

A GOSSIP PROBLEM

For $n \geq 4$, consider a group of n persons, P_1, P_2, \dots, P_n . Every person P_i knows a scandal S_i that only he/she knows. Person P_i can make a phone call to P_j in which all scandals they know get exchanged. (So all scandals that P_i knows get communicated to P_j and vice versa.).

After how many phone calls does everyone know all scandals?

GOSSIP PROBLEM: BRUTE FORCE

Solution I: Each pair of persons makes one phone call.

Why is this a solution? I.e., why is the brute force approach correctly solving the problem?

How many phone calls are that?

$$\binom{n}{2} = \frac{n(n-1)}{2}$$

So, a quadratic number of calls are made. Can we do better?

GOSSIP PROBLEM: LINEAR

Initially, each person knows one secret, visualized for $n=4$, as follows:

P₁	P₂	P₃	P₄
S ₁	S ₂	S ₃	S ₄

now, P₁ calls P₂.

P₁	P₂	P₃	P₄
S ₁ S ₂	S ₁ S ₂	S ₃	S ₄

GOSSIP PROBLEM: LINEAR - ILLUSTRATION

now P_3 calls P_4 :

P_1	P_2	P_3	P_4
$S_1 S_2$	$S_1 S_2$	$S_3 S_4$	$S_3 S_4$

now, P_1 calls P_3 .

P_1	P_2	P_3	P_4
$S_1 S_2 S_3 S_4$	$S_1 S_2$	$S_1 S_2 S_3 S_4$	$S_3 S_4$

GOSSIP PROBLEM: LINEAR

Finally, P_2 calls P_4 .

P_1	P_2	P_3	P_4
$S_1 S_2 S_3 S_4$	$S_1 S_2 S_3 S_4$	$S_1 S_2 S_3 S_4$	$S_1 S_2 S_3 S_4$

So, how calls many were these? 4 calls instead of $\binom{4}{2} = \frac{4(3-1)}{2} = 6$

GOSSIP PROBLEM: LINEAR (RECURSIVE)

Gossip(n)

If $n = 4$ Base case: for four people, use the algorithm described above

else P_{n-1} calls P_n

Gossip($n-1$)

P_{n-1} calls P_n why does that not look like a repeat from above?

endif

ANALYSIS OF RECURSIVE SOLUTION TO GOSSIP

Let $C(n)$ denote the number of calls made by the algorithm for n persons and secrets.

$$C(4) = 4$$

Let $n \geq 5$. we claim that

$$C(n) = C(n-1) + 2 \quad \text{why?} \quad n \geq 5$$

$$\text{Then,} \quad C(4) = 4 \quad C(5) = C(4) + 2 = 6 \quad C(6) = C(5) + 2 = 8 \dots$$

Guess therefore that $C(n) = 2n - 4$ for $n \geq 4$.

use simple induction (sketched here only)

$$\text{with induction step: } C(n) = C(n-1) + 2 = 2(n-1) - 4 + 2 = 2n - 2 - 4 + 2 = 2n - 4 \quad \text{q.e.d.}$$

GOSSIP – RECURSIVE ALGORITHM IS OPTIMAL

- it can be shown (not done here) that **ANY** algorithm that schedules phone calls for $n \geq 4$ persons must make **at least** $2n - 4$ phone calls.
- Therefore, we cannot do better and our algorithm is optimal.
- Check exercise 4.52 to see why we start the base case with $n=4$.
 - If we were to go to $n=2$ as base case, we would need $2n-3$ phone calls.

EUCLID'S ALGORITHM

We might call Euclid's method the granddaddy of all algorithms, because it is the oldest nontrivial algorithm that has survived to the present day.

— Donald E. Knuth, *The Art of Computer Programming*, Vol. 2, 1997

EUCLID'S ALGORITHM

- The **greatest common divisor** of two integers $a \geq 1$ and $b \geq 1$ is the largest integer that divides both a and b . We denote this largest integer by **$\gcd(a, b)$** .

