39. Combination Sum

Medium 5326 142 Add to List Share

Given an array of **distinct** integers `candidates` and a target integer `target`, return *a list of all* ***unique combinations*** *of* `candidates` *where the chosen numbers sum to* `target`. You may return the combinations in **any order**.

The **same** number may be chosen from `candidates` an **unlimited number of times**. Two combinations are unique if the frequency of at least one of the chosen numbers is different.

It is **guaranteed** that the number of unique combinations that sum up to `target` is less than `150` combinations for the given input.

Example 1:

```
Input: candidates = [2,3,6,7], target = 7
Output: [[2,2,3],[7]]
Explanation:
2 and 3 are candidates, and 2 + 2 + 3 = 7. Note that 2 can be used multiple times.
7 is a candidate, and 7 = 7.
These are the only two combinations.
```

Example 2:

```
Input: candidates = [2,3,5], target = 8
Output: [[2,2,2,2],[2,3,3],[3,5]]
```

Example 3:

TUF

39. Combination Sum

Medium 5326 142 Add to List Share

Given an array of **distinct** integers `candidates` and a target integer `target`, return *a list of all **unique combinations** of* `candidates` *where the chosen numbers sum to* `target`. You may return the combinations in **any order**.

The **same** number may be chosen from `candidates` an **unlimited number of times**. Two combinations are unique if the frequency of at least one of the chosen numbers is different.

It is **guaranteed** that the number of unique combinations that sum up to `target` is less than `150` combinations for the given input.

Example 1:

```
Input: candidates = [2,3,6,7], target = 7
Output: [[2,2,3],[7]]
Explanation:
2 and 3 are candidates, and 2 + 2 + 3 = 7. Note that 2 can be used multiple times.
7 is a candidate, and 7 = 7.
These are the only two combinations.
```

Example 2:

```
Input: candidates = [2,3,5], target = 8
Output: [[2,2,2,2],[2,3,3],[3,5]]
```

Example 3:

TUF

Description | Solution | Discuss (999+) | Submissions

39. Combination Sum

Medium 5326 142 Add to List Share

Given an array of **distinct** integers `candidates` and a target integer `target`, return *a list of all **unique combinations** of* `candidates` *where the chosen numbers sum to* `target`. You may return the combinations in **any order**.

The **same** number may be chosen from `candidates` an **unlimited number of times**. Two combinations are unique if the frequency of at least one of the chosen numbers is different.

It is **guaranteed** that the number of unique combinations that sum up to `target` is less than `150` combinations for the given input.

Example 1:

```
Input: candidates = [2,3,6,7], target = 7
Output: [[2,2,3],[7]]
Explanation:
2 and 3 are candidates, and 2 + 2 + 3 = 7. Note that 2 can be used multiple times.
7 is a candidate, and 7 = 7.
These are the only two combinations.
```

Example 2:

```
Input: candidates = [2,3,5], target = 8
Output: [[2,2,2,2],[2,3,3],[3,5]]
```

Example 3:

Promotion till 02:22

TUF


L8. Combination Sum | Recursion | Leetcode | C++ | Java
arr[] = [2, 3, 6, 7]
target = 7
2, 2, 3
TUF
3:01 / 26:59
CC
HD
All changes saved!

