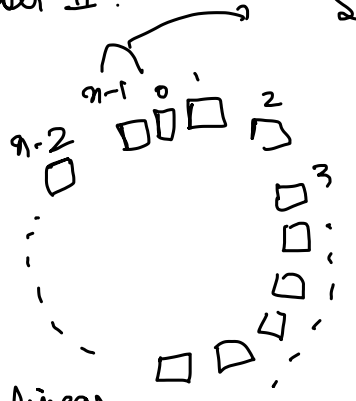
Assignment-1

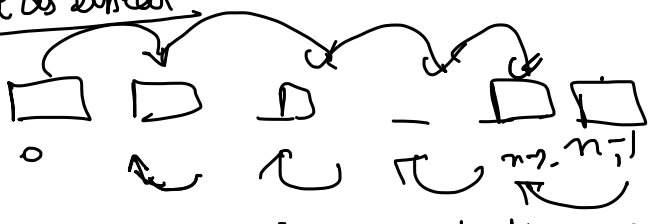
Q3. House Robber II.

So we can either rob [0] or [n-1]



Just like in candy problem we can work using two loops one starting to end & other ending to start.

If we consider it as linear



If I take forward loop from 0 to n-1 & backward loop then there is possibility ki dono choose ho jaye so.

As like in permutation when we permute n people in circular pattern we can write [n-1] and then consider it is straight pattern.

Forward loop → 0 to n-2

Backward loop → n-1 to 1

later from 1 to n-1

Then max (Floop, Bloop) is our answer, because we get answer when first & n-1 house is not robbed and second when 0th house is not robbed.

And for getting maximum rob from 0 to n-2 it is same problem as House Robber I that is discussed in class that if we're standing at ith index we have two options either rob it or not rob it. so take = arr[i] + max[i-2] or ntake = max[i-1];

As DP problems cannot be solved reversely so and we know max rob from $n-1$ to 1 or 1 to $n-1$ is same we will work through 1 to $n-1$ to solve it through DP.

01. As going from $n \to n/2$ has cost 1 (even n) $f(n/2)$ ode

So this will be preferred more as it is reducing number by a good factor and still using cost 1.

If n is odd we have two options either increase or decrease it by 1 ($f(n/2+1)$ $f(n/2-1)$ both steps uses cost as 1).

So first can be brute force recursion where at a certain n we have one option if n is even, and if n is odd we will consider return $\min(f(n+1), f(n-1))+1$ and with a base case of $n == 1$ (return 0;)

So for using DP here as we're counting $f(n)$ multiple times for a certain n. We can use here an unordered map <key (int), min-cost> dp where key resembles for what n we have calculated minimum cost. (Top to down DP).

If $n = 3$

two method

$3 \to 4 \to 2 \to 1$ cost = 3

$3 \to 2 \to 1$ cost = 2 preffred

$n = 4$

$4 \to 2 \to 1$

three method

$n = 5$ $5 \to 4 \to 2 \to 1$ minimum $5 \to 6 \to 3 \to 2 \to 1$ $5 \to 6 \to 3 \to 4 \to 2 \to 1$

$n = 6$ $6 \to 3 \to 2 \to 1$

$n = 7$ $7 \to 6 \to 3 \to 2 \to 1$ $7 \to 8 \to 4 \to 2 \to 1$

So by intution we can observe that if we find n s.t. $n \& 2 == 1$ (Means n has second binary as 1) we get to the code 1 fast and cheaply if we change $n \to n+1$ ($7 \to 8$) and if $n \& 2 == 0$ ($n \to n-1$) ($5 \to 4$)

Because we want that after changing $n \to n+1$ or $n-1$ we get more and more 2 as factor to reduce the number fast

↳ But for n=3

this algo doesn't work as if we change 3→2 & divide by 2 we don't have to increase or decrease further as we got base case.
But for 3→4 we have to divide 4 by 2 (2 times) which will increase cost.

(This exception came because 3 instantly give 1 where we stop when just divided by 2 once in n-1 case)