cse581
Computer Science Fundamentals: Theory

Professor Anita Wasilewska

CM - Lecture 1
Chapter 1: Recurrent Problems - Tower of Hanoi, Josephus Problem, Binary Expansion Solution

Concrete and Discrete Mathematics

The Concrete Mathematics book **B3** was **written**
as an antidote to what authors call an Abstract Mathematics

The Abstract Mathematics is is now called
Discrete Mathematics and was developed as a part of building
the **Foundations of Mathematics**

Both Concrete and Discrete Mathematics play crucial role in
building the **Foundations of Computer Science**

Concrete and Discrete Mathematics

The classical Discrete Mathematics approach includes development of such mathematics fields as Set Theory, Model Theory, Theory of Boolean Algebras, as well as Classical and Non-classical Logics, Number Theory or Graph Theory and many others

We introduce some basic notions of the classical Discrete Mathematics in our Lectures as and when **needed**

What is Concrete Mathematic?
Book Definition

Concrete Mathematics is a controlled **manipulation** of some mathematical formulas **using** a collection of techniques developed for solving problems

We will **learn** various techniques to evaluate horrendously looking finite sums, to solve complex recurrences, and specific manipulations methods for certain classes of them

The. original text of the book was an extension of the chapter "Mathematical Preliminaries" of **Knuth's** classic book "Art of Computer Programming"

Concrete and Discrete Mathematics

Concrete Mathematics is supposed (and hopefully will) to **help** you in the art of writing programs

Discrete Mathematics is supposed to help you to **think** about the art and correctness of programming

B3 - CHAPTER 1
Recurrent Problems

Tower of Hanoi

Josephus Problem

Recurrent Problems in General

We follow the following steps

Abstraction: find a mathematical model for a problem

Recursion: find a recurrent formula describing the problem

Closed Form Formula: find it for a given recurrent one

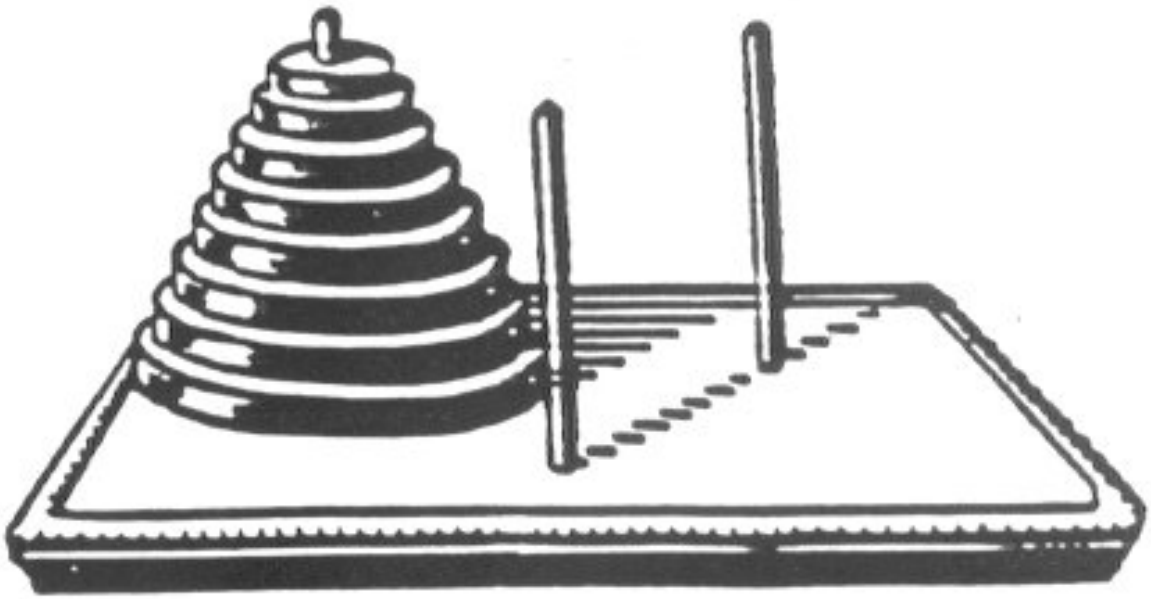
(if exists) and **prove** their equivalency

B3 - CHAPTER 1
PART ONE: Tower of Hanoi

The Tower of Hanoi

Tower of Hanoi **puzzle** is attributed to the French mathematician Edouard Lucas, who came up with it in 1883

His formulation involved three pegs and eight distinctly-sized disks stacked on one of the pegs from the biggest on the **bottom** to the smallest on the **top**



The GOAL

The puzzle goal is to move the stack of disks to one of the other pegs with the following rule:

L - rule

must move one disk at a time

a larger disk cannot be on top of any smaller disks at any time

do it in as few moves as possible

Lucas furnished his puzzle with a romantic legend about Tower of Brahma (64 disks) with monks, gold, diamond needles etc...

The Tower of Hanoi GENERALIZED

Tower has now *n* disks, all stacked in decreasing order from bottom to top on one of three pegs,

Question

what is the minimum number of (legal) **moves** needed to move the stack to one of the other pegs?

Plan

1. we **start** by expressing the minimum number of moves required to move a stack of *n* disks as a **recurrence** relation, i.e. we **find** and **prove** a recursive (recurrent) formula

2. we **find** a closed-form formula for the number of moves required;

3. we **prove** that the closed-form and recurrent formulas are equivalent

The Tower of Hanoi GENERALIZED to n disks

We denote by

T_n - the minimum number of moves that will transfer n disks from one peg to another under the

L - rule:

must move one disk at a time;

a larger disk cannot be on top of any smaller disks at any time

do it in as few moves as possible

$n = 1$ - we have 1 disk- and 1 move, i.e. $T_1 = 1$

$n = 2$ - we have 2 disks- and 3 moves: top (smaller) disk from peg 1 to peg 2, remaining (larger) disk from peg 1 to peg 3, the disk from peg 2 (smaller) on the top of the disk (larger) on peg 3 so L - rule holds and hence $T_2 = 3$

A Strategy for $n = 3$ disks

1. transfer top 2 disks as in previous case for $n = 2$ – we use T_2 moves;
2. move remaining (largest) disk to **empty peg** – we use 1 move;
3. bring the 2 disks to the top of the largest disk as in previous case for $n = 2$ – we use T_2 moves;

together we have

$$T_2 + T_2 + 1 = 3 + 3 + 1 = 7 \text{ moves}$$

Recurrent Strategy to evaluate T_n

1. In order to **move** the bottom disk, we need to **move all** the $n - 1$ disks above it to a empty peg first
2. Then we can **move** the bottom disk to the remaining empty peg, and
3. **move** the $n - 1$ smaller disks back on top of it

