

LIS(0, -1)
 ↑
 index prev-index

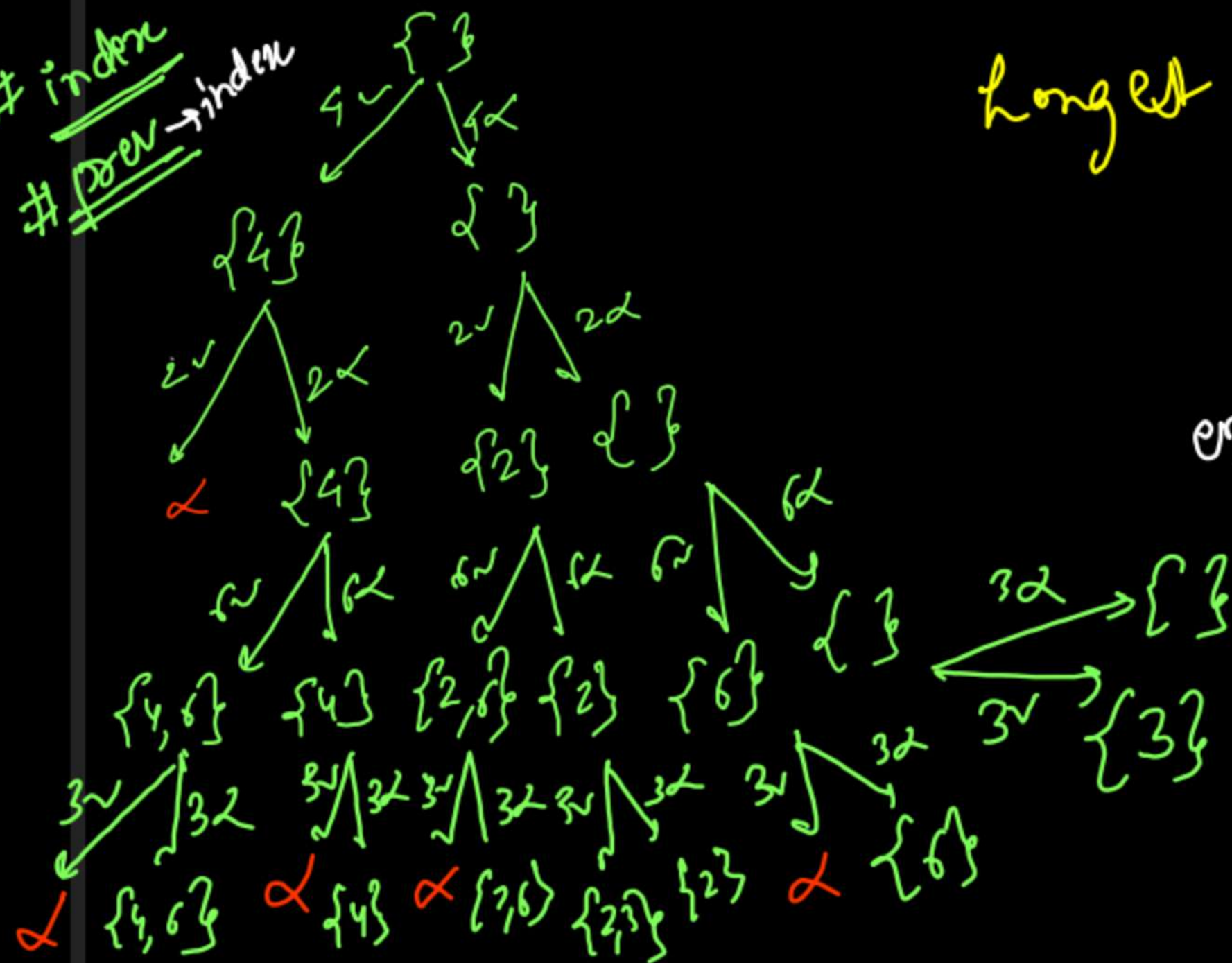
longest strictly increasing subsequence → subset → 2^n total subsets

{ 4, 2, 6, 3, 5, 7, 1, 6, 8, 9, 2 }

longest LIS: { 2, 3, 5, 6, 8, 9 }

length = 6

index
 # prev → index



empty set

arr[ind] > arr[prev]

prev = -1

yes

LIS(ind, prev)

no

LIS(ind+1, ind)

