Extended Josephus Problem

1) Problem Statement:

- A group of ***n*** people, numbered *0* to *n-1*, stands in a circle.
- Starting from the first person (index 0), every ***kth*** person in the circle is eliminated.
- The process continues in a circular manner until only one survivor remains.
- Given the values of *n* (number of people) and *k* (step size for elimination), determine the survivor's position ***J(n, k)*** in the original arrangement.

2) Definitions:

J(n,k) = Survivor's position in a circle of *n* people with every k^{th} person being eliminated.

3) Approach:

1st Approach:

- **Recursive Josephus Problem with Boolean Vector**
- ✓ Initialize a boolean vector person of size n (all 0s, meaning alive).
- ✓ Find the kill position using ***(k - 1) % person_left*** to avoid unnecessary looping.
- ✓ Move to the k^{th} person, skipping eliminated ones *(person[index] == 1).*
- ✓ Mark the person as eliminated (1) and decrement person_left.
- ✓ Find the next alive person and recursively repeat until only one remains.
- ✓ Return the last alive person's index.

- **Implementation**

```
class Solution{
public:
    int winner(vector<bool> &person,int n,int index,int person_left,int k)
    {
        if(person_left==1)
        {
            for(int i=0;i<n;i++)
            if(person[i]==0)
            return i;
        }

        // find the position for kill
        // by taking 'kill' we avoid unnecessay looping in the array
        int kill = (k-1) % person_left;

        while(kill--)
        {
            index = (index+1) % n;
            while(person[index]==1)
            index=(index+1) %n;
        }

        person[index] = 1;

        // next alive person
        while(person[index]==1)
```

```
        index=(index+1) %n;

        return winner(person,n,index,person_left-1,k);
    }

    int findTheWinner(int n, int k) {

        vector<bool> person(n,0);
        return winner(person,n,0,n,k)+1;
    }
};
```

- **Time and Space Complexity Analysis**
  - ✓ Time Complexity:

    Each **recursive call** eliminates one person from the ***n*** people.

    In every call, we **traverse the array** to find the next alive person, which can take up to **O(n)** in the worst case.

Thus, the recurrence relation is:

$$T(n) = T(n-1) + (n) \qquad \text{...(1)}$$

Expanding this recurrence:

$$T(n-1) = T(n-2) + (n-1) \qquad \text{...(2)}$$

Substituting (2) in (1);

$$T(n) = T(n-2) + (n-1) + (n)$$

$$T(n) = T(n-3) + (n-2) + (n-1) + (n)$$

$$T(n) = T(n-k) + (n-(k-1)) + \ldots\ldots + (n-1) + (n) \qquad \text{...(Generalized)}$$

$$T(n) = T(1) + (n-(n-1-1)) + \ldots\ldots + (n-1) + (n) \qquad \text{...}(k = n-1)$$