Recurrent Strategy to evaluate T_n

1. we **move** all the $n-1$ disks above bottom disk to a different (empty) peg - we do it in T_{n-1} moves;
2. we **move** the bottom disk to the remaining empty peg - we do it in 1 moves
3. we **move** $n-1$ disks from peg resulting in 1. to the peg resulting in 2. - another T_{n-1} moves;

How many moves? together we have at most
$T_{n-1} + T_{n-1} + 1 = 2T_{n-1} + 1$ moves i.e we have that

$$T_n \leq 2T_{n-1} + 1, \text{ where } n \geq 1$$

Recursive Formula for T_n

We have proven that $T_n \leq 2T_{n-1} + 1$.

We show (next slide) that **there is no better way**, i.e. that

$$T_n \geq 2T_{n-1} + 1$$

and hence we get the Recursive Formula that gives us the solution for the minimum number of moves T_n required to move a tower with n disks to another peg.

$$T_n = \begin{cases} 0\,, & \text{if } n = 0; \\ 2T_{n-1} + 1\,, & \text{if } n > 0. \end{cases}$$

Recursive Formula for T_n - end of the proof

Observe that in order to move the largest bottom disk anywhere, we have to first get the $n-1$ smaller disks on top of it onto **one** of the other pegs

This will take **at least** T_{n-1} moves

Once this is done, we have to **move** the bottom disk **at least once;** we may move it more than once!

After we're **done** moving the bottom disk, we have to move the $n-1$ other disks back on top of it eventually, which will take again **at least** T_{n-1} moves;

all together we get that $T_n \geq 2T_{n-1}+1$ and hence we **proved** our Recursive Formula

$$T_n = \begin{cases} 0\,, & \text{if } n = 0; \\ 2T_{n-1}+1\,, & \text{if } n > 0. \end{cases}$$

From Recursive Formula to Closed Form Formula

Often the problem with a recurrent solution is in its computational complexity;

Observe that for any recursive formula R_n, in order to calculate its value for a certain n one needs to calculate (recursively) all values for R_k, $k = 1, \ldots, n-1$.

It's easy to see that for large n, this can be quite complex.

So **we would like to find** (if possible) a **non- recursive function** with a formula **f = f(n)**,

Such formula is called a **Closed Form Formula**

Provided that the **Closed Form Formula** computes **the same function** as our original recursive one.

From Recursive Formula to Closed Form Formula

We will **examine classes** of Recursive Formula functions for which the it is **possible to find** corresponding **equivalent** Closed Form Formula function

Of course we have always to **prove** that Recursive Formula functions and Closed Form Formula functions we have found are **equal,** i.e. their corresponding **formulas are equivalent**.

Definition of Equality of Functions

Given two functions f and g such that

$$f: \ A \longrightarrow B \quad \text{and} \quad g: \ A \longrightarrow B$$

we say that f and g are **equal**, or their formulas are equivalent and write symbolically as

$f = g$ if and only if $f(a) = g(a)$, for all $a \in A$, i.e.

$$\forall_{a \in A} \ f(a) = g(a)$$

Proving Equality of Functions

Observe that when the domain of f and g are natural numbers N (or a subset of N), i.e.

$$f : \; N \longrightarrow \; B \quad \text{and} \quad g : \; N \longrightarrow \; B$$

then **proving** that they are **equal**, or their formulas are **equivalent** means proving that

$$\forall_{n \in N} \, f(n) = g(n)$$

We usually carry such proofs by Mathematical Induction over the common domain of both functions

Back to Tower of Hanoi

We proved that the solution for the **Tower of Hanoi** is given by a **Recursive Formula**

$$T_n = \begin{cases} 0\,, & \text{if } n = 0; \\ 2T_{n-1} + 1\,, & \text{if } n > 0. \end{cases}$$

Mathematically it means that we have defined a function

$$T: \quad N \longrightarrow \quad N$$

such that

$$T(0) = 0, \quad T(n) = 2T(n-1) + 1, \text{ for all } n > 0$$

From Recursive Formula to Closed Form Formula

For functions with natural numbers N as the domain we use, as in a case of any sequences a notation $T(n) = T_n$

We write our recursively defined function $T : N \longrightarrow N$

$$T(0) = 0, \quad T(n) = 2T(n-1) + 1, \text{ for all } n > 0$$

as

$$T_0 = 0, \quad T_n = 2T_{n-1} + 1, \text{ for all } n > 0$$

and call it, for short a **recursive formula**

Our **goal** now is to **find** a **Closed Form Formula** equivalent to the obove **recursive formula**

One way to get such a solution is to first come up with a **guess,** and then prove that the **guess** is in fact a **correct solution**

From Recursive Formula to Closed Form Formula

Given our **Recursive Formula**

$$RF: \quad T_0 = 0, \quad T_n = 2T_{n-1} + 1, \text{ for } n > 0$$

We evaluate few values for T_n:
$T_0 = 0,\ T_1 = 1,\ T_2 = 3,\ T_3 = 7,\ T_4 = 15,\ T_5 = 31,\ T_6 = 63, \ldots$
It is easy to observe that values of T_n follows the pattern

$$T_n = 2^n - 1, \text{ for all } n \geq 0$$

We hence **guess** that $T_n = 2^n - 1$ is a Closed Form Formula CF equivalent to our Recursive Formula RF .

Proving RF = CF

We use, after the book, that same ”name” (in this case T_n) for both functions representing Recursive Formula RF and Closed Form Formula CF.

We distinguish them here and in the future investigations by using different colors and notation: RF and CF, respectively.

As both functions has the natural numbers N as their common domain, we carry the proof here (and in the future investigations) by Mathematical Induction over the domain of the functions (always a subset of N).

