22	27	34	40	45	48	50	50	50	58	58	65
11	14	29	31	35	36	41	49	55	55	58	59	61	62	67
15	20	30	32	39	42	42	50	59	60	60	65	68	71	
16	21	35	41	41	43	44	58	62	69	69	70	70	75	
20	22	36	41	42	50	50	61	65	70	75	75	76	78	78
21	25	37	44	44	59	60	62	70	72	75	76	78	80	
22	28	39	48	50	61	62	66	71	75	75	76	78	81	85
26	31	40	56	65	65	65	69	75	78	78	80	82	82	88
29	34	41	61	66	69	72	72	78	80	80	81	82	84	88
31	41	45	66	67	70	72	72	78	82	84	84	85	85	91
32	45	49	67	67	72	78	80	81	86	85	86	86	88	95
40	55	56	70	71	75	81	81	81	86	86	88	91	93	98

Search for $x = 68$

ALGORITHM 4 (SEARCH FOR x):

Start the search for x from the bottom-left corner.

Keep performing the steps below until either you find x or you fall off the grid:

Let y be the number at current location.

- $y = x$: you found the item
- $y < x$: move to the right
- $y > x$: move to the cell above

Searching in a Sorted Grid (Algorithm 4)

$n = 2^m - 1$

$A[1:n, 1:n]$

2	5	10	11	20	22	26	30	31	31	34	34	37	40	45
4	6	15	18	27	30	31	38	39	40	42	42	44	48	48
7	9	16	21	27	31	39	41	41	41	48	50	55	55	59
8	13	22	22	27	34	40	45	48	50	50	50	58	58	65
11	14	29	31	35	36	41	49	55	55	58	59	61	62	67
15	20	30	32	39	42	42	50	59	60	60	65	68	71	
16	21	35	41	41	43	44	58	62	69	69	70	70	75	
20	22	36	41	42	50	50	61	65	70	75	75	76	78	78
21	25	37	44	44	59	60	62	70	72	75	76	78	80	
22	28	39	48	50	61	62	66	71	75	75	76	78	81	85
26	31	40	56	65	65	65	69	75	78	78	80	82	82	88
29	34	41	61	66	69	72	72	78	80	80	81	82	84	88
31	41	45	66	67	70	72	72	78	82	84	84	85	85	91
32	45	49	67	67	72	78	80	81	86	85	86	86	88	95
40	55	56	70	71	75	81	81	81	86	86	88	91	93	98

Search for $x = 68$

ALGORITHM 4 (SEARCH FOR x):

Start the search for x from the bottom-left corner.

Keep performing the steps below until either you find x or you fall off the grid:

Let y be the number at current location.

- $y = x$: you found the item
- $y < x$: move to the right
- $y > x$: move to the cell above

17

(Lecture 1) Introduction

Searching in a Sorted Grid (Algorithm 4)

$n = 2^m - 1$

$A[1:n, 1:n]$

2	5	10	11	20	22	26	30	31	31	34	34	37	40	45
4	6	15	18	27	30	31	38	39	40	42	42	44	48	48
7	9	16	21	27	31	39	41	41	41	48	50	55	55	59
8	13	22	22	27	34	40	45	48	50	50	50	58	58	65
11	14	29	31	35	36	41	49	55	55	58	59	61	62	67
15	20	30	32	39	42	42	50	59	60	60	60	65	68	71
16	21	35	41	41	43	44	58	62	69	69	70	70	70	75
20	22	36	41	42	50	50	61	65	70	75	75	76	78	78
21	25	37	44	44	59	60	62	70	72	75	76	78	78	80
22	28	39	48	50	61	62	66	71	75	75	76	78	81	85
26	31	40	56	65	65	65	69	75	78	78	80	82	82	88
29	34	41	61	66	69	72	72	78	80	80	81	82	84	88
31	41	45	66	67	70	72	72	78	82	84	84	85	85	91
32	45	49	67	67	72	78	80	81	86	85	86	86	88	95
40	55	56	70	71	75	81	81	81	86	86	88	91	93	98

Search for $x = 68$

Let $Q_4(n)$ = number of comparisons performed on an $n \times n$ grid.