arr[] = [2, 3, 6, 7] target = 7

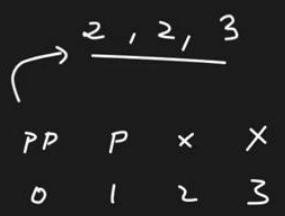
2, 2, 3

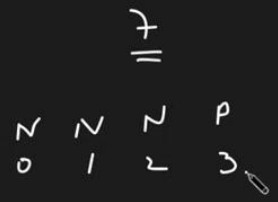
P

0 1 2 3

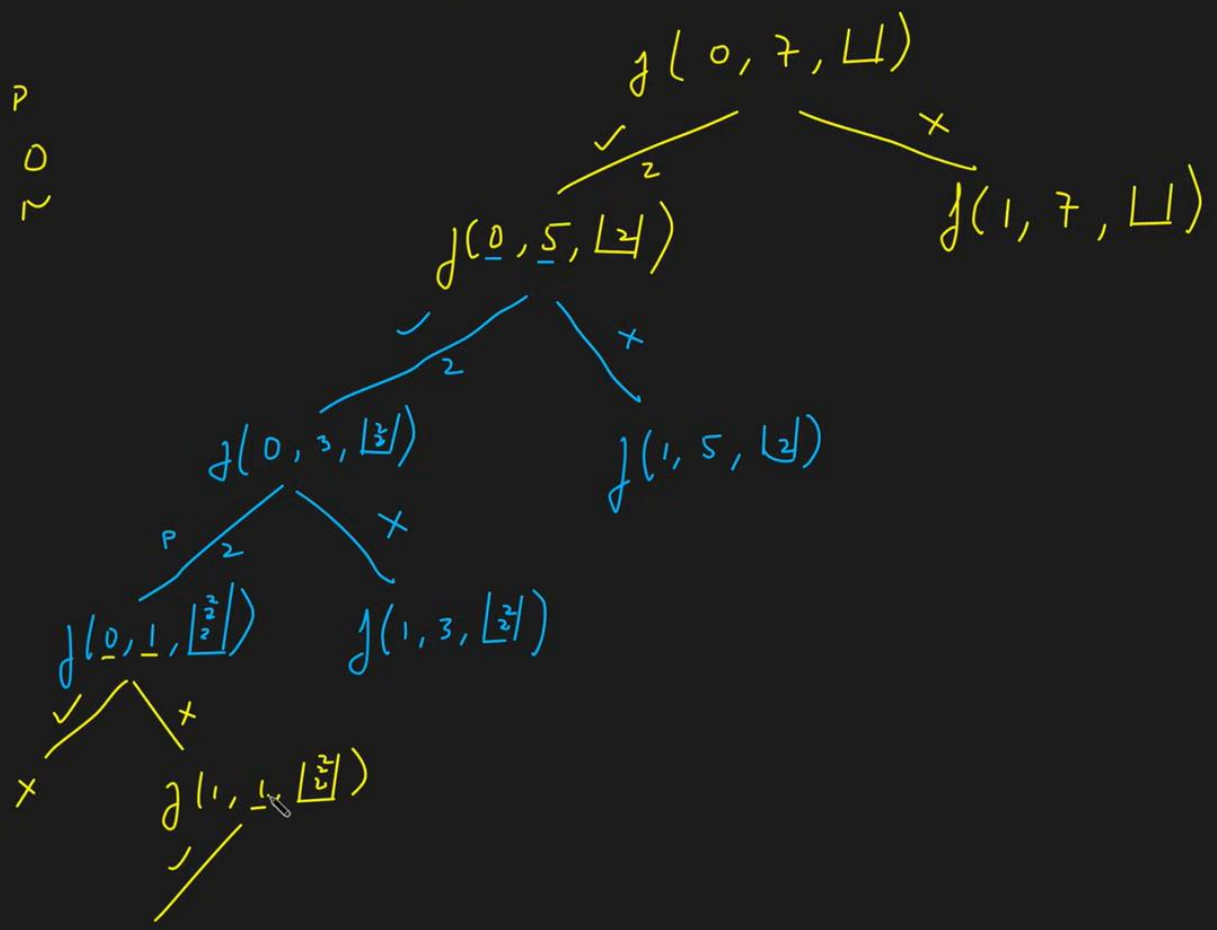