- **Example:** 36: has divisors: 1, 2, 3, 4, 6, 9, **12**, 18, 36
 24: has divisors: 1, 2, 4, 6, **12**, 24

How can we determine the gcd of two numbers efficiently? Efficiency becomes important in particular if we have huge numbers

$a = 371,435,805$ and $b = 137,916,675$

EUCLID'S ALGORITHM

Let us determine a prime factorization of a and b:

$$a = 371,435,805 = 3^2 \cdot 5 \cdot 13^4 \cdot 17^2$$

$$b = 137,916,675 = 3^4 \cdot 5^2 \cdot 13^3 \cdot 31$$

Now, we can see that $3^2 \cdot 5 \cdot 13^3 = 98,865$ is the $\gcd(a,b)$.

EUCLID'S ALGORITHM

No algorithm is known which does prime factorization efficiently.

(This would really help in attacks in Cryptography)

So, prime factorization is not a path we can take.

We introduce first the modulo operation:

MODULO

Let $a \geq 1$ and $b \geq 1$ be integers. By dividing a by b , we obtain a quotient q and a remainder r , which are the unique integers that satisfy

$$a = qb + r, q \geq 0, \text{ and } 0 \leq r \leq b - 1.$$

We denote the modulo operation by $a \bmod b$.

$$a \bmod b = r.$$

Examples:

$$20 \bmod 4 = 0 \quad (20 = 5 \cdot 4 + 0).$$

$$21 \bmod 4 = 1 \quad (21 = 5 \cdot 4 + 1)$$

$$14 \bmod 4 = 2 \quad (14 = 3 \cdot 4 + 2)$$

$$4 \bmod 4 = 0 \quad (4 = 1 \cdot 4 + 0)$$

EUCLID'S ALGORITHM

Algorithm EUCLID(a, b):

// a and b are integers with $a \geq b \geq 1$

$r = a \bmod b$;

if $r = 0$

then return b

else EUCLID(b, r)

// observe that $b > r \geq 1$

endif

EUCLID'S ALGORITHM: EXAMPLE

Call to: $\text{Euclid}(75, 45)$,

the algorithm computes $75 \bmod 45$, which is 30.

Then, calls $\text{Euclid}(45, 30)$,

the algorithm computes $45 \bmod 30$, which is 15.

Now, call $\text{Euclid}(30, 15)$,

the algorithm computes $30 \bmod 15$, which is 0.

Since $r = 0$, the algorithm returns 15,

15 is the greatest common divisor of our input; 75 and 45.

EUCLID'S ALGORITHM: WHY DOES IT WORK?

We need some help for this given through this lemma:

Lemma 4.5.1 Let a and b be integers with $a \geq b \geq 1$, and let $r = a \bmod b$.

1. If $r = 0$, then $\gcd(a, b) = b$.
2. If $r \geq 1$, then $\gcd(a, b) = \gcd(b, r)$.

Proof Because $a \geq b$ we can write a as: $a = qb + r$, where q, r are integers such that

$$q \geq 1, \quad \text{and} \quad 0 \leq r \leq b - 1.$$

Should r be 0, we are done as $\gcd(a, b) = b$ in that case.

So, assume that $r > 0$.

EUCLID'S ALGORITHM: PROOF OF LEMMA CONT'D

So, assume that $r > 0$. In that case, the common divisors of a and b are the same as the common divisors of b and r . Why is this true?

1. Let $d \geq 1$ be an integer that divides both a and b .

Since $r = a - qb$, it follows that d divides r . Thus, d divides both b and r .

2. Let $d \geq 1$ be an integer that divides both b and r .

Since $a = qb + r$, it follows that d divides a . Thus, d divides both a and b .

Since the two pairs a, b and b, r have the same common divisors, their greatest common divisors are equal as well. q.e.d.

EUCLID'S ALGORITHM: THEOREM

Theorem 4.5.2 For any two integers a and b with $a \geq b \geq 1$, algorithm $\text{Euclid}(a, b)$ returns $\text{gcd}(a, b)$.

