南方科技大学
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

CS215 DISCRETE MATHEMATICS FOR COMPUTER SCIENCE

Dr. Qi Wang

Department of Computer Science and Engineering
Office: Room413, CoE South Tower
Email: wangqi@sustech.edu.cn

Growth Rates of Solutions to Recurrences

- Divide and conquer algorithms
- Iterating recurrences
- Three different behaviors

Divide and conquer algorithms

- We just analyzed recurrences of the form

$$T(n) = \begin{cases} b & \text{if } n = 0 \\ r \cdot T(n-1) + a & \text{if } n > 0 \end{cases}$$

Divide and conquer algorithms

- We just analyzed recurrences of the form

$$T(n) = \begin{cases} b & \text{if } n = 0 \\ r \cdot T(n-1) + a & \text{if } n > 0 \end{cases}$$

These corresponded to the analysis of recursive algorithms in which a problem of size n is solved by recursively solving a problem of size $n - 1$.

Divide and conquer algorithms

- We just analyzed recurrences of the form

$$T(n) = \begin{cases} b & \text{if } n = 0 \\ r \cdot T(n-1) + a & \text{if } n > 0 \end{cases}$$

These corresponded to the analysis of recursive algorithms in which a problem of size n is solved by recursively solving a problem of size $n-1$.

$$T(n) = r^n b + a\frac{1-r^n}{1-r}$$

Divide and conquer algorithms

- We just analyzed recurrences of the form

$$T(n) = \begin{cases} b & \text{if } n = 0 \\ r \cdot T(n-1) + a & \text{if } n > 0 \end{cases}$$

These corresponded to the analysis of recursive algorithms in which a problem of size n is solved by recursively solving a problem of size $n-1$.

$$T(n) = r^n b + a\frac{1-r^n}{1-r}$$

We will now look at recurrences of the form

$$T(n) = \begin{cases} \text{something given} & \text{if } n \leq n_0 \\ r \cdot T(n/m) + a & \text{if } n > n_0 \end{cases}$$

Binary Search

- Someone has chosen a number x between 1 and n. We need to discover x.

Binary Search

- Someone has chosen a number x between 1 and n. We need to discover x.

  We are only allowed to ask two types of questions:

Binary Search

- Someone has chosen a number x between 1 and n. We need to discover x.

  We are only allowed to ask two types of questions:

  - ◇ Is x greater than k?
  - ◇ Is x equal to k?

Binary Search

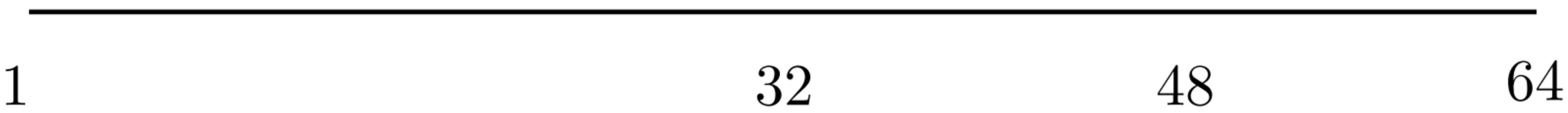
- Someone has chosen a number x between 1 and n. We need to discover x.

  We are only allowed to ask two types of questions:

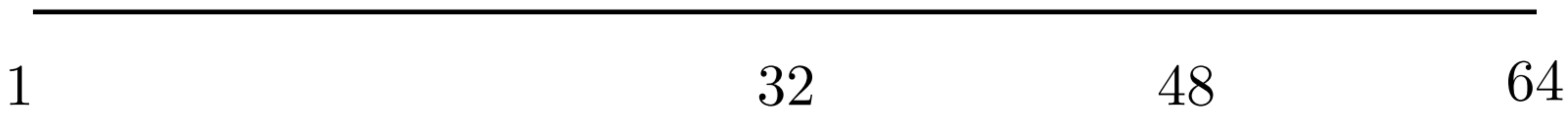
  - Is x greater than k?
  - Is x equal to k?

  Our strategy will be to always ask greater than questions, at each step halving our search range, until the range only contains one number, when we ask a final equal to question.

Binary Search Example

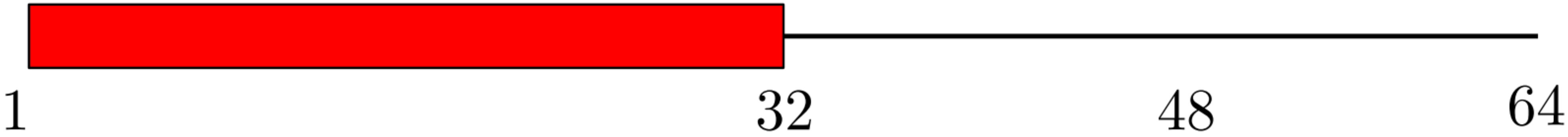


Binary Search Example



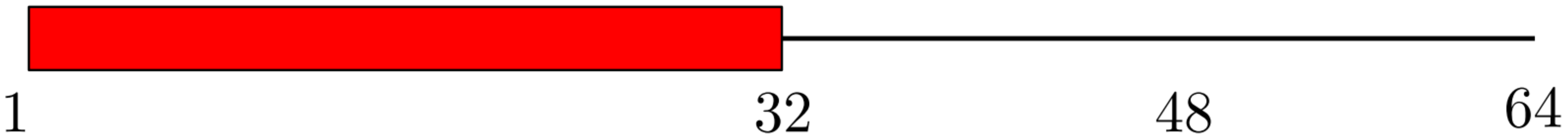
Is $x > 32$?

Binary Search Example



Is $x > 32$?   Answer: Yes

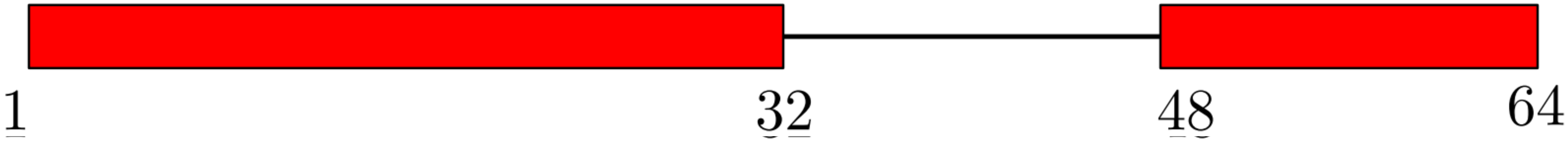
Binary Search Example



Is $x > 32?$ Answer: Yes

Is $x > 48?$

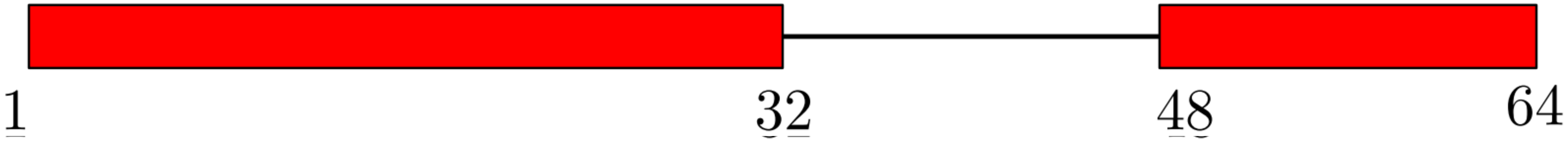
Binary Search Example



Is $x > 32?$ Answer: Yes

Is $x > 48?$ Answer: No

Binary Search Example

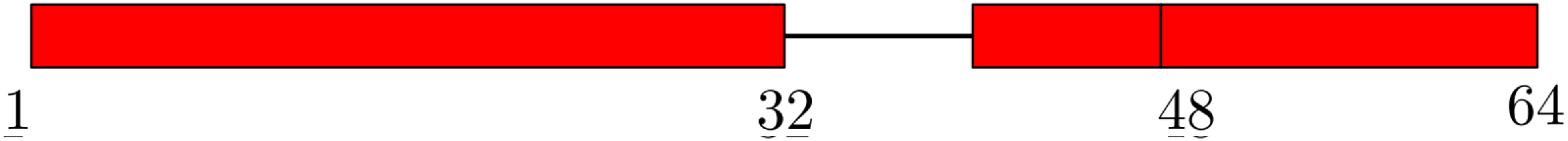


Is $x > 32$? Answer: Yes

Is $x > 48$? Answer: No

Is $x > 40$?

Binary Search Example

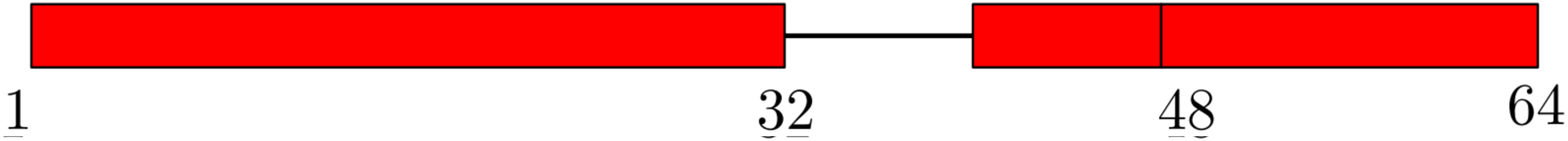


Is $x > 32?$ Answer: Yes

Is $x > 48?$ Answer: No

Is $x > 40?$ Answer: No

Binary Search Example



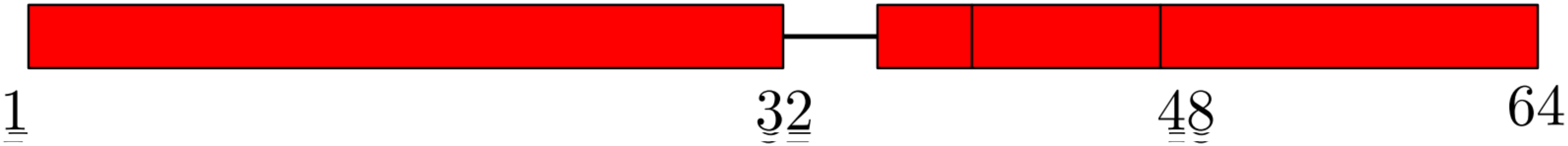
Is $x > 32?$ Answer: Yes

Is $x > 48?$ Answer: No

Is $x > 40?$ Answer: No

Is $x > 36?$

Binary Search Example



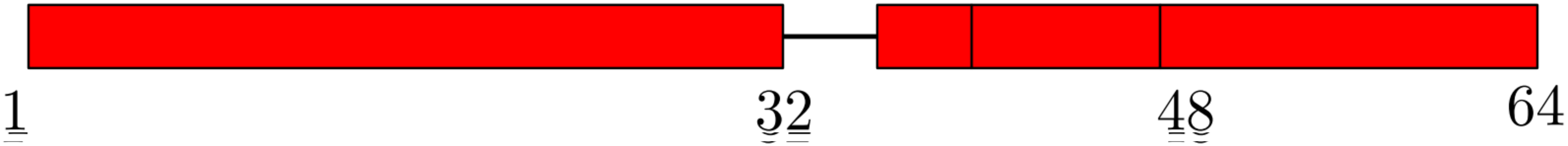
Is $x > 32$? Answer: Yes

Is $x > 48$? Answer: No

Is $x > 40$? Answer: No

Is $x > 36$? Answer: No

Binary Search Example



Is $x > 32$? Answer: Yes

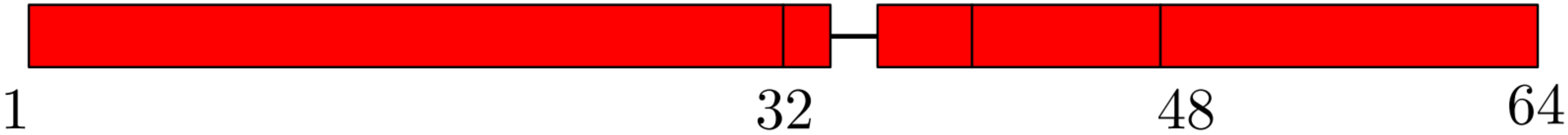
Is $x > 48$? Answer: No

Is $x > 40$? Answer: No

Is $x > 36$? Answer: No

Is $x > 34$?

Binary Search Example



Is $x > 32?$ Answer: Yes

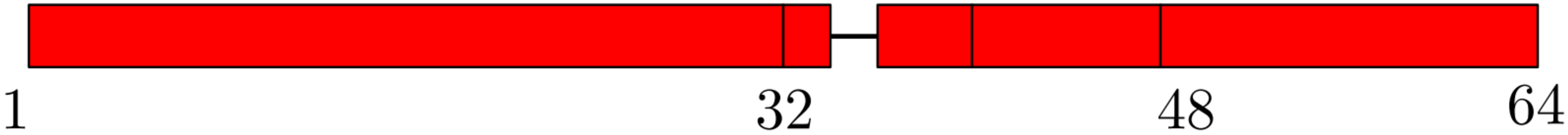
Is $x > 48?$ Answer: No

Is $x > 40?$ Answer: No

Is $x > 36?$ Answer: No

Is $x > 34?$ Answer: Yes

Binary Search Example



Is $x > 32$? Answer: Yes

Is $x > 48$? Answer: No

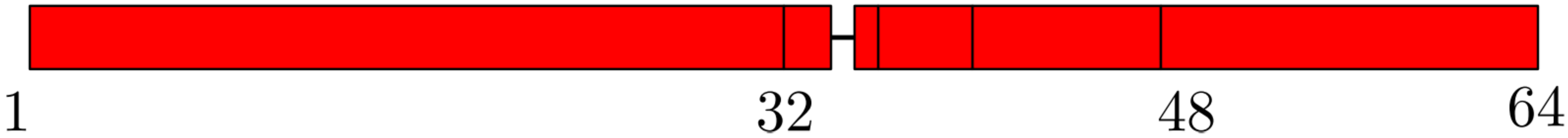
Is $x > 40$? Answer: No

Is $x > 36$? Answer: No

Is $x > 34$? Answer: Yes

Is $x > 35$?

Binary Search Example



Is $x > 32$? Answer: Yes

Is $x > 48$? Answer: No

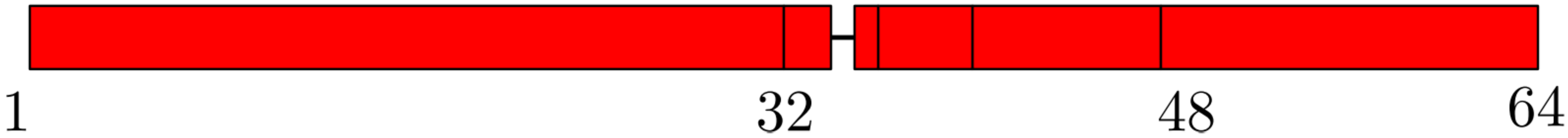
Is $x > 40$? Answer: No

Is $x > 36$? Answer: No

Is $x > 34$? Answer: Yes

Is $x > 35$? Answer: No

Binary Search Example



Is $x > 32$? Answer: Yes

Is $x > 48$? Answer: No

Is $x > 40$? Answer: No

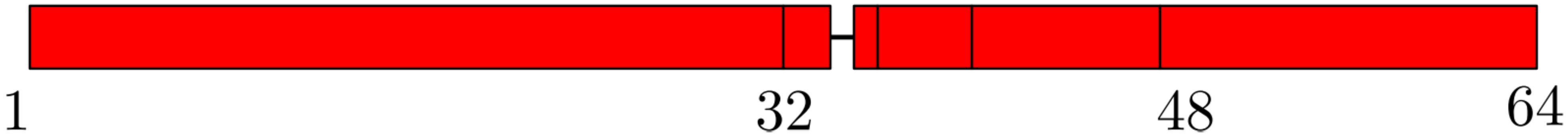
Is $x > 36$? Answer: No

Is $x > 34$? Answer: Yes

Is $x > 35$? Answer: No

Is $x = 35$?

Binary Search Example



Is $x > 32$? Answer: Yes

Is $x > 48$? Answer: No

Is $x > 40$? Answer: No

Is $x > 36$? Answer: No

Is $x > 34$? Answer: Yes

Is $x > 35$? Answer: No

Is $x = 35$? Answer: BINGO!

Binary Search Example

- Method: Each guess reduces the problem to one in which the range is only half as big.

Binary Search Example

- Method: Each guess reduces the problem to one in which the range is only half as big.

  This divides the original problem into one that is only half as big; we can now (recursively) conquer this smaller problem.

Binary Search Example

- Method: Each guess reduces the problem to one in which the range is only half as big.

  This divides the original problem into one that is only half as big; we can now (recursively) conquer this smaller problem.

  Note: When n is a power of 2, $T(n)$, the number of questions in a binary search on $[1, n]$, satisfies

$$T(n) = \begin{cases} 1 & \text{if } n = 1 \\ T(n/2) + 1 & \text{if } n \geq 2 \end{cases}$$

Binary Search Example

- Method: Each guess reduces the problem to one in which the range is only half as big.

  This divides the original problem into one that is only half as big; we can now (recursively) conquer this smaller problem.

  Note: When n is a power of 2, $T(n)$, the number of questions in a binary search on $[1, n]$, satisfies

$$T(n) = \begin{cases} 1 & \text{if } n = 1 \\ T(n/2) + 1 & \text{if } n \geq 2 \end{cases}$$

  This can also be proved inductively, similar to the tower of Hanoi recurrence.

Binary Search Example

- $T(n)$: number of questions in a binary search on $[1, n]$

Binary Search Example

- $T(n)$: number of questions in a binary search on $[1, n]$

  Assume: n is a power of 2. Give recurrence for $T(n)$

Binary Search Example

- $T(n)$: number of questions in a binary search on $[1, n]$

  Assume: n is a power of 2.   Give recurrence for $T(n)$

$$T(n) = \begin{cases} 1 & \text{if } n = 1 \\ T(n/2) + 1 & \text{if } n \geq 2 \end{cases}$$

Binary Search Example

- $T(n)$: number of questions in a binary search on $[1, n]$

  Assume: n is a power of 2. Give recurrence for $T(n)$

$$T(n) = \begin{cases} 1 & \text{if } n = 1 \\ T(n/2) + 1 & \text{if } n \geq 2 \end{cases}$$

  Number of questions needed for binary search on n items is:

Binary Search Example

- $T(n)$: number of questions in a binary search on $[1, n]$

  Assume: n is a power of 2. Give recurrence for $T(n)$

$$T(n) = \begin{cases} 1 & \text{if } n = 1 \\ T(n/2) + 1 & \text{if } n \geq 2 \end{cases}$$

Number of questions needed for binary search on n items is:

first step

$+$

time to perform binary search on the remaining $n/2$ items

Binary Search Example

- $T(n)$: number of questions in a binary search on $[1, n]$

  Assume: n is a power of 2. Give recurrence for $T(n)$

$$T(n) = \begin{cases} 1 & \text{if } n = 1 \\ T(n/2) + 1 & \text{if } n \geq 2 \end{cases}$$

Number of questions needed for binary search on n items is:

first step

$+$

time to perform binary search on the remaining $n/2$ items

Base case (1 item): $T(1) = 1$ to ask: "Is the number k?"

Binary Search Example

- $$(*) \qquad T(n) = \begin{cases} C_1 & \text{if } n = 1 \\ T(\lceil n/2 \rceil) + C_2 & \text{if } n \geq 2 \end{cases}$$

  For simplicity, we will (usually) assume that n is a power of 2 (or sometimes 3 or 4) and also often that constants such as C_1, C_2 are 1. This will let us replace a recurrence such as (*) by one such as (**).