TUF

arr[] = [2, 3, 6, 7] target = 7

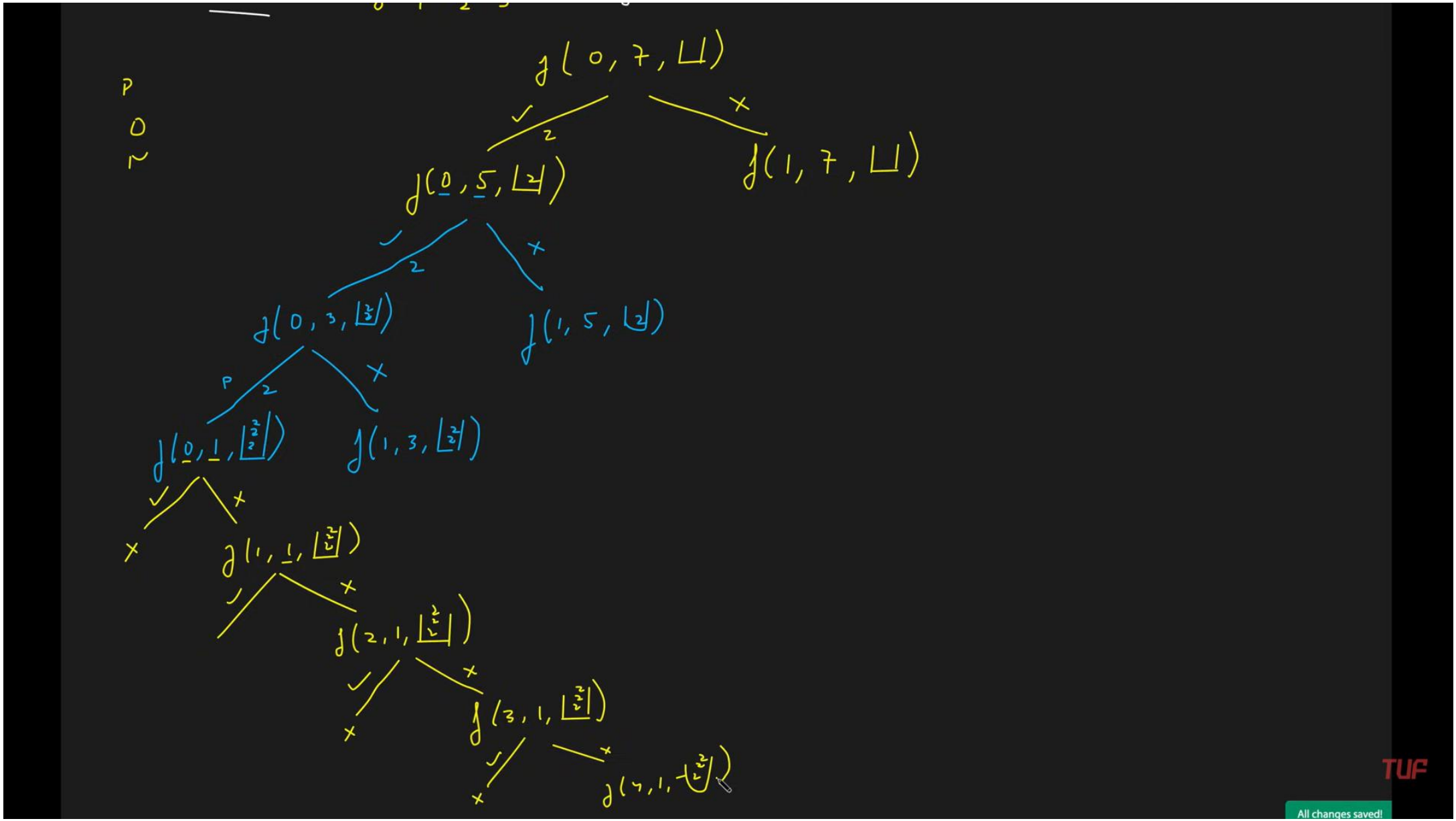


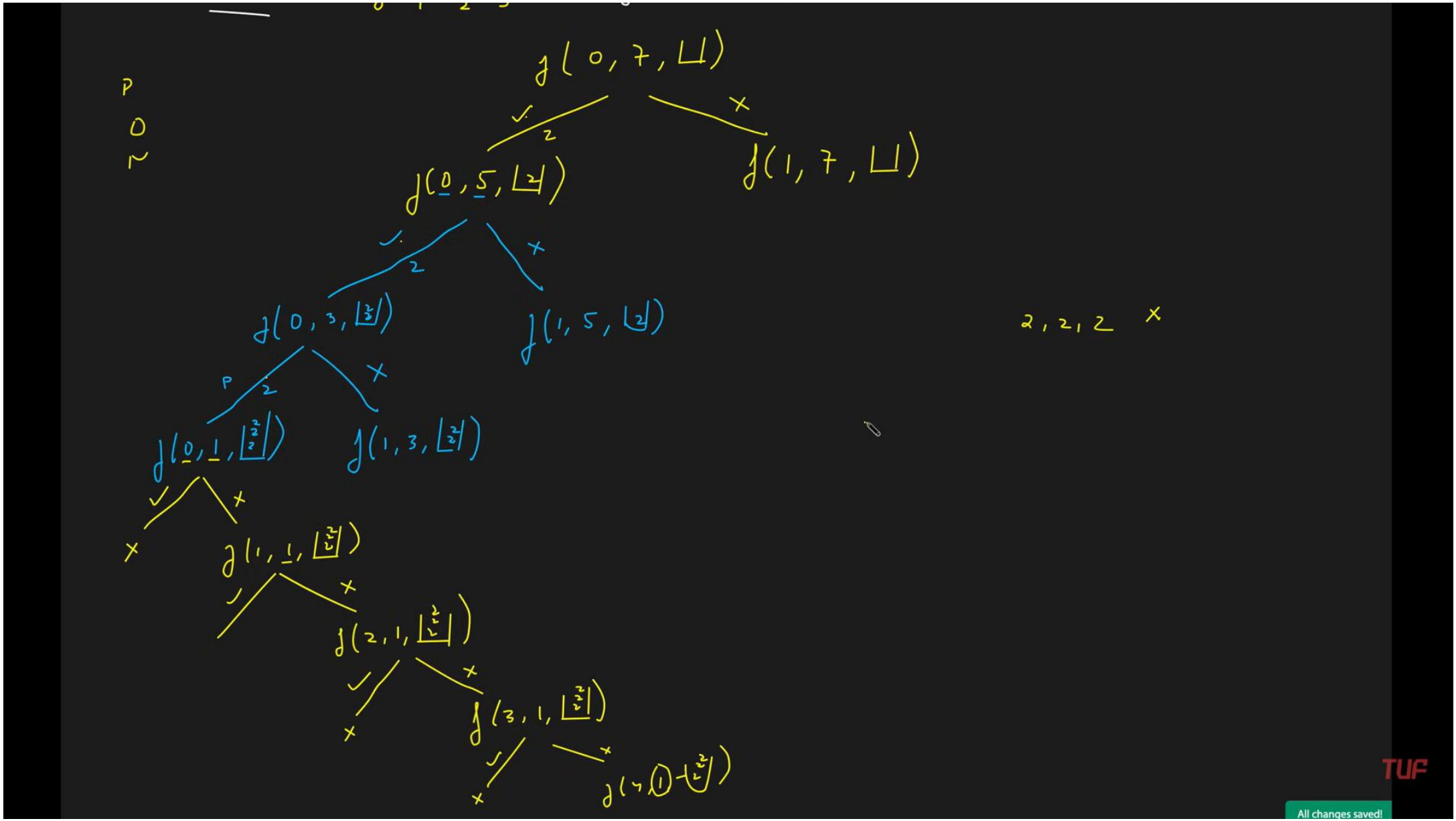


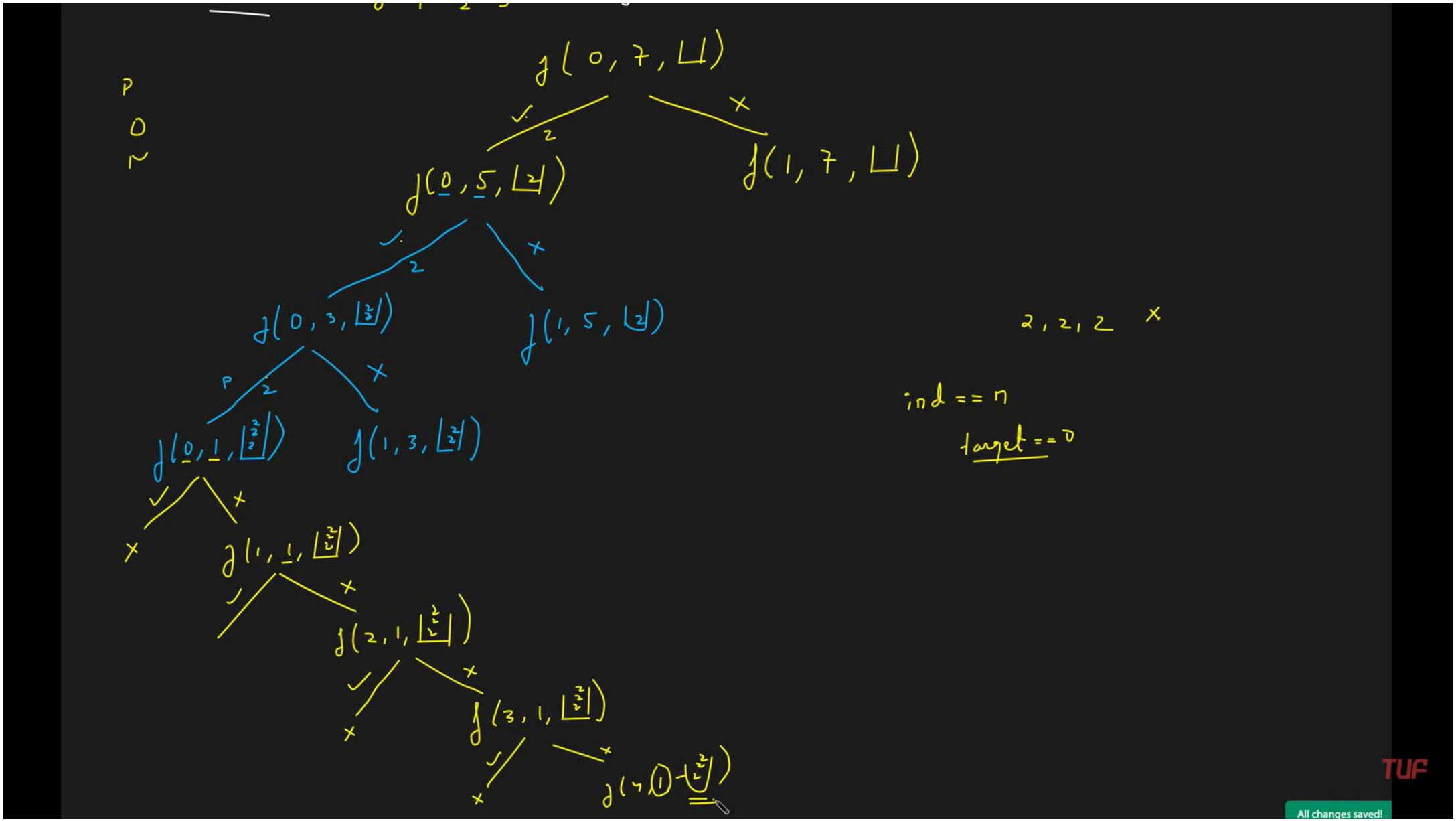