$$T(n) = C + (2+3+4\ldots\ldots+n)$$

$$T(n) = 1 + 2 + 3 + 4 \ldots\ldots + n \qquad \text{...(let C=1)}$$

$$T(n) = n(n+1)/2 \rightarrow O(n^2)$$

Worst-case time complexity: **$O(n^2)$**

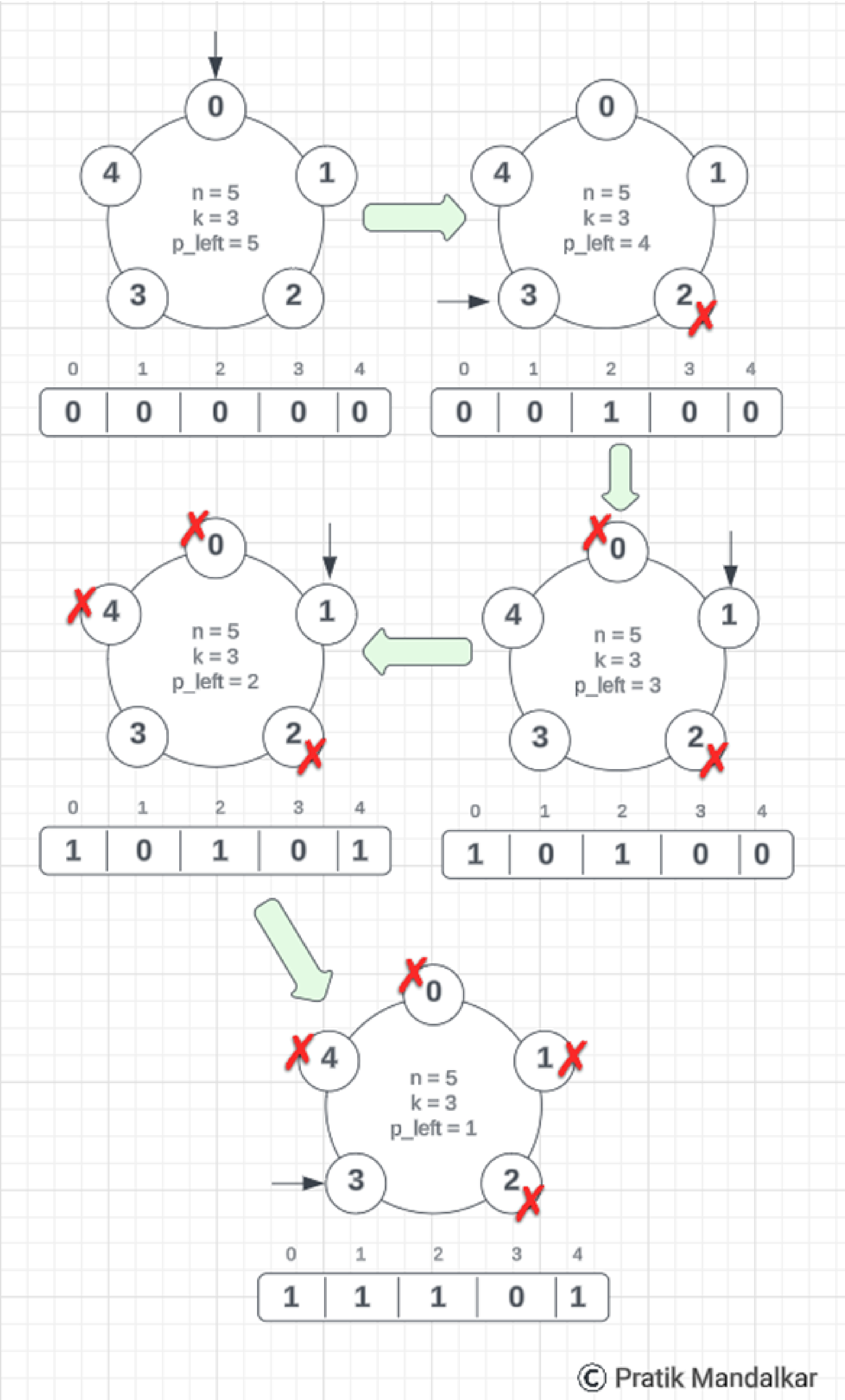
  - ✓ **Space Complexity**

    The boolean **vector** person stores n elements → O(n).

    The recursive **call stack** depth goes up to n in the worst case → O(n).

    Total space complexity: O(n) + O(n) = **O(n).**

- **Dry Run with Visualization**



2st Approach:

- Recursive Josephus Problem with **Mapping Approach**

How the Algorithm Works

The problem follows the **Josephus recurrence**:

$$J(n,k) = (J(n-1,k) + k) \bmod n$$

where:

- ***J(n,k)*** is the position of the survivor in a group of n people.
- ***J(n−1,k)*** is the position of the survivor in a smaller problem (when n-1 people remain).
- The ***+k*** represents skipping k persons for elimination.
- ***%n*** ensures circular movement.
- **Base case:** When only 1 person is left, they are the survivor, i.e.,

$$J(1, k) = 0$$

Why We Use Mapping (Recurrence Shift Explanation)