Proof The correctness is clear via the previous lemma. We only need to establish that the algorithm indeed terminates. That follows from the fact that in each recursive call to Euclid , the second argument decreases. Since that argument is a positive integer, the algorithm must terminate. q.e.d.

EUCLID'S ALGORITHM: RUNNING TIME

- How many mod operations, denoted by $M(a,b)$, does the algorithm perform?
- Recall the example the number of mod operations is 3.

Call to: $\text{Euclid}(75, 45)$,

the algorithm computes $75 \bmod 45$, which is 30.

Then, calls $\text{Euclid}(45, 30)$,

the algorithm computes $45 \bmod 30$, which is 15.

Now, call $\text{Euclid}(30, 15)$,

the algorithm computes $30 \bmod 15$, which is 0.

Since $r = 0$, the algorithm returns 15 no more mod operations performed.

EUCLID'S ALGORITHM: RUNNING TIME

First crude analysis: $M(a,b)$ is linear in b

- initial call $\text{Euclid}(a,b)$; the second argument (initially b) always decreases. So, for all integers a and b with $a \geq b \geq 1$, $M(a, b) \leq b$.

Second more refined analysis: $M(a,b)$ is logarithmic in b

Recall Fibonacci: $f_0 = 0, f_1 = 1, f_n = f_{n-1} + f_{n-2}$ if $n \geq 2$

Usually, we do time complexity analysis as a function of the input size. Here we do something somewhat strange. We fix a value m for the running time $M(a,b)$ and see how big a and b must be at least. That means we give a lower bound on both a and b in terms of m .

LEMMA TO HELP UNDERSTAND THE COMPLEXITY

Lemma 4.5.3 Let a and b be integers with $a > b \geq 1$, and let $m = M(a, b)$.

Then $a \geq f_{m+2}$ and $b \geq f_{m+1}$.

Since Fibonacci increases exponential for the given m , conversely m must be logarithmic in terms of a and b . We will make that more precise later.

Proof (Lemma) We use induction on m .

Base case: $m = 1$ Since $a > b \geq 1$, $a \geq b+1 \geq 2 = f_3$ and $b \geq 1 = f_2$

thus $a \geq f_{1+2}$ and $b \geq f_{1+1}$

PROOF OF LEMMA CONT'D

Induction step: assume that $m \geq 2$. Let the integers q and r satisfy $a = qb + r$, $q \geq 1$, and $0 \leq r \leq b - 1$.

Algorithm Euclid(a, b) computes the value $a \bmod b$, which is equal to r .

Since $m \geq 2$, we have $r \geq 1$ and the total number of modulo operations, $M(b, r)$, performed during the recursive call Euclid(b, r) is equal to $m - 1$.

Therefore, by induction hypothesis, we have $b \geq f_{m+1}$ and $r \geq f_m$.

Now observe that $a = qb + r \geq b + r \geq f_{m+1} + f_m = f_{m+2}$.

q.e.d.

EUCLID'S ALGORITHMS IS LOGARITHMIC

Recall, for the n^{th} Fibonacci number we have shown that: $f_n = \frac{\varphi^n - \psi^n}{\sqrt{5}}$ furthermore,

$\varphi^2 = \varphi + 1$ with that it is easily shown that $f_n \geq \varphi^{n-2}$. (induction)

Theorem 4.5.4 Let a and b be integers with $a \geq b \geq 1$. Then,

$$M(a, b) \leq 1 + \log_{\varphi} b,$$

i.e., the total number of modulo operations performed by algorithm Euclid(a, b)

is $O(\log b)$.

EUCLID'S ALGORITHMS IS LOGARITHMIC PROOF

Proof If $a=b$ then $M(a,b) = 1$ – in this case, the claim holds easily.

Assume therefore, that $a > b$. By Lemma 4,5,3 and $f_n \geq \varphi^{n-2}$ we get

$b \geq f_{m+1} \geq \varphi^{m-1}$ Now, take the log to the base φ on both sides and we get

$\log_{\varphi} b \geq m-1$. Thus, $M(a,b) = m \leq 1 + \log_{\varphi} b = O(\log b)$. q.e.d.