Proof of RF = CF for Tower of Hanoi Solution

RF: $T_0 = 0, \quad T_n = 2T_{n-1} + 1, \quad n > 0$

CF: $T_n = 2^n - 1, \quad n \geq 0$

We prove by Mathematical Induction that RF = CF, i.e. that

$$\forall_{n \in N} \; T_n = T_n = 2^n - 1$$

Base Case $n = 0$

We verify: $T_0 = 0, \quad T_0 = 2^0 - 1 = 0$ and we get that Base Case is true: $T_0 = T_0$

Proof of RF = CF for Tower of Hanoi Solution

RF: $T_0 = 0, \quad T_n = 2T_{n-1} + 1, \quad n > 0$

CF: $T_n = 2^n - 1, \quad n \geq 0$

Inductive Assumption: $T_{n-1} = T_{n-1} = 2^{n-1} - 1$

Inductive Thesis: $T_n = T_n = 2^n - 1$

Proof:

$$\begin{aligned} T_k &=^{def} 2T_{k-1} + 1 \\ &=^{ind} 2(2^{k-1} - 1) + 1 \\ &= 2^k - 2 + 1 \\ &= 2^k - 1 = T_k \end{aligned}$$

Another Proof of RF = CF for Tower of Hanoi Solution

Here is an interesting way to find a closed-form solution without having to guess that the solution is $T_n = 2^n - 1$. Consider what happens when we add 1 to the recursive formula RF

$$T_n + 1 = \begin{cases} 1\,, & \text{if } n = 0; \\ 2T_{n-1} + 2 = 2(T_{n-1} + 1)\,, & \text{if } n > 0. \end{cases}$$

Now, letting $U_n = T_n + 1$, we get the following recurrence:

$$U_n = \begin{cases} 1\,, & \text{if } n = 0; \\ 2U_{n-1}\,, & \text{if } n > 0. \end{cases}$$

It's pretty easy (in any case easier than for the T_n) to see that the solution (proof by Mathematical Induction) to this recurrence is $U_n = 2^n$. Since $U_n = T_n + 1$, we get

$$T_n = U_n - 1 = 2^n - 1.$$

Book3 - CHAPTER 1
PART TWO: The Josephus Problem

Josephus Story

Flavius Josephus was a historian of 1st century

During Jewish-Roman war Josephus was among 41 Jewish rebels captured by the Romans

They preferred **suicide** to the **capture** and decided to form a circle and to **kill** every third person until **no one** was left

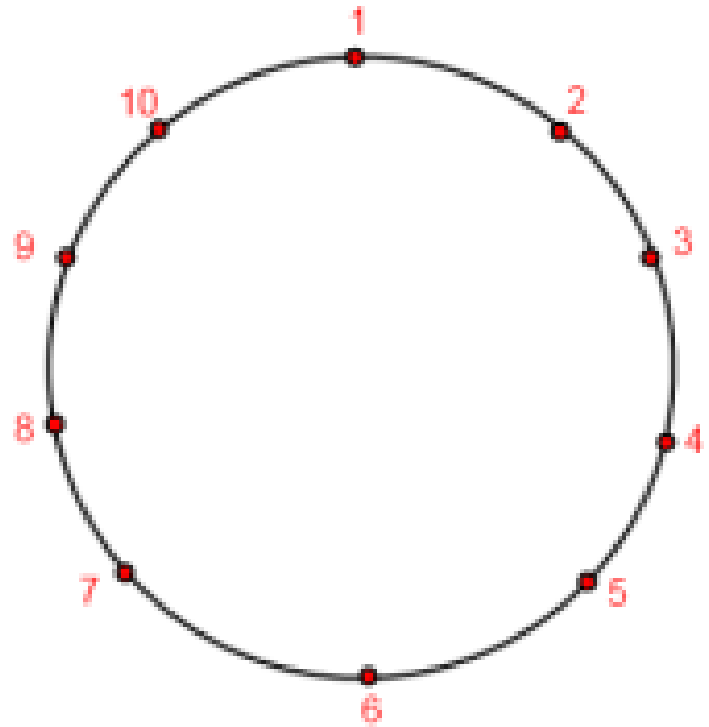
Josephus with with one friend wanted none of this **suicide** nonsense and he calculated **where** he and his friend should **stand** to avoid being killed and they were **saved**

The Josephus Problem - Our variation

n people around the **circle** and we **eliminate** each second remaining person **until** one survives

We denote by J(n) the **position** of a surviver

Example n = 10



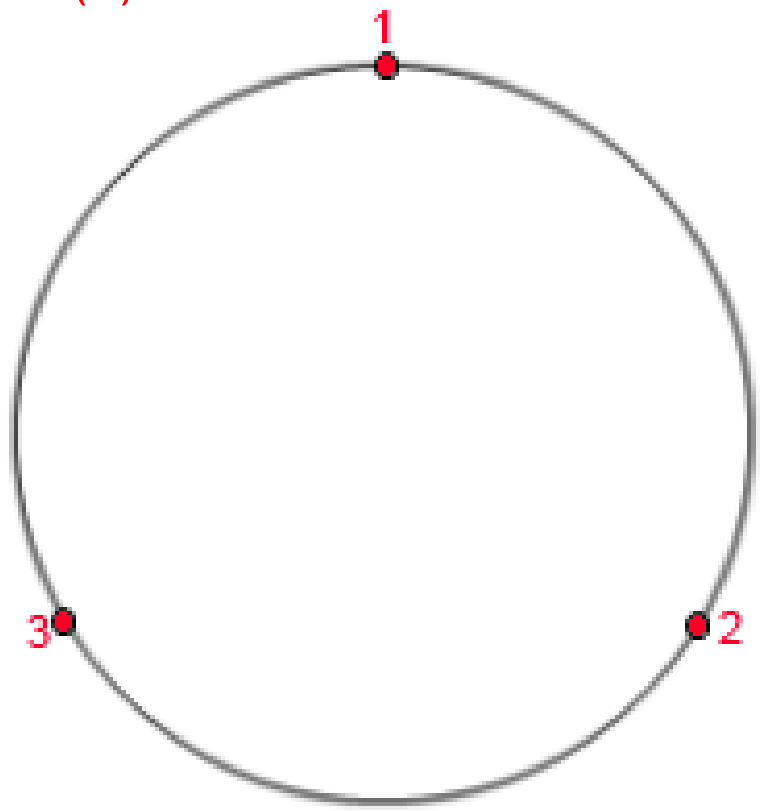
Elimination order: 2, 4, 6, 8, 10, 3, 7, 1, 9.