- ✓ If we observe the pattern, we see that 'n' is decreasing linearly.
- ✓ In n=4, we can't say we have the node values as <0,1,2,3>, what if they are?
- ✓ If we can make the values of present nodes look something like <0 to n-1>, then we will not require the extra array to keep the track of eliminated peoples.
- ✓ For that we need to establish the relationship between the actual node values to the assumed node values
- ✓ Which is achieved by ***{actual values = (assumed value + k) %n}***

- **Implementation**

```
class Solution {
public:
    int winner(int n,int k)
    {
        //base condition
        if(n == 1)
        return 0;

        return (winner(n-1,k)+k)%n;
    }

    int findTheWinner(int n, int k) {

        return winner(n,k)+1;
    }
};
```

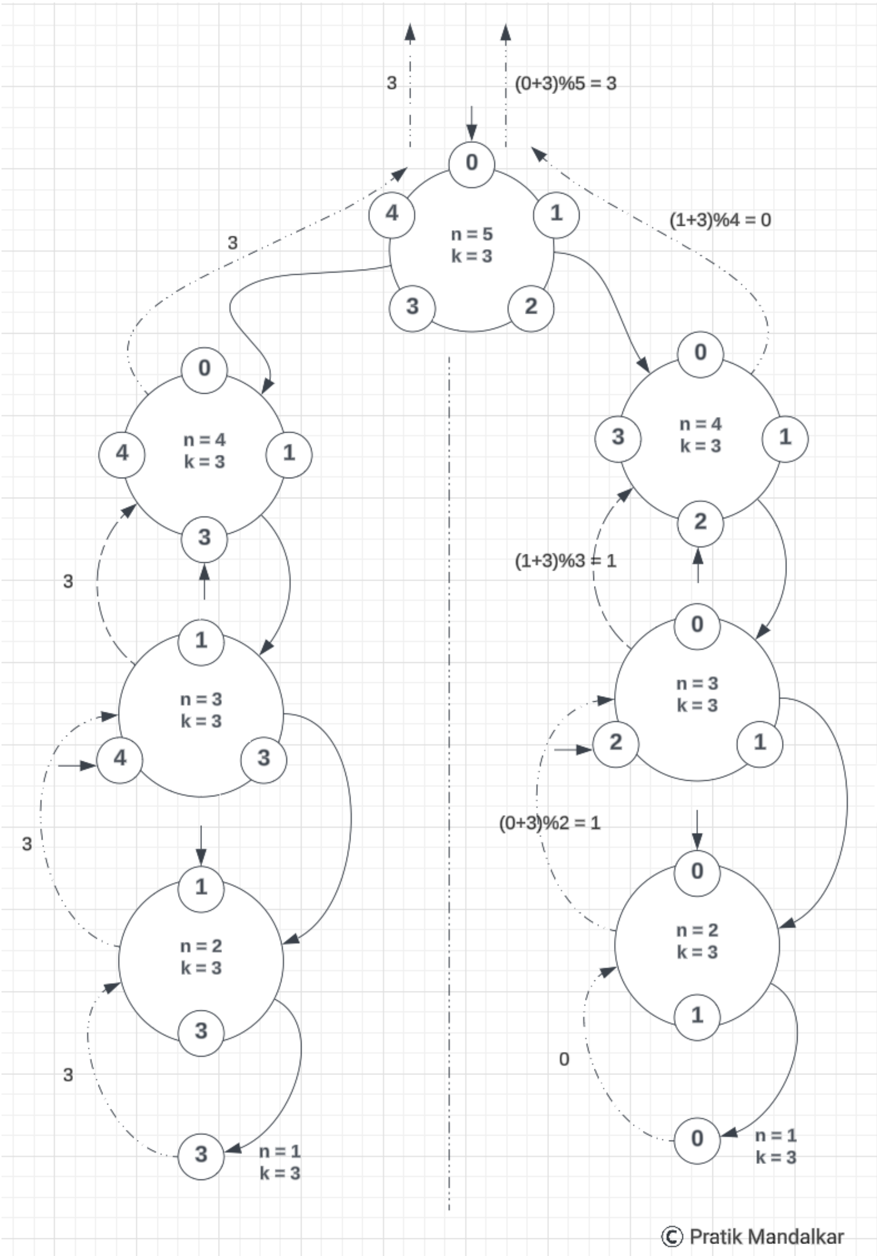
- **Time and Space Complexity Analysis**
  - ✓ **Time Complexity**:

    The function reduces n by 1 in each recursive call.
    The depth of recursion is **O(n)**.
    Each call does **O(1)** work.
    Overall time complexity: O(n).