LIS(ind+1, prev)

```

class Solution {
    public int memo(int curr, int prev, int[] nums, int[][] dp){
        if(curr == nums.length) return 0;
        if(dp[curr][prev + 1] != -1) return dp[curr][prev + 1];

        int yes = (prev == -1 || nums[prev] < nums[curr])
            ? memo(curr + 1, curr, nums, dp) + 1 : 0;

        int no = memo(curr + 1, prev, nums, dp);

        return dp[curr][prev + 1] = Math.max(yes, no);
    }

    public int lengthOfLIS(int[] nums) {
        int n = nums.length;
        int[][] dp = new int[n + 1][n + 1];
        for(int i=0; i<=n; i++){
            for(int j=0; j<=n; j++){
                dp[i][j] = -1;
            }
        }

        return memo(0, -1, nums, dp);
    }
}

```

to store
prev = -1
also in
the
DP

Memorization

Time $\rightarrow O(N \times N)$

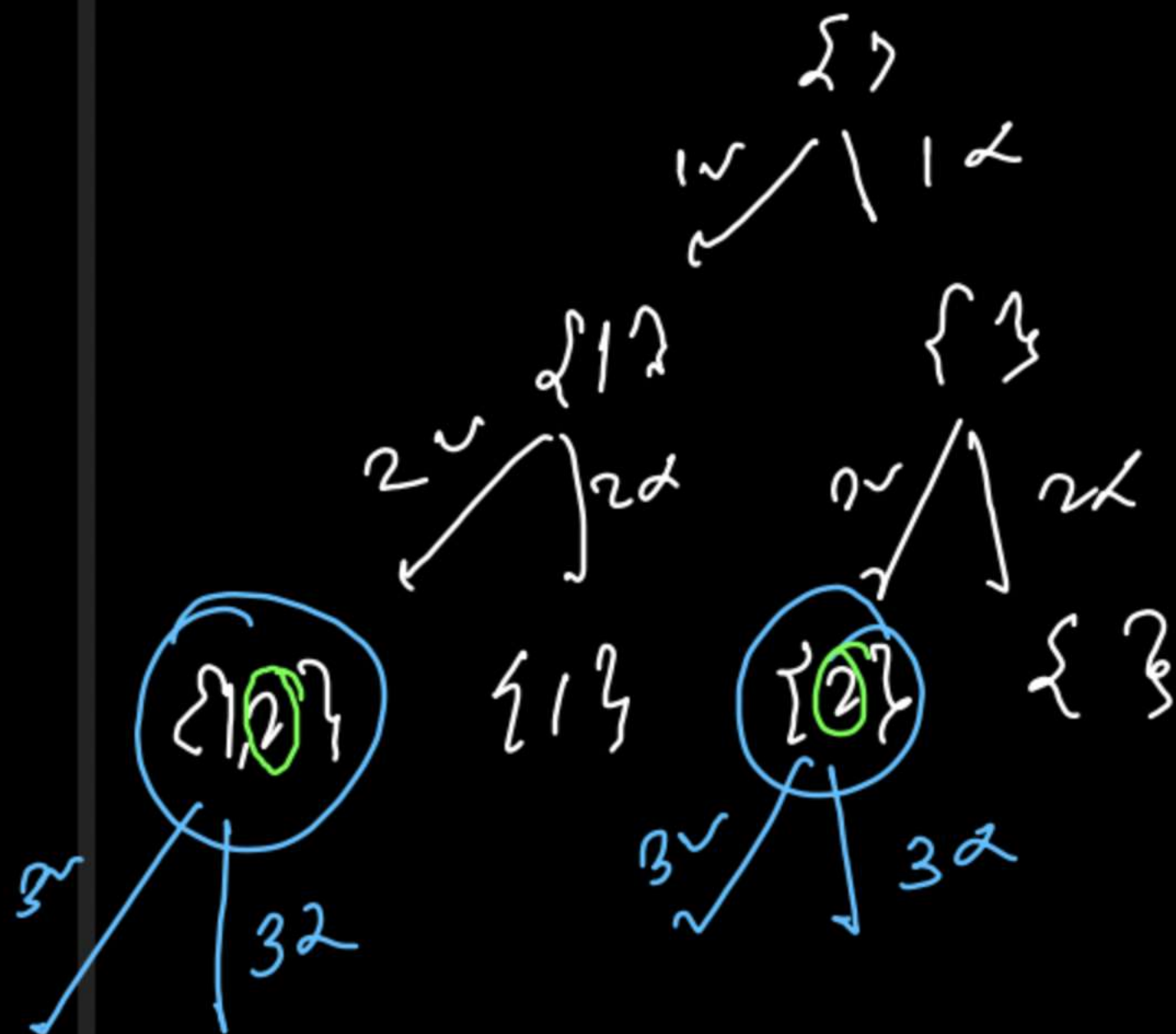
curr
prev

curr element
is included
 \Rightarrow length \uparrow by 1

Space $\rightarrow O(N \times N)$