As a result, number 5 survives, i.e. J(10) = 5

Problem: Determine survivor number J(n)

We **evaluate** now J(n) for n=1,2, . . . 6

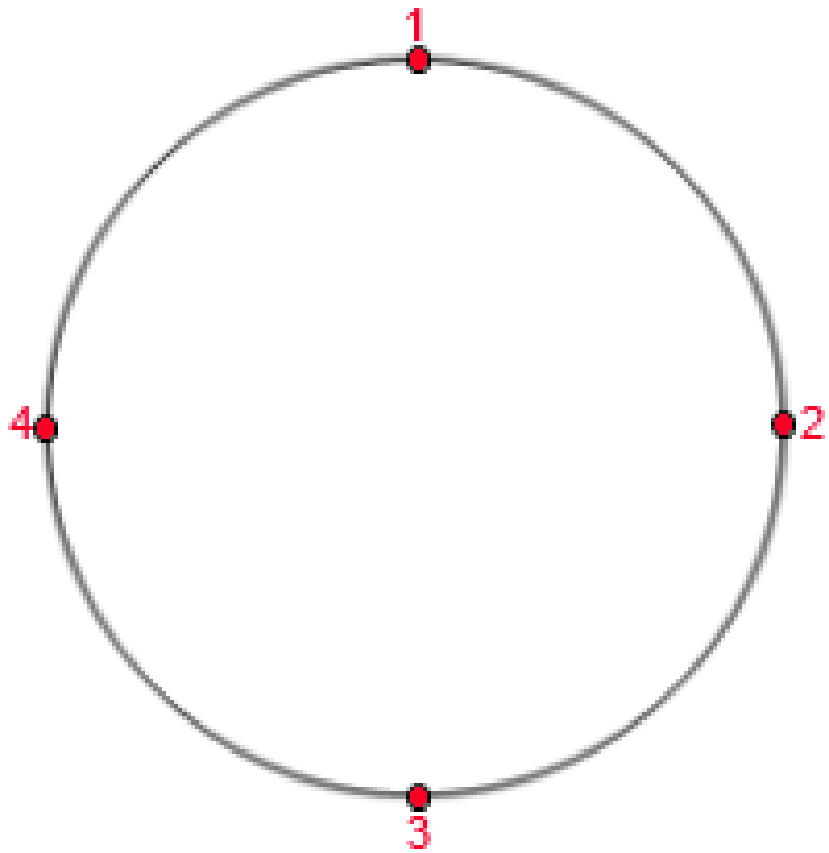
J(1)=1, J(2) = 1, J(3):



We get that J(3)=3

Determine survivor number J(n)

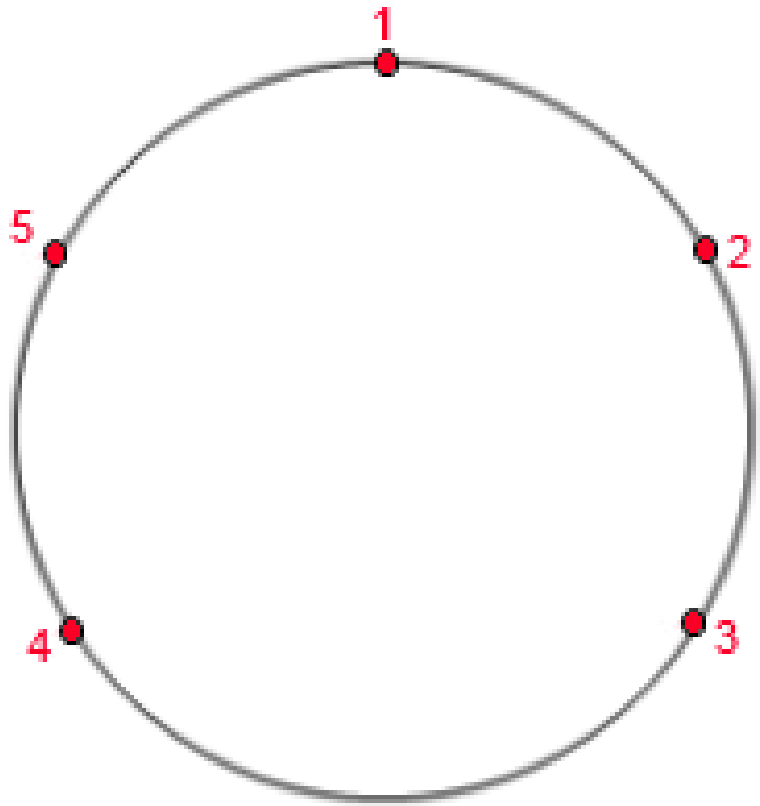
Picture for J(4):



We get J(4)=1

Determine survivor number J(n)

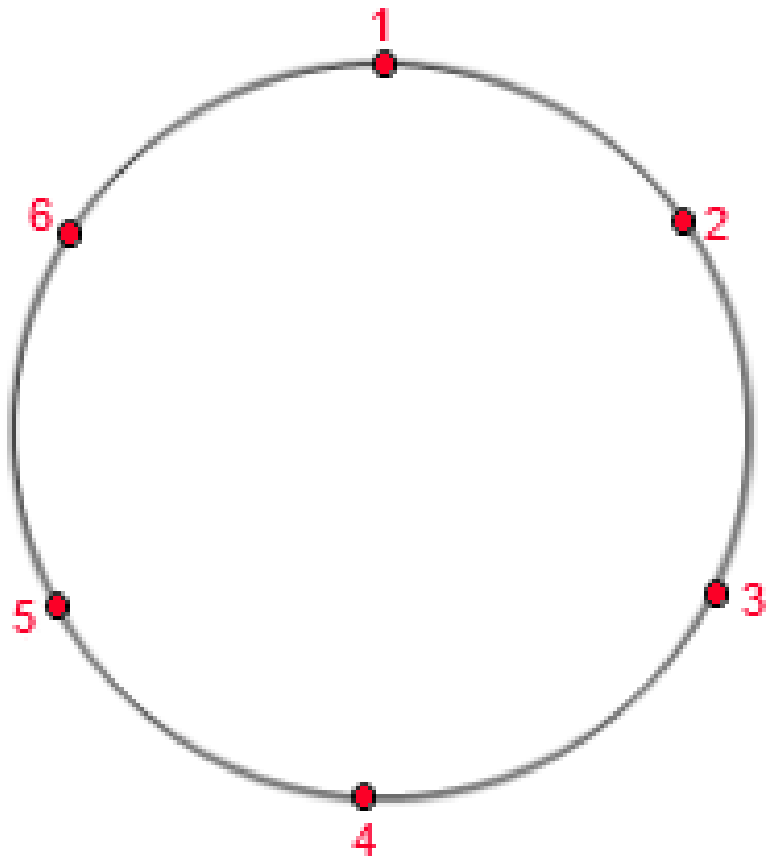
Picture for J(5):