Binary Search Example

- $$(*) \qquad T(n) = \begin{cases} C_1 & \text{if } n = 1 \\ T(\lceil n/2 \rceil) + C_2 & \text{if } n \geq 2 \end{cases}$$

  For simplicity, we will (usually) assume that n is a power of 2 (or sometimes 3 or 4) and also often that constants such as C_1, C_2 are 1. This will let us replace a recurrence such as (*) by one such as (**).

  $$(**) \qquad T(n) = \begin{cases} 1 & \text{if } n = 1 \\ T(n/2) + 1 & \text{if } n \geq 2 \end{cases}$$

Binary Search Example

- $$(*) \qquad T(n) = \begin{cases} C_1 & \text{if } n = 1 \\ T(\lceil n/2 \rceil) + C_2 & \text{if } n \geq 2 \end{cases}$$

  For simplicity, we will (usually) assume that n is a power of 2 (or sometimes 3 or 4) and also often that constants such as C_1, C_2 are 1. This will let us replace a recurrence such as (*) by one such as (**).

  $$(**) \qquad T(n) = \begin{cases} 1 & \text{if } n = 1 \\ T(n/2) + 1 & \text{if } n \geq 2 \end{cases}$$

  In practice, the solution of (*) will be very close to that of (**) (this can be proved mathematically). Hence, we can restrict attention to (**).

Growth Rates of Solutions to Recurrences

- Divide and conquer algorithms

- Iterating recurrences

- Three different behaviors

Iterating Recurrences: Example 1

- $$(*) \qquad T(n) = \begin{cases} T(1) & \text{if } n = 1 \\ 2T(n/2) + n & \text{if } n \geq 2 \end{cases}$$

Iterating Recurrences: Example 1

- $$(*) \qquad T(n) = \begin{cases} T(1) & \text{if } n = 1 \\ 2T(n/2) + n & \text{if } n \geq 2 \end{cases}$$

  This corresponds to solving a problem of size n, by

  (i) solving 2 subproblems of size $n/2$ and
  (ii) doing n units of additional work

  or using $T(1)$ work for "bottom" case of $n = 1$

Iterating Recurrences: Example 1

- $$(*) \qquad T(n) = \begin{cases} T(1) & \text{if } n = 1 \\ 2T(n/2) + n & \text{if } n \geq 2 \end{cases}$$

This corresponds to solving a problem of size n, by

(i) solving 2 subproblems of size $n/2$ and
(ii) doing n units of additional work

or using $T(1)$ work for "bottom" case of $n = 1$

In the course "Analysis of Algorithms", this is exactly how Mergesort works.

Iterating Recurrences: Example 1

- $$(*) \qquad T(n) = \begin{cases} T(1) & \text{if } n = 1 \\ 2T(n/2) + n & \text{if } n \geq 2 \end{cases}$$

This corresponds to solving a problem of size n, by

(i) solving 2 subproblems of size $n/2$ and
(ii) doing n units of additional work

or using $T(1)$ work for "bottom" case of $n = 1$

In the course "Analysis of Algorithms", this is exactly how Mergesort works.

We now see how to solve (*) by algebraically iterating the recurrence.

Iterating Recurrences: Example 1

- Algebraically iterating the recurrence

$$T(n) = 2T\left(\frac{n}{2}\right) + n$$

Iterating Recurrences: Example 1

- Algebraically iterating the recurrence

  Assume that n is a power of 2

$$T(n) = 2T\left(\frac{n}{2}\right) + n \qquad = 2\left(2T\left(\frac{n}{4}\right) + \frac{n}{2}\right) + n$$

Iterating Recurrences: Example 1

- Algebraically iterating the recurrence

  Assume that n is a power of 2

$$\begin{aligned} T(n) = 2T\left(\tfrac{n}{2}\right) + n \quad &= 2\left(2T\left(\tfrac{n}{4}\right) + \tfrac{n}{2}\right) + n \\ = 4T\left(\tfrac{n}{4}\right) + 2n \quad &= 4\left(2T\left(\tfrac{n}{8}\right) + \tfrac{n}{4}\right) + 2n \end{aligned}$$

Iterating Recurrences: Example 1

- Algebraically iterating the recurrence

  Assume that n is a power of 2

$$\begin{aligned} T(n) = 2T\left(\frac{n}{2}\right) + n \quad & = 2\left(2T\left(\frac{n}{4}\right) + \frac{n}{2}\right) + n \\ = 4T\left(\frac{n}{4}\right) + 2n \quad & = 4\left(2T\left(\frac{n}{8}\right) + \frac{n}{4}\right) + 2n \\ = 8T\left(\frac{n}{8}\right) + 3n \quad & \end{aligned}$$

Iterating Recurrences: Example 1

- Algebraically iterating the recurrence

  Assume that n is a power of 2

$$
\begin{aligned}
T(n) = 2T\left(\tfrac{n}{2}\right) + n \quad & = 2\left(2T\left(\tfrac{n}{4}\right) + \tfrac{n}{2}\right) + n \\
= 4T\left(\tfrac{n}{4}\right) + 2n \quad & = 4\left(2T\left(\tfrac{n}{8}\right) + \tfrac{n}{4}\right) + 2n \\
= 8T\left(\tfrac{n}{8}\right) + 3n \quad & \\
\vdots \qquad \vdots \quad & \\
= 2^i T\left(\tfrac{n}{2^i}\right) + in \quad &
\end{aligned}
$$

Iterating Recurrences: Example 1

- Algebraically iterating the recurrence

  Assume that n is a power of 2

$$T(n) = 2T\left(\frac{n}{2}\right) + n \qquad = 2\left(2T\left(\frac{n}{4}\right) + \frac{n}{2}\right) + n$$

$$= 4T\left(\frac{n}{4}\right) + 2n \qquad = 4\left(2T\left(\frac{n}{8}\right) + \frac{n}{4}\right) + 2n$$

$$= 8T\left(\frac{n}{8}\right) + 3n$$

$$\vdots \qquad \vdots$$

$$= 2^i T\left(\frac{n}{2^i}\right) + in$$

$$\vdots \qquad \vdots$$

$$= 2^{\log_2 n} T\left(\frac{n}{2^{\log_2 n}}\right) + (\log_2 n)n$$

End when $i = \log_2 n$

Iterating Recurrences: Example 1

- Algebraically iterating the recurrence

  Assume that n is a power of 2

$$T(n) = 2T\left(\frac{n}{2}\right) + n \qquad = 2\left(2T\left(\frac{n}{4}\right) + \frac{n}{2}\right) + n$$

$$= 4T\left(\frac{n}{4}\right) + 2n \qquad = 4\left(2T\left(\frac{n}{8}\right) + \frac{n}{4}\right) + 2n$$

$$= 8T\left(\frac{n}{8}\right) + 3n$$

$$\vdots \qquad \vdots$$

$$= 2^i T\left(\frac{n}{2^i}\right) + in$$

End when $i = \log_2 n$

$$\vdots \qquad \vdots$$

$$= 2^{\log_2 n} T\left(\frac{n}{2^{\log_2 n}}\right) + (\log_2 n)n$$

$$= nT(1) + n\log_2 n$$

Iterating Recurrences: Example 1

- We just iterated the recurrence to derive that the solution to

$$(*) \qquad T(n) = \begin{cases} T(1) & \text{if } n = 1 \\ 2T(n/2) + n & \text{if } n \geq 2 \end{cases}$$

is $nT(1) + n\log_2 n$.

Iterating Recurrences: Example 1

- We just iterated the recurrence to derive that the solution to

$$(*) \qquad T(n) = \begin{cases} T(1) & \text{if } n = 1 \\ 2T(n/2) + n & \text{if } n \geq 2 \end{cases}$$

is $nT(1) + n\log_2 n$.

Note: Technically, we still need to use **induction** to prove that our solution is correct. Practically, we never explicitly perform this step, since it is obvious how the induction would work.

Iterating Recurrences: Example 2

- $$(*) \qquad T(n) = \begin{cases} 1 & \text{if } n = 1 \\ T(n/2) + 1 & \text{if } n \geq 2 \end{cases}$$

Iterating Recurrences: Example 2

- $$(*) \qquad T(n) = \begin{cases} 1 & \text{if } n = 1 \\ T(n/2) + 1 & \text{if } n \geq 2 \end{cases}$$

$$T(n) = T\left(\frac{n}{2}\right) + 1$$

Iterating Recurrences: Example 2

- $$(*) \qquad T(n) = \begin{cases} 1 & \text{if } n = 1 \\ T(n/2) + 1 & \text{if } n \geq 2 \end{cases}$$

$$T(n) = T\left(\frac{n}{2}\right) + 1 \qquad = \left(T\left(\frac{n}{2^2}\right) + 1\right) + 1$$

Iterating Recurrences: Example 2

- $$(*) \qquad T(n) = \begin{cases} 1 & \text{if } n = 1 \\ T(n/2) + 1 & \text{if } n \geq 2 \end{cases}$$

$$T(n) = T\left(\frac{n}{2}\right) + 1 \qquad = \left(T\left(\frac{n}{2^2}\right) + 1\right) + 1$$

$$= T\left(\frac{n}{2^2}\right) + 2$$

Iterating Recurrences: Example 2

- $$(*) \qquad T(n) = \begin{cases} 1 & \text{if } n = 1 \\ T(n/2) + 1 & \text{if } n \geq 2 \end{cases}$$

$$\begin{aligned} T(n) &= T\left(\frac{n}{2}\right) + 1 & &= \left(T\left(\frac{n}{2^2}\right) + 1\right) + 1 \\ &= T\left(\frac{n}{2^2}\right) + 2 & &= \left(T\left(\frac{n}{2^3}\right) + 1\right) + 2 \end{aligned}$$

Iterating Recurrences: Example 2

- $$(*) \qquad T(n) = \begin{cases} 1 & \text{if } n = 1 \\ T(n/2) + 1 & \text{if } n \geq 2 \end{cases}$$

$$\begin{aligned} T(n) &= T\left(\frac{n}{2}\right) + 1 & &= \left(T\left(\frac{n}{2^2}\right) + 1\right) + 1 \\ &= T\left(\frac{n}{2^2}\right) + 2 & &= \left(T\left(\frac{n}{2^3}\right) + 1\right) + 2 \\ &= T\left(\frac{n}{2^3}\right) + 3 \end{aligned}$$

Iterating Recurrences: Example 2

- $$(*) \qquad T(n) = \begin{cases} 1 & \text{if } n = 1 \\ T(n/2) + 1 & \text{if } n \geq 2 \end{cases}$$

$$\begin{aligned} T(n) &= T\left(\tfrac{n}{2}\right) + 1 & &= \left(T\left(\tfrac{n}{2^2}\right) + 1\right) + 1 \\ &= T\left(\tfrac{n}{2^2}\right) + 2 & &= \left(T\left(\tfrac{n}{2^3}\right) + 1\right) + 2 \\ &= T\left(\tfrac{n}{2^3}\right) + 3 & & \\ &\ \ \vdots \qquad \vdots & & \\ &= T\left(\tfrac{n}{2^i}\right) + i & & \end{aligned}$$

Iterating Recurrences: Example 2

- $$(*) \qquad T(n) = \begin{cases} 1 & \text{if } n = 1 \\ T(n/2) + 1 & \text{if } n \geq 2 \end{cases}$$

$$\begin{aligned} T(n) &= T\left(\frac{n}{2}\right) + 1 & &= \left(T\left(\frac{n}{2^2}\right) + 1\right) + 1 \\ &= T\left(\frac{n}{2^2}\right) + 2 & &= \left(T\left(\frac{n}{2^3}\right) + 1\right) + 2 \\ &= T\left(\frac{n}{2^3}\right) + 3 \\ &\ \vdots \qquad \vdots \\ &= T\left(\frac{n}{2^i}\right) + i \\ &\ \vdots \qquad \vdots \\ &= T\left(\frac{n}{2^{\log_2 n}}\right) + \log_2 n \end{aligned}$$

Iterating Recurrences: Example 2

- $$(*) \qquad T(n) = \begin{cases} 1 & \text{if } n = 1 \\ T(n/2) + 1 & \text{if } n \geq 2 \end{cases}$$

$$\begin{aligned}
T(n) &= T\left(\frac{n}{2}\right) + 1 & &= \left(T\left(\frac{n}{2^2}\right) + 1\right) + 1 \\
&= T\left(\frac{n}{2^2}\right) + 2 & &= \left(T\left(\frac{n}{2^3}\right) + 1\right) + 2 \\
&= T\left(\frac{n}{2^3}\right) + 3 \\
&\ \ \vdots \qquad \vdots \\
&= T\left(\frac{n}{2^i}\right) + i \\
&\ \ \vdots \qquad \vdots \\
&= T\left(\frac{n}{2^{\log_2 n}}\right) + \log_2 n & &= 1 + \log_2 n
\end{aligned}$$

Iterating Recurrences: Example 3

- $$(*) \qquad T(n) = \begin{cases} 1 & \text{if } n = 1 \\ T(n/2) + n & \text{if } n \geq 2 \end{cases}$$

Iterating Recurrences: Example 3

- $$(*) \qquad T(n) = \begin{cases} 1 & \text{if } n = 1 \\ T(n/2) + n & \text{if } n \geq 2 \end{cases}$$

$$\begin{aligned} T(n) &= T\left(\frac{n}{2}\right) + n \\ &= T\left(\frac{n}{2^2}\right) + \frac{n}{2} + n \end{aligned}$$

Iterating Recurrences: Example 3

- $$(*) \qquad T(n) = \begin{cases} 1 & \text{if } n = 1 \\ T(n/2) + n & \text{if } n \geq 2 \end{cases}$$

$$\begin{aligned} T(n) &= T\left(\frac{n}{2}\right) + n \\ &= T\left(\frac{n}{2^2}\right) + \frac{n}{2} + n \\ &= T\left(\frac{n}{2^3}\right) + \frac{n}{2^2} + \frac{n}{2} + n \end{aligned}$$

Iterating Recurrences: Example 3

- $$(*) \qquad T(n) = \begin{cases} 1 & \text{if } n = 1 \\ T(n/2) + n & \text{if } n \geq 2 \end{cases}$$

$$\begin{aligned} T(n) &= T\left(\frac{n}{2}\right) + n \\ &= T\left(\frac{n}{2^2}\right) + \frac{n}{2} + n \\ &= T\left(\frac{n}{2^3}\right) + \frac{n}{2^2} + \frac{n}{2} + n \\ &\vdots \qquad \vdots \\ &= T\left(\frac{n}{2^i}\right) + \frac{n}{2^{i-1}} + \cdots + \frac{n}{2^2} + \frac{n}{2} + n \end{aligned}$$

Iterating Recurrences: Example 3

- $$(*) \qquad T(n) = \begin{cases} 1 & \text{if } n = 1 \\ T(n/2) + n & \text{if } n \geq 2 \end{cases}$$

$$\begin{aligned}
T(n) &= T\left(\frac{n}{2}\right) + n \\
&= T\left(\frac{n}{2^2}\right) + \frac{n}{2} + n \\
&= T\left(\frac{n}{2^3}\right) + \frac{n}{2^2} + \frac{n}{2} + n \\
&\quad \vdots \qquad \vdots \\
&= T\left(\frac{n}{2^i}\right) + \frac{n}{2^{i-1}} + \cdots + \frac{n}{2^2} + \frac{n}{2} + n \\
&\quad \vdots \qquad \vdots \\
&= T\left(\frac{n}{2^{\log_2 n}}\right) + \frac{n}{2^{\log_2 n - 1}} + \cdots + \frac{n}{2^2} + \frac{n}{2} + n
\end{aligned}$$

Iterating Recurrences: Example 3

- $$(*) \qquad T(n) = \begin{cases} 1 & \text{if } n = 1 \\ T(n/2) + n & \text{if } n \geq 2 \end{cases}$$

$$\begin{aligned}
T(n) &= T\left(\frac{n}{2}\right) + n \\
&= T\left(\frac{n}{2^2}\right) + \frac{n}{2} + n \\
&= T\left(\frac{n}{2^3}\right) + \frac{n}{2^2} + \frac{n}{2} + n \\
&\ \ \vdots \qquad \vdots \\
&= T\left(\frac{n}{2^i}\right) + \frac{n}{2^{i-1}} + \cdots + \frac{n}{2^2} + \frac{n}{2} + n \\
&\ \ \vdots \qquad \vdots \\
&= T\left(\frac{n}{2^{\log_2 n}}\right) + \frac{n}{2^{\log_2 n-1}} + \cdots + \frac{n}{2^2} + \frac{n}{2} + n \\
&= 1 + 2 + 2^2 + \cdots + \frac{n}{2^2} + \frac{n}{2} + n
\end{aligned}$$

Iterating Recurrences: Example 3

- $$(*) \qquad T(n) = \begin{cases} 1 & \text{if } n = 1 \\ T(n/2) + n & \text{if } n \geq 2 \end{cases}$$

$$\begin{aligned}
T(n) &= T\left(\frac{n}{2}\right) + n \\
&= T\left(\frac{n}{2^2}\right) + \frac{n}{2} + n \\
&= T\left(\frac{n}{2^3}\right) + \frac{n}{2^2} + \frac{n}{2} + n \\
&\ \vdots \qquad \vdots \\
&= T\left(\frac{n}{2^i}\right) + \frac{n}{2^{i-1}} + \cdots + \frac{n}{2^2} + \frac{n}{2} + n \\
&\ \vdots \qquad \vdots \\
&= T\left(\frac{n}{2^{\log_2 n}}\right) + \frac{n}{2^{\log_2 n - 1}} + \cdots + \frac{n}{2^2} + \frac{n}{2} + n \\
&= 1 + 2 + 2^2 + \cdots + \frac{n}{2^2} + \frac{n}{2} + n \qquad = \Theta(n)
\end{aligned}$$

Iterating Recurrences: Example 4

- $$(*) \qquad T(n) = \begin{cases} 1 & \text{if } n < 3 \\ 3T(n/3) + n & \text{if } n \geq 3 \end{cases}$$

Iterating Recurrences: Example 4

- $$(*) \qquad T(n) = \begin{cases} 1 & \text{if } n < 3 \\ 3T(n/3) + n & \text{if } n \geq 3 \end{cases}$$

$$T(n) = 3T\left(\frac{n}{3}\right) + n$$

Iterating Recurrences: Example 4

- $$(*) \qquad T(n) = \begin{cases} 1 & \text{if } n < 3 \\ 3T(n/3) + n & \text{if } n \geq 3 \end{cases}$$

$$T(n) = 3T\left(\frac{n}{3}\right) + n \qquad = 3\left(3T\left(\frac{n}{3^2}\right) + \frac{n}{3}\right) + n$$

Iterating Recurrences: Example 4

- $$(*) \qquad T(n) = \begin{cases} 1 & \text{if } n < 3 \\ 3T(n/3) + n & \text{if } n \geq 3 \end{cases}$$

$$T(n) = 3T\left(\frac{n}{3}\right) + n \qquad = 3\left(3T\left(\frac{n}{3^2}\right) + \frac{n}{3}\right) + n$$

$$= 3^2 T\left(\frac{n}{3^2}\right) + 2n$$

Iterating Recurrences: Example 4

- $$(*) \qquad T(n) = \begin{cases} 1 & \text{if } n < 3 \\ 3T(n/3) + n & \text{if } n \geq 3 \end{cases}$$

$$\begin{aligned} T(n) = 3T\left(\tfrac{n}{3}\right) + n \qquad &= 3\left(3T\left(\tfrac{n}{3^2}\right) + \tfrac{n}{3}\right) + n \\ = 3^2T\left(\tfrac{n}{3^2}\right) + 2n \quad &= 3^2\left(3T\left(\tfrac{n}{3^3}\right) + \tfrac{n}{3^2}\right) + 2n \end{aligned}$$

Iterating Recurrences: Example 4

- $$(*) \qquad T(n) = \begin{cases} 1 & \text{if } n < 3 \\ 3T(n/3) + n & \text{if } n \geq 3 \end{cases}$$

$$\begin{aligned} T(n) &= 3T\left(\tfrac{n}{3}\right) + n & &= 3\left(3T\left(\tfrac{n}{3^2}\right) + \tfrac{n}{3}\right) + n \\ &= 3^2T\left(\tfrac{n}{3^2}\right) + 2n & &= 3^2\left(3T\left(\tfrac{n}{3^3}\right) + \tfrac{n}{3^2}\right) + 2n \\ &= 3^3T\left(\tfrac{n}{3^3}\right) + 3n \end{aligned}$$