TUF

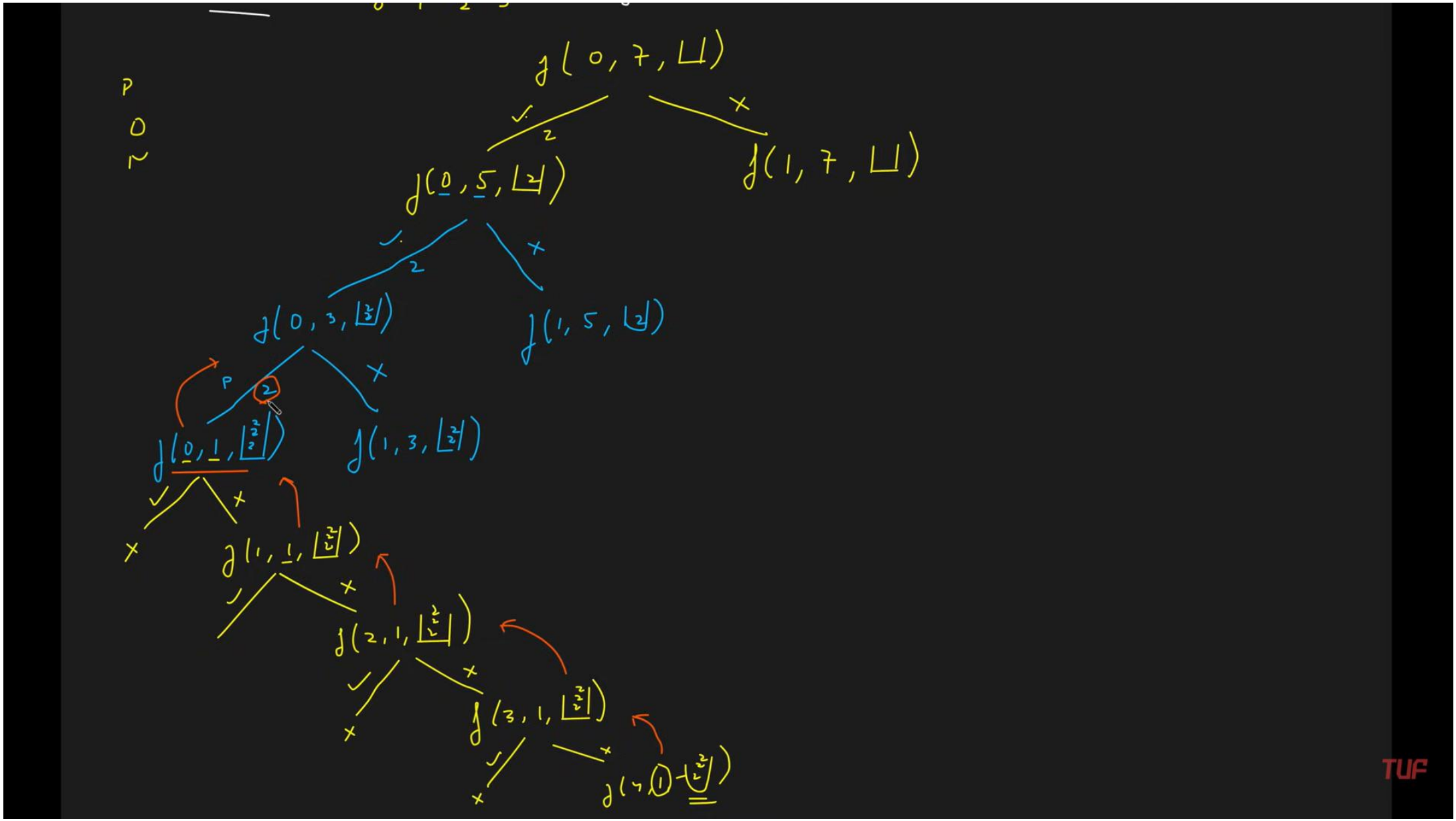


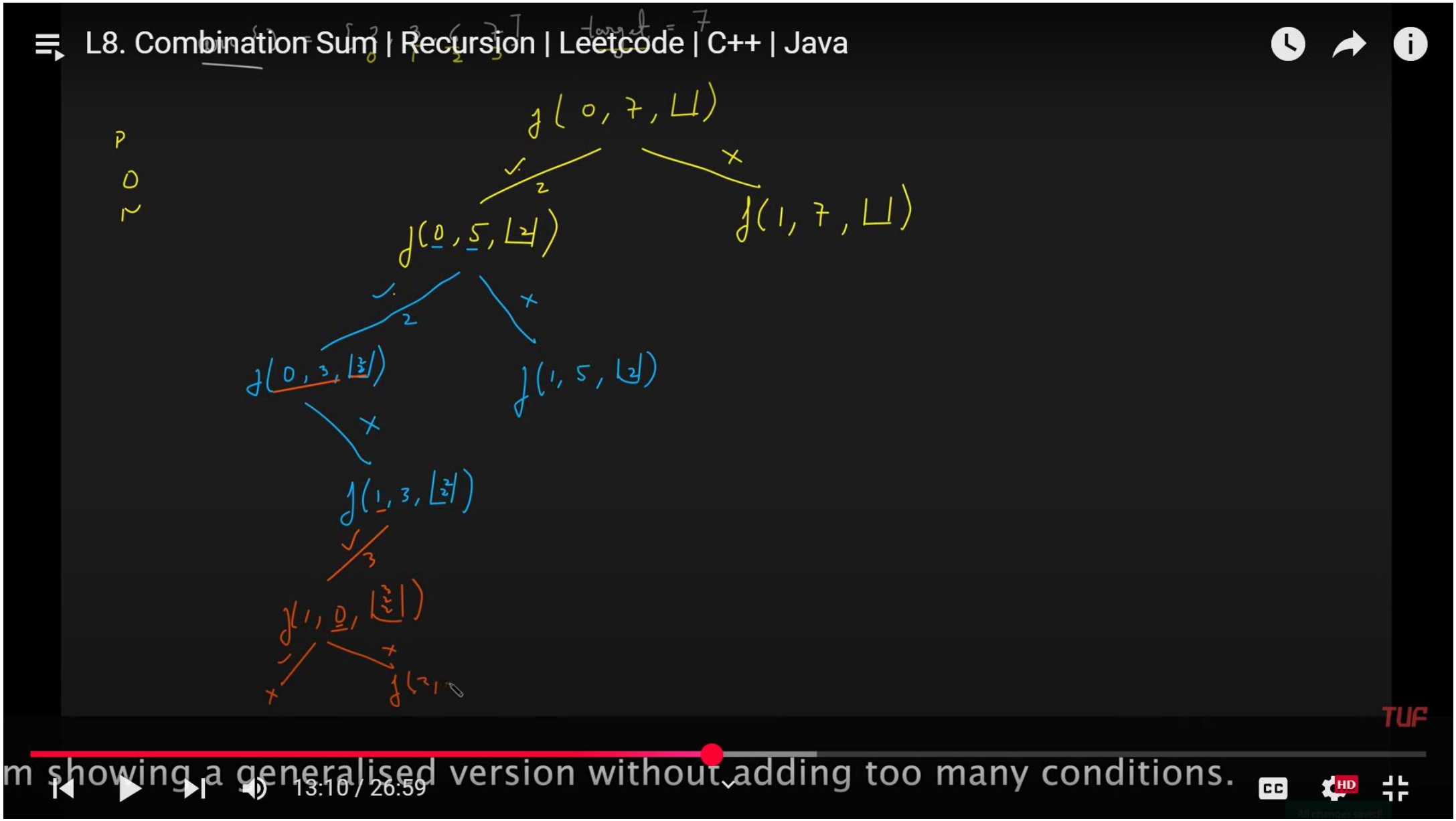
TUF
Autosaving

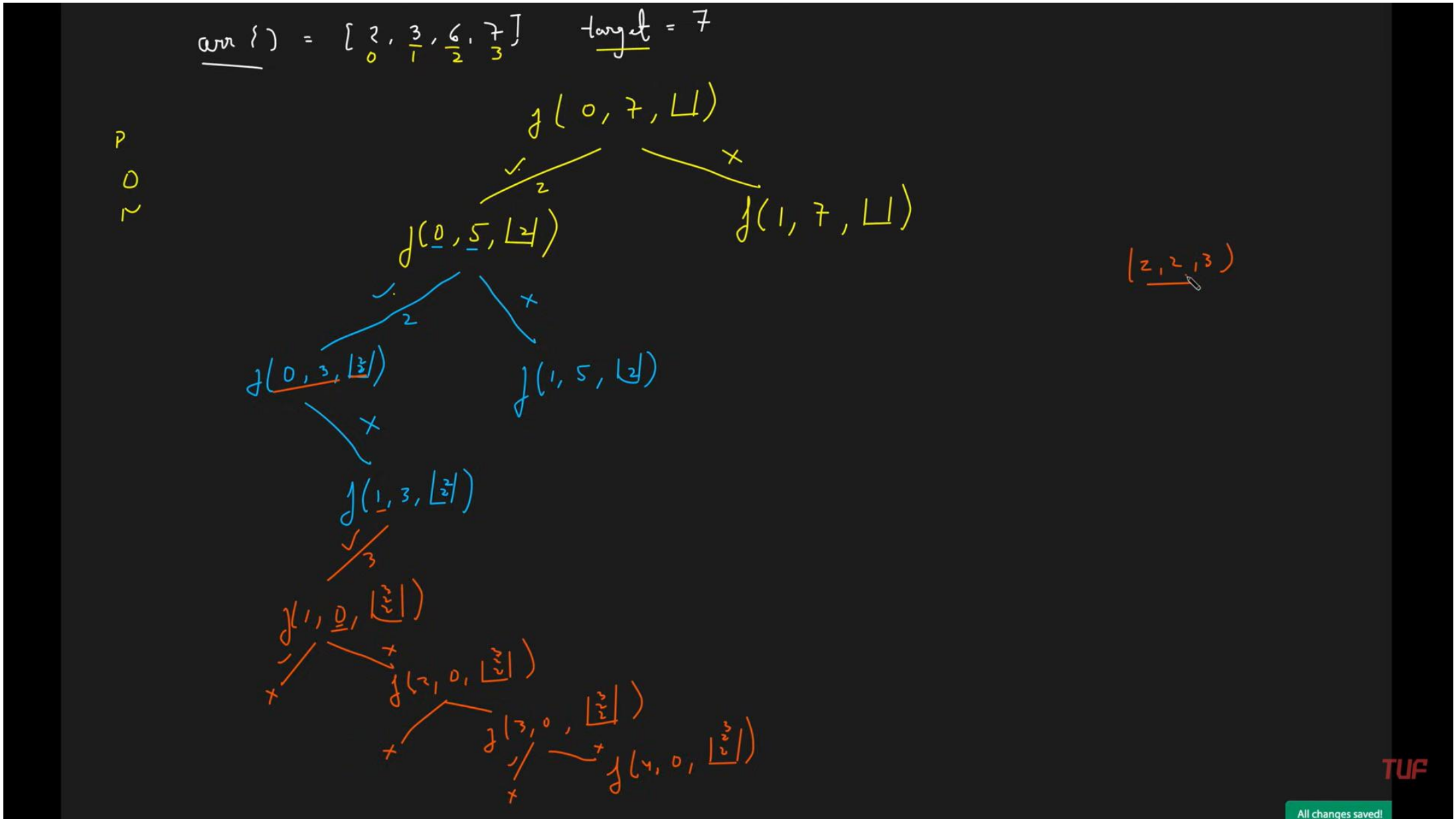

f(0, 7, ⌊1⌋)
f(0, 5, ⌊½⌋)
f(1, 7, ⌊1⌋)
f(0, 3, ⌊⅓⌋)
f(1, 5, ⌊½⌋)
f(0, 1, ⌊⅔⌋)
f(1, 3, ⌊⅓⌋)
f(1, 1, ⌊⅔⌋)
f(2, 1, ⌊⅔⌋)
f(3, 1, ⌊⅔⌋)
f(4, 1, ⌊⅔⌋)
P
2
x
All changes saved!