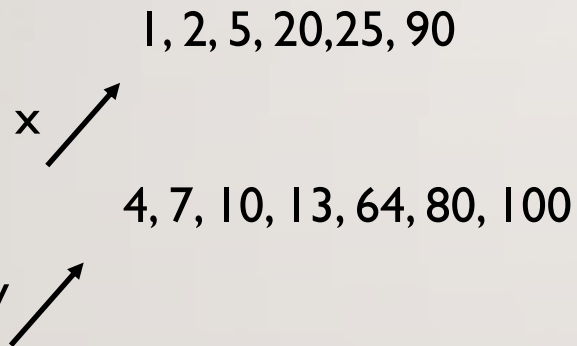
(Recall: Base change law for logs: $\log_b(x) = \log_c(x) / \log_c(b)$;

so we can, for the big oh notation, omit the log base as conversion between any two bases is

just a multiplicative constant $1 / \log_c(b)$)

SORTING TWO SORTED LISTS

How do we sort two lists L_1 and L_2 that are sorted (in increasing order)



- Let x be the first element of L_1 and let y be the first element of L_2 .
- If $x \leq y$, then remove x from L_1 and append it to L (i.e., add x at the end of L).
- Otherwise (i.e., if $x > y$), remove y from L_2 and append it to L .

See class 1, 2, 4, 5, 7, 10, 13, 20, 25, 64, 80, 90, 100

Important note: the total number of comparisons of the form $x \leq y$, is linear in the size of the lists, where x and y are elements from the lists.

SORTING TWO SORTED LISTS

How do we sort two lists L_1 and L_2
that are sorted (in increasing order)

pseudo-code

Time complexity: $O(|L|)$

Algorithm MERGE(L_1, L_2):

```
//  $L_1$  and  $L_2$  are sorted lists
 $L$  = empty list;
while  $L_1$  is not empty and  $L_2$  is not empty
do  $x$  = first element of  $L_1$ ;
     $y$  = first element of  $L_2$ ;
    if  $x \leq y$ 
    then remove  $x$  from  $L_1$ ;
        append  $x$  to  $L$ 
    else remove  $y$  from  $L_2$ ;
        append  $y$  to  $L$ 
    endif
endwhile;
if  $L_1$  is empty
then append  $L_2$  to  $L$ 
else append  $L_1$  to  $L$ 
endif;
return  $L$ 
```

MERGE-SORT

We use the above Merge procedure in Algorithm MergeSort.

MergeSort: A recursive algorithm which sorts n numbers a_1, a_2, \dots, a_n as follows:

1. pick the element a_m with $m=n/2$
2. recursive sort using MergeSort a_1, a_2, \dots, a_m and store the result in a list L_1
3. recursive sort using MergeSort $a_{m+1}, a_{m+2}, \dots, a_n$ and store the result in a list L_2
4. Merge the two lists L_1 and L_2 into one sorted list L .

In PseudoCode next

MERGE-SORT

Algorithm MERGESORT(L, n):

// L is a list of $n \geq 0$ numbers

if $n \geq 2$

then $m = \lfloor n/2 \rfloor$;

L_1 = list consisting of the first m elements of L ;

L_2 = list consisting of the last $n - m$ elements of L ;

L_1 = MERGESORT(L_1, m);

L_2 = MERGESORT($L_2, n - m$);

L = MERGE(L_1, L_2)

endif;

return L

MERGE-SORT: CORRECTNESS

- Why does MergeSort correctly sort a list of n elements?
- First observe that all elements of L_1 and L_2 will appear in L and only those when calling procedure Merge. I will not prove the correctness of Merge.
- Now induction proof for MergeSort
- Base cases $n=0$ and $n=1$.
 - In those cases the algorithm simply returns L which is correct.

```
// MergeSort
if  $n \geq 2$ 
  then  $m = \lfloor n/2 \rfloor$ 
         $L_1 = \text{MergeSort}(L[1:m])$ 
         $L_2 = \text{MergeSort}(L[m+1:n])$ 
         $L_1 = \text{Merge}(L_1, L_2)$ 
         $L_2 = \text{Merge}(L_1, L_2)$ 
         $L = \text{Merge}(L_1, L_2)$ 
endif;
return  $L$ 
```

MERGESORT CORRECTNESS

Let $n \geq 2$

Induction hypothesis: assume that for any integer $0 \leq k < n$ and for any List L' of k numbers, algorithm MergeSort correctly returns the sorted list L' containing all and only those elements that were originally in L' .