Iterating Recurrences: Example 4

- $$(*) \qquad T(n) = \begin{cases} 1 & \text{if } n < 3 \\ 3T(n/3) + n & \text{if } n \geq 3 \end{cases}$$

$$\begin{aligned} T(n) &= 3T\left(\tfrac{n}{3}\right) + n && = 3\left(3T\left(\tfrac{n}{3^2}\right) + \tfrac{n}{3}\right) + n \\ &= 3^2T\left(\tfrac{n}{3^2}\right) + 2n && = 3^2\left(3T\left(\tfrac{n}{3^3}\right) + \tfrac{n}{3^2}\right) + 2n \\ &= 3^3T\left(\tfrac{n}{3^3}\right) + 3n \\ &\ \ \vdots \qquad\quad \vdots \\ &= 3^iT\left(\tfrac{n}{3^i}\right) + in \end{aligned}$$

Iterating Recurrences: Example 4

- $$(*) \qquad T(n) = \begin{cases} 1 & \text{if } n < 3 \\ 3T(n/3) + n & \text{if } n \geq 3 \end{cases}$$

$$\begin{aligned} T(n) &= 3T\left(\frac{n}{3}\right) + n & &= 3\left(3T\left(\frac{n}{3^2}\right) + \frac{n}{3}\right) + n \\ &= 3^2 T\left(\frac{n}{3^2}\right) + 2n & &= 3^2\left(3T\left(\frac{n}{3^3}\right) + \frac{n}{3^2}\right) + 2n \\ &= 3^3 T\left(\frac{n}{3^3}\right) + 3n \\ &\vdots & &\vdots \\ &= 3^i T\left(\frac{n}{3^i}\right) + in \\ &\vdots & &\vdots \\ &= 3^{\log_3 n} T\left(\frac{n}{3^{\log_3 n}}\right) + n \log_3 n \end{aligned}$$

Iterating Recurrences: Example 4

- $$(*) \qquad T(n) = \begin{cases} 1 & \text{if } n < 3 \\ 3T(n/3) + n & \text{if } n \geq 3 \end{cases}$$

$$\begin{aligned} T(n) &= 3T\left(\tfrac{n}{3}\right) + n & &= 3\left(3T\left(\tfrac{n}{3^2}\right) + \tfrac{n}{3}\right) + n \\ &= 3^2T\left(\tfrac{n}{3^2}\right) + 2n & &= 3^2\left(3T\left(\tfrac{n}{3^3}\right) + \tfrac{n}{3^2}\right) + 2n \\ &= 3^3T\left(\tfrac{n}{3^3}\right) + 3n \\ &\vdots \qquad \vdots \\ &= 3^iT\left(\tfrac{n}{3^i}\right) + in \\ &\vdots \qquad \vdots \\ &= 3^{\log_3 n}T\left(\tfrac{n}{3^{\log_3 n}}\right) + n\log_3 n & &= n + n\log_3 n \end{aligned}$$

Iterating Recurrences: Example 5

- $$(*) \qquad T(n) = \begin{cases} 1 & \text{if } n = 1 \\ 4T(n/2) + n & \text{if } n \geq 2 \end{cases}$$

Iterating Recurrences: Example 5

- $$(*) \qquad T(n) = \begin{cases} 1 & \text{if } n = 1 \\ 4T(n/2) + n & \text{if } n \geq 2 \end{cases}$$

$$T(n) = 4T\left(\frac{n}{2}\right) + n$$

Iterating Recurrences: Example 5

- $$(*) \qquad T(n) = \begin{cases} 1 & \text{if } n = 1 \\ 4T(n/2) + n & \text{if } n \geq 2 \end{cases}$$

$$T(n) = 4T\left(\frac{n}{2}\right) + n \qquad = 4\left(4T\left(\frac{n}{2^2}\right) + \frac{n}{2}\right) + n$$

Iterating Recurrences: Example 5

- $$(*) \qquad T(n) = \begin{cases} 1 & \text{if } n = 1 \\ 4T(n/2) + n & \text{if } n \geq 2 \end{cases}$$

$$T(n) = 4T\left(\frac{n}{2}\right) + n \qquad = 4\left(4T\left(\frac{n}{2^2}\right) + \frac{n}{2}\right) + n$$

$$= 4^2 T\left(\frac{n}{2^2}\right) + \frac{4}{2}n + n$$

Iterating Recurrences: Example 5

- $$(*) \qquad T(n) = \begin{cases} 1 & \text{if } n = 1 \\ 4T(n/2) + n & \text{if } n \geq 2 \end{cases}$$

$$T(n) = 4T\left(\tfrac{n}{2}\right) + n \qquad = 4\left(4T\left(\tfrac{n}{2^2}\right) + \tfrac{n}{2}\right) + n$$

$$= 4^2 T\left(\tfrac{n}{2^2}\right) + \tfrac{4}{2}n + n \qquad = 4^2\left(4T\left(\tfrac{n}{2^3}\right) + \tfrac{n}{2^2}\right) + \tfrac{4}{2}n + n$$

Iterating Recurrences: Example 5

- $$(*) \qquad T(n) = \begin{cases} 1 & \text{if } n = 1 \\ 4T(n/2) + n & \text{if } n \geq 2 \end{cases}$$

$$\begin{aligned} T(n) &= 4T\left(\tfrac{n}{2}\right) + n & &= 4\left(4T\left(\tfrac{n}{2^2}\right) + \tfrac{n}{2}\right) + n \\ &= 4^2 T\left(\tfrac{n}{2^2}\right) + \tfrac{4}{2}n + n & &= 4^2\left(4T\left(\tfrac{n}{2^3}\right) + \tfrac{n}{2^2}\right) + \tfrac{4}{2}n + n \\ &= 4^3 T\left(\tfrac{n}{2^3}\right) + \tfrac{4^2}{2^2}n + \tfrac{4}{2}n + n \end{aligned}$$

Iterating Recurrences: Example 5

- $$(*) \qquad T(n) = \begin{cases} 1 & \text{if } n = 1 \\ 4T(n/2) + n & \text{if } n \geq 2 \end{cases}$$

$$\begin{aligned} T(n) &= 4T\left(\tfrac{n}{2}\right) + n & &= 4\left(4T\left(\tfrac{n}{2^2}\right) + \tfrac{n}{2}\right) + n \\ &= 4^2T\left(\tfrac{n}{2^2}\right) + \tfrac{4}{2}n + n & &= 4^2\left(4T\left(\tfrac{n}{2^3}\right) + \tfrac{n}{2^2}\right) + \tfrac{4}{2}n + n \\ &= 4^3T\left(\tfrac{n}{2^3}\right) + \tfrac{4^2}{2^2}n + \tfrac{4}{2}n + n \\ &\vdots \qquad \vdots \\ &= 4^iT\left(\tfrac{n}{2^i}\right) + \tfrac{4^{i-1}}{2^{i-1}}n + \cdots + \tfrac{4^2}{2^2}n + n \end{aligned}$$

Iterating Recurrences: Example 5

- $$(*) \qquad T(n) = \begin{cases} 1 & \text{if } n = 1 \\ 4T(n/2) + n & \text{if } n \geq 2 \end{cases}$$

$$\begin{aligned}
T(n) &= 4T\left(\frac{n}{2}\right) + n & &= 4\left(4T\left(\frac{n}{2^2}\right) + \frac{n}{2}\right) + n \\
&= 4^2T\left(\frac{n}{2^2}\right) + \frac{4}{2}n + n & &= 4^2\left(4T\left(\frac{n}{2^3}\right) + \frac{n}{2^2}\right) + \frac{4}{2}n + n \\
&= 4^3T\left(\frac{n}{2^3}\right) + \frac{4^2}{2^2}n + \frac{4}{2}n + n \\
&\vdots \qquad \vdots \\
&= 4^iT\left(\frac{n}{2^i}\right) + \frac{4^{i-1}}{2^{i-1}}n + \cdots + \frac{4^2}{2^2}n + n \\
&\vdots \qquad \vdots \\
&= 4^{\log_2 n}T\left(\frac{n}{2^{\log_2 n}}\right) + \frac{4^{\log_2 n-1}}{2^{\log_2 n-1}}n + \cdots + \frac{4}{2}n + n
\end{aligned}$$

Iterating Recurrences: Example 5

- $$(*) \qquad T(n) = \begin{cases} 1 & \text{if } n = 1 \\ 4T(n/2) + n & \text{if } n \geq 2 \end{cases}$$

$$\begin{aligned}
T(n) &= 4T\left(\tfrac{n}{2}\right) + n & &= 4\left(4T\left(\tfrac{n}{2^2}\right) + \tfrac{n}{2}\right) + n \\
&= 4^2T\left(\tfrac{n}{2^2}\right) + \tfrac{4}{2}n + n & &= 4^2\left(4T\left(\tfrac{n}{2^3}\right) + \tfrac{n}{2^2}\right) + \tfrac{4}{2}n + n \\
&= 4^3T\left(\tfrac{n}{2^3}\right) + \tfrac{4^2}{2^2}n + \tfrac{4}{2}n + n \\
&\vdots \qquad \vdots \\
&= 4^iT\left(\tfrac{n}{2^i}\right) + \frac{4^{i-1}}{2^{i-1}}n + \cdots + \frac{4^2}{2^2}n + n \\
&\vdots \qquad \vdots \\
&= 4^{\log_2 n}T\left(\frac{n}{2^{\log_2 n}}\right) + \frac{4^{\log_2 n-1}}{2^{\log_2 n-1}}n + \cdots + \frac{4}{2}n + n \\
&= 2n^2 - n
\end{aligned}$$

Growth Rates of Solutions to Recurrences

- Divide and conquer algorithms
- Iteration recurrences
- Three different behaviors

Three Different Behaviors

- Compare the iteration for the recurrences

$$T(n) = 2T(n/2) + n$$

$$T(n) = T(n/2) + n$$

$$T(n) = 4T(n/2) + n$$

Three Different Behaviors

- Compare the iteration for the recurrences

$$T(n) = 2T(n/2) + n$$

$$T(n) = T(n/2) + n$$

$$T(n) = 4T(n/2) + n$$

◇ all three recurrences iterate $\log_2 n$ times

◇ in each case, size of subproblem in next iteration is half the size in the preceding iteration level

Three Different Behaviors

- **Theorem** Suppose that we have a recurrence of the form
$$T(n) = aT(n/2) + n,$$
where a is a positive integer and $T(1)$ is nonnegative. Then we have the following big Θ bounds on the solution:

  1. If $a < 2$, then $T(n) = \Theta(n)$.
  2. If $a = 2$, then $T(n) = \Theta(n \log n)$.
  3. If $a > 2$, then $T(n) = \Theta(n^{\log_2 a})$

Three Different Behaviors

- **Theorem** Suppose that we have a recurrence of the form

$$T(n) = aT(n/2) + n,$$

where a is a positive integer and $T(1)$ is nonnegative. Then we have the following big Θ bounds on the solution:

1. If $a < 2$, then $T(n) = \Theta(n)$.
2. If $a = 2$, then $T(n) = \Theta(n \log n)$.
3. If $a > 2$, then $T(n) = \Theta(n^{\log_2 a})$

Proof

We already proved Case 1 when $a = 1$ in Example 3.

(will not prove it for $1 < a < 2$)

We already proved Case 2 in Example 1.

We will now prove Case 3.

Iterating Recurrences

- $T(n) = aT(n/2) + n$, where $a > 2$. Assume that $n = 2^i$.

Iterating Recurrences

- $T(n) = aT(n/2) + n$, where $a > 2$. Assume that $n = 2^i$.

  Iterating as in Example 5 gives

$$T(n) = a^i T\left(\frac{n}{2^i}\right) + \left(\frac{a^{i-1}}{2^{i-1}} + \frac{a^{i-2}}{2^{i-2}} + \cdots \frac{a}{2} + 1\right) n$$

Iterating Recurrences

- $T(n) = aT(n/2) + n$, where $a > 2$. Assume that $n = 2^i$.

  Iterating as in Example 5 gives

$$T(n) = a^i T\left(\tfrac{n}{2^i}\right) + \left(\tfrac{a^{i-1}}{2^{i-1}} + \tfrac{a^{i-2}}{2^{i-2}} + \cdots \tfrac{a}{2} + 1\right) n$$

$$T(n) = a^{\log_2 n} T(1) + n \textstyle\sum_{i=0}^{\log_2 n-1} \left(\tfrac{a}{2}\right)^i$$

Work at
"bottom"

Iterated
Work

Total work

- The total work is

$$T(n) = a^{\log_2 n} T(1) + n \sum_{i=0}^{\log_2 n - 1} \left(\frac{a}{2}\right)^i$$

Total work

- The total work is

$$T(n) = a^{\log_2 n} T(1) + n \sum_{i=0}^{\log_2 n - 1} \left(\frac{a}{2}\right)^i$$

Since $a > 2$, the geometric series is Θ of the largest term.

$$n \sum_{i=0}^{\log_2 n - 1} \left(\frac{a}{2}\right)^i = n \frac{1-(a/2)^{\log_2 n}}{1-a/2} = n\Theta((a/2)^{\log_2 n - 1})$$

Total work

- n times the largest term in the geometric series is

$$n\left(\frac{a}{2}\right)^{\log_2 n-1} = \frac{2}{a}\cdot\frac{n\cdot a^{\log_2 n}}{2^{\log_2 n}} = \frac{2}{a}\cdot\frac{n\cdot a^{\log_2 n}}{n} = \frac{2}{a}\cdot a^{\log_2 n}$$

Total work

- n times the largest term in the geometric series is

$$n\left(\frac{a}{2}\right)^{\log_2 n-1} = \frac{2}{a}\cdot\frac{n\cdot a^{\log_2 n}}{2^{\log_2 n}} = \frac{2}{a}\cdot\frac{n\cdot a^{\log_2 n}}{n} = \frac{2}{a}\cdot a^{\log_2 n}$$

Notice that

$$a^{\log_2 n} = \left(2^{\log_2 a}\right)^{\log_2 n} = \left(2^{\log_2 n}\right)^{\log_2 a} = n^{\log_2 a}$$

Total work

- n times the largest term in the geometric series is

$$n\left(\frac{a}{2}\right)^{\log_2 n-1} = \frac{2}{a}\cdot\frac{n\cdot a^{\log_2 n}}{2^{\log_2 n}} = \frac{2}{a}\cdot\frac{n\cdot a^{\log_2 n}}{n} = \frac{2}{a}\cdot a^{\log_2 n}$$

Notice that

$$a^{\log_2 n} = \left(2^{\log_2 a}\right)^{\log_2 n} = \left(2^{\log_2 n}\right)^{\log_2 a} = n^{\log_2 a}$$

So the total work is

$$a^{\log_2 n}T(1) + n\sum_{i=0}^{\log_2 n-1}\left(\frac{a}{2}\right)^i$$

Total work

- n times the largest term in the geometric series is

$$n\left(\frac{a}{2}\right)^{\log_2 n-1} = \frac{2}{a}\cdot\frac{n\cdot a^{\log_2 n}}{2^{\log_2 n}} = \frac{2}{a}\cdot\frac{n\cdot a^{\log_2 n}}{n} = \frac{2}{a}\cdot a^{\log_2 n}$$

Notice that

$$a^{\log_2 n} = \left(2^{\log_2 a}\right)^{\log_2 n} = \left(2^{\log_2 n}\right)^{\log_2 a} = n^{\log_2 a}$$

So the total work is

$$a^{\log_2 n}T(1) + n\sum_{i=0}^{\log_2 n-1}\left(\frac{a}{2}\right)^i$$

$$\Theta\left(n^{\log_2 a}\right) \qquad \Theta\left(n^{\log_2 a}\right)$$

Example 5 Recap

- $$(*) \qquad T(n) = \begin{cases} 1 & \text{if } n = 1 \\ 4T(n/2) + n & \text{if } n \geq 2 \end{cases}$$

Example 5 Recap

- $$(*) \qquad T(n) = \begin{cases} 1 & \text{if } n = 1 \\ 4T(n/2) + n & \text{if } n \geq 2 \end{cases}$$

$a = 4$, so the Theorem says that

$$T(n) = \Theta\left(n^{\log_2 a}\right) = \Theta\left(n^{\log_2 4}\right) = \Theta(n^2)$$

Example 5 Recap

- $$(*) \qquad T(n) = \begin{cases} 1 & \text{if } n = 1 \\ 4T(n/2) + n & \text{if } n \geq 2 \end{cases}$$

$a = 4$, so the Theorem says that

$$T(n) = \Theta\left(n^{\log_2 a}\right) = \Theta\left(n^{\log_2 4}\right) = \Theta(n^2)$$

This matches with the exact answer of $2n^2 - n$.

Three Different Behaviors

- **Theorem** Suppose that we have a recurrence of the form
$$T(n) = aT(n/2) + n,$$
where a is a positive integer and $T(1)$ is nonnegative. Then we have the following big Θ bounds on the solution:

  1. If $a < 2$, then $T(n) = \Theta(n)$.
  2. If $a = 2$, then $T(n) = \Theta(n \log n)$.
  3. If $a > 2$, then $T(n) = \Theta(n^{\log_2 a})$

The Master Theorem

- **Theorem** Suppose that we have a recurrence of the form
$$T(n) = aT(n/b) + cn^d,$$
where a is a positive integer, $b \geq 1$, c, d are real numbers with c positive and d nonnegative, and $T(1)$ is nonnegative. Then we have the following big Θ bounds on the solution:

  1. If $a < b^d$, then $T(n) = \Theta(n^d)$.
  2. If $a = b^d$, then $T(n) = \Theta(n^d \log n)$.
  3. If $a > b^d$, then $T(n) = \Theta(n^{\log_b a})$

Counting

- Assume we have a set of objects with certain properties

Counting

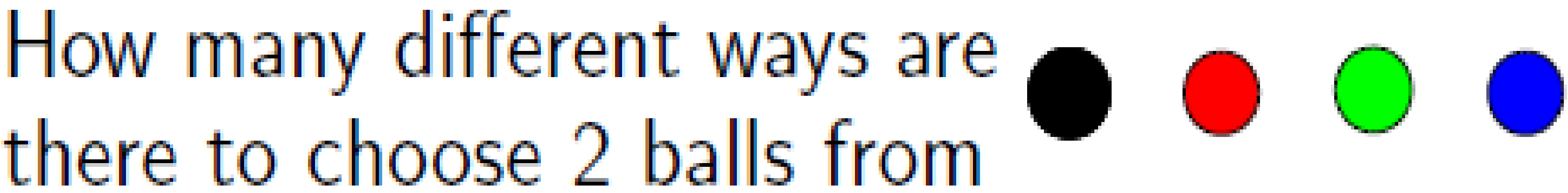
- Assume we have a set of objects with certain properties

  Counting is used to determine the number of these objects.

Counting

- Assume we have a set of objects with certain properties

  Counting is used to determine the number of these objects.

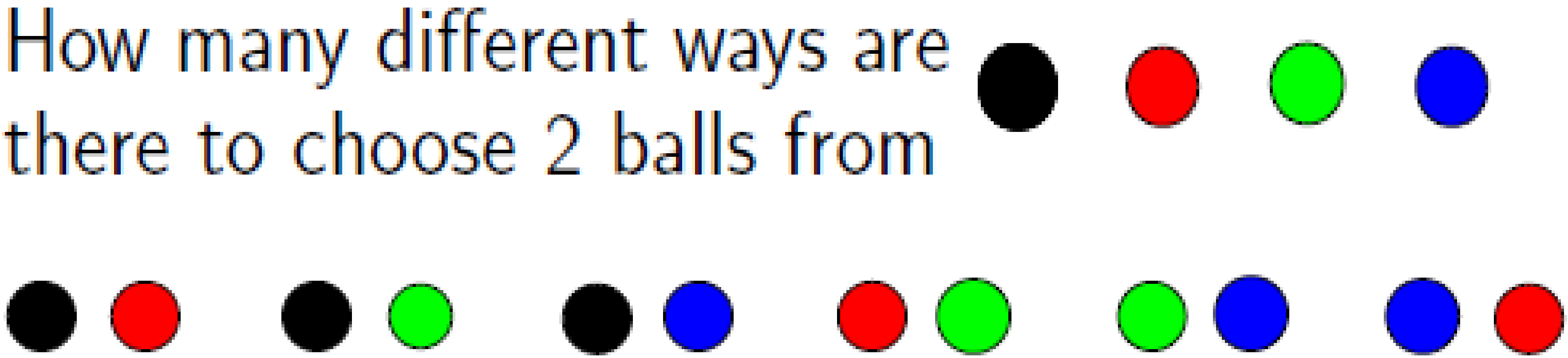


How many different ways are there to choose 2 balls from

Counting

- Assume we have a set of objects with certain properties

  Counting is used to determine the number of these objects.