2D DP

$\{1, 2, 3, 4\}$ overlapping subproblems



Tabulation

② {2, 6}

③ {2, 3, 5}

④ {2, 3, 5, 7}

3 {2, 3, 7}

2 {6, 7}

2 {2, 5}

2 {2, 7}

2 {4, 5}

2 {4, 7}

2 {1, 6}

④ {2, 3, 5, 6}

3 {2, 3, 6}

2 {2, 6}

2 {4, 6}

4 {7, 3, 5, 8}

3 {2, 3, 8}

2 {6, 8}

3 {2, 6, 8}

2 {4, 8}

5 {2, 3, 5, 6, 8}

5 {2, 3, 7, 7, 8}

2 {1, 4}

③ {2, 3, 4}

2 {3, 4}

1	1	1	1	1	1	1	1	1	1
{4}	{2}	{6}	{3}	{5}	{7}	{1}	{6}	{8}	{4}

4

②

6

③

⑤

7

1

⑥

⑧

4

dp[i] = longest increasing subsequence length
ending at index i

```

public int lengthOfLIS(int[] nums) {
    int n = nums.length;
    int[] dp = new int[n];
    int maxLIS = 0;
    for(int i=0; i<nums.length; i++){
        dp[i] = 1; // If Prev Does not Exist, then current element can have yes

        for(int j=0; j<i; j++){
            if(nums[j] < nums[i]){
                dp[i] = Math.max(dp[i], dp[j] + 1);
            }
        }

        maxLIS = Math.max(maxLIS, dp[i]);
    }

    return maxLIS;
}