Induction step: Let L be a list of n elements. By induction hypothesis L_1 and L_2 are sorted correctly

$L_1 =$ list consisting of the first m elements of L ;
 $L_2 =$ list consisting of the last $n - m$ elements of L ;
 $L_1 = \text{MERGESORT}(L_1, m)$;
 $L_2 = \text{MERGESORT}(L_2, n - m)$;

MERGESORT: CORRECTNESS

Algorithm Merge then correctly merges the two sorted lists L_1 and L_2 to produce the output list L .

$$L = \text{MERGE}(L_1, L_2)$$

L is sorted and contains all and only those elements that were in the input.

q.e.d.

MERGESORT: RUNNING TIME

MergeSort is a recursive algorithm. We will count the number, $T(n)$, of comparisons made by mergesort. That means we count only the operations of the type $x \leq y$ as they appeared in algorithm Merge.

```
y ← first element of L2;  
if  $x \leq y$   
then remove  $x$  from  $L$ 
```

base case $n = 1$ $T(1) = 0$

COMPLEXITY RECURSIVE

- we do 2 recursive steps on lists L_1 and L_2 . These lists have size $n/2$. No further comparisons (except for the recursive calls) are made.
- Furthermore, we do execute $\text{Merge}(L_1, L_2)$ taking a linear number of comparisons:
 - $|L_1| + |L_2| = |L| = n$

Recursive Step: $T(n) \leq T(n/2) + T(n/2) + n = 2 T(n/2) + n$

SOLVING THE RECURRENCE

So how do we solve this recurrence ?

Let us assume that n is a power of 2?

There are many techniques (see 3804). Here we give one.

It is easy, but you need to be careful not to forget some terms.

$$T(1) = 0$$

$$T(n) \leq 2T(n/2) + n \text{ for } n \geq 2 \text{ and } n \text{ is a power of 2, i.e., } n = 2^k.$$

UNFOLDING

- $T(n) \leq 2T(n/2) + n$
 - what is $T(n/2)$? $T(n/2) \leq 2T(n/2^2) + n/2$



- $T(n) \leq 2T(n/2) + n \leq 2(2T(n/2^2) + n/2) + n$
 $= 2^2 T(n/2^2) + n + n = 2^2 T(n/2^2) + 2n$

UNFOLDING

- Using $T(n) \leq 2 T(n/2) + n$

what is $T(n/2)$? $T(n/2) \leq 2 T(n/4) + n/2$



- $T(n) \leq 2^2 T(n/4) + 2n \leq 2^2(2 T(n/8) + n/4) + 2n = 2^3 T(n/8) + n + 2n = 2^3 T(n/8) + 3n$
- so do we see a pattern here?

UNFOLDING

- $T(n) \leq 2^i T(n/2^i) + in$
 - easily shown by induction. (see textbook e.g.)

Now, how long can we unfold? Until we cannot divide n anymore by 2.

We know that $n = 2^k$ (by initial assumption). Thus, $k = \log_2 n$.

Plugging k into the inequality for i above we get:

- $T(n) \leq 2^k T(n/2^k) + kn$, where $k = \log_2 n$.

UNFOLDING COMPLETE

- $T(n) \leq 2^k T(n/2^k) + kn$, where $k = \log_2 n$.

- $$\begin{aligned} T(n) &\leq 2^{\log_2 n} T(n/2^{\log_2 n}) + \log_2 n * n \\ &= n T(1) + n \log_2 n = n \log_2 n \quad (\text{recall: } T(1) = 0) \end{aligned}$$

We only counted element comparisons. However, it is easily seen that the total number of elementary operations is a constant factor of the element comparisons.