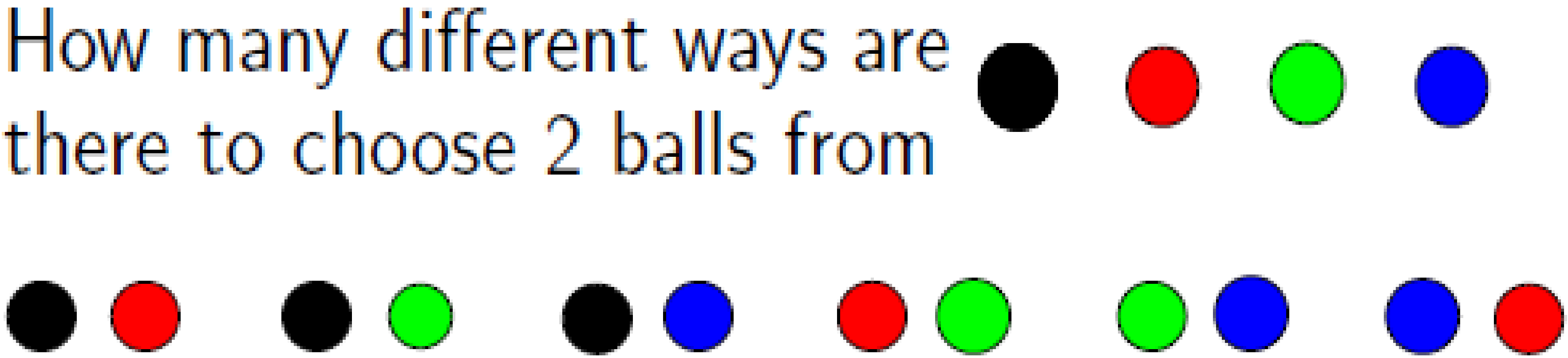


How many different ways are there to choose 2 balls from

Counting

- Assume we have a set of objects with certain properties

  Counting is used to determine the number of these objects.



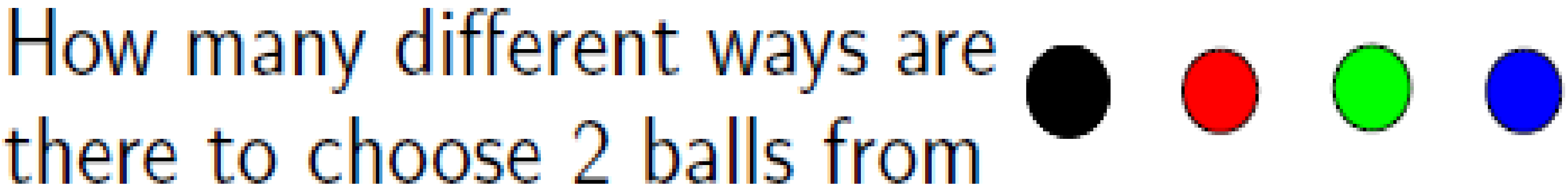
How many different ways are there to choose 2 balls from

What about when order counts?

Counting

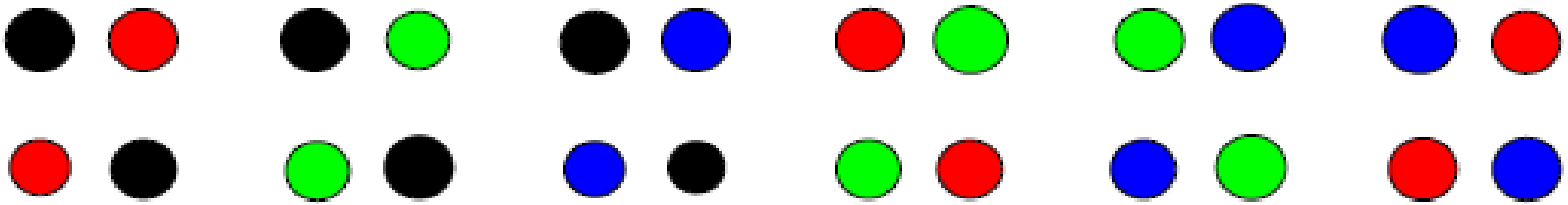
- Assume we have a set of objects with certain properties

  Counting is used to determine the number of these objects.



How many different ways are there to choose 2 balls from

What about when order counts?



Counting

- Assume we have a set of objects with certain properties

  Counting is used to determine the number of these objects.

Counting

- Assume we have a set of objects with certain properties

  Counting is used to determine the number of these objects.

  Examples

  - ◇ the number of steps in a computer program
  - ◇ the number of passwords between $6-10$ characters
  - ◇ the number of telephone numbers with 8 digits

Counting

- Assume we have a set of objects with certain properties

  Counting is used to determine the number of these objects.

  Examples

  ◇ the number of steps in a computer program

  ◇ the number of passwords between $6 - 10$ characters

  ◇ the number of telephone numbers with 8 digits

  Counting may be very hard, not trivial.

Counting

- Assume we have a set of objects with certain properties

  Counting is used to determine the number of these objects.

  Examples

  ◇ the number of steps in a computer program

  ◇ the number of passwords between $6-10$ characters

  ◇ the number of telephone numbers with 8 digits

  Counting may be very hard, not trivial.

  – simplify the solution by decomposing the problem

Basic Counting Rules

- *the Product Rule*

- *the Sum Rule*

Basic Counting Rules

- *the Product Rule*

  ◇ A count decomposes into a sequence of dependent counts (each element in the first count is associated with all elements of the second count)

- *the Sum Rule*

  ◇ A count decomposes into a set of independent counts (elements of counts are alternatives)

The Product Rule

- A count decomposes into a sequence of dependent counts (each element in the first count is associated with all elements of the second count)

The Product Rule

- A count decomposes into a sequence of dependent counts (each element in the first count is associated with all elements of the second count)

Example

In an auditorium, the seats are labeled by a letter and numbers in between 1 to 50 (e.g., $A23$). What is the total number of seats?

The Product Rule

- A count decomposes into a sequence of dependent counts (each element in the first count is associated with all elements of the second count)

Example

In an auditorium, the seats are labeled by a letter and numbers in between 1 to 50 (e.g., $A23$). What is the total number of seats?

We may either list all or use the product rule.

$$26 \times 50 = 1300$$

The Product Rule

- **Product Rule**: If a count of elements can be broken down into a sequence of dependent counts where the first count yields n_1 elements, the second n_2 elements, and kth count n_k elements, then the total number of elements is

$$n = n_1 \cdot n_2 \cdot \dots \cdot n_k$$

The Product Rule

- **Product Rule**: If a count of elements can be broken down into a sequence of dependent counts where the first count yields n_1 elements, the second n_2 elements, and kth count n_k elements, then the total number of elements is

$$n = n_1 \cdot n_2 \cdot \cdots \cdot n_k$$

Example

How many different bit strings of length 7 are there?

The Product Rule

- **Product Rule**: If a count of elements can be broken down into a sequence of dependent counts where the first count yields n_1 elements, the second n_2 elements, and kth count n_k elements, then the total number of elements is

$$n = n_1 \cdot n_2 \cdot \dots \cdot n_k$$

Example

How many different bit strings of length 7 are there?

How many different functions are there from a set with m elements to a set with n elements?

The Product Rule

- **Product Rule**: If a count of elements can be broken down into a sequence of dependent counts where the first count yields n_1 elements, the second n_2 elements, and kth count n_k elements, then the total number of elements is
$$n = n_1 \cdot n_2 \cdot \cdots \cdot n_k$$

Example

How many different bit strings of length 7 are there?

How many different functions are there from a set with m elements to a set with n elements?

How many one-to-one functions are there from a set with m elements to a set with n elements?

The Product Rule

- **Product Rule**: If a count of elements can be broken down into a sequence of dependent counts where the first count yields n_1 elements, the second n_2 elements, and kth count n_k elements, then the total number of elements is
$$n = n_1 \cdot n_2 \cdot \cdots \cdot n_k$$

Example

How many different bit strings of length 7 are there?

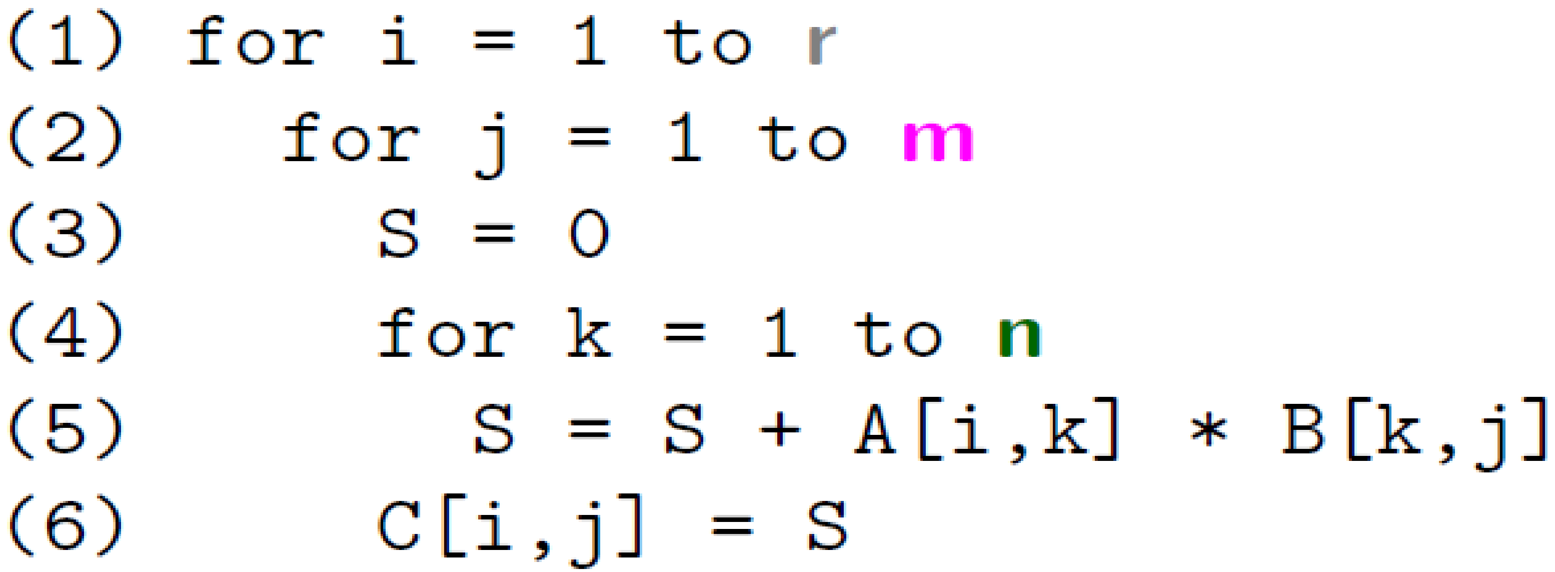
How many different functions are there from a set with m elements to a set with n elements?

How many one-to-one functions are there from a set with m elements to a set with n elements?

How many onto functions?

The Product Rule

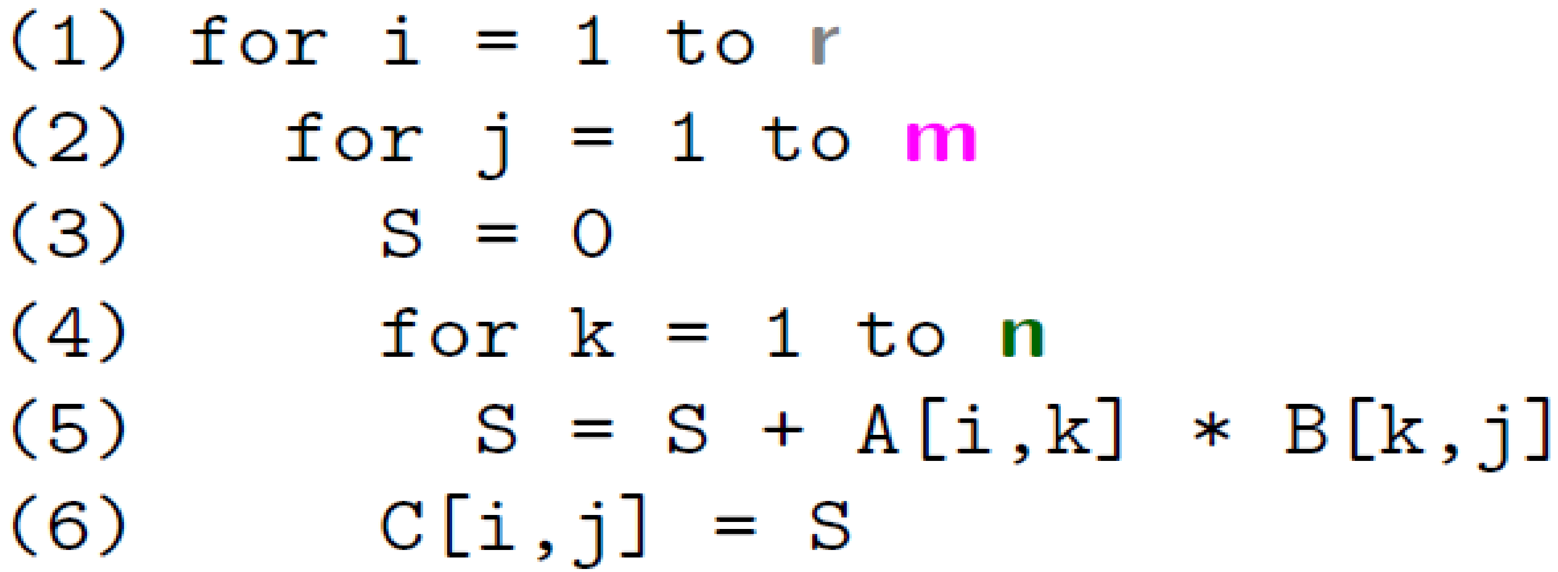
- The following loop is a part of program computing the product of two matrices.



```
for i = 1 to r
    for j = 1 to m
        S = 0
        for k = 1 to n
            S = S + A[i,k] * B[k,j]
        C[i,j] = S
```

The Product Rule

- The following loop is a part of program computing the product of two matrices.



```
(1) for i = 1 to r
(2)   for j = 1 to m
(3)     S = 0
(4)     for k = 1 to n
(5)       S = S + A[i,k] * B[k,j]
(6)     C[i,j] = S
```

How many multiplications (in terms of r, m, n) does this program carry out in total among all iterations of line 5?

The Sum Rule

- A count decomposes into a set of independent counts (elements of counts are alternatives)

The Sum Rule

- A count decomposes into a set of independent counts (elements of counts are alternatives)

Example

You need to travel from city A to B. You may either fly, take a train, or a bus. There are 12 different flights, 5 different trains and 10 buses. How many options do you have to get from A to B?

The Sum Rule

- A count decomposes into a set of independent counts (elements of counts are alternatives)

Example

You need to travel from city A to B. You may either fly, take a train, or a bus. There are 12 different flights, 5 different trains and 10 buses. How many options do you have to get from A to B?

We may use the sum rule.

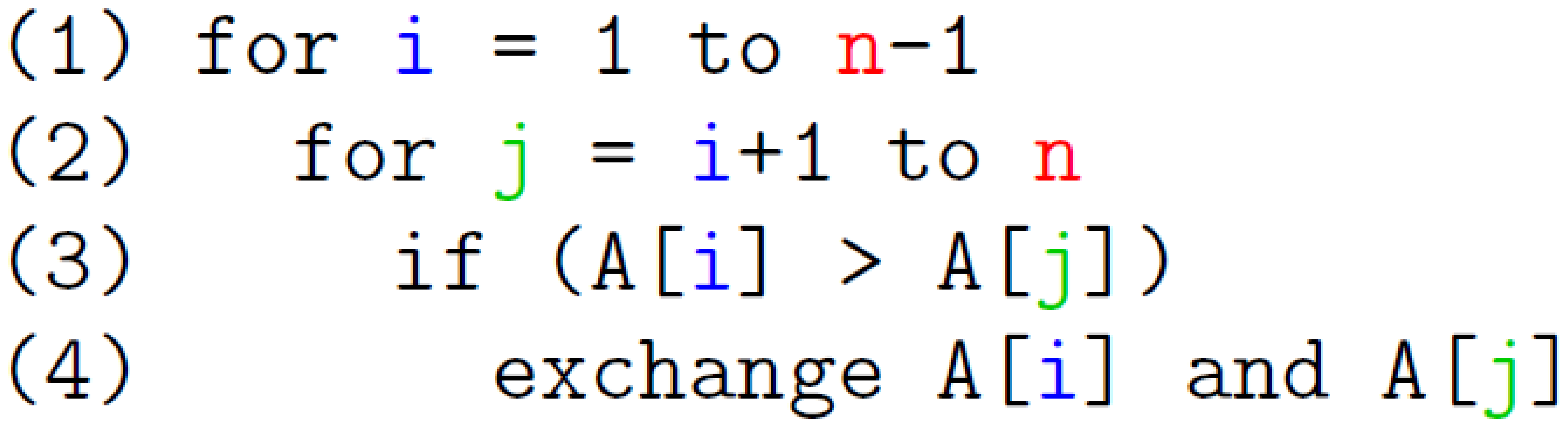
$$12 + 5 + 10$$

The Sum Rule

- **Sum Rule**: If a count of elements can be broken down into a set of independent counts where the first count yields n_1 elements, the second n_2 elements, and kth count n_k elements, then the total number of elements is
$$n = n_1 + n_2 + \cdots + n_k$$

The Sum Rule

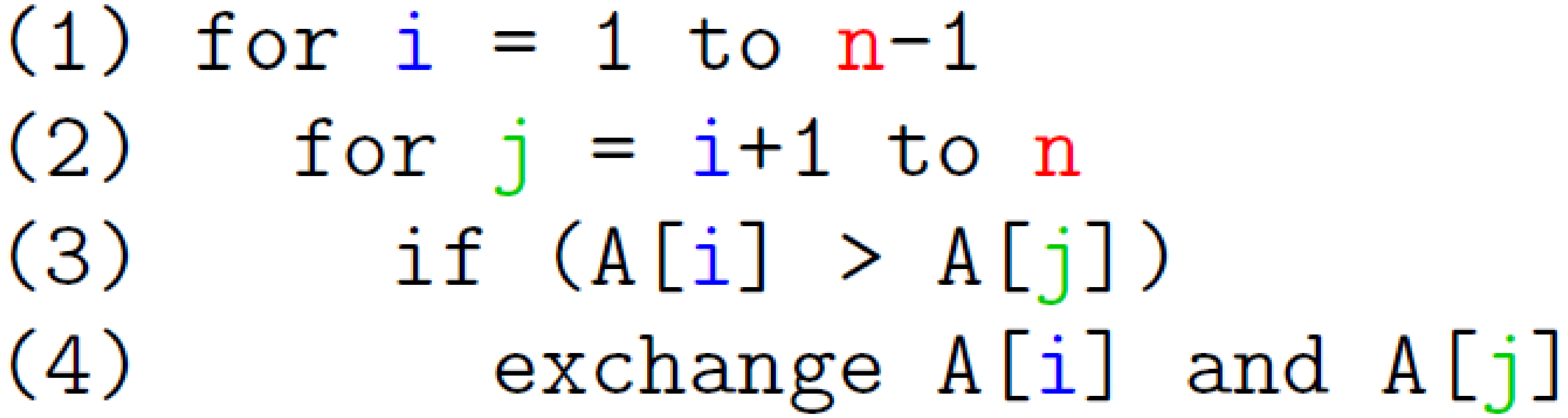
- The following loop is from selection sort.



```
for i = 1 to n-1
    for j = i+1 to n
        if (A[i] > A[j])
            exchange A[i] and A[j]
```

The Sum Rule

- The following loop is from selection sort.



```
(1) for i = 1 to n-1
(2)    for j = i+1 to n
(3)       if (A[i] > A[j])
(4)          exchange A[i] and A[j]
```

How many comparisons (in terms of *n*) does this program carry out in total among all iterations of line 3?