Then $Q_4(n) \leq 2n - 1 < 2n$

Comparing the Four (4) Grid Search Algorithms

Searching in a Sorted Grid (Algorithm 1)

$n = 2^m - 1$

$A[1:n, 1:n]$

2	5	10	11	20	22	26	30	31	31	34	34	37	40	45
4	6	15	18	27	30	31	38	39	40	42	42	44	48	48
7	9	16	21	27	31	39	41	41	41	48	50	55	55	59
8	13	22	22	27	34	40	45	48	50	50	50	58	58	65
11	14	29	31	35	36	41	49	55	55	58	59	61	62	67
15	20	30	32	39	42	42	50	59	60	60	60	65	68	71
16	21	35	41	41	43	44	58	62	69	69	70	70	70	75
20	22	36	41	42	50	50	61	65	70	75	75	76	78	78
21	25	37	44	44	59	60	62	70	72	75	76	78	78	80
22	28	39	48	50	61	62	66	71	75	75	76	78	81	85
26	31	40	56	65	65	65	69	75	78	78	80	82	82	88
29	34	41	61	66	69	72	72	78	80	80	81	82	84	88
31	41	45	66	67	70	72	72	78	82	84	84	85	85	91
32	45	49	67	67	72	78	80	81	86	85	86	86	88	95
40	55	56	70	71	75	81	81	81	86	86	88	91	93	98

Let $T_1(n)$ = running time of algorithm 1 on an $n \times n$ grid.

Though we were able to compute an exact worst-case bound for $Q_1(n)$, the same cannot be done for $T_1(n)$ because it depends on many other external factors such as CPU speed, programming style, compiler and optimization level used, etc.

But for large values of n , $T_1(n)$'s worst-case value will be within a constant factor of that of $Q_1(n)$. That constant is generally unknown, and depends on the specific hardware and compiler used, expertise of the programmer, etc.

Searching in a Sorted Grid (Algorithm 1)

$n = 2^m - 1$

$A[1:n, 1:n]$

2	5	10	11	20	22	26	30	31	31	34	34	37	40	45
4	6	15	18	27	30	31	38	39	40	42	42	44	48	48
7	9	16	21	27	31	39	41	41	41	48	50	55	55	59
8	13	22	22	27	34	40	45	48	50	50	50	58	58	65
11	14	29	31	35	36	41	49	55	55	58	59	61	62	67
15	20	30	32	39	42	42	50	59	60	60	60	65	68	71
16	21	35	41	41	43	44	58	62	69	69	70	70	70	75
20	22	36	41	42	50	50	61	65	70	75	75	76	78	78
21	25	37	44	44	59	60	62	70	72	75	76	78	78	80
22	28	39	48	50	61	62	66	71	75	75	76	78	81	85
26	31	40	56	65	65	65	69	75	78	78	80	82	82	88
29	34	41	61	66	69	72	72	78	80	80	81	82	84	88
31	41	45	66	67	70	72	72	78	82	84	84	85	85	91
32	45	49	67	67	72	78	80	81	86	85	86	86	88	95
40	55	56	70	71	75	81	81	81	86	86	88	91	93	98

ALGORITHM 1 (SEARCH FOR x):

1. for $i = 1$ to n do

2. for $j = 1$ to n do

3. if $A[i, j] = x$ then return "item found"

4. end for

5. end for

6. return "item not found"