  - ✓ **Space Complexity**:

    The function uses **recursive calls**, which take O(n) space in the **call stack**.
    No extra data structures (like arrays) are used.
    Overall space complexity: **O(n)** (due to recursion).

- **Dry Run with Visualization**

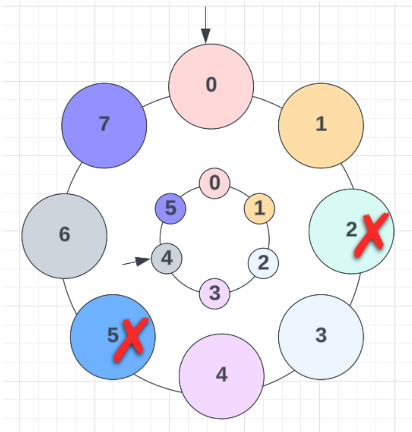


3rd Approach:

- Recursive Josephus Problem with **Efficient Mapping**:
  - ✓ Instead of eliminating 1 person in an iteration, we eliminate as many number of people we can elimiate in a single iteration.
  - ✓ If n is the number of people in the circle, we can remove $\lfloor n/k \rfloor$ in a single iteration. This process is repeated till n becomes less than k (step size).
  - ✓ When n<k, we follow the approach 2 given by the recurrence:

$$J(n, k) = (J(n-1,k)+k)\%n$$

  - ✓ When n>=k, we follow the generalized approach which computes ***J(n,k)*** in a more efficient way.



$$J(n,k) = \left\lfloor \frac{k\left(J\left(n - \left\lfloor \frac{n}{k} \right\rfloor, k\right) - n \mod k\right)}{k-1} \right\rfloor \mod n$$

- **Derivation:**
  - ✓ Consider a circle with *j* people, which are going to get eliminated in a sequential manner. After the first iteration exactly $j - \left\lfloor \frac{j}{k} \right\rfloor$ people are left in the circle.
  - ✓ We define this new circle to contain ***i*** people where:

$$i = j - \left\lfloor \frac{j}{k} \right\rfloor$$

  - ✓ To reiterate, we are considering two circles now –
    a) One having ***j*** people.
    b) Second having ***i*** people(after 1 iteration of elimination).

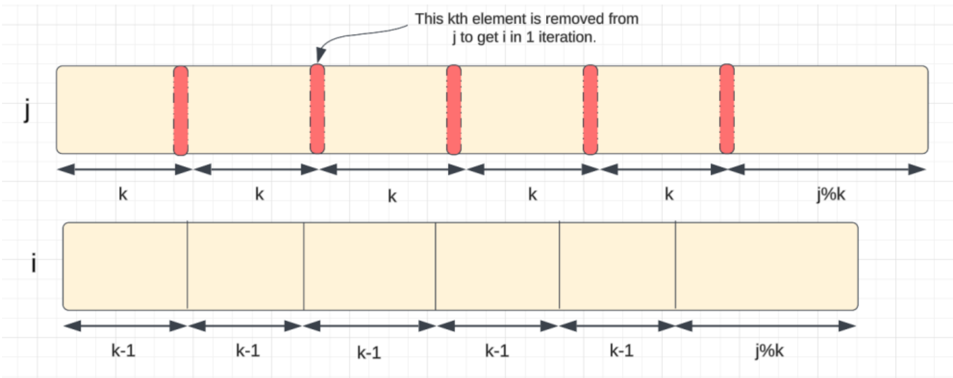
- ✓ Thus, ***j*** can be written as :