THEOREM FOR MERGESORT


Also, note that if n is not a power of 2, we can slightly modify the recurrence and its solution

$$T(n) = 0 \text{ for } n = 0 \text{ or } n = 1$$

$$T(n) \leq T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + n \text{ for } n \geq 2$$

Theorem: For any list L of n numbers, the running time of algorithm MergeSort(L, n) is $O(n \log n)$.

CHAPTER 5: DISCRETE PROBABILITIES

Your weather when it really matters™							
	Mainly sunny	Mainly sunny	Cloudy with sunny breaks	Mainly sunny	Flurries	Mainly sunny	A mix of sun and clouds
							
	-7°	-10°	-14°	-14°	-13°	-12°	-9°
Feels like	-13	-16	-19	-19	-18	-18	-12
Night	-16°	-21°	-22°	-18°	-21°	-19°	-11°
POP	20 %	20 %	30 %	20 %	60 %	20 %	30 %
Wind (km/h)	16 W	14 NW	10 N	10 NW	11 NE	16 W	10 E
Wind gust (km/h)	24	21	15	15	17	24	15
Hrs Of Sun	8 h	8 h	2 h	6 h	3 h	6 h	4 h
24 Hr Snow	-	-	-	1-3 cm	1-3 cm	-	-

"PoP" (probability of precipitation) statement, which is often expressed as the "chance of precipitation".

DISCRETE PROBABILITIES: 6/49

- probability of winning the jackpot in the 6/49 lottery = $1 / 13,983,816$
- the probability of winning the lottery =
the number of winning lottery numbers / the total number of possible lottery numbers

We will now make the notion of probability and discrete probability clearer next.

First though, a very interesting and unintuitive example.

ANONYMOUS BROADCASTING

Group: Persons P_1, P_2, \dots, P_n for some integer $n \geq 3$. One person, say P_k , wants to broadcast a message so that:

- everybody in the group receives the message,
- nobody knows that the message was broadcast by P_k .

What does the task mean exactly

When P_i (with $i \neq k$) receives the message, P_i only knows that it **was broadcast by one of** $P_1, \dots, P_{i-1}, P_{i+1}, \dots, P_n$; P_i **cannot determine who** broadcast the message.

Seems impossible – right?

Wrong!

DAVID CHAUM SHOWED IT IS POSSIBLE

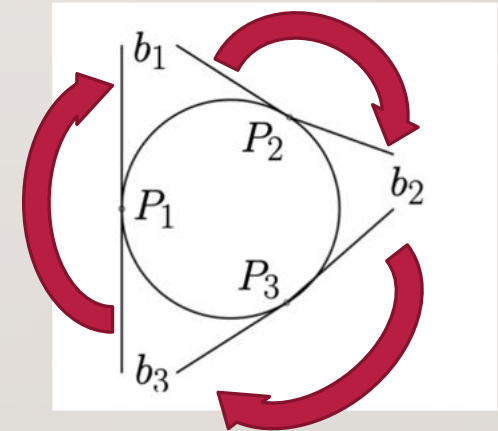
- The dining cryptographers problem: *Unconditional sender and recipient untraceability*
- David Chaum
- *Journal of Cryptology* **volume 1**, pages 65–75 (1988)
- Abstract
- Keeping confidential who sends which messages, in a world where any physical transmission can be traced to its origin, seems impossible. The solution presented here is unconditionally or cryptographically secure, depending on whether it is based on one-time-use keys or on public keys, respectively. It can be adapted to address efficiently a wide variety of practical considerations.

SOLUTION FOR THREE PERSONS

Assume that P_1 , P_2 and P_3 are sitting at a table, in clockwise order of their indices. Let b be the current bit that the broadcaster wants to announce.