In the worst case,

– line 3 will be executed n^2 times,

– variable j in line 2 will be updated n^2 times,

– variable i in line 1 will be updated n times, and

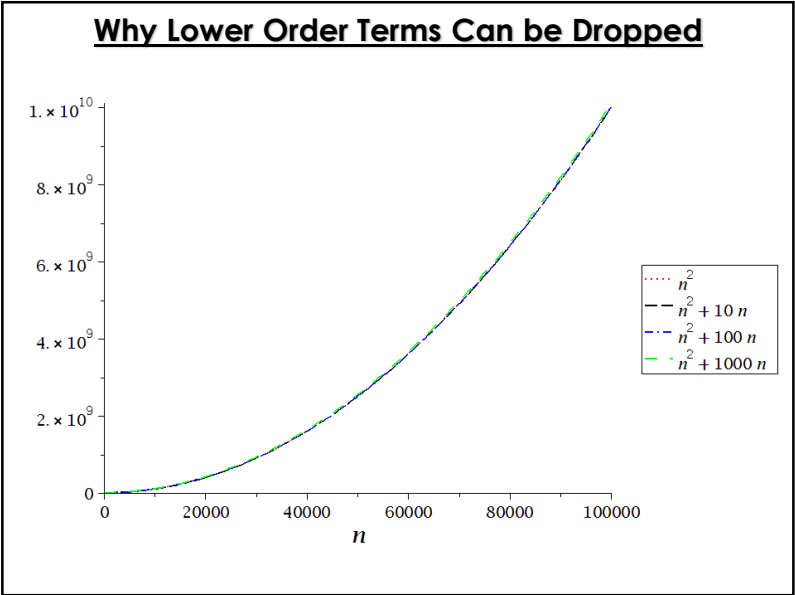
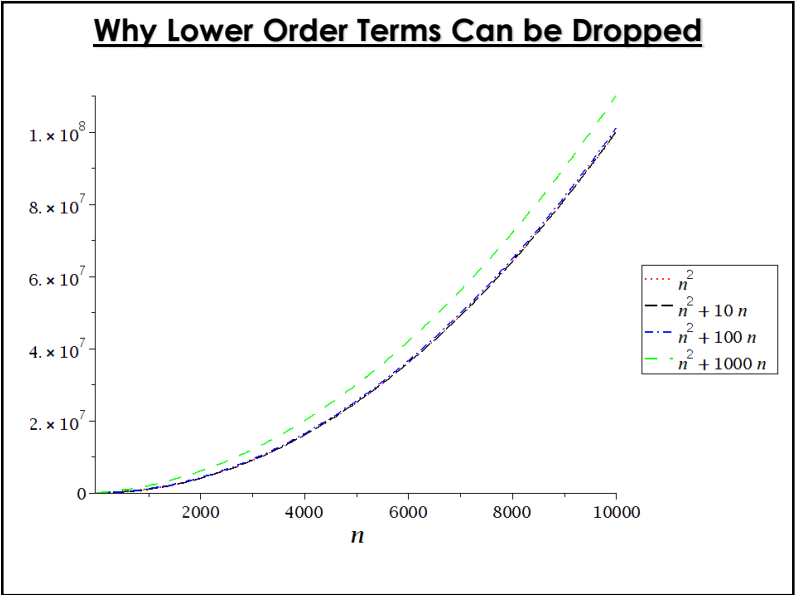
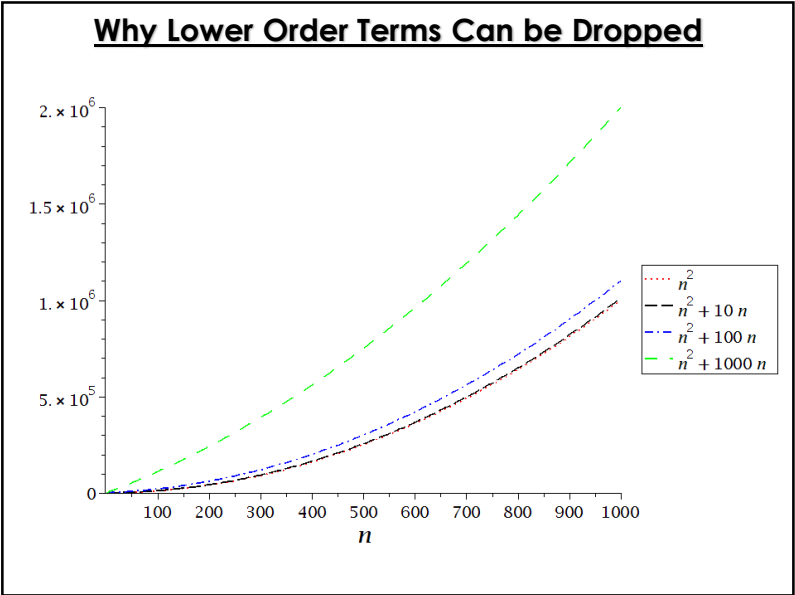
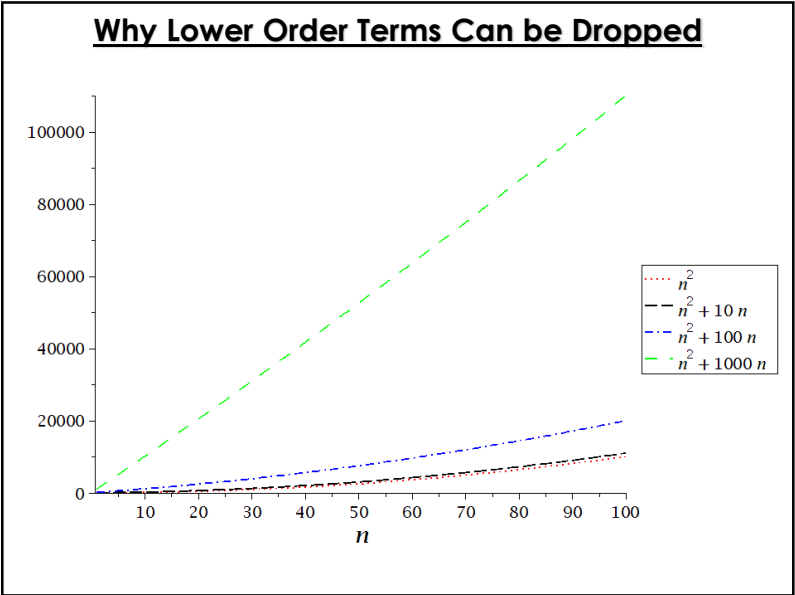
– line 6 will be executed will be executed 1 time.

