Dynamic Programming

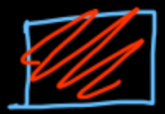
→ Lecture - 8 9 am to 12 pm Saturday
30 April

1. Paint Fence
2. Ugly No - II
3. Ugly No - III
4. Frog Jump
5. Jump Game - All Paths

Paint Fence

(n houses, k colors) no more than 2 houses have the same color

N=1

k=1  ①

k=2  ,  ②

k=3  ,  ,  ③

⋮

k=D (k)

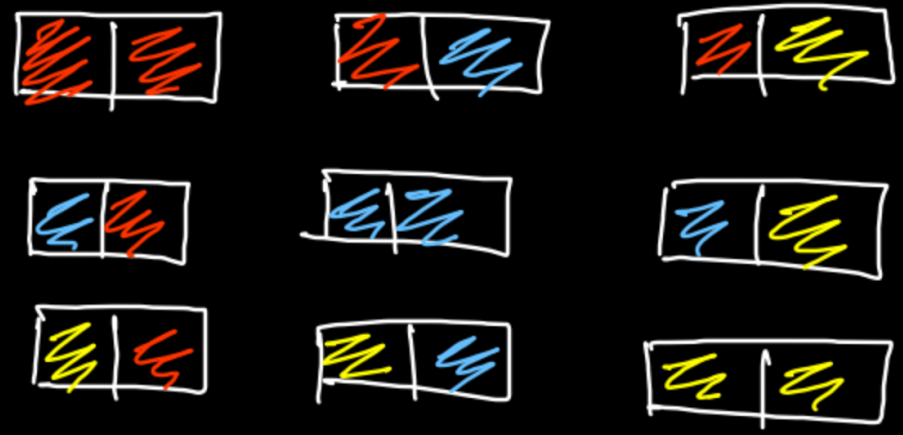
N=2

k=1  ①

k=2  ④

N = 2

k = 3



9

N = 2

k → k^2

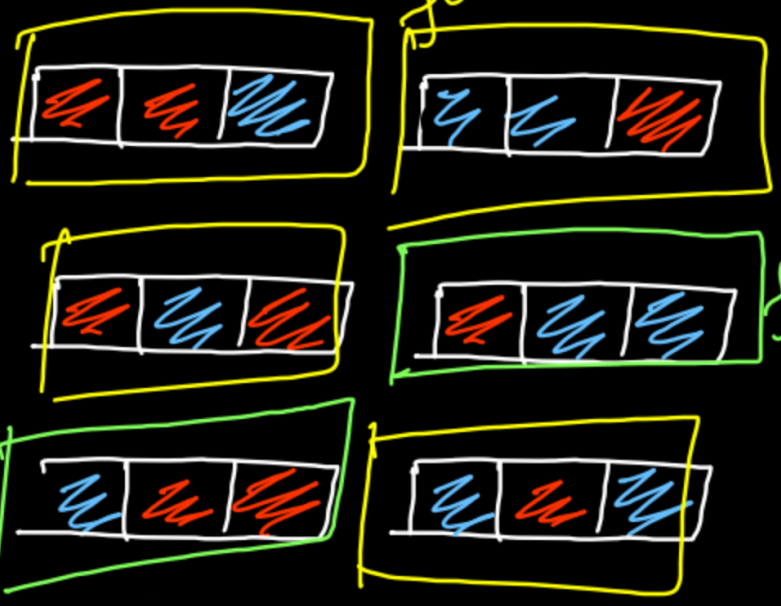
N=3

k=1

6

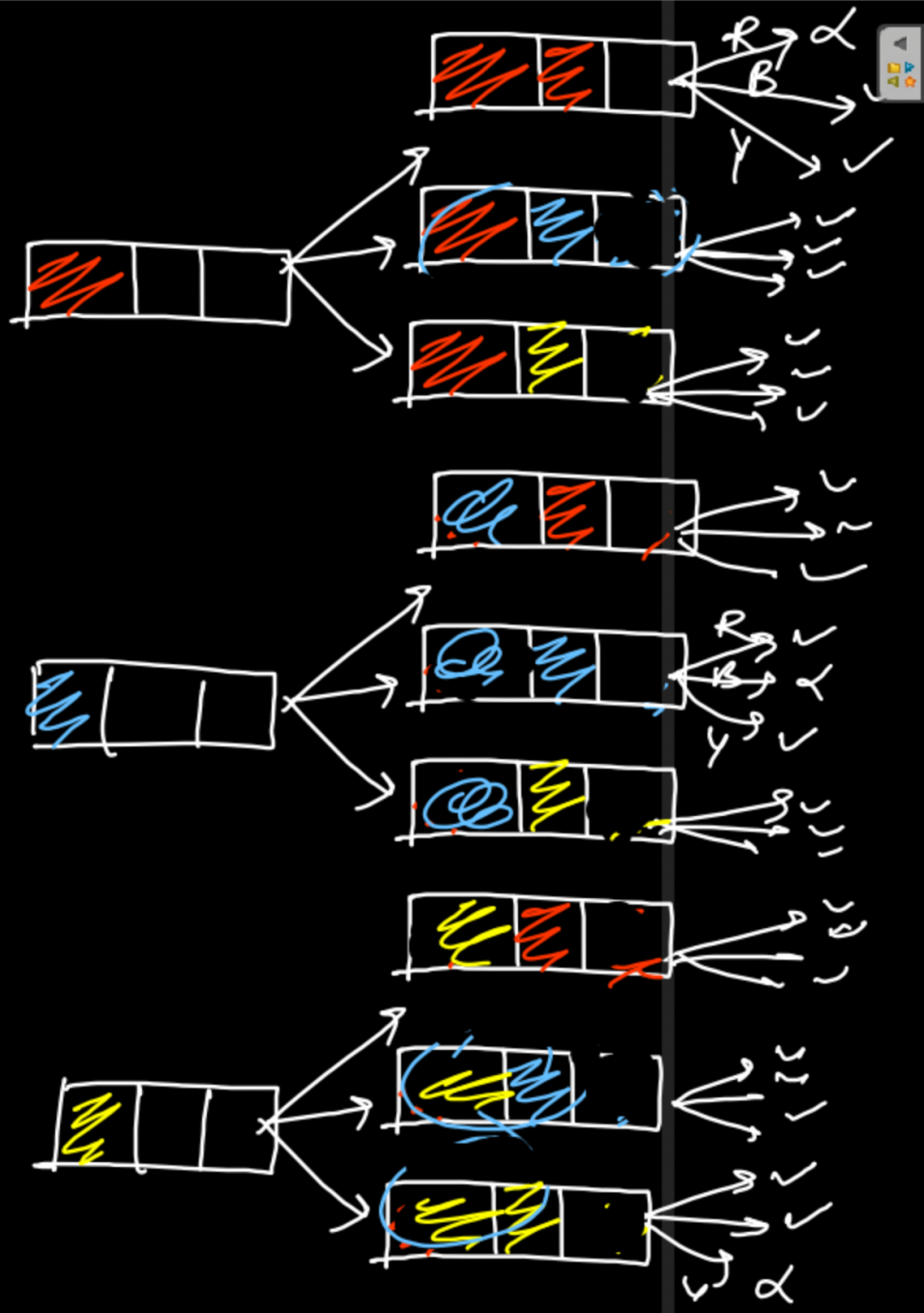
k=2

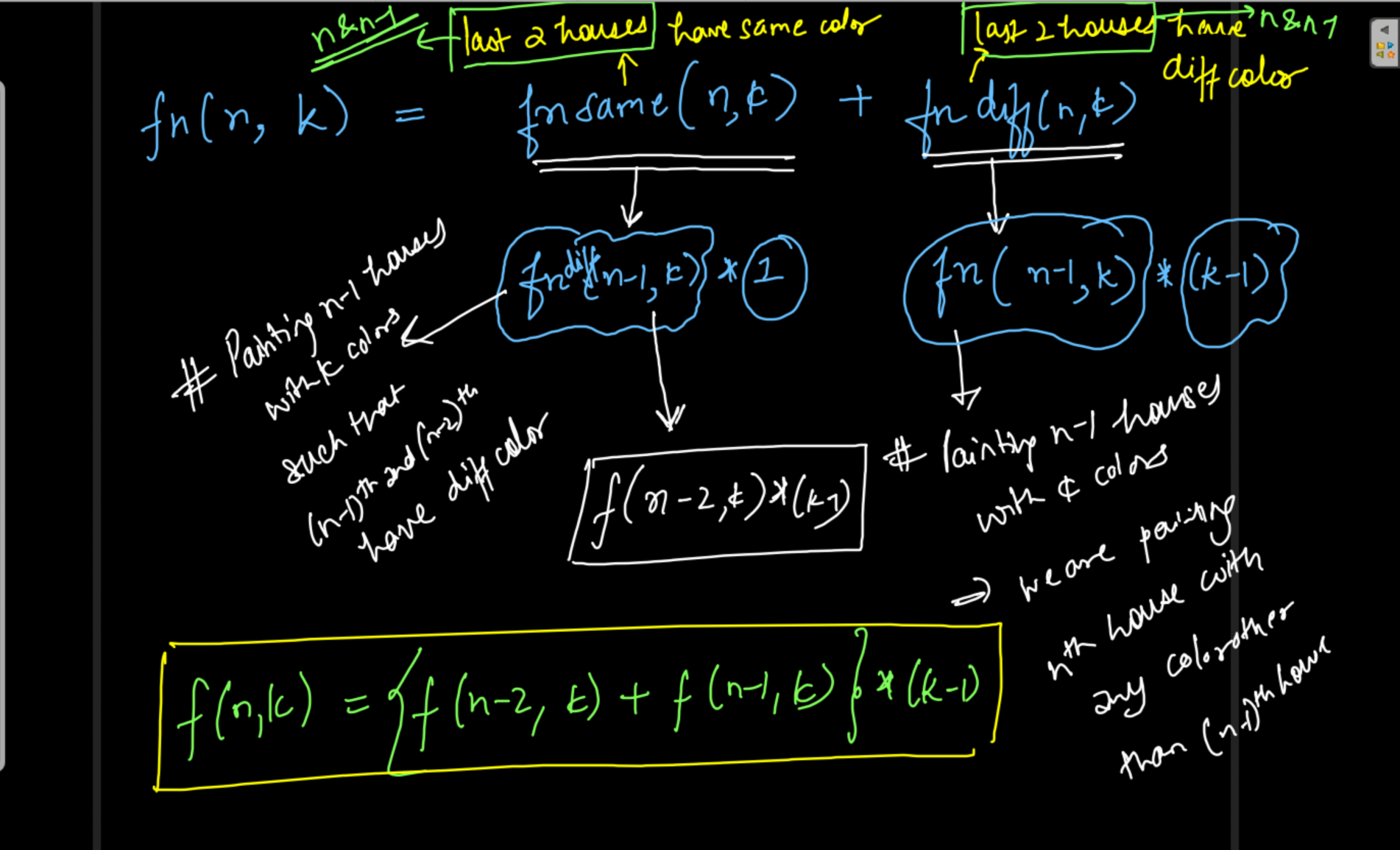
6



k=3

24




n & n-1
last 2 houses have same color
last 2 houses have n & n-1 diff color
fn(n, k) = fn same(n, k) + fn diff(n, k)