More Complex Counting

- Typically requies a combination of the sum and product rules.

More Complex Counting

- Typically requies a combination of the sum and product rules.

Example

Each password is 6 to 8 characters long, where each character is an lowercase letter or a digit. Each password must contain at least one digit. How many possible passwords are there?

More Complex Counting

- Typically requies a combination of the sum and product rules.

Example

Each password is 6 to 8 characters long, where each character is an lowercase letter or a digit. Each password must contain at least one digit. How many possible passwords are there?

$$P = P_6 + P_7 + P_8$$

Tree Diagrams

- A *tree* is a structure that consists of a root, branches and leaves.

Tree Diagrams

- A *tree* is a structure that consists of a root, branches and leaves.

  Can be useful to represent a counting problem and record the choices we made for alternatives. The count appears on the leaves.

Tree Diagrams

- A *tree* is a structure that consists of a root, branches and leaves.

  Can be useful to represent a counting problem and record the choices we made for alternatives. The count appears on the leaves.

  Example

  What is the number of bit strings of length 4 that do not have two consecutive 1's?

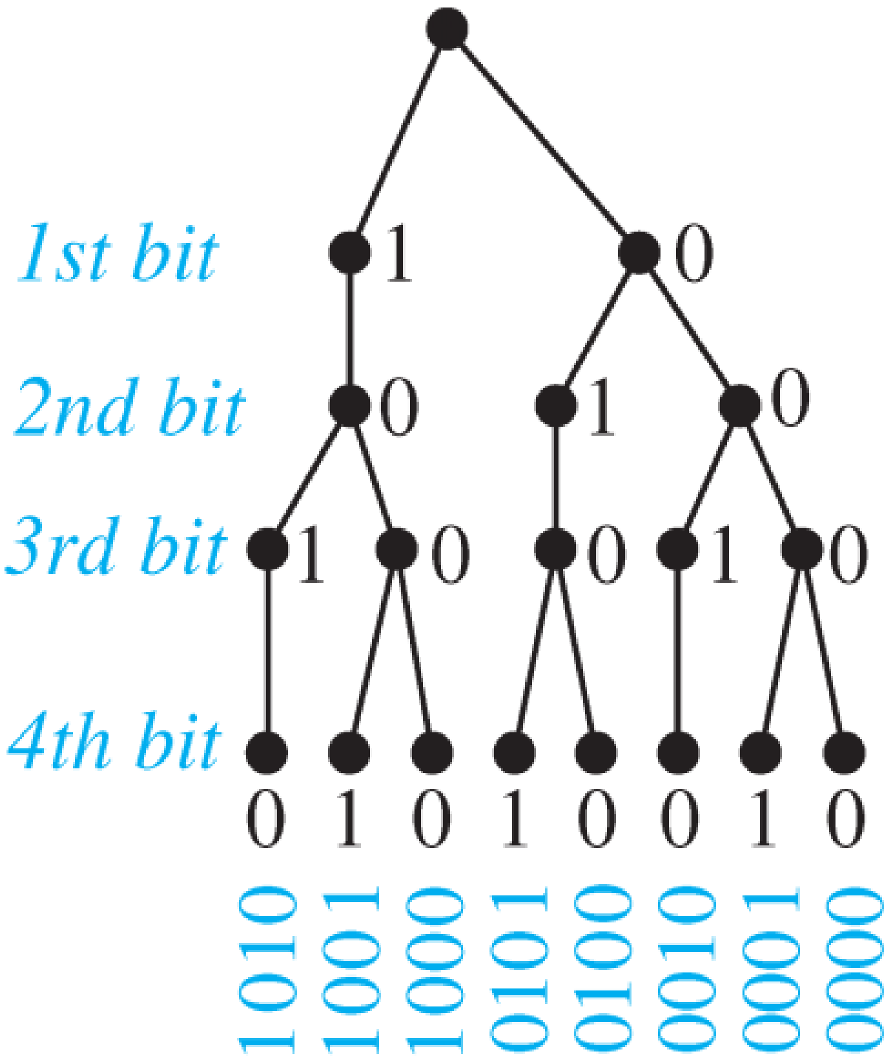
Tree Diagrams

- A *tree* is a structure that consists of a root, branches and leaves.

  Can be usefu ɔlem and record the choices w count appears on the leaves.

  Example

  What is the ı h 4 that do not have two con

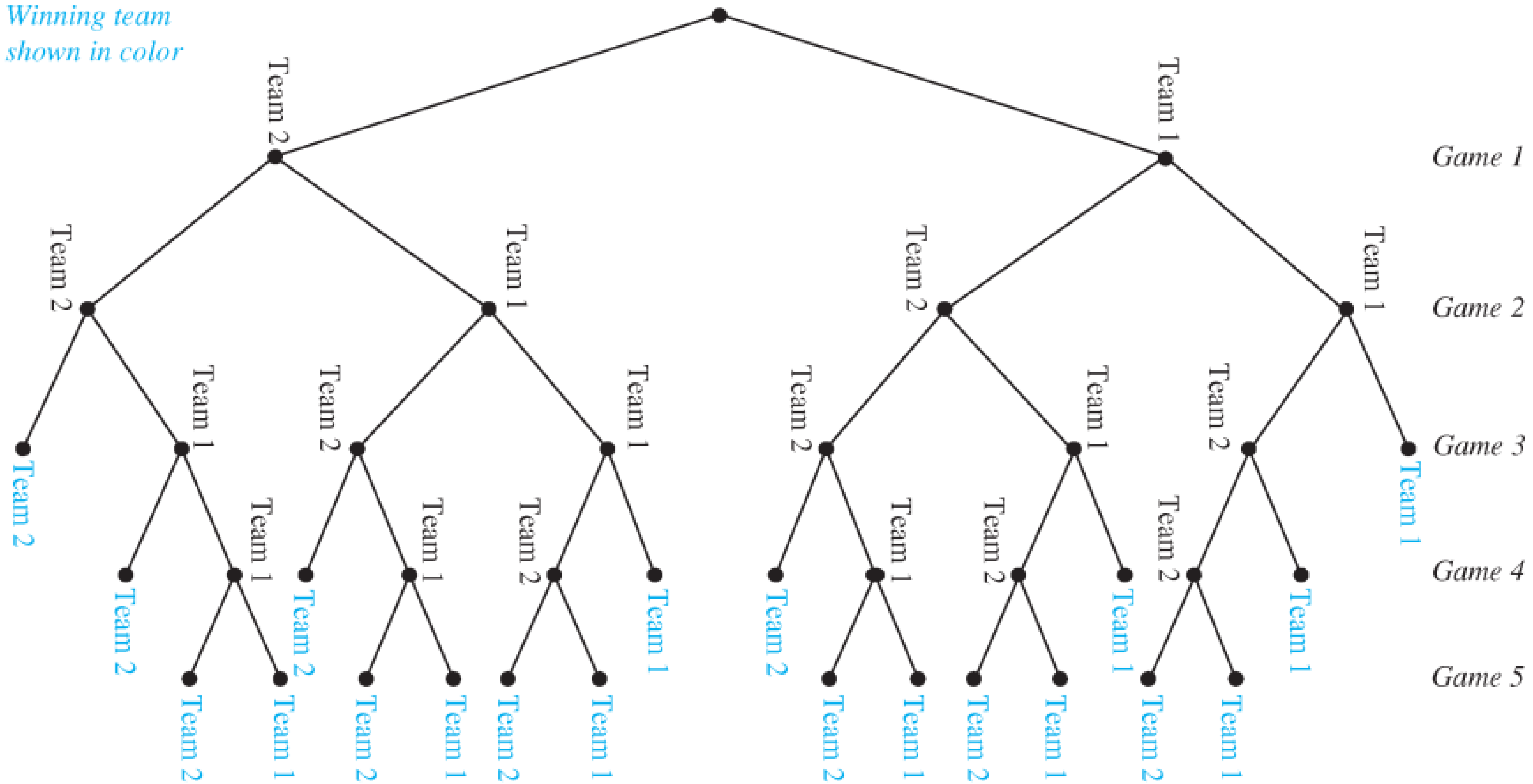


Tree Diagram

- How many different ways can a "best 3 of 5" playoff occur?

Tree Diagram

- How many different ways can a "best 3 of 5" playoff occur?



Pigeonhole Principle

- Assume that there are a set of objects and a set of bins to store them.

Pigeonhole Principle

- Assume that there are a set of objects and a set of bins to store them.

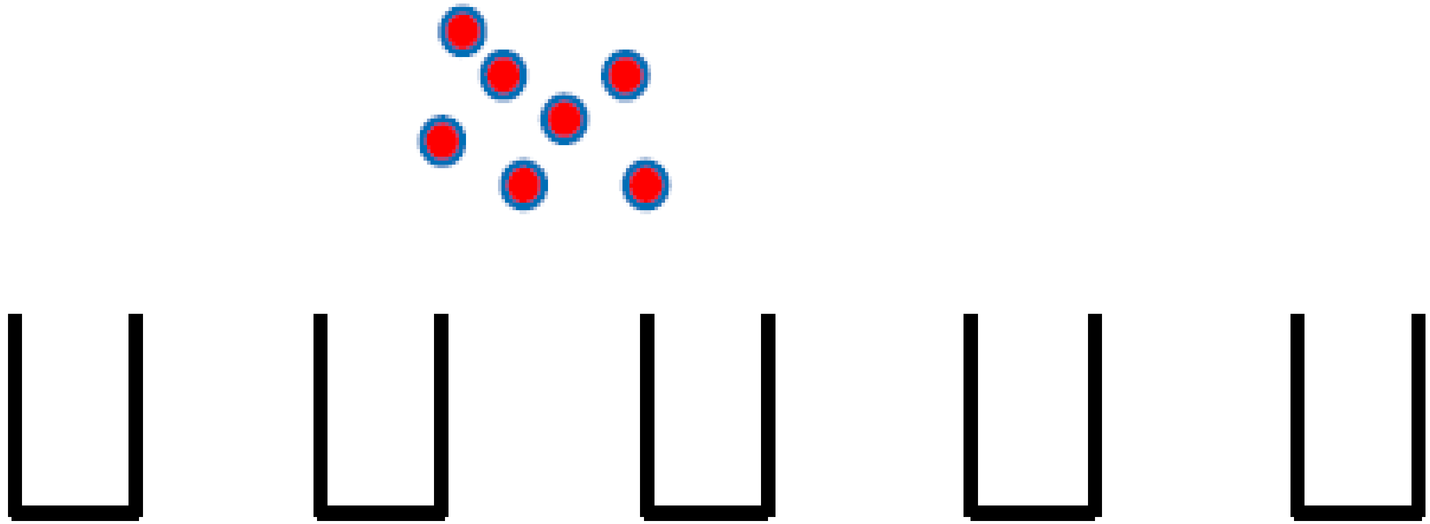
  The pigeonhole principle states that if there are more objects than bins then there is at least one bin with more than one object.

Pigeonhole Principle

- Assume that there are a set of objects and a set of bins to store them.

  The pigeonhole principle states that if there are more objects than bins then there is at least one bin with more than one object.

Example: 7 balls and 5 bins to store them

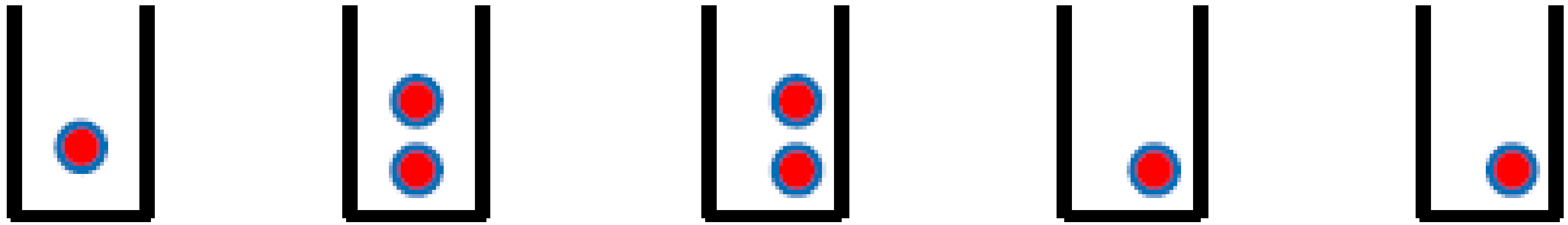


Pigeonhole Principle

- Assume that there are a set of objects and a set of bins to store them.

  The pigeonhole principle states that if there are more objects than bins then there is at least one bin with more than one object.

  Example: 7 balls and 5 bins to store them



Pigeonhole Principle

- **Theorem** If there are $k+1$ objects and k bins, then there is at least one bin with two or more objects.

Pigeonhole Principle

- **Theorem** If there are $k+1$ objects and k bins, then there is at least one bin with two or more objects.

  Proof by contradiction

Pigeonhole Principle

- **Theorem** If there are $k+1$ objects and k bins, then there is at least one bin with two or more objects.

  Proof by contradiction

  Example

  Assume that there are 367 students. Are there any two people who have the same birthday?

  There are 5 bins and 12 objects. Then there must be a bin with at least 3 objects. Why?

Generalized Pigeonhole Principle

- If N objects are placed into k bins, then there is at least one bin containing at least $\lceil N/k \rceil$ objects.

Generalized Pigeonhole Principle

- If N objects are placed into k bins, then there is at least one bin containing at least $\lceil N/k \rceil$ objects.

Example

Assume there are 100 students. How many of them were born in the same month?

Bijections and Permutations

- A function that is both one-to-one and onto is called a *bijection*, or a *one-to-one correspondence*.

Bijections and Permutations

- A function that is both one-to-one and onto is called a *bijection*, or a *one-to-one correspondence*.

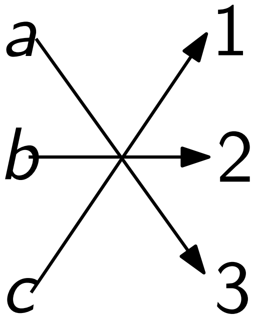
  How many bijections are there?

Bijections and Permutations

- A function that is both one-to-one and onto is called a *bijection*, or a *one-to-one correspondence*.

  How many bijections are there?

$\boxed{\text{$f : \{a, b, c\} \to \{1, 2, 3\}$ defined by $f(a) = 3, f(b) = 2, f(c) = 1$ is a bijection.}}$

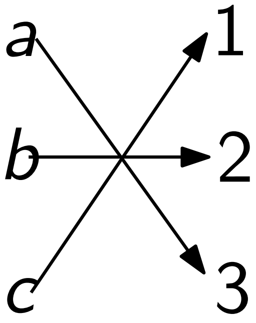


Bijections and Permutations

- A function that is both one-to-one and onto is called a *bijection*, or a *one-to-one correspondence*.

  How many bijections are there?

  $f : \{a, b, c\} \to \{1, 2, 3\}$ defined by $f(a) = 3, f(b) = 2, f(c) = 1$ is a bijection.

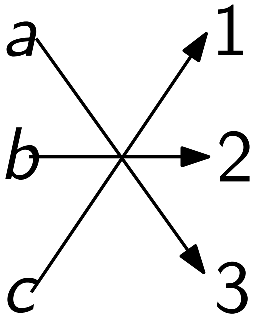
  A bijection from a set onto itself is called a *permutation*.

Bijections and Permutations

- A function that is both one-to-one and onto is called a *bijection*, or a *one-to-one correspondence*.

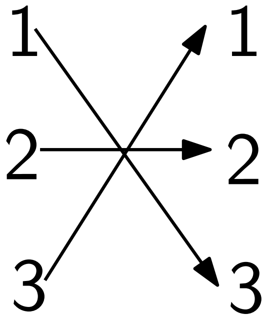
  How many bijections are there?

  $f : \{a, b, c\} \to \{1, 2, 3\}$ defined by $f(a) = 3, f(b) = 2, f(c) = 1$ is a bijection.



  A bijection from a set onto itself is called a *permutation*.

  $f : \{1, 2, 3\} \to \{1, 2, 3\}$ defined by $f(1) = 3, f(2) = 2, f(3) = 1$ is a bijection.

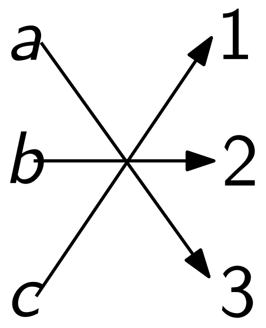


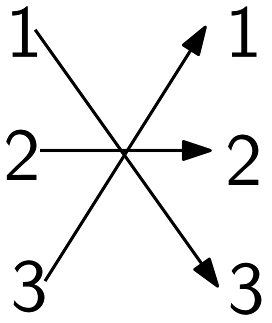
Bijections and Permutations

- A function that is both one-to-one and onto is called a *bijection*, or a *one-to-one correspondence*.

  A bijection from a set onto itself is called a *permutation*.

  In a bijection,
  exactly one arrow leaves each item on the left and exactly one arrow arrives at each item on the right.





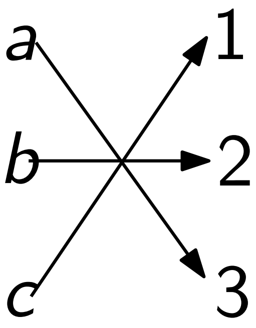
Bijections and Permutations

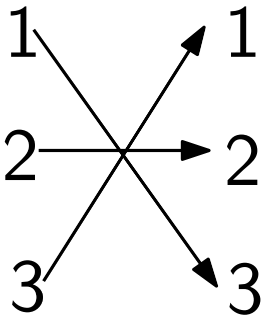
- A function that is both one-to-one and onto is called a *bijection*, or a *one-to-one correspondence*.

  A bijection from a set onto itself is called a *permutation*.

  In a bijection,
  exactly one arrow leaves each item on the left and exactly one arrow arrives at each item on the right.

  Thus,
  the left and right sides must have the same size.