$$j = \left\lfloor \frac{j}{k} \right\rfloor * \; k \; + \; j \,\%\, k$$

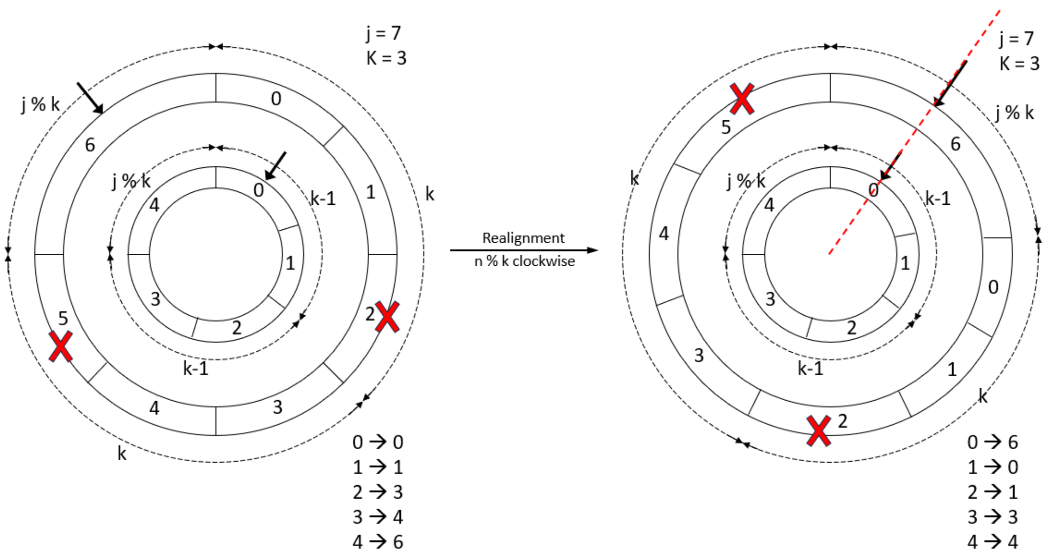
- ✓ And i can be written as :

$$i = \left\lfloor \frac{j}{k} \right\rfloor * (k-1) \; + \; j \,\%\, k$$

- ✓ We can see that j as $\left\lfloor \frac{j}{k} \right\rfloor$ parts of length k and a length *j%k.* Similarly, i can be seen as made up of $\left\lfloor \frac{j}{k} \right\rfloor$ parts of (k-1) length and a length j%k.



- ✓ It is clear from the above diagram that a portion of length of the circle *(j%k)* remains **constant** while the k $* \left\lfloor \frac{j}{k} \right\rfloor$ part of the circle reduces to (k −1)$* \left\lfloor \frac{j}{k} \right\rfloor$.
- ✓ Now, let us take the example of a circle having 7 people (i.e. n = 7) and k = 3. Since $j - \lfloor j/k \rfloor = 7 - \lfloor 7\ 3 \rfloor = 5$. There are 5 people in the second circle. This can be visualized as follows:

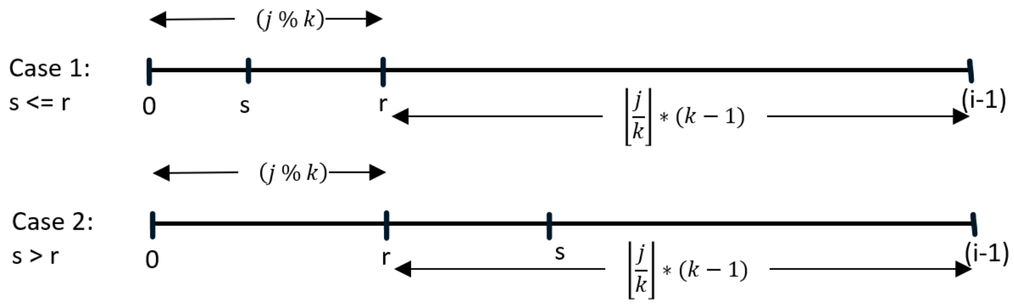


- ✓ We realign the outer circle (*j*) by rotation about the axis in either clockwise direction by (*n%k)* or anticlockwise by $\left\lfloor \frac{j}{k} \right\rfloor * k$. We did this because we observe that there are two portions of each circle. The *j % k* portion does not change in size while the by $\left\lfloor \frac{j}{k} \right\rfloor * k$ portion changes in size. Depending upon the location of the last survivor in the i circle, we can calculate the corresponding location in the j circle. In doing so, the difference in the growth of the two portions of the circle play a big role. We aligned the same length portion of the two circles together so that we can compare the lengths of the other portions easily.