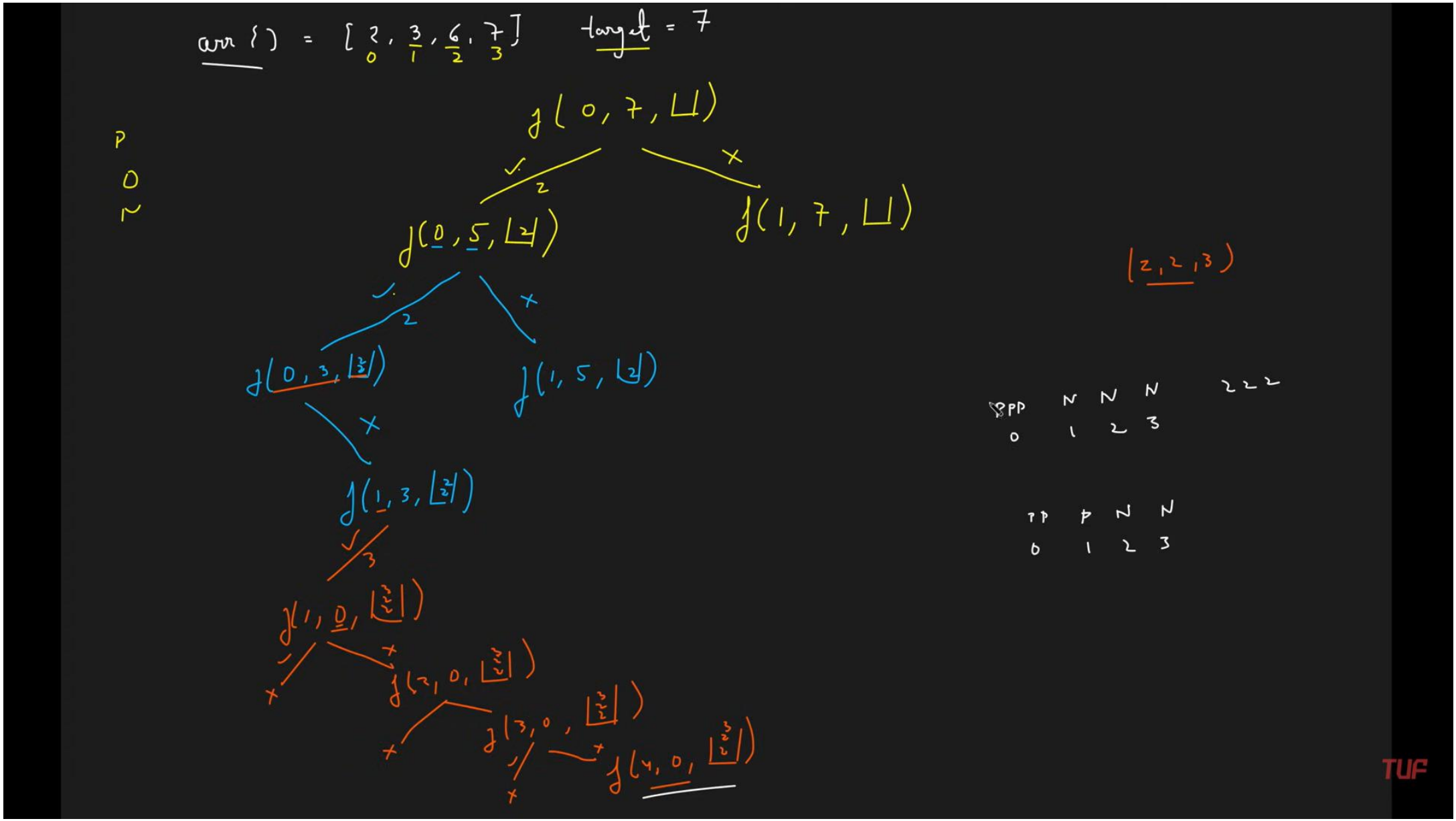

f(0, 7, ⌊7/2⌋)
f(0, 5, ⌊5/2⌋)
f(1, 7, ⌊7/2⌋)
f(0, 3, ⌊3/2⌋)
f(1, 5, ⌊5/2⌋)
f(0, 1, ⌊1/2⌋)
f(1, 3, ⌊3/2⌋)
f(1, 1, ⌊1/2⌋)
f(2, 1, ⌊2/2⌋)
f(3, 1, ⌊3/2⌋)
2, 2, 2 x
All changes saved!

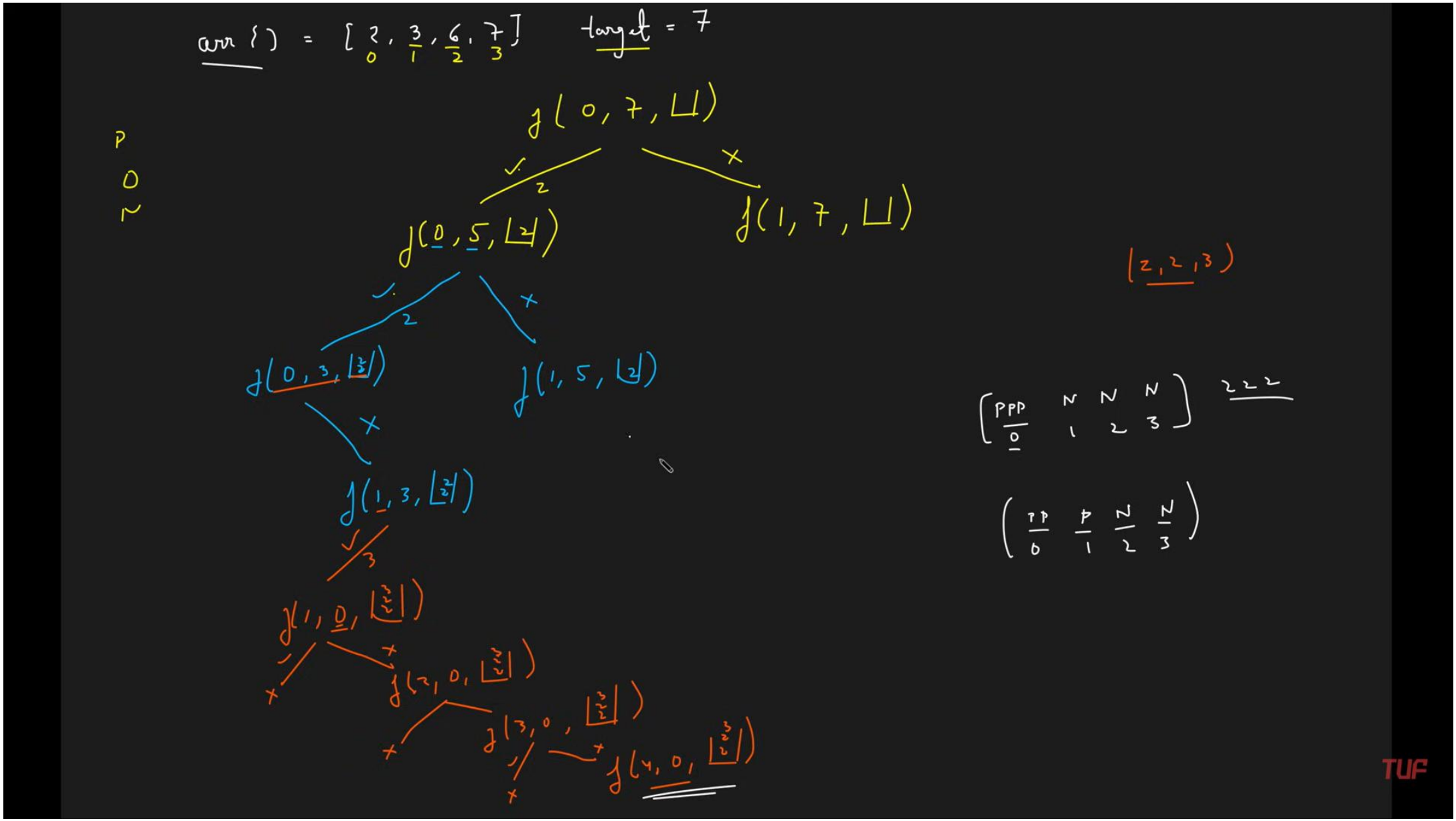

f(0, 7, [1])
f(0, 5, [2])
f(1, 7, [1])
f(0, 3, [2,2])
f(1, 5, [2])
f(0, 1, [2,2,2])
f(1, 3, [2,2])
f(1, 1, [2,2,2])
f(2, 1, [2,2,2])
f(3, 1, [2,2,2])
2, 2, 2
ind == n
target == 0