Hence, $T_1(n) \leq a_1 n^2 + a_2 n + a_3$, where a_1 , a_2 and a_3 are constants.

Clearly, $T_1(n) \leq (a_1 + 1)n^2 = (a_1 + 1)Q_1(n)$, when $n \geq a_2 + a_3$.

18

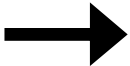
(Lecture 1) Introduction



Running Times of the Four (4) Algorithms for Large n

GRID SEARCHING ALGORITHM	WORST-CASE BOUND ON #COMPARISONS	WORST-CASE BOUND ON RUNNING TIMES
ALGORITHM 1	$Q_1(n) \leq n^2$	$T_1(n) \leq c_1 n^2$
ALGORITHM 2	$Q_2(n) \leq 2(n+1)^{1.6}$	$T_2(n) \leq c_2(n+1)^{1.6}$
ALGORITHM 3	$Q_3(n) \leq n \log_2(n+1)$	$T_3(n) \leq c_3 n \log_2(n+1)$
ALGORITHM 4	$Q_4(n) \leq 2n$	$T_4(n) \leq c_4 n$

c_1, c_2, c_3 and c_4 are constants



Why Faster Algorithms?

As the input gets large a faster algorithm run on a slow computer will eventually beat a slower algorithm run on a fast computer!

Suppose we run ALGORITHM 4 on computer A that can execute only 1 million instructions per second. The algorithm was implemented by an inexperienced programmer, and so $c_4 = 10$.

Suppose we run ALGORITHM 1 on computer B that is 1000 times faster than A , and the algorithm was implemented by an expert programmer, and so $c_1 = 1$.

Let's run both algorithm on a large grid with $n = 100,000$.

Then ALGORITHM 1 will require up to $\frac{1 \times (100000)^2}{1000000000} = 10$ seconds, while ALGORITHM 4 will terminate in only $\frac{10 \times 100000}{1000000} = 1$ second!



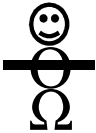
Asymptotic Bounds

We compute performance bounds as functions of input size n .

Asymptotic bounds are obtained when $n \rightarrow \infty$.

Several types of asymptotic bounds

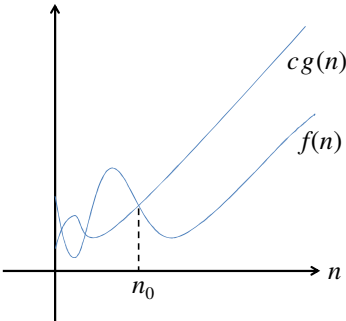
- upper bound (O -notation)
- strict upper bound (o -notation)
- lower bound (Ω -notation)
- strict lower bound (ω -notation)
- tight bound (Θ -notation)



Asymptotic Stickman
(by Aleksandra Patrzalek)



Asymptotic Upper Bound (O -notation)



$$O(g(n)) = \left\{ f(n): \text{there exist positive constants } c \text{ and } n_0 \text{ such that } 0 \leq f(n) \leq cg(n) \text{ for all } n \geq n_0 \right\}$$