fn diff(n-1, k) * (1)
fn(n-1, k) * (k-1)
# Painting n-1 houses with k colors such that (n-1)th and (n-2)th have diff color
f(n-2, k) * (k-1)
# painting n-1 houses with k colors
⇒ we are painting nth house with any color other than (n-1)th house
f(n, k) = f(n-2, k) + f(n-1, k) * (k-1)

Tabulation

$k = 3$

	N=1	N=2	N=3	N=4
least 2 houses same	1	3	3*2	18
diff	1	3*2	(3+3*2)*2 = 18	(18+6)*2 = 48

TC ⇒ O(N)

SC ⇒ O(N) → "Space can be optimized"

```java
public class Solution {
    public int memo(int n, int k, int[] dp){
        if(n == 1) return k;
        if(n == 2) return k * k;
        if(dp[n] != -1) return dp[n];

        int ans = (memo(n - 1, k, dp) + memo(n - 2, k, dp)) * (k - 1);
        return dp[n] = ans;
    }

    public int numWays(int n, int k) {
        if(n == 1) return k;
        if(n == 2) return k * k;
        if(k == 1) return 0;

        int[] dp = new int[n + 1];
        Arrays.fill(dp, -1);
        return memo(n, k, dp);
    }
}
```

0.7 x

```java
public int numWays(int n, int k) {
    if(n == 1) return k;
    if(n == 2) return k * k;
    if(k == 1) return 0;

    int[] same = new int[n + 1];
    int[] diff = new int[n + 1];
    same[2] = k; diff[2] = k * (k - 1);

    for(int i=3; i<=n; i++){
        same[i] = diff[i - 1];
        diff[i] = (same[i - 1] + diff[i - 1]) * (k - 1);
    }

    return same[n] + diff[n];
}
```