We get J(5)=3

Problem: Determine survivor number J(n)

Picture for J(6):



We get J(6)=5

Determine survivor number J(n)

We put our results in a table:

n	1	2	3	4	5	6
J(n)	1	1	3	1	3	5

Observation

All our J(n) after the **first run** are odd numbers

Fact

First trip **eliminates** all even numbers

Determine survivor number J(n)

Fact

First trip **eliminates** all even numbers

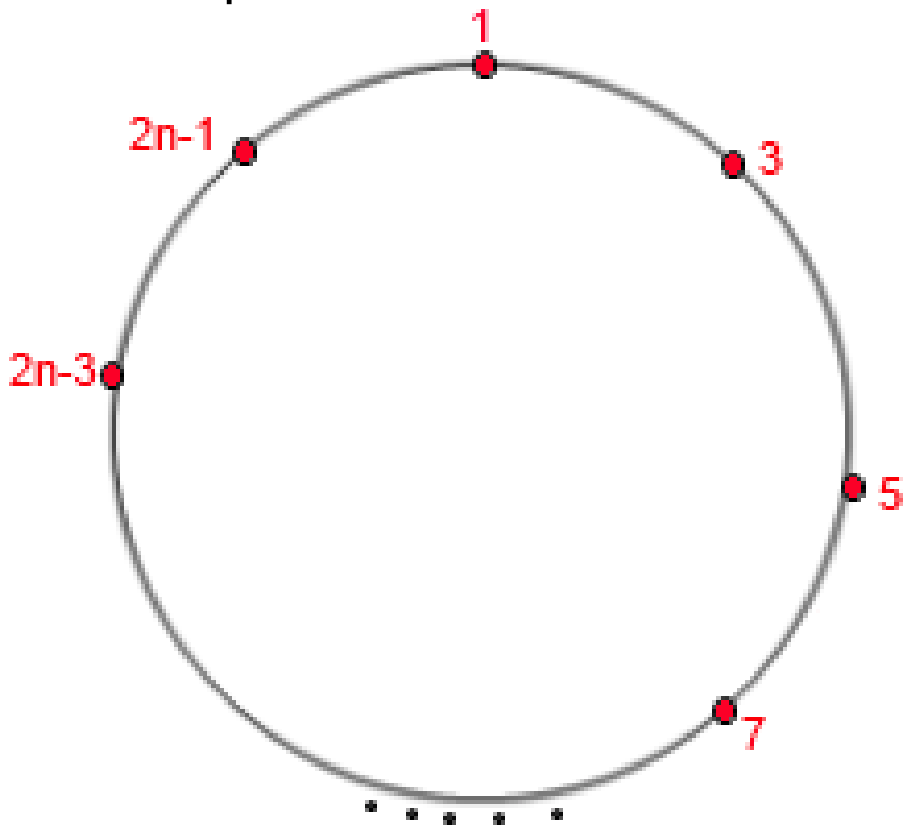
Observation

If $n \in$ EVEN we arrive to a similar situation we started with with half as many people (numbering has changed)

Determine survivor number J(n)

ASSUME that we START with 2n people

After first trip we have



3 goes out next

This is like starting with n except **each** person has been doubled and decreased by 1

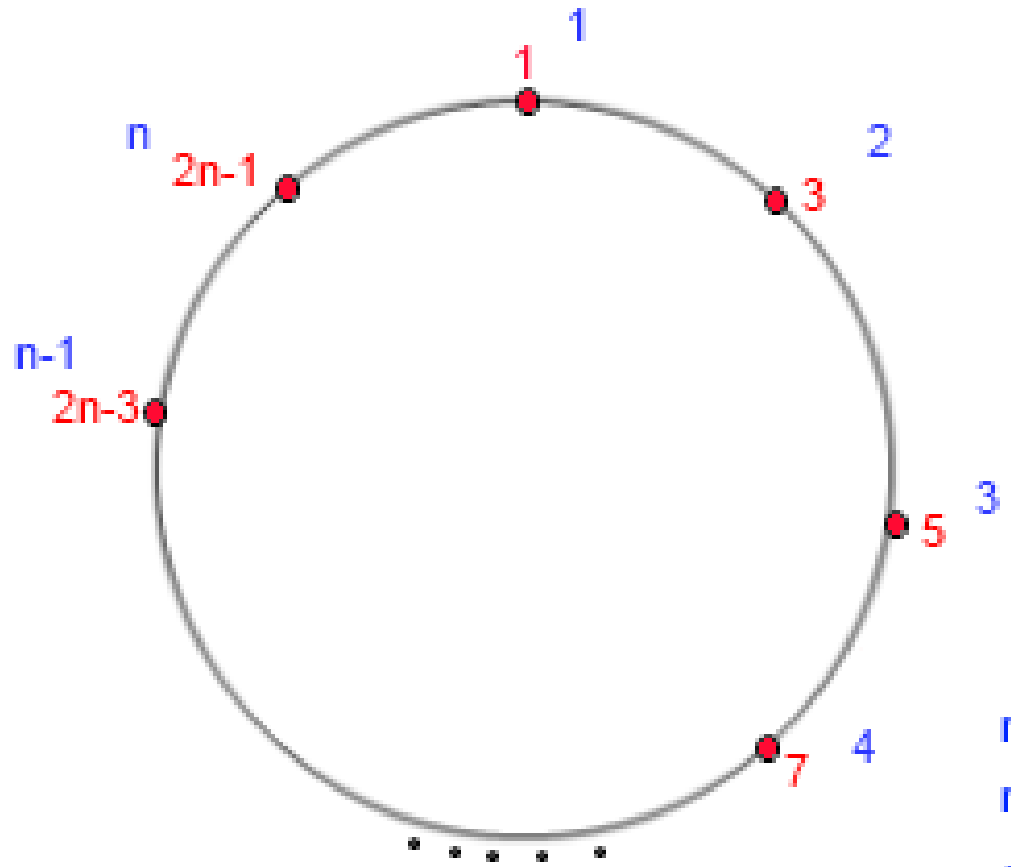
Determine survivor number J(n)

Case n=2n

We get **J(2n)=2J(n) -1** (each person has been doubled and decreased by 1)