The Bijection Principle

- The following loop is a part of program to determine the number of triangles formed by n points in the plane.

```
trianglecount = 0
  for i = 1 to n
    for j = i+1 to n
      for k = j+1 to n
        if points i, j, k are not collinear
          trianglecount = trianglecount + 1
```

The Bijection Principle

- The following loop is a part of program to determine the number of triangles formed by n points in the plane.

```
(1) trianglecount = 0
(2)   for i = 1 to n
(3)     for j = i+1 to n
(4)       for k = j+1 to n
(5)         if points i, j, k are not collinear
(6)           trianglecount = trianglecount + 1
```

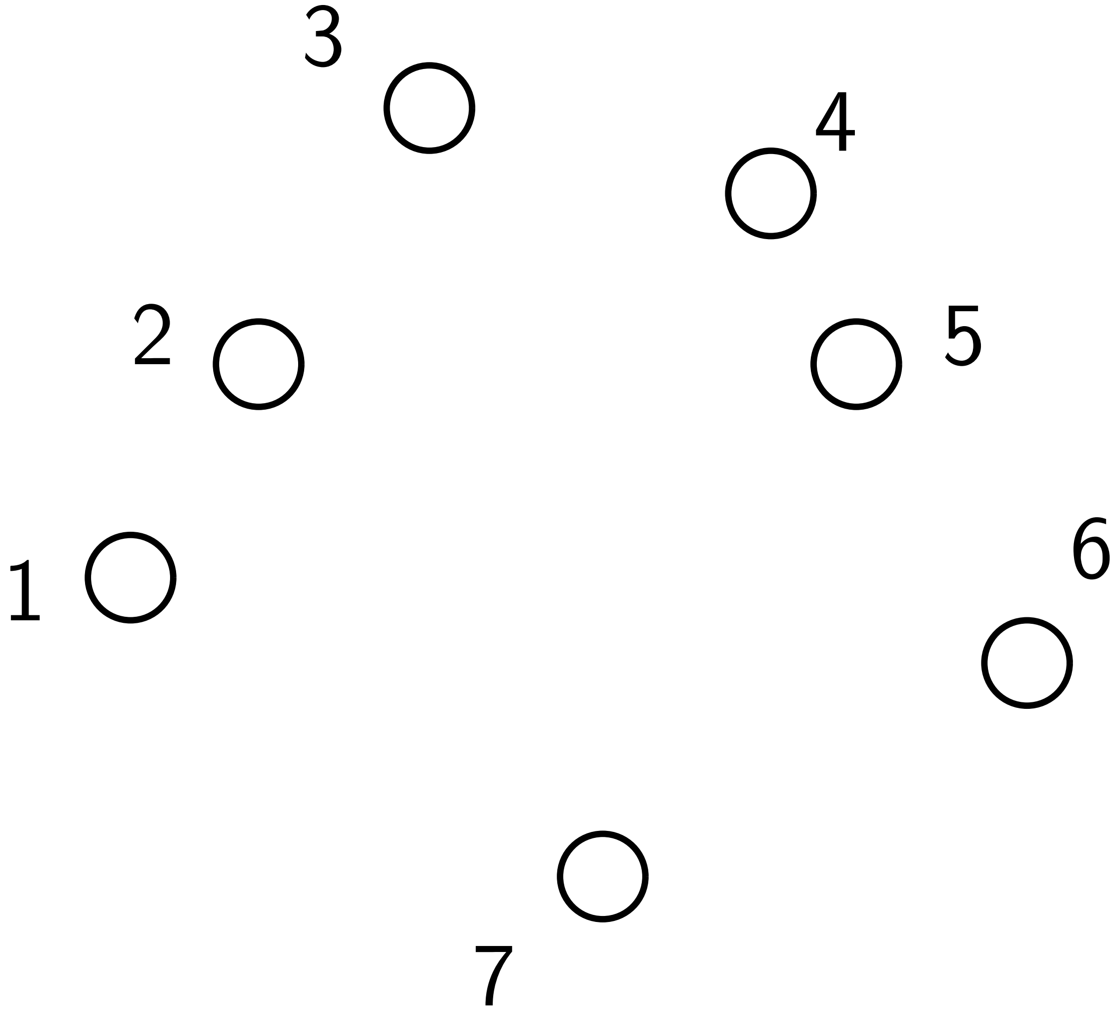
Among all iterations of line 5, what is the total number of times this line checks three points to see if they are collinear?

Counting Triangles

- 3 points form a triangle if and only if they are non collinear

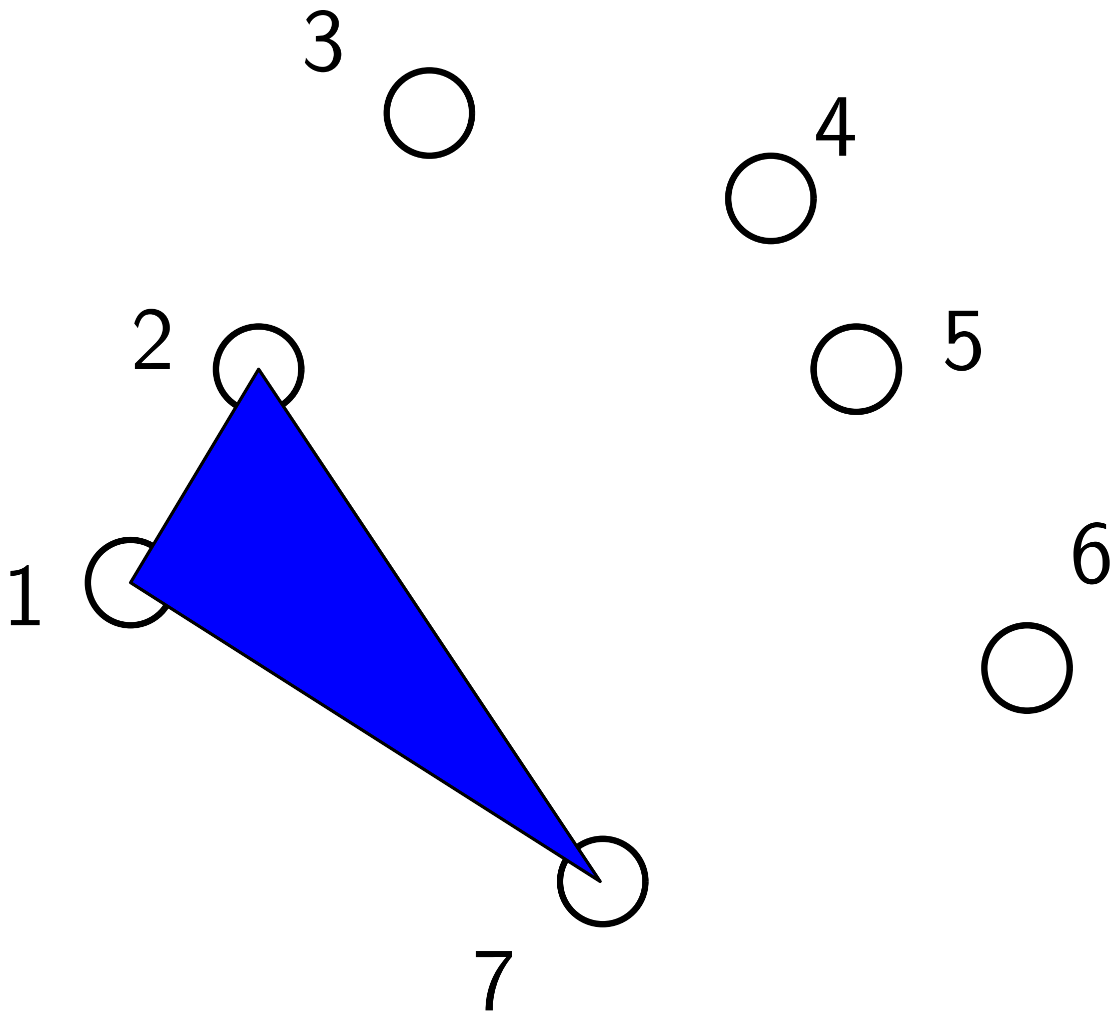
Counting Triangles

- 3 points form a triangle if and only if they are non collinear



Counting Triangles

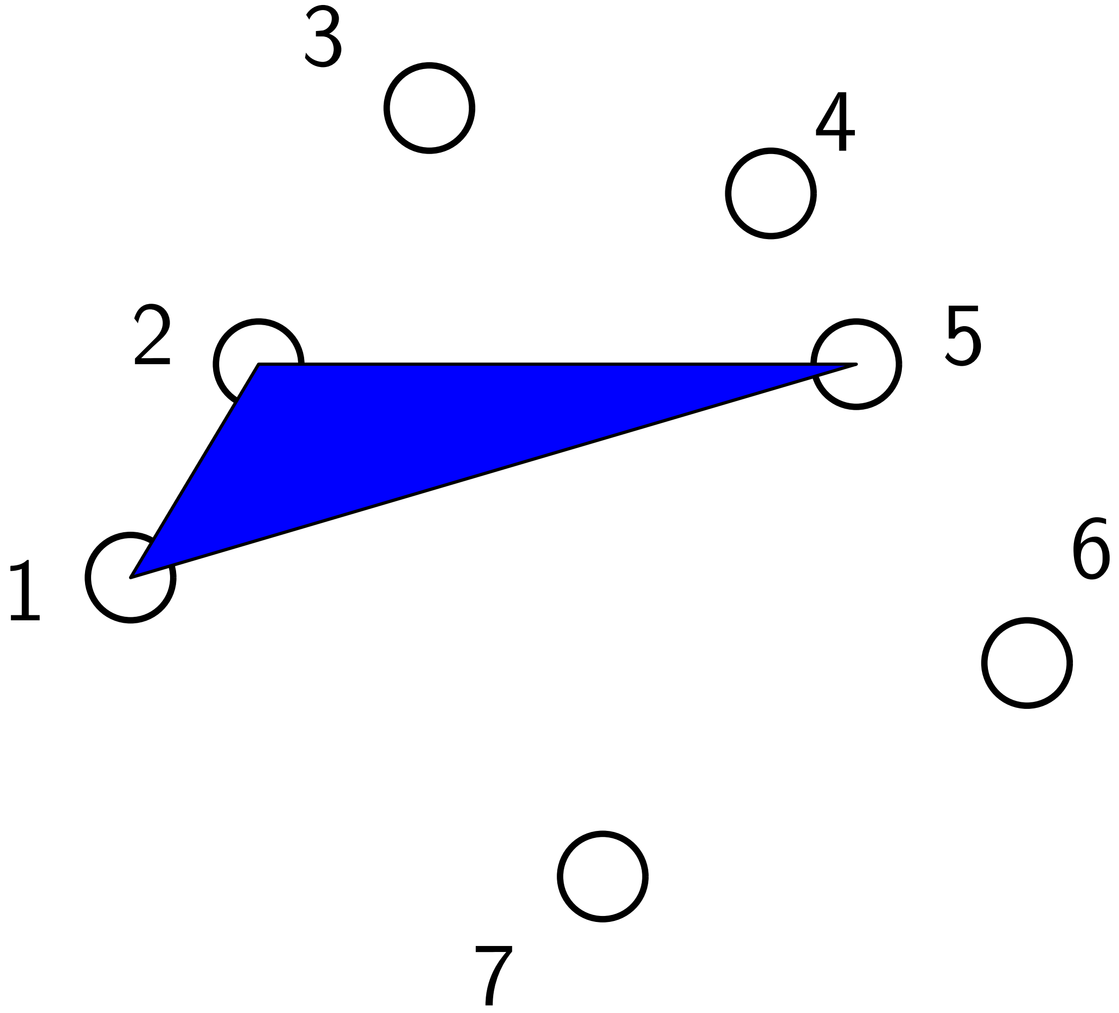
- 3 points form a triangle if and only if they are non collinear