```

Tabulation

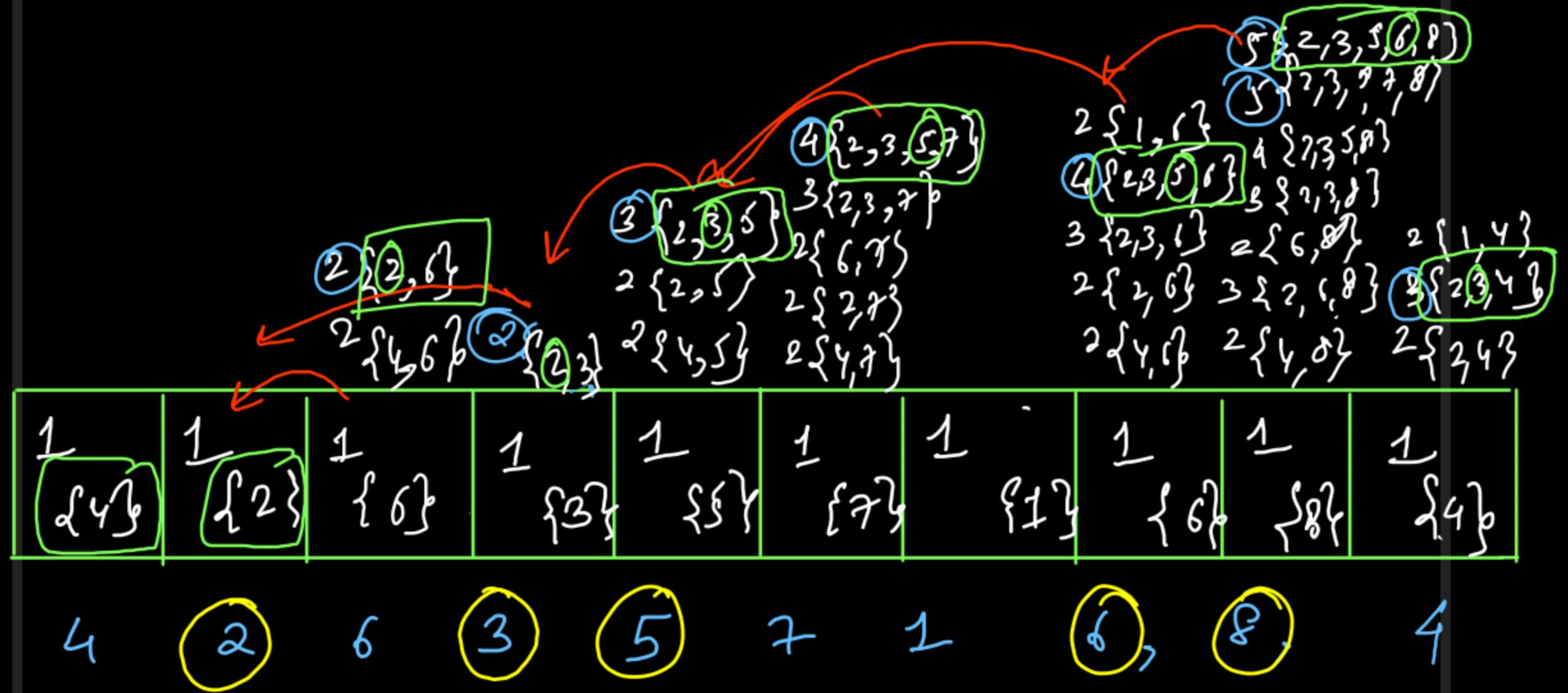
Time $\rightarrow O(N^2)$

Space $\rightarrow O(N)$

1D DP

Printing LIS

Any one \rightarrow Arraylist LIS store at each index
 Print All \rightarrow Backtracking



Print Any one LIS

✓ Time

$O(N \times N \times N)$
 $= O(N^3)$

✓ Space

$O(N \times N)$
storing AL
at each index

```
public static void solution(int[] nums){
    int n = nums.length;
    ArrayList<Integer>[] dp = new ArrayList[n];

    int maxLIS = 0, maxLISind = 0;

    for(int i=0; i<nums.length; i++){
        dp[i] = new ArrayList<>();
        dp[i].add(nums[i]);

        for(int j=0; j<i; j++){
            if(nums[j] < nums[i]){
                if(dp[j].size() + 1 > dp[i].size()){
                    dp[i] = new ArrayList<>(dp[j]);
                    dp[i].add(nums[i]);
                }
            }
        }

        if(dp[i].size() > maxLIS){
            maxLIS = dp[i].size();
            maxLISind = i;
        }
    }

    for(int val: dp[maxLISind]){
        System.out.print(val + " ");
    }
}
```

* deep copy → $O(N)$


```

public static void solution(int[] nums){
    int n = nums.length;
    int[] dp = new int[n];
    int[] prev = new int[n];

    int maxLIS = 0, lisidx = 0;

    for(int i=0; i<nums.length; i++){
        dp[i] = 1; // Length
        prev[i] = -1; // Empty Subset -> Current Element

        for(int j=0; j<i; j++){
            if(nums[j] < nums[i]){
                if(dp[j] + 1 > dp[i]){
                    dp[i] = dp[j] + 1;
                    prev[i] = j;
                }
            }
        }

        if(dp[i] > maxLIS){
            maxLIS = dp[i];
            lisidx = i;
        }
    }

    ArrayList<Integer> LIS = new ArrayList<>();

    while(lisidx != -1){
        LIS.add(nums[lisidx]);
        lisidx = prev[lisidx];
    }

    Collections.reverse(LIS);
    System.out.println(LIS);
}