We know that J(10)=5, so J(20) = 2J(10)-1 = 2*5-1 = 9

Re-numbering



1= 2*1-1

#2

3=2*2- 1

3=new # 2

#3

5=2*3-1

5=new #3

#4

7=2*4-1

new #2 = 2*2-1=3

new #3 = 2*3-1=5

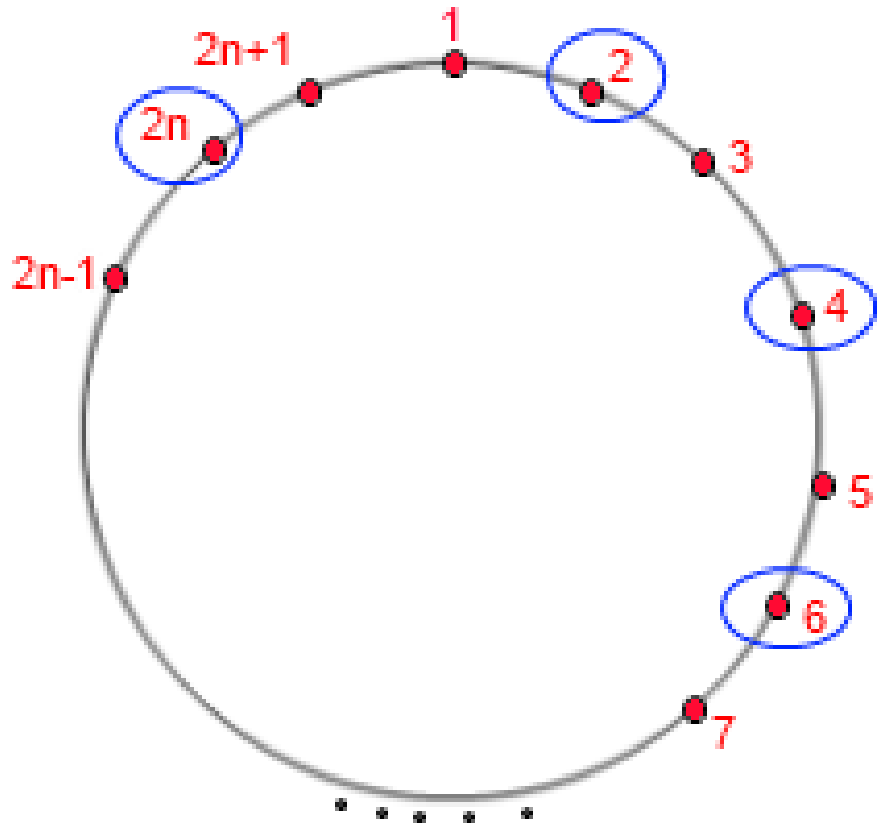
new #4 = 3*4-1=7

Determine survivor number J(n)

Case n=2n+1

ASSUME that we start with 2n+1 people:

First looks like that

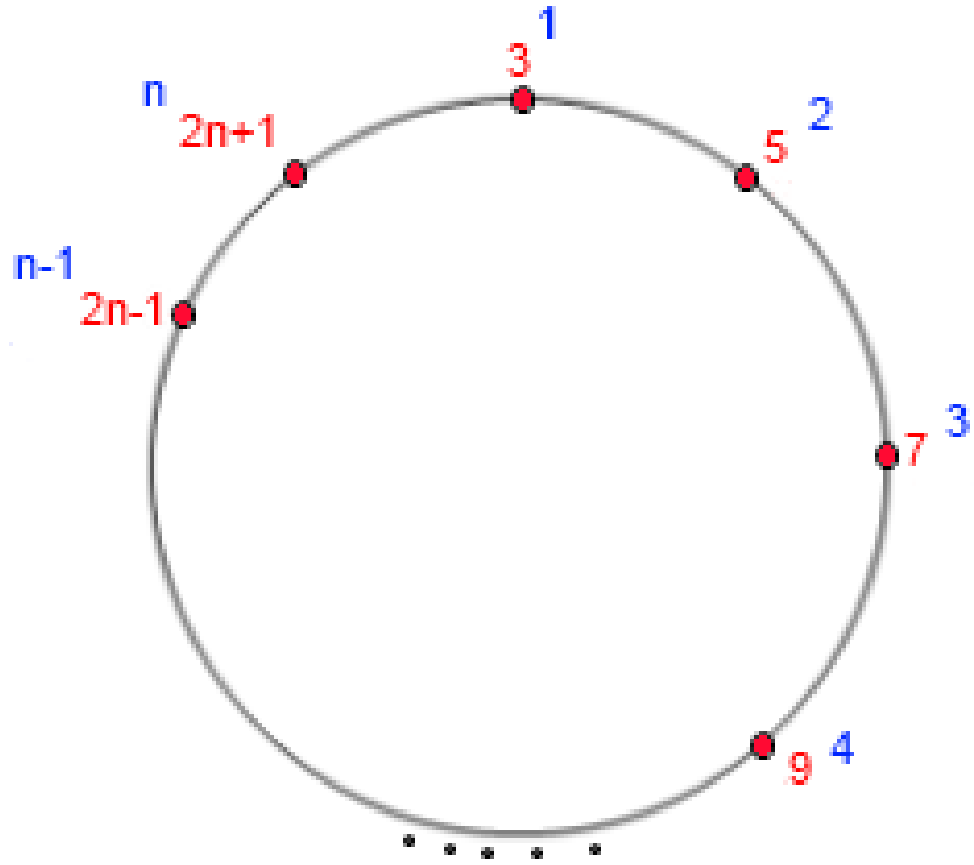


1 is wipped out after 2n

We want to have n-elements after **first** round

Determine survivor number J(n)