$1 - 2 - 7$: yes

Counting Triangles

- 3 points form a triangle if and only if they are non collinear

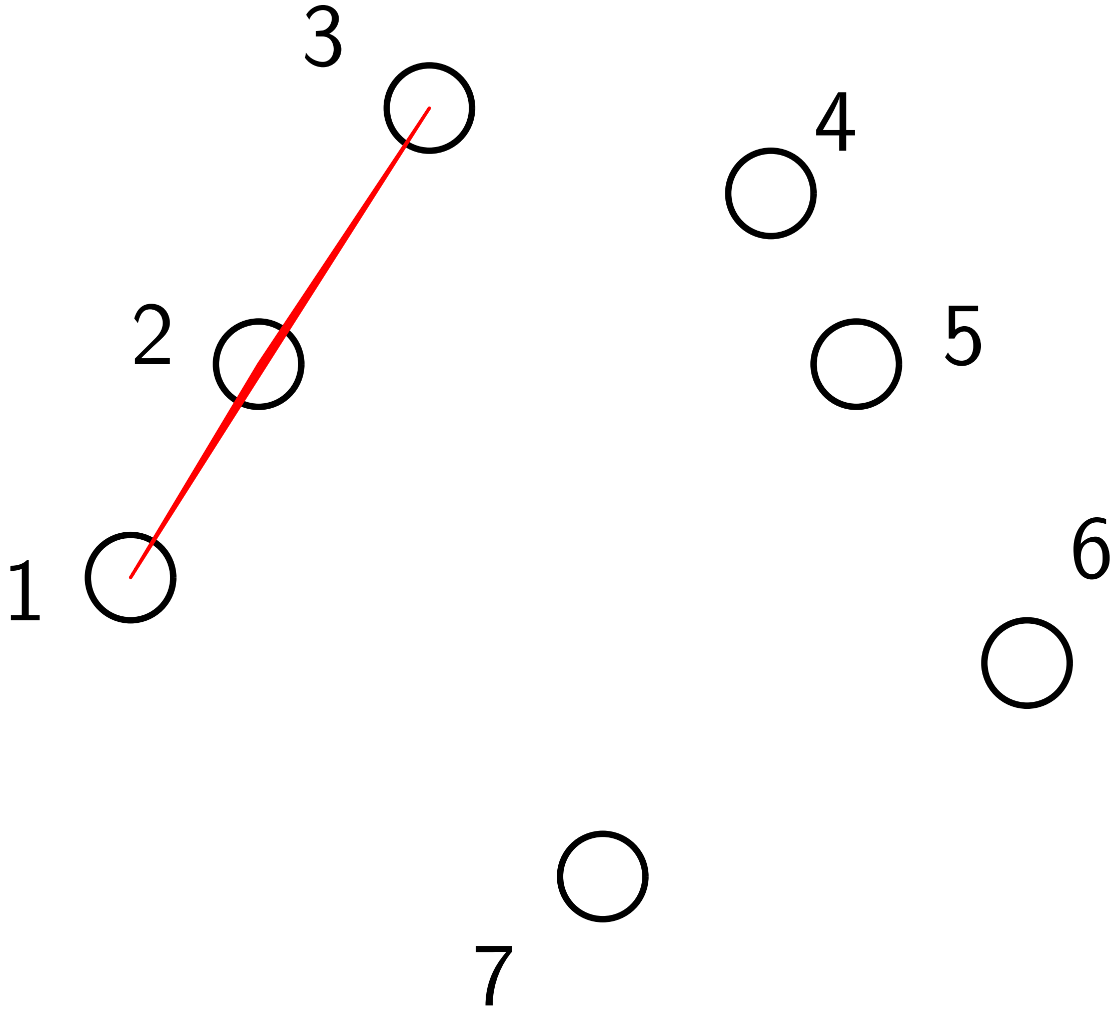


1 – 2 – 7: yes

1 – 2 – 5: yes

Counting Triangles

- 3 points form a triangle if and only if they are non collinear



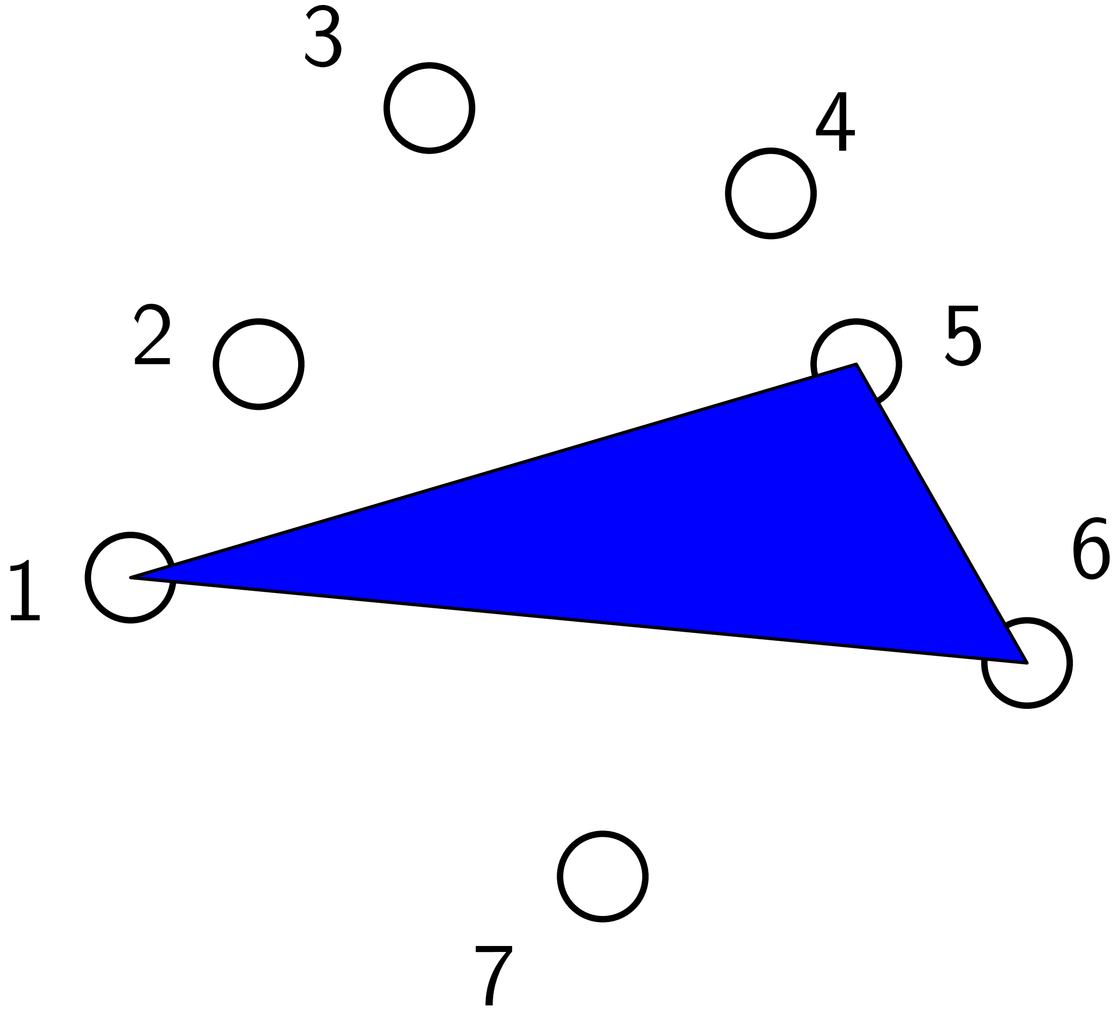
1 – 2 – 7: yes

1 – 2 – 5: yes

1 – 2 – 3: no

Counting Triangles

- 3 points form a triangle if and only if they are non collinear



$1 - 2 - 7$: yes

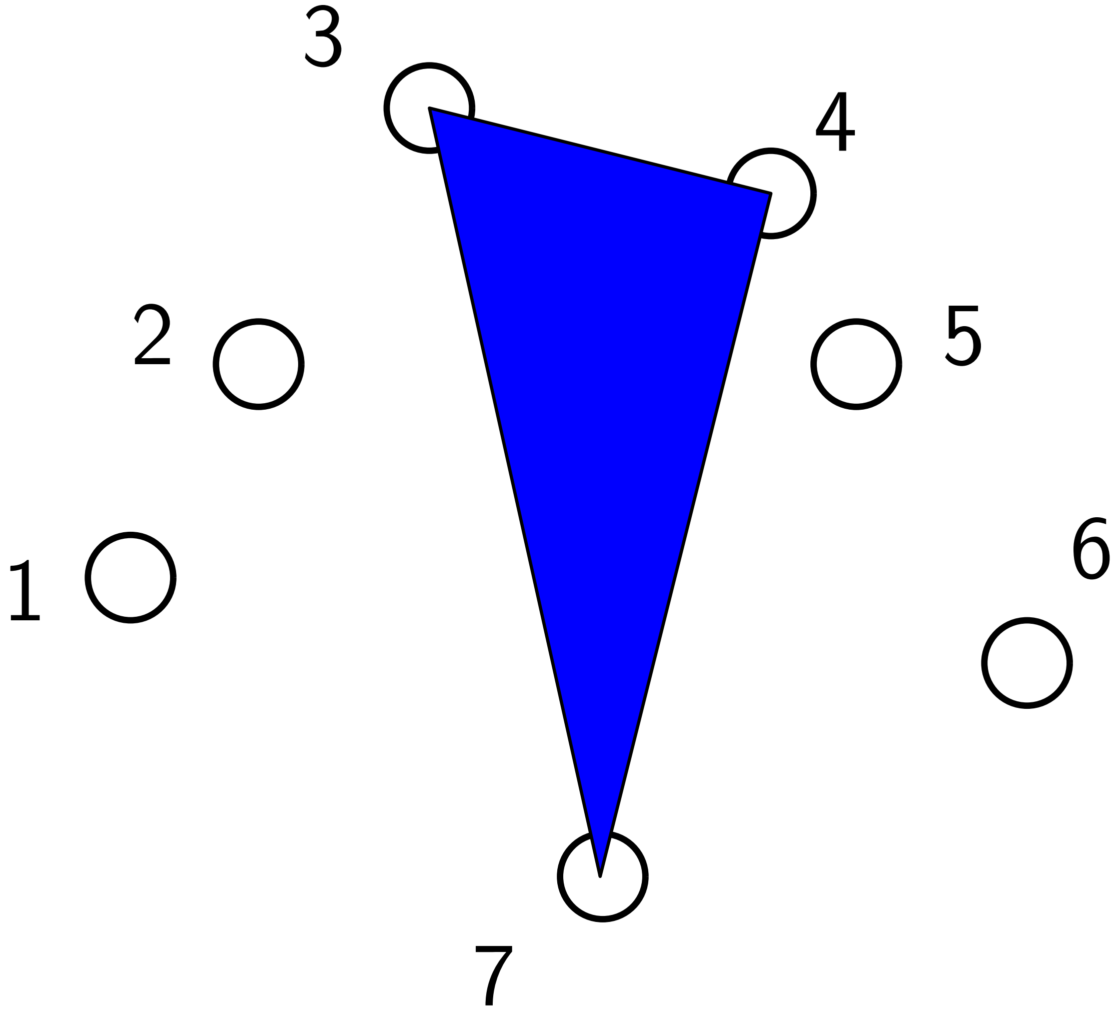
$1 - 2 - 5$: yes

$1 - 2 - 3$: no

$1 - 5 - 6$: yes

Counting Triangles

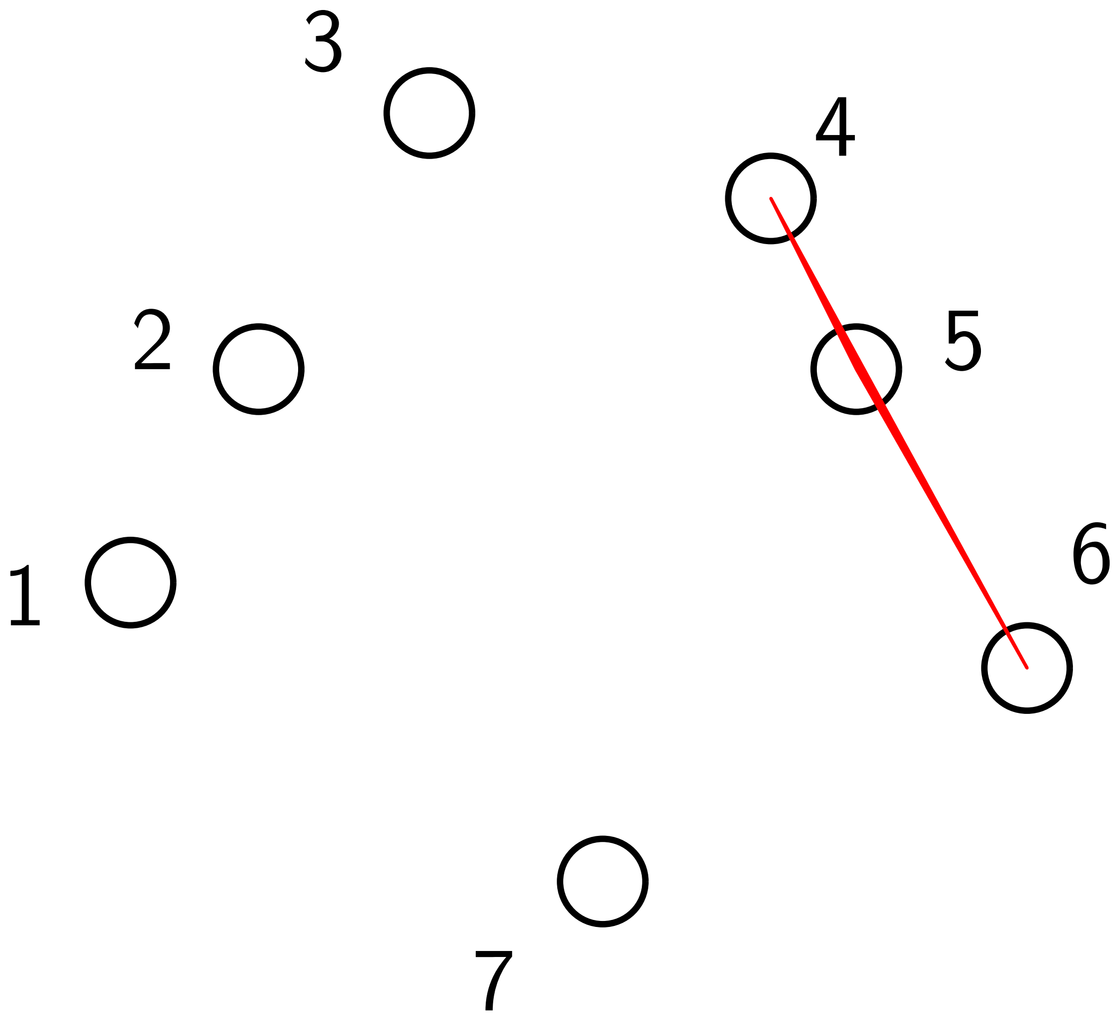
- 3 points form a triangle if and only if they are non collinear



1 — 2 — 7:	yes
1 — 2 — 5:	yes
1 — 2 — 3:	no
1 — 5 — 6:	yes
3 — 4 — 7:	yes

Counting Triangles

- 3 points form a triangle if and only if they are non collinear



1 – 2 – 7: yes

1 – 2 – 5: yes

1 – 2 – 3: no

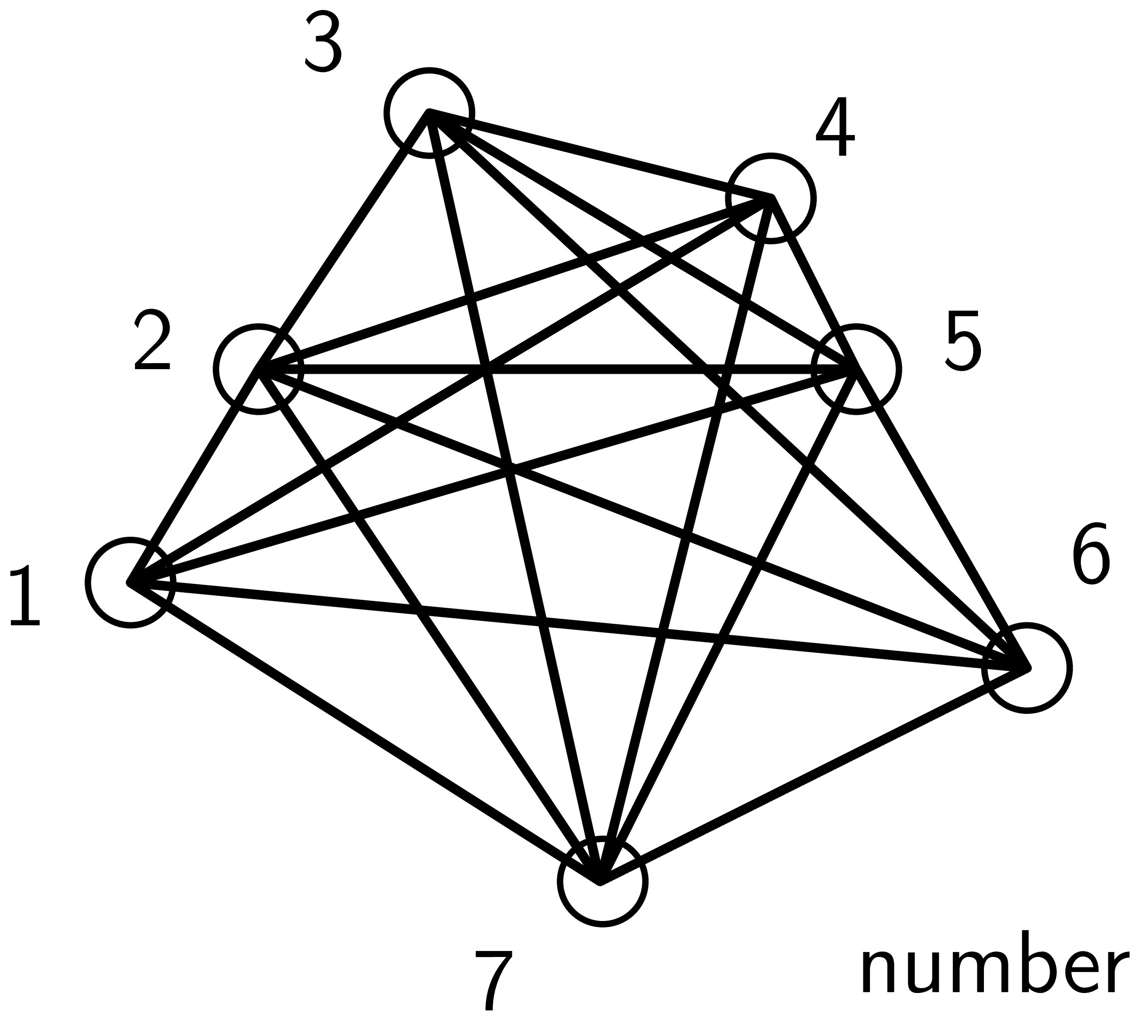
1 – 5 – 6: yes

3 – 4 – 7: yes

4 – 5 – 6: no

Counting Triangles

- 3 points form a triangle if and only if they are non collinear



1 – 2 – 7: yes

1 – 2 – 5: yes

1 – 2 – 3: no

1 – 5 – 6: yes

3 – 4 – 7: yes

4 – 5 – 6: no

number of triangles: 33

Counting Triangles

```
triangleecount = 0
  for i = 1 to n
    for j = i+1 to n
      for k = j+1 to n
        if points i, j, k are not collinear
          trianglecount = trianglecount + 1
```

Counting Triangles

```
trianglecount = 0
  for i = 1 to n
    for j = i+1 to n
      for k = j+1 to n
        if points i, j, k are not collinear
          trianglecount = trianglecount + 1
```

A loop

Counting Triangles

```
trianglecount = 0
  for i = 1 to n
    for j = i+1 to n
      for k = j+1 to n
        if points i, j, k are not collinear
          trianglecount = trianglecount + 1
```

A loop embedded in a loop

Counting Triangles

```
trianglecount = 0
  for i = 1 to n
    for j = i+1 to n
      for k = j+1 to n
        if points i, j, k are not collinear
          trianglecount = trianglecount + 1
```

A loop embedded in a loop embedded in another loop.

Counting Triangles

```
triangle count = 0
  for i = 1 to n
    for j = i+1 to n
      for k = j+1 to n
        if points i, j, k are not collinear
          trianglecount = trianglecount + 1
```

A loop embedded in a loop embedded in another loop.

Second loop begins with $j = i + 1$ and j increases up to n.
Third loop begins with $k = j + 1$ and k increases up to n.

Counting Triangles

```
triangleconut = 0
```

```
triaglecount = 0
```

A loop embedded in a loop embedded in another loop.

Second loop begins with $j = i + 1$ and j increases up to n.
Third loop begins with $k = j + 1$ and k increases up to n.

Thus each triple i, j, k with $i < j < k$ is examined exactly once.

Counting Triangles

```
trianglecount = 0
  for i = 1 to n
    for j = i+1 to n
      for k = j+1 to n
        if points i, j, k are not collinear
          trianglecount = trianglecount + 1
```

A loop embedded in a loop embedded in another loop.

Second loop begins with $j = i + 1$ and j increases up to n.
Third loop begins with $k = j + 1$ and k increases up to n.

Thus each triple i, j, k with $i < j < k$ is examined exactly once.

For example, if $n = 4$, then triples (i, j, k) used by algorithm are (1,2,3), (1,2,4), (1,3,4), and (2,3,4).

Counting Triangles

- Want to compute the number of *increasing triples* (i, j, k) with $1 \leq i < j < k \leq n$.

Counting Triangles

- Want to compute the number of *increasing triples* (i, j, k) with $1 \leq i < j < k \leq n$.

  Claim: Number of increasing triples is exactly the same as number of 3-element subsets from $\{1, 2, \ldots, n\}$

Counting Triangles

- Want to compute the number of *increasing triples* (i, j, k) with $1 \leq i < j < k \leq n$.

Claim: Number of increasing triples is exactly the same as number of 3-element subsets from $\{1, 2, \ldots, n\}$

Why? Let $X =$ set of increasing triples and $Y =$ set of 3-element subsets from $\{1, 2, \ldots, n\}$

Counting Triangles

- Want to compute the number of
  increasing triples (i, j, k) with $1 \leq i < j < k \leq n$.

  Claim: Number of increasing triples is exactly the same as number of 3-element subsets from $\{1, 2, \ldots, n\}$

  Why? Let $X =$ set of increasing triples and
  $Y =$ set of 3-element subsets from $\{1, 2, \ldots, n\}$

  Define: $f : X \to Y$ by $f((i, j, k)) = \{i, j, k\}$
  Claim: f is a **bijection** (why) so $|X| = |Y|$

Counting Triangles

- Want to compute the number of
  increasing triples (i, j, k) with $1 \leq i < j < k \leq n$.

Claim: Number of increasing triples is exactly the same as number of 3-element subsets from $\{1, 2, \ldots, n\}$

Why? Let $X =$ set of increasing triples and
$Y =$ set of 3-element subsets from $\{1, 2, \ldots, n\}$

Define: $f : X \to Y$ by $f((i, j, k)) = \{i, j, k\}$
Claim: f is a **bijection** (why) so $|X| = |Y|$

f is a bijection because
f is one-to-one
if $(i, j, k) \neq (i', j', k') \Rightarrow f(\,(i, j, k)\,) \neq f(\,(i', j', k'))$
f is onto
if γ is a 3-element subset then it can be written as $\gamma = \{i, j, k\}$
where $i < j < k$ so $f(\,(i, j, k)\,) = \gamma$.

Counting Pairs

- The number of

  increasing pairs (i, j) with $1 \leq i < j \leq n$

  is the same as the number of

  2-sets from $\{1, 2, \ldots, n\}$

Counting Pairs

- The number of

  increasing pairs (i, j) with $1 \leq i < j \leq n$

  is the same as the number of

  2-sets from $\{1, 2, \ldots, n\}$

  Define $f : X \to Y$ by $f(\,(i, j)\,) = \{i, j\}$

  Claim: f is a **bijection** so $|X| = |Y|$

Counting Pairs

- The number of

  increasing pairs (i, j) with $1 \leq i < j \leq n$

  is the same as the number of

  2-sets from $\{1, 2, \ldots, n\}$

  Define $f : X \to Y$ by $f(\,(i, j)\,) = \{i, j\}$

  Claim: f is a **bijection** so $|X| = |Y|$

  We actually already saw that $|X| = |Y| = \binom{n}{2}$

The Bijection Principle

- Two sets have the same size if and only if there is a one-to-one function from one set onto the other.

The Bijection Principle

- Two sets have the same size if and only if there is a one-to-one function from one set onto the other.

  A standard first step in counting the size of a set is to use a bijection to show that it has the same size as a 2nd set, and then count the 2nd set instead.

The Bijection Principle

- Two sets have the same size if and only if there is a one-to-one function from one set onto the other.

  A standard first step in counting the size of a set is to use a bijection to show that it has the same size as a 2nd set, and then count the 2nd set instead.

  In practice, in real problems we often only *implicitly* use the bijection and don't *explicitly* describe it

The Bijection Principle

- Two sets have the same size if and only if there is a one-to-one function from one set onto the other.

  A standard first step in counting the size of a set is to use a bijection to show that it has the same size as a 2nd set, and then count the 2nd set instead.

  In practice, in real problems we often only *implicitly* use the bijection and don't *explicitly* describe it

  Currently, we started with the problem of counting the $\#$ of increasing triples and changed it to the problem of counting the $\#$ of 3-element sets from $\{1, 2, \ldots, n\}$

Inclusion-Exclusion Principle

- Used in counts where the decomposition yields two independent counting tasks with overlapping elements

Inclusion-Exclusion Principle

- Used in counts where the decomposition yields two independent counting tasks with overlapping elements

  If we use the sum rule, some elements would be counted twice.

Inclusion-Exclusion Principle

- Used in counts where the decomposition yields two independent counting tasks with overlapping elements

  If we use the sum rule, some elements would be counted twice.

  Inclusion-Exclusion Principle: uses a sum rule and then corrects for the overlapping elements.

Inclusion-Exclusion Principle

- Used in counts where the decomposition yields two independent counting tasks with overlapping elements

  If we use the sum rule, some elements would be counted twice.

Inclusion-Exclusion Principle: uses a sum rule and then corrects for the overlapping elements.

$$|A \cup B| = |A| + |B| - |A \cap B|$$

Inclusion-Exclusion Principle

- **Example**

  How many bit strings of length 8 either start with a ‘1’ bit or end with the two bits ‘00’?

Inclusion-Exclusion Principle

- **Example**

  How many bit strings of length 8 either start with a ‘1’ bit or end with the two bits ‘00’?

  ◇ it is easy to count bit strings starting with ‘1’:

Inclusion-Exclusion Principle

- **Example**

  How many bit strings of length 8 either start with a ‘1’ bit or end with the two bits ‘00’?

  ◇ it is easy to count bit strings starting with ‘1’: 2^7

Inclusion-Exclusion Principle

- **Example**

  How many bit strings of length 8 either start with a '1' bit or end with the two bits '00'?

  - ◇ it is easy to count bit strings starting with '1': 2^7
  - ◇ it is easy to count bit strings ending with '00':

Inclusion-Exclusion Principle

- **Example**

  How many bit strings of length 8 either start with a ‘1’ bit or end with the two bits ‘00’?

  - ◇ it is easy to count bit strings starting with ‘1’: 2^7
  - ◇ it is easy to count bit strings ending with ‘00’: 2^6

Inclusion-Exclusion Principle

- **Example**

  How many bit strings of length 8 either start with a ‘1’ bit or end with the two bits ‘00’?

  ◇ it is easy to count bit strings starting with ‘1’: 2^7

  ◇ it is easy to count bit strings ending with ‘00’: 2^6

  Overcounting!!!

Inclusion-Exclusion Principle

- **Example**

  How many bit strings of length 8 either start with a ‘1’ bit or end with the two bits ‘00’?

  ◇ it is easy to count bit strings starting with ‘1’: 2^7

  ◇ it is easy to count bit strings ending with ‘00’: 2^6

  Overcounting!!!

  ◇ deduct the number of strings starting with ‘1’ and ending with “00”:

Inclusion-Exclusion Principle

- **Example**

How many bit strings of length 8 either start with a ‘1’ bit or end with the two bits ‘00’?