- ✓ Let us define s as the position of the survivor in the i circle and **r = j % k**. Therefore,

$$s = J\left(j - \left\lfloor \frac{j}{k} \right\rfloor, k\right)$$

- ✓ We have 2 cases as *s* can be anywhere in the *i* circle:



Case I- (s < r):
Since, s lies in the portion (*j % k)* which remains of the same length in the two circles, there is no other additional shift in the position of *s* in the *j* circle. However, since we realigned the axis of the outer *j* circle, a shift of $\left\lfloor \frac{j}{k} \right\rfloor * k$ is introduced (since we rotated by that amount).
Therefore,

$$J\,(j+k) \;=\; s \;+\; \left\lfloor \frac{j}{k} \right\rfloor * k \;=\; s + j - j\%k$$

Case II- (s >= r):
Since, *s* lies in the portion that changes in length, there would be an additional shift while mapping from the *i* circle to the *j* circle. This would be accompanied by the shift caused by the realignment of the axis.
We know that,

$$\left\lfloor \frac{j}{k} \right\rfloor \;=\; \left\lfloor \frac{i}{k-1} \right\rfloor \;=\; number\ of\ blocks\ of\ size\ k(j\ circle)\ or\ (k-1)\ (i\ circle)$$

Therefore, for a distance of *(s – r)* in the *i* circle, a distance of $\frac{k*(s-r)}{k-1}$ there in the *j* circle. Thus, the additional shift is:

$$\boldsymbol{Additional}\ Shift\ = \frac{k*(s-r)}{k-1} - (s-r)$$
$$= (s-r)\left[\frac{k}{k-1} - 1\right]$$
$$= \frac{(s-r)}{(k-1)}$$

Therefore,

$$J(j,k)\ =\ (s\ +\ \left\lfloor\frac{j}{k}\right\rfloor * k\ +\ \frac{(s\ -r)}{(k\ -1)})\ \%\ j$$
$$= (s\ +\ j\ - (j\ \%\ k) + \frac{(s-r)}{(k-1)})\ \%\ j$$

But $j\ \%\ k\ =\ r$
Hence,

$$J(j,k)\ =\ ((s\ -\ r)(1\ +\ \frac{1}{(k\ -1)})\ \%\ j$$
$$= \boldsymbol{\frac{k*(s-r)}{(k-1)}}\ \%\ \boldsymbol{j}$$

- **Pseudo Code:**
  Input: Given the number of people in a circle n and the k^{th} person being executed in every iteration.
  Output: Survivor's position in the initial circle.
  1. **if** $n = 1$ **then return** 0
  2. **if** $k = 1$ **then return** $n - 1$
  3. **if** $k > n$ **then return** $(J(n-1,k) + k)\ mod\ n$
  4. $N \leftarrow\ J(n\ -\ \left\lfloor\frac{j}{k}\right\rfloor, k) - n\ mod\ k$
  5. **if** $N < 0$ **then** $N \leftarrow N + n$ **else** $N \leftarrow N + \frac{N}{(k-1)}$
  6. **return** *N*

4) <u>Final Recurrence :</u>

$$J(n,k)\ =\ \begin{cases} 0 & if\ n\ =\ 1, \\ (J(n-1,k)+k)\ mod\ n & if\ 1 <\ n < k \\ \left\{\begin{matrix} N\ +n & if\ N\ <\ 0, \\ \left\lfloor\frac{k\ *\ N}{k\ -1}\right\rfloor\ mod\ n & if\ N \geq\ 0 \end{matrix}\right\} & if\ k\ \leq\ n \end{cases}$$

5) Table of comparison:

Approach	**Time Complexity**	**Space Complexity**
1	$O(n^2)$	O(n)
2	O(n)	O(n)
3	O (k*log n)	O(n)