After the first trip we have

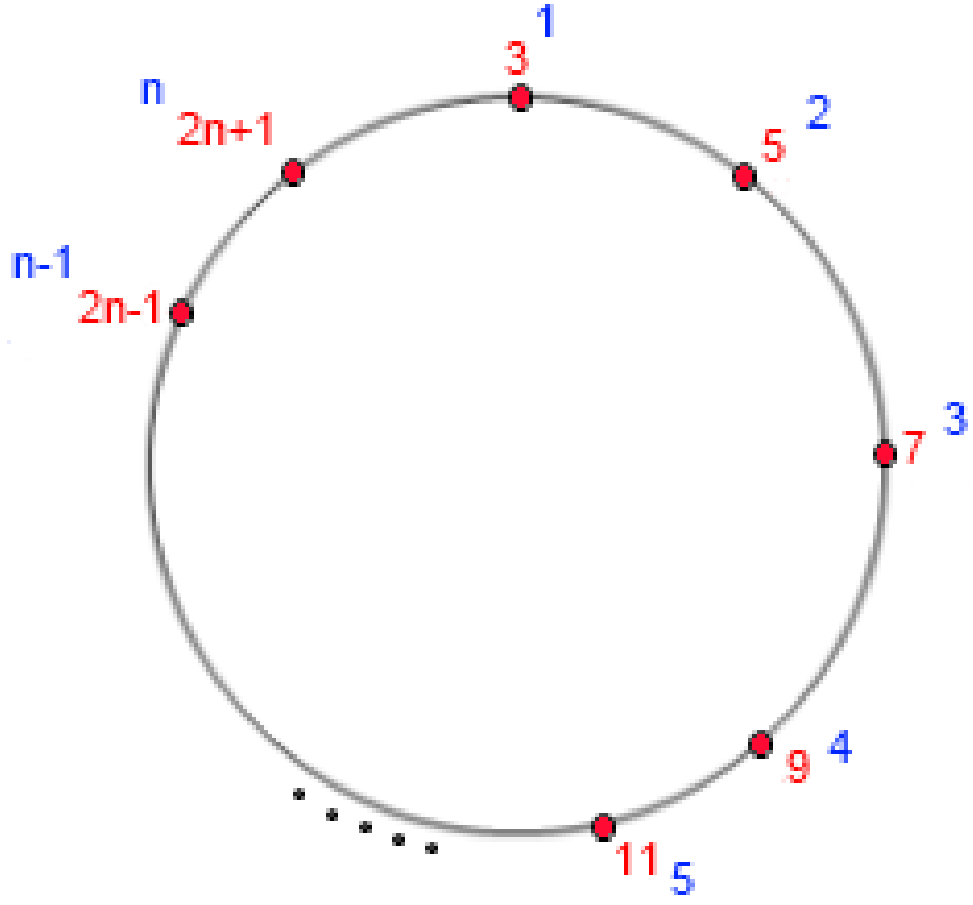


This is like starting with n except that now each person is doubled and increased by 1

Determine survivor number J(n)

CASE n=2n + 1 c.d.

Re-numbering



3=2*1-1
5=2*2+1
7=2*3+1

3=new #1 3=2*1+1
5=new #2 5=2*2+1

new 1 =2*1+1=3 → Old
new 2=2*2+1=5 → Old
new 3=2*3+1=7 → Old

Formula: new number k = 2k+1

J(2n+1)=new number J(n)

J(2n+1)=2J(n)+1

Recurrence Formula for J(n)

The Recurrence Formula RF for J(n) is:

$J(1) = 1$

$J(2n) = 2J(n) - 1$

$J(2n+1) = 2J(n) + 1$

Remember that J(k) is a position of the **survivor**

This formula is more efficient then getting F(n) from F(n-1)

It reduces n by factor 2 each time it is applied

We need only 19 application to evaluate $J(10^6)$

From Recursive Formula to Closed Form Formula

In order to find a **Closed Form Formula** (CF) equivalent to given **Recursive Formula** RF we ALWAYS follow the the Steps 1 - 4 listed below.

- Step 1 Compute from recurrence RF a TABLE for some initial values. In our case RF is: $J(1) = 1,\ J(2n) = 2J(n) - 1,\ J(2n+1) = 2J(n) + 1$
- Step 2 **Look** for a pattern formed by the values in the TABLE
- Step 3 **Find** - guess a closed form formula CF for the pattern
- Step 4 **Prove** by Mathematical Induction that RF = CF

TABLE FOR J(n)

n	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
J(n)	1	1	3	1	3	5	7	1	3	5	7	9	11	13	15	1
	G1	G2		G3				G4								G5

Observation: $J(n) = 1$ for $n = 2^k, \quad k = 0, 1, ..$

Next step: we form groups of J(n) for n consecutive powers of 2 and observe that

J(n)	G1	G2	G3	G4	G5	...
n	2^0	$2^1 + l$	$2^2 + l$	$2^3 + l$	$2^4 + l$	...

for $0 \leq l < 2^{(k-1)}$ and $k = 1, 2, ...5,$

Computation of J(n)

Observe that for each **group** G_k the corresponding n are $n = 2^{k-1} + l$ for all $0 \leq l < 2^{(k-1)}$ and the value of $J(n)$ for $n = 2^k + l$ i.e. $J(n) = J(2^k + l)$ **increases** by 2 within the **group**

Let's now make a TABLE for the group G3

J(n)	1	3	5	7 = 2l+1
n	2^2	$2^2 + l$	$2^2 + 2$	$2^2 + 3$
	l=0	l=1	l=2	l=3

Guess for CF formula for J(n)

Given $n = 2^{k-1} + l$ we **observed** that $J(n) = 2l + 1$

We **guess** that our CF formula is

$$J(2^k + l) = 2l + 1,$$

for any $k \geq 0$, $0 \leq l < 2^k$

Representation of n

$n = 2^k + l$ is called a **representation** of n when
l is a **remainder** by dividing n by 2^k and
k is the largest power of 2 not exceeding n

Observe that $2^k \leq n < 2^{k+1}$, $l = n - 2^k$ and so
$0 \leq l < 2^{k+1} - 2^k = 2^m$, i.e.

$$0 \leq l < 2^k$$

Proof RF = CF

RF: $J(1) = 1, J(2n) = 2J(n) - 1, J(2n+1) = 2J(n) + 1$

CF: $J(2^k + l) = 2l + 1$, for $n = 2^k + l, \quad k \geq 0,\ 0 \leq l < 2^k$

Proof: by Mathematical Induction on k

Base Case: k=0.

Observe that $0 \geq l < 2^0 = 1$, and l = 0 , $n = 2^0 + 0 = 1$, i.e. $n = 1$.

We evaluate $J(1) = 1$, $J(2^0) = 1$, i.e.

$$RF = CF$$

Proof RF = CF

Induction Step over k has two cases

c1: **n** $\in$ **even** and $J(2n) = 2J(n) - 1$

c2: **n** $\in$ **odd** and $J(2n+1) = 2J(n) + 1$

Induction Assumption for k is

$J(2^{k-1} + l) = 2l + 1$, for $0 \leq l < 2^{k-1}$

case c1: **n** $\in$ **even**

put n:= 2n, i.e. $2^k + l = 2n$, $0 \leq l < 2^k$

Observe that

$2^k + l = 2n$ iff $l \in$ even, i.e. $l = 2m$, and

l/2 = m $\in N$ and $0 \leq \frac{l}{2} < 2^{k-1}$.