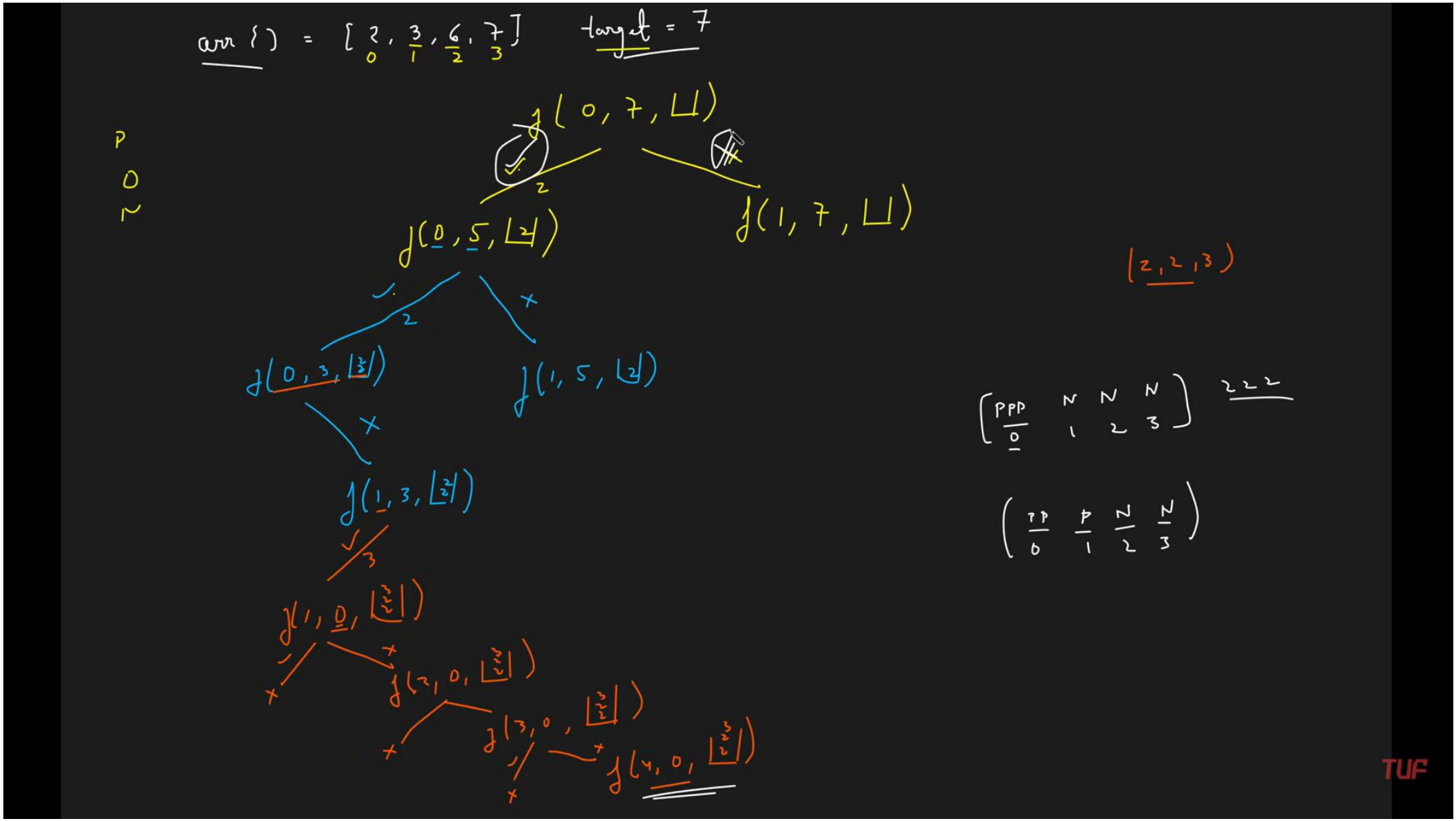

f(0, 7, 11)
f(0, 5, 4)
f(1, 7, 11)
f(0, 3, 2)
f(1, 5, 2)
f(0, 1, 1)
f(1, 3, 3)
f(1, 1, 1)
f(2, 1, 1)
f(3, 1, 1)
f(4, 1, 1)
P
2
TUF