0.7 x

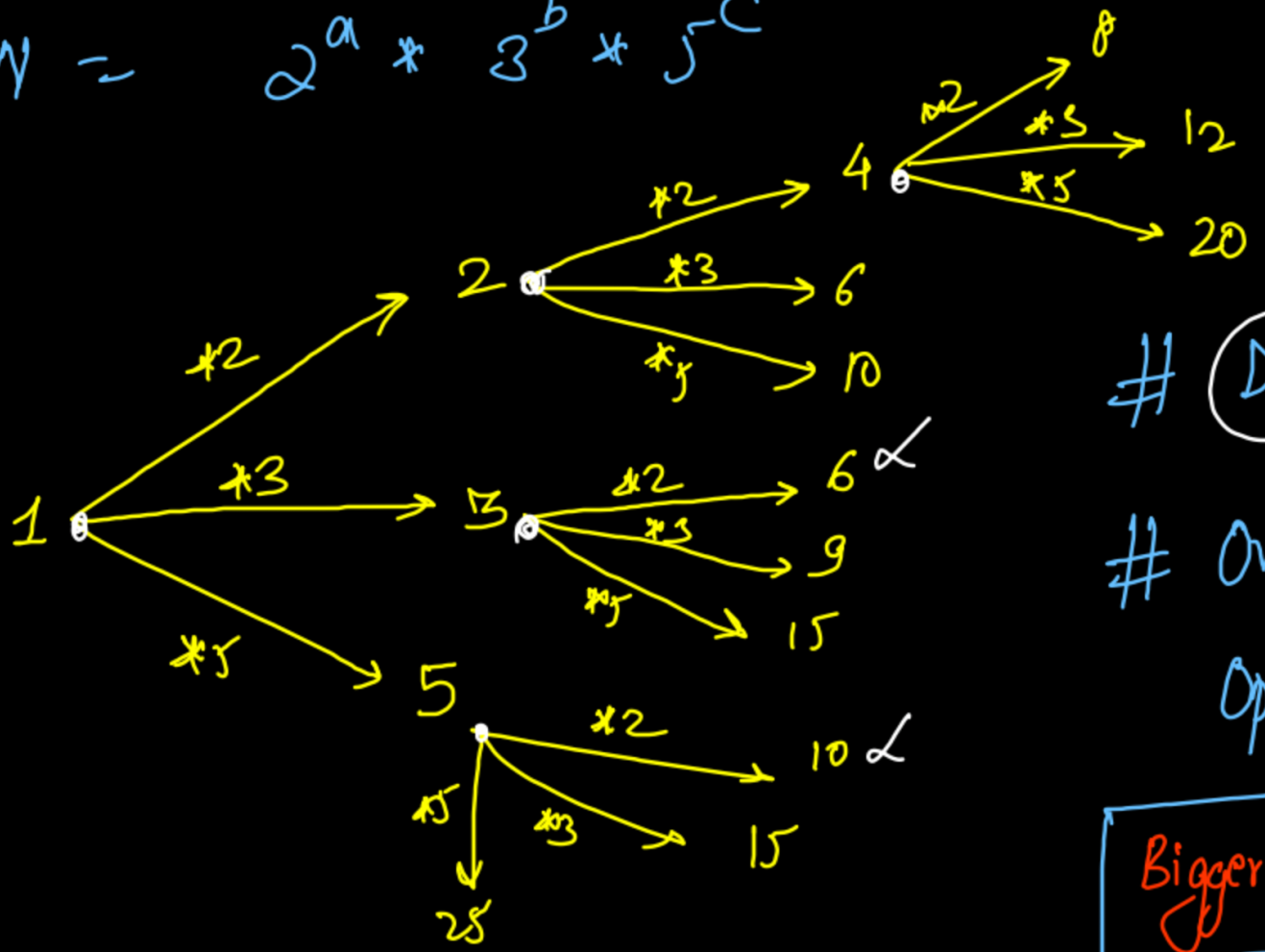
```java
public int numWays(int n, int k) {
    if(n == 1) return k;
    if(n == 2) return k * k;
    if(k == 1) return 0;

    int same = k;
    int diff = k * (k - 1);

    for(int i=3; i<=n; i++){
        int newSame = diff;
        int newDiff = (same + diff) * (k - 1);

        same = newSame; diff = newDiff;
    }

    return same + diff;
}
```

Ugly No - 11

0 1 2 3 4 5 6 8

1, 2, 3, 4, 5, 6, 8, 10 →

$N = 2^a * 3^b * 5^c$



DFS or BFS ?