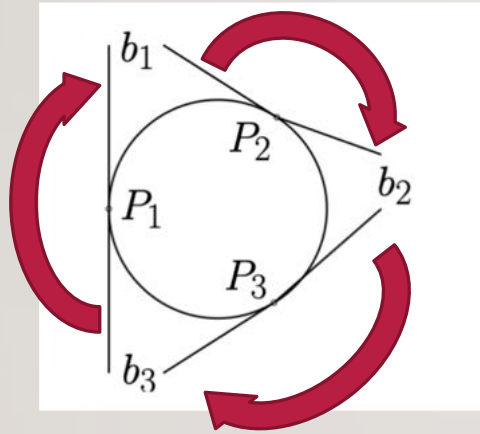
Step 1: Each person P_i **generates a random bit** b_i , for example, by flipping a coin. Thus, with 50% probability, $b_i = 0$ and with 50% probability, $b_i = 1$.

Step 2: Each person P_i **shows the bit b_i** to her **clockwise neighbour**.
Sitting clockwise around a table.



SOLUTION FOR THREE PERSONS

So, after Step 2,



P_1 knows: b_1 and b_3 but not b_2

P_2 knows: b_2 and b_1 but not b_3

P_3 knows: b_3 and b_2 but not b_1

SOLUTION FOR THREE PERSONS

Step 3: Each person P_i computes the sum s_i (modulo 2) of the bits that she knows. Thus,

- P_1 computes $s_1 = (b_1 + b_3) \bmod 2$,
- P_2 computes $s_2 = (b_1 + b_2) \bmod 2$,
- P_3 computes $s_3 = (b_2 + b_3) \bmod 2$.

SOLUTION FOR THREE PERSONS

Step 4: Each person P_i does the following:

- If P_i is **not the broadcaster**, she sets $t_i = s_i$
- If P_i **is the broadcaster**, she sets $t_i = (s_i + b) \bmod 2$.

Recall that b is the current bit that the broadcaster wants to announce.

(Thus, if $b = 1$, then P_i “secretly” flips the bit s_i and stores the result in t_i .)

SOLUTION FOR THREE PERSONS

Step 5: Each person P_i shows her bit t_i to the other two people.

Step 6: Each person P_i computes the sum (modulo 2) of the three bits t_1 , t_2 , and t_3 , i.e., the value $(t_1 + t_2 + t_3) \bmod 2$.

WHY DOES IT WORK?

That requires only a little thought.

First: for any bit x , holds that: $(x + x) \bmod 2 = 0$. Therefore, the last announcement by all:

$(t_1 + t_2 + t_3) \bmod 2$ is b as we can see from the below. All addition are done mod 2.

$$(t_1 + t_2 + t_3) = s_1 + s_2 + s_3 + b \quad \text{recall } t_i = s_i \quad \text{or } t_i = (s_i + b) \bmod 2 \text{ for the broadcaster}$$

$$= (b_1 + b_3) + (b_1 + b_2) + (b_2 + b_3) + b$$

$$= (b_1 + b_1) + (b_2 + b_2) + (b_3 + b_3) + b$$

$$= b \quad \text{which is the bit, the broadcaster wanted to announce.}$$

- P_1 computes $s_1 = (b_1 + b_3) \bmod 2$,
- P_2 computes $s_2 = (b_1 + b_2) \bmod 2$,
- P_3 computes $s_3 = (b_2 + b_3) \bmod 2$.

WHY IS THE BROADCASTER ANONYMOUS?

Assume that the broadcaster wanted to announce $b=1$.

In the setting, the seating of each person is symmetric. So, focus on P_2 being not the broadcaster. We need to show that P_2 cannot find out whether P_1 or P_3 is the broadcaster.

What does P_2 know? b_1, b_2, t_1, t_2 , and t_3 but **not b_3** .

Unfortunately, the proof is based on case analysis. On the highest case level we distinguish between: $b_1 = b_2$ and $b_1 \neq b_2$.