```

Time $\Rightarrow O(N^2 + N)$
 $\approx O(N^2)$

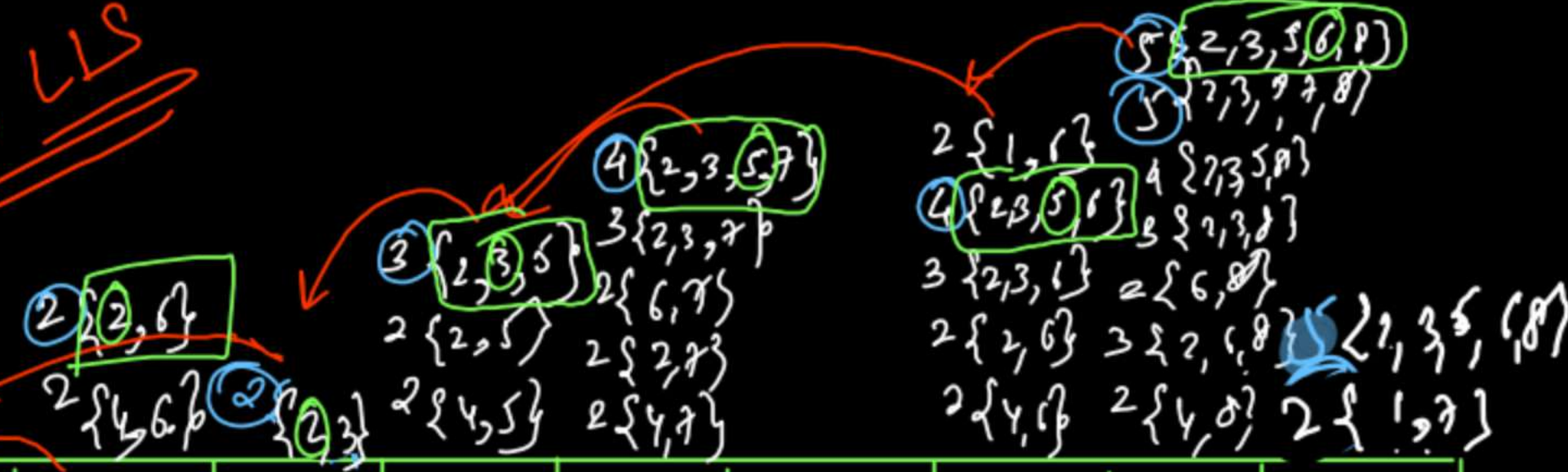
Space $\rightarrow O(2 \times N)$
 $+ O(N)$
 $= O(N)$

} Backtracking $\rightarrow O(\text{LIS length})$
 $\approx O(N)$ in worst case

max LIS - 5
Print ALL LIS

1	1	1	1	1	1	1	1	1	1
{4}	{2}	{6}	{3}	{5}	{7}	{1}	{6}	{8}	{7}

4 2 6 3 5 7 1 6 8 7



DFS(7) {7}

DFS(6) {6, 7}

DFS(5) {5, 6, 7}

DFS(3) {3, 5, 6, 7}

DFS(2) {2, 3, 5, 6, 7}

DFS(8)

DFS(6)

DFS(7)

DFS(5)

DFS(3)

LIS of length = 5

2, 3, 5, 6, 8

2, 3, 5, 7, 8

2, 3, 5, 6, 7

LIS

end at same index
 diff in den more than 2 LIS is possible

{2, 3, 5, 6, 8}

{2, 3, 5, 7, 8}


```

public static void DFS(int curr, int[] nums, int[] dp, String psf) {
    if (dp[curr] == 1) {
        System.out.println(psf);
        return;
    }
    for (int prev = curr - 1; prev >= 0; prev--) {
        if (nums[prev] < nums[curr] && dp[curr] == dp[prev] + 1) {
            DFS(prev, nums, dp, nums[prev] + " -> " + psf);
        }
    }
}

```

2 LIS can have increasing index

Increasing

Longest

Time
 # Worst case \rightarrow exponential
 Any case \rightarrow O(polynomial)

```

public static void solution(int[] nums) {
    int n = nums.length;
    int[] dp = new int[n];

    int maxLIS = 0;

    for (int i = 0; i < nums.length; i++) {
        dp[i] = 1; // Length

        for (int j = 0; j < i; j++) {
            if (nums[j] < nums[i]) {
                dp[i] = Math.max(dp[i], dp[j] + 1);
            }
        }

        if (dp[i] > maxLIS) {
            maxLIS = dp[i];
        }
    }

    System.out.println(maxLIS);
}

```

```

for (int i = n - 1; i >= 0; i--) {
    // Start DFS from each node at which Increasing Subset is of
    // Longest Length

    if (dp[i] == maxLIS) {
        DFS(i, nums, dp, "" + nums[i]);
    }
}

```

} Multiple Sources
 LIS can end at diff index

Space
 $O(N)$ \rightarrow DP
 \rightarrow DFS

LIS → LIS length → Time optimization → $O(N \log N)$
(Binary Search)

→ Count
LIS