Recursion

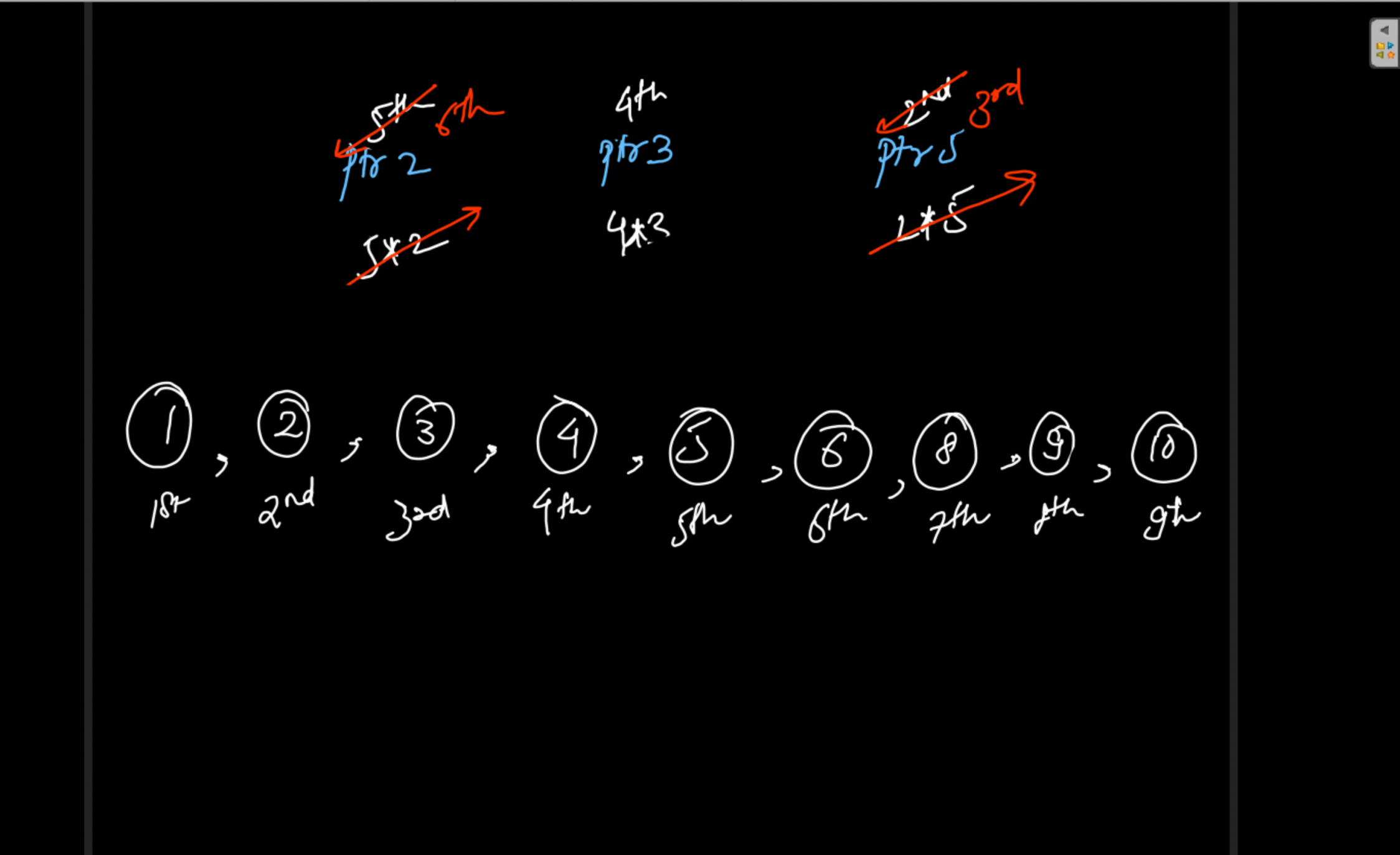
queue

Overlapping subproblems (DP)