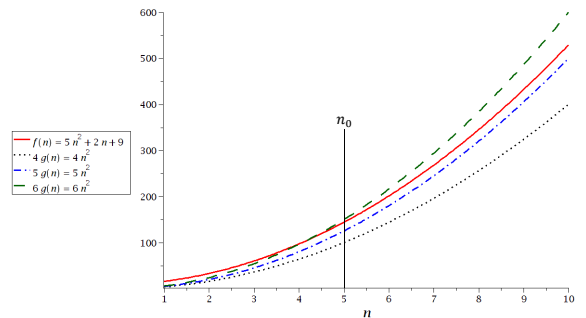
$$O(g(n)) = \left\{ f(n): \text{there exists a positive constant } c \text{ such that } \lim_{n \rightarrow \infty} \left(\frac{f(n)}{g(n)} \right) \leq c \right\}$$

(Lecture 1) Introduction

Asymptotic Upper Bound (O-notation)

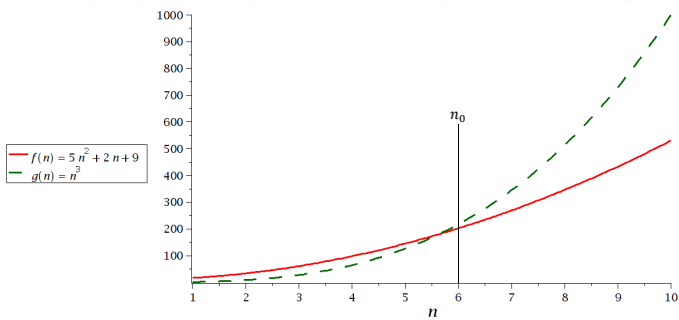
Recall that for Algorithm 1 we had, $T_1(n) \leq f(n)$, where,
 $f(n) = a_1n^2 + a_2n + a_3$, for constants a_1, a_2 and a_3 .
Suppose, $a_1 = 5, a_2 = 2$ and $a_3 = 9$.
Then $f(n) = 5n^2 + 2n + 9$.
We will now derive asymptotic bounds for $f(n)$.

Asymptotic Upper Bound (O-notation)



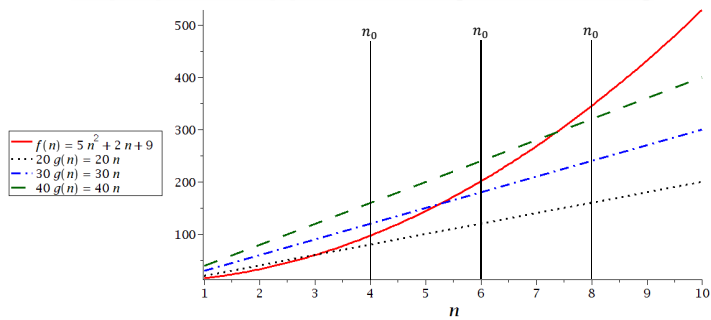
Let $g(n) = n^2$.
Then $f(n) = O(n^2)$ because:
 $0 \leq f(n) \leq cg(n)$ for $c = 6$ and $n \geq 5$.

Asymptotic Upper Bound (O-notation)



Let $g(n) = n^3$.
Then $f(n) = O(n^3)$ because:
 $0 \leq f(n) \leq cg(n)$ for $c = 1$ and $n \geq 6$.

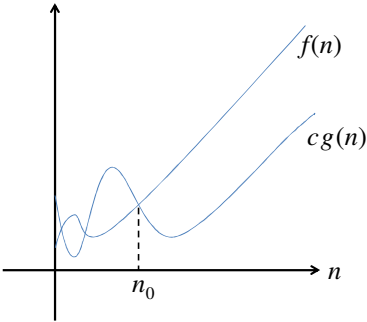
Asymptotic Upper Bound (O-notation)



Let $g(n) = n$.
Then $f(n) \neq O(n)$ because:
 $f(n) > cg(n)$ for any c and $n \geq \frac{c}{5}$.