Proof RF = CF

We evaluate n from $2^k + l = 2n$ as follows

$n = \frac{2^k + l}{2}$,

$n = 2^{k-1} + \frac{l}{2}$, for $0 \leq \frac{l}{2} < 2^{k-1}$, $\frac{l}{2} \in N$

Proof in case **c1: n** $\in$ **even** and $J(2n) = 2J(n) - 1$

Reminder: CF: $J(2^k + l) = 2l + 1$ for $n = 2^k + l$

$J(2^k + l) =^{reprn} 2J(2^{k-1} + \frac{l}{2}) - 1$

$=^{ind} 2(2\frac{l}{2} + 1) - 1 = 2l + 2 - 1$

$= 2l + 1$

Proof RF = CF

Proof in case **c2:** **n** $\in$ **odd** and $J(2n+1) = 2J(n)+1$

Inductive Assumption: $J(2^{k-1}+l) = 2l+1$, for

$0 \leq l < 2^{k-1}$

Inductive Thesis: $J(2^k+l) = 2l+1$, for $0 \leq l < 2^k$

We put $n := 2n+1$ and observe that

$2^k + l = 2n+1$ iff $l \in$ odd, i.e.

l = 2m+1, for certain $m \in N$, l -1 = 2m, and $\frac{l-1}{2} = m \in N$

Proof of RF = CF

Let $J(2n+1) = 2J(n) + 1$

We evaluate, as before n from $2^k + l = 2n + 1$

$2^k + l - 1 = 2n$ and we get the representation of n

$n = 2^{k-1} + \frac{l-1}{2}$

Reminder: CF: $J(2^k + l) = 2l + 1$ for $n = 2^k + l$

Proof RF = CF in case **c2: n ∈ odd** and

$J(2n+1) = 2J(n) + 1$ is now as follows

$J(2^k + l) =^{reprn} 2J(2^{k-1} + \frac{l-1}{2}) + 1$

$=^{ind} 2(2\frac{l-1}{2} + 1) + 1 = 2(l - 1 + 1) + 1$

$= 2l + 1$

Some Facts

Fact 1 $\forall_m\ J(2^m) = 1$

Proof by induction over m

Observe that $2^m \in even$, so we use the formula

$J(2n) = 2J(n) - 1$, and get

$J(2^m) = J(2 * 2^{m-1}) =^{Jdef} 2J(2^{m-1}) - 1 =^{ind} 2 * 1 - 1 = 1$

Hence we also have

Fact 2

First person will always **survive** whenever n is a power of 2

General Case

Fact 3

Let $n = 2^m + l$

The first remaining person, the **survivor** is number $2l + 1$

Our solution for the **proof**

Observe that the number of people is **reduced** to power of 2 after there have been l **executions**

$$J(2^m + l) = 2l + 1$$

where $n = 2^m + l$ and $0 \leq l < 2^m$ depends heavily on powers of 2

Let's look now at the binary expansion of n and see how we can **simplify** the computations

Binary Expansion of n

Definition

$$n = (b_m b_{m-1} ... b_1 b_0)_2$$

stands for

$$n = b_m 2^m + b_{m-1} 2^{m-1} + ..b_1 2 + b_0$$

for

$$b_i \in 0, 1, \quad b_m = 1$$

Binary Expansion of n

EXAMPLE: n=100

$n = (1\ 1\ 0\ 0\ 1\ 0\ 0)_2$

$2^6 2^5 2^4 2^3 2^2 2^1 2^0$

$n = 2^6 + 2^5 + 2^2 = 64 + 32 + 4 + 100$

Binary Expansion of n

Let now :

$n = 2^m + l, \quad 0 \leq l < 2^m$

we have the following binary expansions:

1) $l = (0, b_{m-1}, .., b_1, b_0)_2$ as $l < 2^m$

2) $2l = (b_{m-1}, .., b_1, b_0, 0)_2$ as

$$l = b_{m-1}2^{m-1} + .. + b_1 2 + b_0$$
$$2l = b_{m-1}2^m + .. + b_1 2^2 + b_0 2 + 0$$

3) $2^m = (1, 0, ..., 0)_2, \quad 1 = (0...1)_2$

4) $n = 2^m + l$

$n = (1, b_{m-1}, .., b_1, b_0)_2$ from 1 + 3

5) $2l + 1 = (b_{m-1}, b_{m-2}, .., b_0, 1)_2$ from 2 + 3

Binary Expansion Josephus

Consider now a **closed formula**

CF : $J(n) = 2l + 1$, for $n = 2^m + l$

We use

5) $2l + 1 = (b_{m-1}, b_{m-2}, .., b_0, 1)_2$

and re-write the **closed** formula CF as a **binary expansion** formula BF as follows

$BF : \quad J((b_m, b_{m-1}, .., b_1, b_0)_2) = (b_{m-1}, .., b_1, b_0, b_m)_2$

because $b_m = 1$ in the binary expansion of n, we get

$$BF : \quad J((1, b_{m-1}, .., b_1, b_0)_2) = (b_{m-1}, .., b_1, b_0, 1)_2$$

Binary Expansion Josephus

Example: Find J(100)

$n = 100 = (1100100)_2$

$$J(100) = J((1100100)_2) =^{BF} (1001001)_2$$

J(100) = 64 + 8 + 1 = 73

$$BF: \quad J((1, b_{m-1}, .., b_1, b_0)_2) = ((b_{m-1}, .., b_1, b_0, 1)_2$$

Josephus Generalization

Our function $J : N - \{0\} \longrightarrow N$ is defined as

$J(1) = 1, \quad J(2n) = 2J(n) - 1, \quad J(2n+1) = 2J(n) + 1$ for $n > 1$

We generalize it to function $f : N - \{0\} \longrightarrow N$ defined as follows

$$f(1) = \alpha$$

$$f(2n) = 2f(n) + \beta, \qquad n \geq 1$$

$$f(2n+1) = 2f(n) + \gamma, \qquad n \geq 1$$

Observe that J = f for $\alpha = 1, \beta = -1, \gamma = 1$

NEXT STEP: Find a **Closed** Formula for f