↳ 2*3 = 3*2

Optimal substructure (Faith)

Bigger Ugly = Smaller Ugly * (2/3/5)


5th
6th
ptr 2
5*2
4th
ptr 3
4*3
2nd
3rd
ptr 5
1*5
1 , 2 , 3 , 4 , 5 , 6 , 8 , 9 , 10
1st
2nd
3rd
4th
5th
6th
7th
8th
9th

```java
// O(N) Time, O(N) Space
public int nthUglyNumber(int n) {
    if(n == 1) return 1;

    // Pointers Pointing to Indices not Values
    int ptr2 = 0, ptr3 = 0, ptr5 = 0;

    ArrayList<Integer> ugly = new ArrayList<>(); // visited array
    ugly.add(1); // to add the 1st ugly no at index 0

    for(int i=1; i<n; i++){
        int a = ugly.get(ptr2) * 2;
        int b = ugly.get(ptr3) * 3;
        int c = ugly.get(ptr5) * 5;

        int min = Math.min(a, Math.min(b, c));
        ugly.add(min);

        if(min == a) ptr2++;
        if(min == b) ptr3++;
        if(min == c) ptr5++;
    }

    return ugly.get(n - 1);
}
```

0.7 x

```java
// O(N * Log N) Time, O(N) Space
public int nthUglyNumber(int n) {
    if(n == 1) return 1;

    PriorityQueue<Long> q = new PriorityQueue<>();
    q.add(1l);
    HashSet<Long> vis = new HashSet<>();

    int idx = 0;
    while(q.size() > 0){
        long min = q.remove();
        if(vis.contains(min) == true)
            continue;

        idx++;
        if(idx == n) return (int)min;

        vis.add(min);
        q.add(min * 2l);
        q.add(min * 3l);
        q.add(min * 5l);
    }

    return 1;
}
```

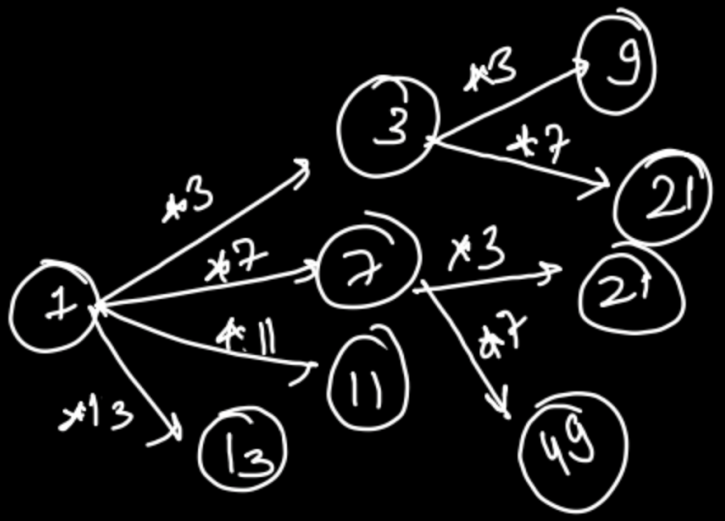
0.5 x

Super Ugly No

K (Input) primes = {3, 7, 11, 13}

N=7 → 7th ugly ?

(Extra) =O(K) pointers = {4, 3, 2, 2}



Res (Extra) = O(N)

1 (1st), 3 (2nd), 7 (3rd), 9 (4th), 11 (5th), 13 (6th), 21 (7th)

Space Comp O(N+K)

Res

pointers index

```java
// O(N * K) Time, O(N + K) Space
public int nthSuperUglyNumber(int n, int[] primes) {
    int[] ptr = new int[primes.length];

    ArrayList<Integer> ugly = new ArrayList<>();
    ugly.add(1); // Add 1st Ugly No at Index 0

    for(int i=1; i<n; i++){
        // Finding the next Smallest Ugly No
        int min = Integer.MAX_VALUE;
        for(int j=0; j<primes.length; j++)
            min = Math.min(min, ugly.get(ptr[j]) * primes[j]);

        ugly.add(min);

        // Updating All Pointers Pointing to Min
        for(int j=0; j<primes.length; j++)
            if(ugly.get(ptr[j]) * primes[j] == min)
                ptr[j]++;
    }

    return ugly.get(n - 1);
}
```