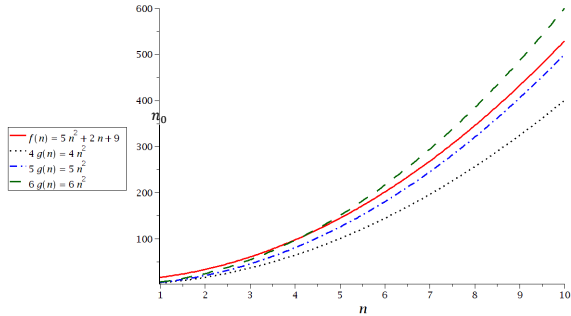
(Lecture 1) Introduction

Asymptotic Lower Bound (Ω-notation)



$$\Omega(g(n)) = \left\{ f(n): \text{there exist positive constants } c \text{ and } n_0 \text{ such that } 0 \leq cg(n) \leq f(n) \text{ for all } n \geq n_0 \right\}$$
$$\Omega(g(n)) = \left\{ f(n): \text{there exists a positive constant } c \text{ such that } \lim_{n \rightarrow \infty} \left(\frac{f(n)}{g(n)} \right) \geq c \right\}$$

Asymptotic Lower Bound (Ω-notation)

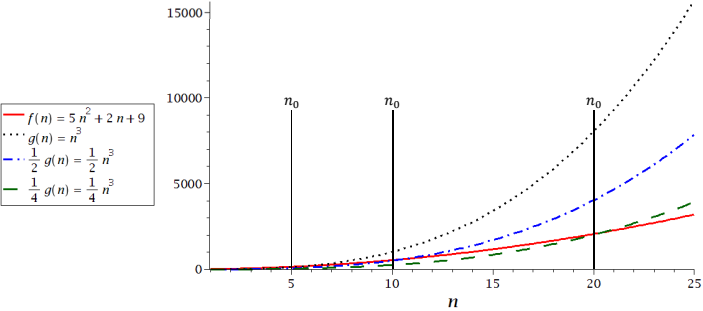


Let $g(n) = n^2$.

Then $f(n) = \Omega(n^2)$ because:

$$0 \leq cg(n) \leq f(n) \text{ for } c = 5 \text{ and } n \geq 1.$$

Asymptotic Lower Bound (Ω-notation)

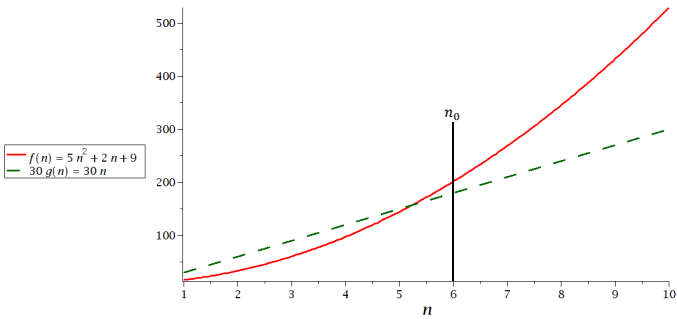


Let $g(n) = n^3$.

Then $f(n) \neq \Omega(n^3)$ because:

$$cg(n) > f(n) \text{ for any } c \text{ and } n \geq \frac{5}{c}.$$

Asymptotic Lower Bound (Ω-notation)



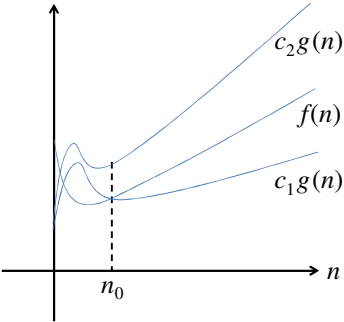
Let $g(n) = n$.

Then $f(n) = \Omega(n)$ because:

$$0 \leq cg(n) \leq f(n) \text{ for } c = 30 \text{ and } n \geq 6.$$

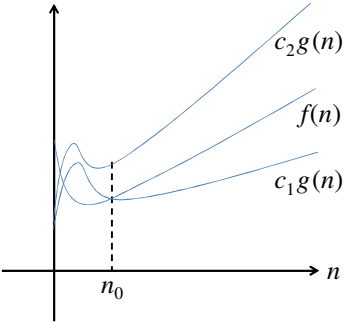
(Lecture 1) Introduction

Asymptotic Tight Bound (Θ-notation)



$$\Theta(g(n)) = \left\{ f(n): \text{there exist positive constants } c_1, c_2 \text{ and } n_0 \text{ such that } 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n) \text{ for all } n \geq n_0 \right\}$$
$$\Theta(g(n)) = \left\{ f(n): \text{there exist positive constants } c_1 \text{ and } c_2 \text{ such that } c_1 \leq \lim_{n \rightarrow \infty} \left(\frac{f(n)}{g(n)} \right) \leq c_2 \right\}$$

Asymptotic Tight Bound (Θ-notation)



$$(f(n) = O(g(n))) \wedge (f(n) = \Omega(g(n))) \Leftrightarrow (f(n) = \Theta(g(n)))$$

Asymptotic Tight Bound (Θ-notation)

$f(n) = 5n^2 + 2n + 9$

$f(n) = \Theta(n^2)$ because both $f(n) = O(n^2)$ and $f(n) = \Omega(n^2)$ hold.