◇ it is easy to count bit strings starting with ‘1’: 2^7

◇ it is easy to count bit strings ending with ‘00’: 2^6

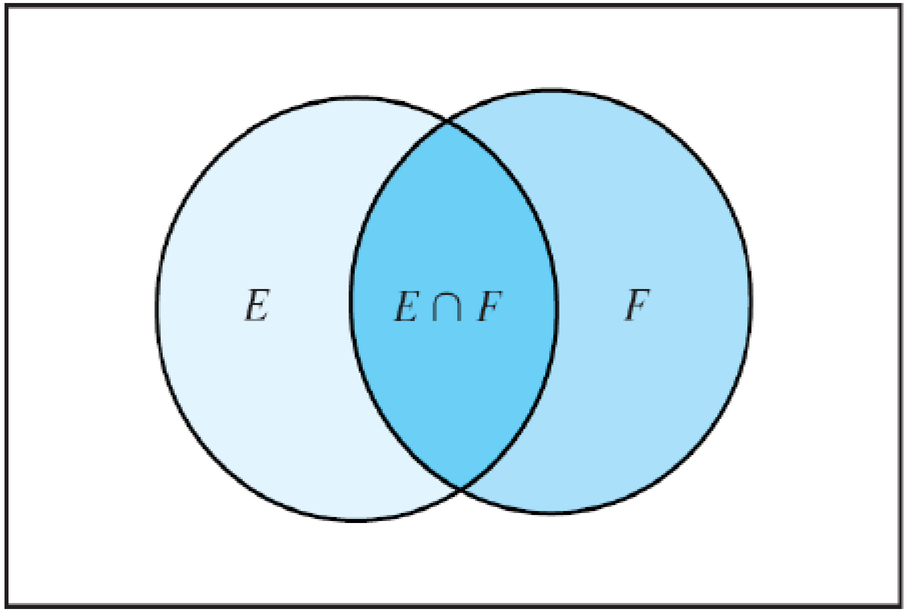
Overcounting!!!

◇ deduct the number of strings starting with ‘1’ and ending with “00”: 2^5

Inclusion-Exclusion Principle

- Two sets

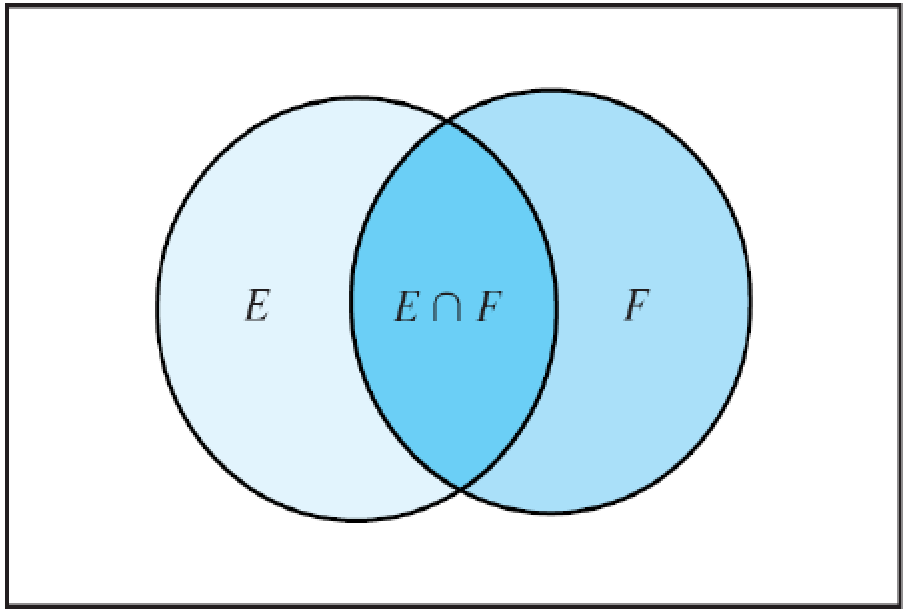
$$|E \cup F| = |E| + |F| - |E \cap F|$$



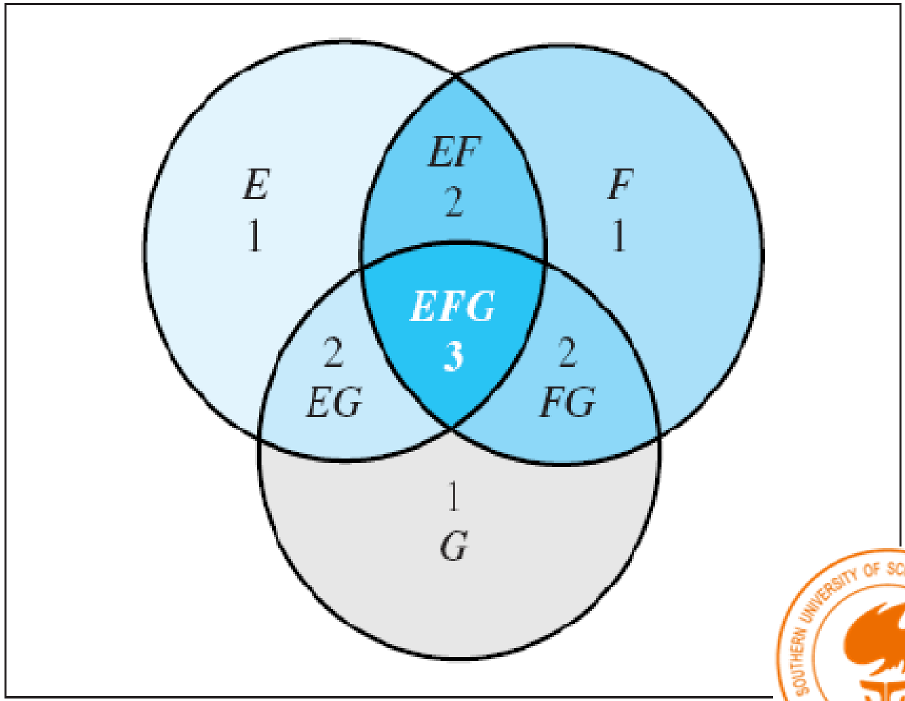
Inclusion-Exclusion Principle

- Two sets

$$|E \cup F| = |E| + |F| - |E \cap F|$$



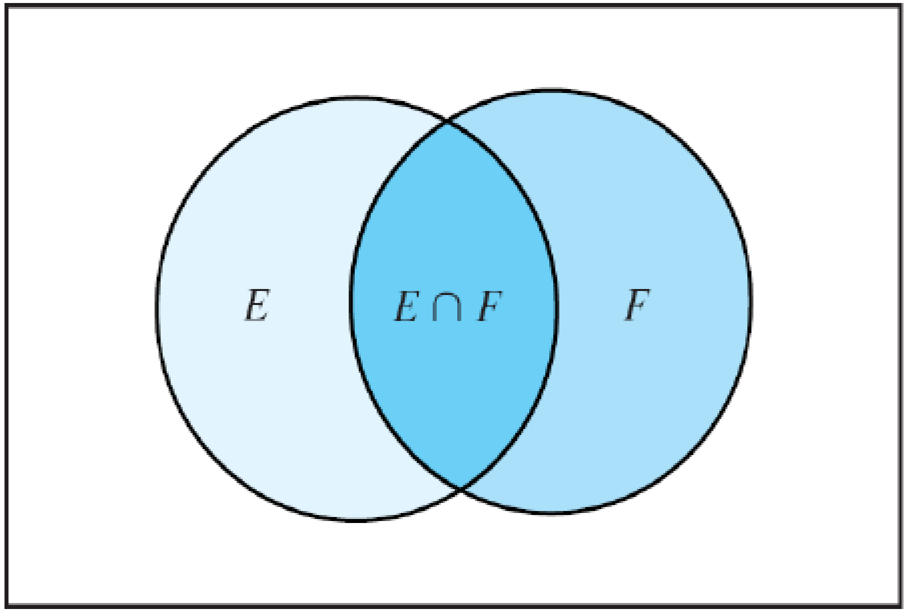
Three sets



Inclusion-Exclusion Principle

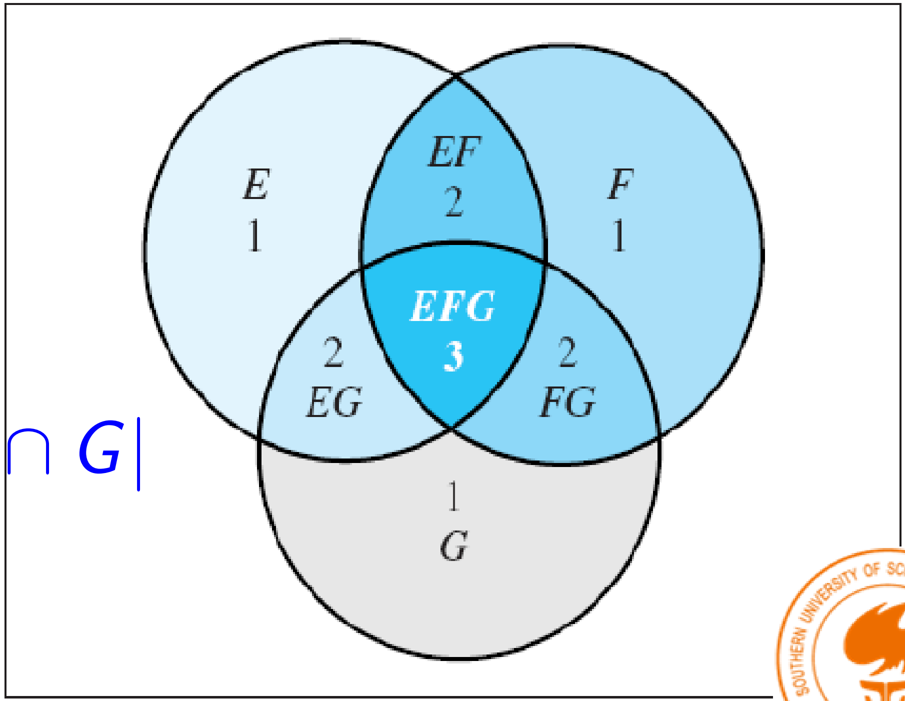
- Two sets

$$|E \cup F| = |E| + |F| - |E \cap F|$$



Three sets

$$\begin{aligned} &|E \cup F \cup G| \\ &= |E| + |F| + |G| \\ &\quad -|E \cap F| - |E \cap G| - |F \cap G| \\ &\quad +|E \cap F \cap G| \end{aligned}$$



Inclusion-Exclusion Principle

- $$|\cup_{i=1}^{n} E_i| = \sum_{k=1}^{n} (-1)^{k+1} \sum_{1 \le i_1 < i_2 < \cdots < i_k \le n} |E_{i_1} \cap E_{i_2} \cap \cdots \cap E_{i_k}|$$

Inclusion-Exclusion Principle

- $$|\cup_{i=1}^{n} E_i| = \sum_{k=1}^{n} (-1)^{k+1} \sum_{1 \le i_1 < i_2 < \cdots < i_k \le n} |E_{i_1} \cap E_{i_2} \cap \cdots \cap E_{i_k}|$$

Proof by induction

Inclusion-Exclusion Principle

- $$|\cup_{i=1}^{n} E_i| = \sum_{k=1}^{n} (-1)^{k+1} \sum_{1 \le i_1 < i_2 < \cdots < i_k \le n} |E_{i_1} \cap E_{i_2} \cap \cdots \cap E_{i_k}|$$

Proof by induction

Base case $(n = 2)$

$$|E \cup F| = |E| + |F| - |E \cap F|$$

Inclusion-Exclusion Principle

- $$|\cup_{i=1}^{n} E_i| = \sum_{k=1}^{n} (-1)^{k+1} \sum_{1 \le i_1 < i_2 < \cdots < i_k \le n} |E_{i_1} \cap E_{i_2} \cap \cdots \cap E_{i_k}|$$

Proof by induction

Base case $(n = 2)$

$$|E \cup F| = |E| + |F| - |E \cap F|$$

Inductive Hypothesis

$$|\cup_{i=1}^{n-1} E_i| = \sum_{k=1}^{n-1} (-1)^{k+1} \sum_{1 \le i_1 < i_2 < \cdots < i_k \le n-1} |E_{i_1} \cap E_{i_2} \cap \cdots \cap E_{i_k}|$$

Inclusion-Exclusion Principle

- Inductive step

  Set $E = E_1 \cup \cdots \cup E_{n-1}$, and $F = E_n$.

Inclusion-Exclusion Principle

- Inductive step

  Set $E = E_1 \cup \cdots \cup E_{n-1}$, and $F = E_n$.

  By $|E \cup F| = |E| + |F| - |E \cap F|$

Inclusion-Exclusion Principle

- Inductive step

  Set $E = E_1 \cup \cdots \cup E_{n-1}$, and $F = E_n$.

  By $|E \cup F| = |E| + |F| - |E \cap F|$

$$|\cup_{i=1}^{n} E_i| = \left|\cup_{i=1}^{n-1} E_i\right| + |E_n| - \left|\left(\cup_{i=1}^{n-1} E_i\right) \cap E_n\right|$$

Inclusion-Exclusion Principle

- Inductive step

Set $E = E_1 \cup \cdots \cup E_{n-1}$, and $F = E_n$.

By $|E \cup F| = |E| + |F| - |E \cap F|$

$$|\cup_{i=1}^{n} E_i| = \boxed{|\cup_{i=1}^{n-1} E_i|} + |E_n| - \left|\left(\cup_{i=1}^{n-1} E_i\right) \cap E_n\right|$$

The first term is given by i.h.

Inclusion-Exclusion Principle

- Inductive step

Set $E = E_1 \cup \cdots \cup E_{n-1}$, and $F = E_n$.

By $|E \cup F| = |E| + |F| - |E \cap F|$

$$|\cup_{i=1}^{n} E_i| = \boxed{|\cup_{i=1}^{n-1} E_i|} + |E_n| - \boxed{\left|(\cup_{i=1}^{n-1} E_i) \cap E_n\right|}$$

The first term is given by i.h.

For the third term, by distributive law,

$$\left|(\cup_{i=1}^{n-1} E_i) \cap E_n\right| = |\cup_{i=1}^{n-1} (E_i \cap E_n)| = |\cup_{i=1}^{n-1} G_i|$$

where $G_i = E_i \cap E_n$.

Inclusion-Exclusion Principle

- So far

$$\left|\cup_{i=1}^{n} E_i\right| = \left|\cup_{i=1}^{n-1} E_i\right| + |E_n| - \left|\cup_{i=1}^{n-1} G_i\right|$$

where $G_i = E_i \cap E_n$.

Inclusion-Exclusion Principle

- So far

$$|\cup_{i=1}^{n} E_i| = \left|\cup_{i=1}^{n-1} E_i\right| + |E_n| - \left|\cup_{i=1}^{n-1} G_i\right|$$

where $G_i = E_i \cap E_n$.

Note that (why?)

$$-(-1)^{k+1}|G_{i_1} \cap G_{i_2} \cap \cdots \cap G_{i_k}|$$
$$= (-1)^{k+2}|E_{i_1} \cap E_{i_2} \cap \cdots \cap E_{i_k} \cap E_n|$$

Inclusion-Exclusion Principle

- So far

$$\left|\cup_{i=1}^{n} E_i\right| = \left|\cup_{i=1}^{n-1} E_i\right| + |E_n| - \left|\cup_{i=1}^{n-1} G_i\right|$$

where $G_i = E_i \cap E_n$.

Note that (why?)

$$-(-1)^{k+1}|G_{i_1} \cap G_{i_2} \cap \cdots \cap G_{i_k}|$$
$$= (-1)^{k+2}|E_{i_1} \cap E_{i_2} \cap \cdots \cap E_{i_k} \cap E_n|$$

Some discussion:
first summation sums $(-1)^{k+1}|E_{i_1} \cap E_{i_2} \cap \cdots \cap E_{i_k}|$ over all lists $i_1, i_2, \ldots, i_k$ that do not contain n
$|E_n|$ and second summation together sum $(-1)^{k+1}|E_{i_1} \cap E_{i_2} \cap \cdots \cap E_{i_k}|$ over all lists $i_1, i_2, \ldots, i_k$ that do contain n

Inclusion-Exclusion Principle

- $$|\cup_{i=1}^{n} E_i| = \sum_{k=1}^{n} (-1)^{k+1} \sum_{1 \le i_1 < i_2 < \cdots < i_k \le n} |E_{i_1} \cap E_{i_2} \cap \cdots \cap E_{i_k}|$$

Inclusion-Exclusion Principle

- $$|\cup_{i=1}^{n} E_i| = \sum_{k=1}^{n} (-1)^{k+1} \sum_{1\le i_1<i_2<\cdots<i_k\le n} |E_{i_1} \cap E_{i_2} \cap \cdots \cap E_{i_k}|$$

  This can be used to determine the number of onto functions

Inclusion-Exclusion Principle

- $$|\cup_{i=1}^{n} E_i| = \sum_{k=1}^{n} (-1)^{k+1} \sum_{1 \le i_1 < i_2 < \cdots < i_k \le n} |E_{i_1} \cap E_{i_2} \cap \cdots \cap E_{i_k}|$$

  This can be used to determine the number of onto functions

  A, B are two sets with $|A| = m$ and $|B| = n$.

Inclusion-Exclusion Principle

- $$|\cup_{i=1}^{n} E_i| = \sum_{k=1}^{n} (-1)^{k+1} \sum_{1 \le i_1 < i_2 < \cdots < i_k \le n} |E_{i_1} \cap E_{i_2} \cap \cdots \cap E_{i_k}|$$

This can be used to determine the number of onto functions

A, B are two sets with $|A| = m$ and $|B| = n$.

(a) How many onto functions are there from A to B?

(b) How many functions are there from A to B that map nothing to at least one element of B?

Inclusion-Exclusion Principle

- $$|\cup_{i=1}^{n} E_i| = \sum_{k=1}^{n} (-1)^{k+1} \sum_{1 \le i_1 < i_2 < \cdots < i_k \le n} |E_{i_1} \cap E_{i_2} \cap \cdots \cap E_{i_k}|$$

This can be used to determine the number of onto functions

A, B are two sets with $|A| = m$ and $|B| = n$.

(a) How many onto functions are there from A to B?

(b) How many functions are there from A to B that map nothing to at least one element of B?

$$\#(a) + \#(b) = n^m$$

Inclusion-Exclusion Principle

- This can be used to determine the number of onto functions

  A, B are two sets with $|A| = m$ and $|B| = n$.

  (a) How many onto functions are there from A to B?

  (b) How many functions are there from A to B that map nothing to at least one element of B?

$$\#(a) + \#(b) = n^m$$

Inclusion-Exclusion Principle

- This can be used to determine the number of onto functions

  A, B are two sets with $|A| = m$ and $|B| = n$.

  (a) How many onto functions are there from A to B?

  (b) How many functions are there from A to B that map nothing to at least one element of B?

$$\#(a) + \#(b) = n^m$$

Set E_i – set of functions that map nothing to element i of B

Inclusion-Exclusion Principle

- This can be used to determine the number of onto functions

  A, B are two sets with $|A| = m$ and $|B| = n$.

  (a) How many onto functions are there from A to B?

  (b) How many functions are there from A to B that map nothing to at least one element of B?

$$\#(a) + \#(b) = n^m$$

Set E_i – set of functions that map nothing to element i of B

$\#(b) = |\cup_{i=1}^{n} E_i|$

Inclusion-Exclusion Principle

- This can be used to determine the number of onto functions

  A, B are two sets with $|A| = m$ and $|B| = n$.

  (a) How many onto functions are there from A to B?

  (b) How many functions are there from A to B that map nothing to at least one element of B?

$$\#(a) + \#(b) = n^m$$

Set E_i – set of functions that map nothing to element i of B

$\#(b) = |\cup_{i=1}^{n} E_i|$

$= \sum_{k=1}^{n}(-1)^{k+1} \sum_{1\le i_1 < i_2 < \cdots < i_k \le n} |E_{i_1} \cap E_{i_2} \cap \cdots \cap E_{i_k}|$

Inclusion-Exclusion Principle

- This can be used to determine the number of onto functions

  A, B are two sets with $|A| = m$ and $|B| = n$.

  (a) How many onto functions are there from A to B?

  (b) How many functions are there from A to B that map nothing to at least one element of B?

$$\#(a) + \#(b) = n^m$$

Set E_i – set of functions that map nothing to element i of B

$\#(b) = |\cup_{i=1}^{n} E_i|$

$= \sum_{k=1}^{n}(-1)^{k+1} \sum_{1 \le i_1 < i_2 < \cdots < i_k \le n} |E_{i_1} \cap E_{i_2} \cap \cdots \cap E_{i_k}|$

$= \sum_{k=1}^{n}(-1)^{k+1} \binom{n}{k}(n-k)^m$

Next Lecture

- recurrence ...