CASE ANALYSIS

I. $b_1 = b_2$

Now, we need to distinguish between two subcases depending on the value of b_3

I.1 $b_3 = b_1$

Thus, $b_1 = b_2 = b_3$

If P_1 is the broadcaster, then (again all arithmetic ops done mod 2) $t_i = (s_i + b) \bmod 2$.

$$t_1 = s_1 + 1 = b_1 + b_3 + 1 = 1$$

and $t_3 = s_3 = b_2 + b_3 = 0$

$t_i = s_i$ P_3 computes $s_3 = (b_2 + b_3) \bmod 2$

CASE ANALYSIS CONT'D

analogously,

If P_3 is the broadcaster, then (again all arithmetic ops done mod 2)

$$t_1 = 0 \quad \text{and} \quad t_3 = 1.$$

So, the broadcaster is the person whose t -bit is equal to 1.

CASE ANALYSIS CONT'D

1.2. $b_3 \neq b_1$ and thus $b_3 \neq b_2$ (we still in the case $b_1 = b_2$)

If P_1 is the broadcaster, then

$$t_1 = s_1 + 1 = b_1 + b_3 + 1 = 0$$

and $t_3 = s_3 = b_2 + b_3 = 1$

analogously, if P_3 is the broadcaster, then

$$t_1 = 1 \quad \text{and} \quad t_3 = 0.$$

Again, the broadcaster is the person whose t-bit is equal to 1.

CASE ANALYSIS CONT'D

Is that not bad news? *the broadcaster is the person whose t -bit is equal to 1.*

Well P_2 knows b_1 and b_2 , so know when Case I occurs. But P_2 does not know b_3 which is good news, so P_2 cannot figure out if we are in Case I.1 or I.2. Thus, in Case I, P_2 cannot determine if P_1 or P_3 is the broadcaster.

CASE ANALYSIS CONT'D - CASE 2.1

Case 2: $b_1 \neq b_2$. Again, this case has two subcases depending on the value of b_3

Case 2.1: $b_3 = b_1$ and, thus, $b_3 \neq b_2$.

If P_1 is the broadcaster, then

$$t_1 = s_1 + 1 = b_1 + b_3 + 1 = 1$$

and

$$t_3 = s_3 = b_2 + b_3 = 1.$$

CASE ANALYSIS CONT'D - CASE 2.1

If P_3 is the broadcaster, then

$$t_1 = s_1 = b_1 + b_3 = 0$$

and

$$t_3 = s_3 + I = b_2 + b_3 + I = 0.$$

So, t_1 is always equal to t_3 , no matter whether P_1 or P_3 is the broadcaster.

CASE ANALYSIS CONT'D - CASE 2.2

Recall Case 2: $b_1 \neq b_2$. Again, this case has two subcases depending on the value of b_3

Case 2.2: $b_3 \neq b_1$ and, thus, $b_3 = b_2$.

analogously, to Case 2.1, we can show that in this case t_1 and t_3 are identical no matter if P_1 or P_3 is the broadcaster but their values are flipped

if P_1 is broadcaster, t_1 and t_3 are 0 if P_3 is broadcaster they are equal to 1.

P_2 knows b_1 and b_2 , so she knows if $b_1 \neq b_2$. But, she does not know b_3 which means she does not know if we are in subcase 2.1 or 2.2.

CASE ANALYSIS CONT'D

So far, we assumed that the bit, b , to be communicated is equal to 1.

In either Case 1 or 2, the process is anonymous.

So, what happens if the bit is equal to 0?

If P_i is the broadcaster, she sets $t_i = (s_i + b) \bmod 2$.

(Thus, if $b = 1$, then P_i “secretly” flips the bit s_i and stores the result in t_i .)

No secret bit flipping when $b=0$.

CASE ANALYSIS CONT'D

In that case, everyone uses $t_i = s_i$

Since $t_1=s_1$, $t_2=s_2$ and $t_3=s_3$ and each person P_i shows her bit t_i to the other two people. P_2 knows bit b_3

But she has no way to determine whether P_1 or P_3 is the broadcaster.

—

How about longer messages than one bit. You repeat the ENTIRE procedure for every bit of the message! B.t.w., exactly one person must be the broadcaster.