$f(n) \neq \Theta(n^3)$ because though $f(n) = O(n^3)$ holds, $f(n) \neq \Omega(n^3)$.

$f(n) \neq \Theta(n)$ because though $f(n) = \Omega(n)$ holds, $f(n) \neq O(n)$.

Asymptotic Strict Upper Bound (o-notation)

$$o(g(n)) = \left\{ f(n): \text{there exist positive constants } c \text{ and } n_0 \text{ such that } 0 \leq f(n) \leq cg(n) \text{ for all } n \geq n_0 \right\}$$
$$o(g(n)) = \left\{ f(n): \text{there exists a positive constant } c \text{ such that } \lim_{n \rightarrow \infty} \left(\frac{f(n)}{g(n)} \right) \leq c \right\}$$

$$o(g(n)) = \left\{ f(n): \lim_{n \rightarrow \infty} \left(\frac{f(n)}{g(n)} \right) = 0 \right\}$$

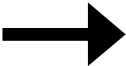
(Lecture 1) Introduction

Asymptotic Strict Lower Bound (ω -notation)

$$\Omega(g(n)) = \left\{ f(n): \text{there exist positive constants } c \text{ and } n_0 \text{ such that } 0 \leq cg(n) \leq f(n) \text{ for all } n \geq n_0 \right\}$$

$$\Omega(g(n)) = \left\{ f(n): \text{there exists a positive constant } c \text{ such that } \lim_{n \rightarrow \infty} \left(\frac{f(n)}{g(n)} \right) \geq c \right\}$$

$$\omega(g(n)) = \left\{ f(n): \lim_{n \rightarrow \infty} \left(\frac{g(n)}{f(n)} \right) = 0 \right\}$$



Comparing Functions: Transitivity

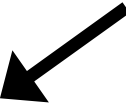
$$f(n) = O(g(n)) \text{ and } g(n) = O(h(n)) \Rightarrow f(n) = O(h(n))$$

$$f(n) = \Omega(g(n)) \text{ and } g(n) = \Omega(h(n)) \Rightarrow f(n) = \Omega(h(n))$$

$$f(n) = \Theta(g(n)) \text{ and } g(n) = \Theta(h(n)) \Rightarrow f(n) = \Theta(h(n))$$

$$f(n) = o(g(n)) \text{ and } g(n) = o(h(n)) \Rightarrow f(n) = o(h(n))$$

$$f(n) = \omega(g(n)) \text{ and } g(n) = \omega(h(n)) \Rightarrow f(n) = \omega(h(n))$$



Comparing Functions: Reflexivity

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

$$f(n) = \Omega(f(n))$$

$$f(n) = \Theta(f(n))$$



Comparing Functions: Symmetry

$$f(n) = \Theta(g(n)) \text{ if and only if } g(n) = \Theta(f(n))$$

Comparing Functions: Transpose Symmetry

$$f(n) = O(g(n)) \text{ if and only if } g(n) = \Omega(f(n))$$

$$f(n) = \Omega(g(n)) \text{ if and only if } g(n) = O(f(n))$$

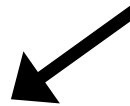


Adding Functions

$$O(f(n)) + O(g(n)) = O(f(n) + g(n))$$

$$\Omega(f(n)) + \Omega(g(n)) = \Omega(f(n) + g(n))$$

$$\Theta(f(n)) + \Theta(g(n)) = \Theta(f(n) + g(n))$$



Multiplying Functions by Constants

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

$$\Omega(c f(n)) = \Omega(f(n))$$

$$\Theta(c f(n)) = \Theta(f(n))$$



Multiplying Two Functions

$$O(f(n)) \times O(g(n)) = O(f(n) \times g(n))$$

$$\Omega(f(n)) \times \Omega(g(n)) = \Omega(f(n) \times g(n))$$

$$\Theta(f(n)) \times \Theta(g(n)) = \Theta(f(n) \times g(n))$$