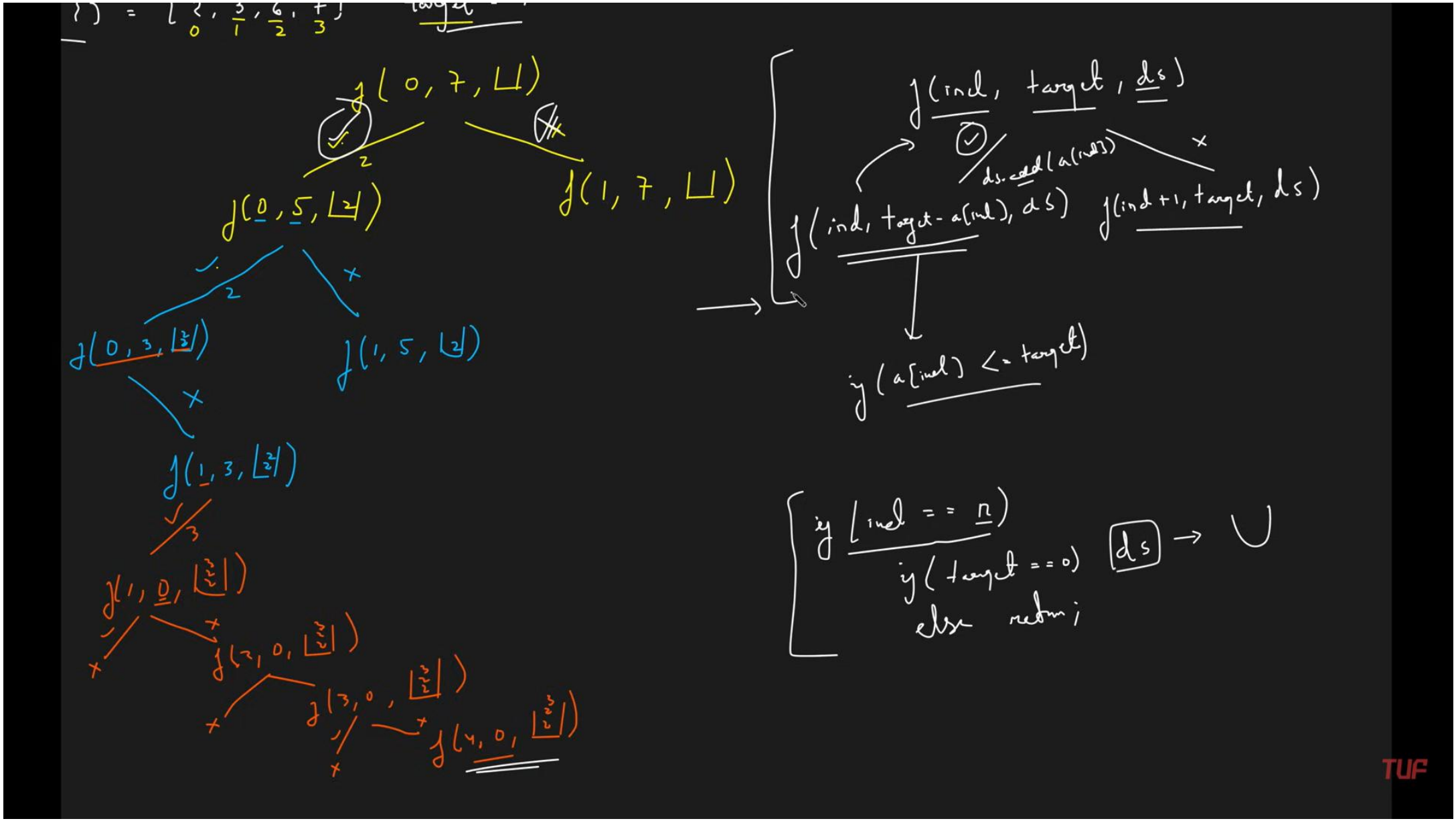

L8. Combination Sum | Recursion | Leetcode | C++ | Java
f(0, 7, [])
f(0, 5, [2])
f(1, 7, [])
f(0, 3, [2,2])
f(1, 5, [2])
f(1, 3, [2,2])
f(1, 0, [2,2,3])
TUF
m showing a generalised version without adding too many conditions.
13:10 / 26:59


arr[] = [2, 3, 6, 7]
target = 7
f(0, 7, [])
f(0, 5, [2])
f(1, 7, [])
f(0, 3, [2,3])
f(1, 5, [2])
f(1, 3, [2,3])
f(1, 0, [2,3])
f(2, 0, [2,3])
f(3, 0, [2,3])
f(4, 0, [2,3])
[2, 2, 3]
TUF
All changes saved!

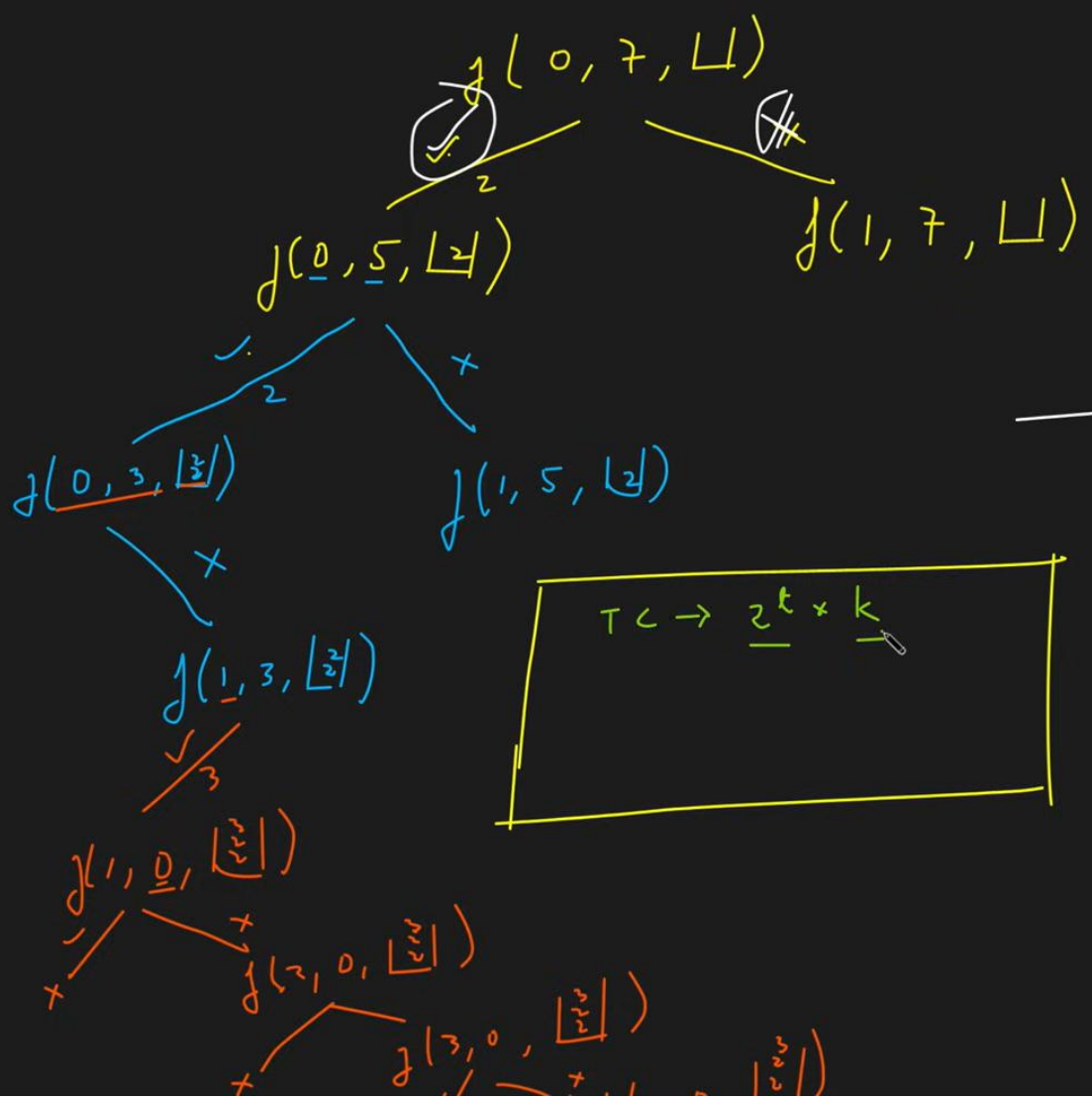

arr[] = [2, 3, 6, 7]
target = 7
f(0, 7, [])
f(0, 5, [2])
f(1, 7, [])
f(0, 3, [2,3])
f(1, 5, [2])
f(1, 3, [2,3])
f(1, 0, [2,2,3])
f(2, 0, [2,2,3])
f(3, 0, [2,2,3])
f(4, 0, [2,2,3])
[2, 2, 3]
TUF

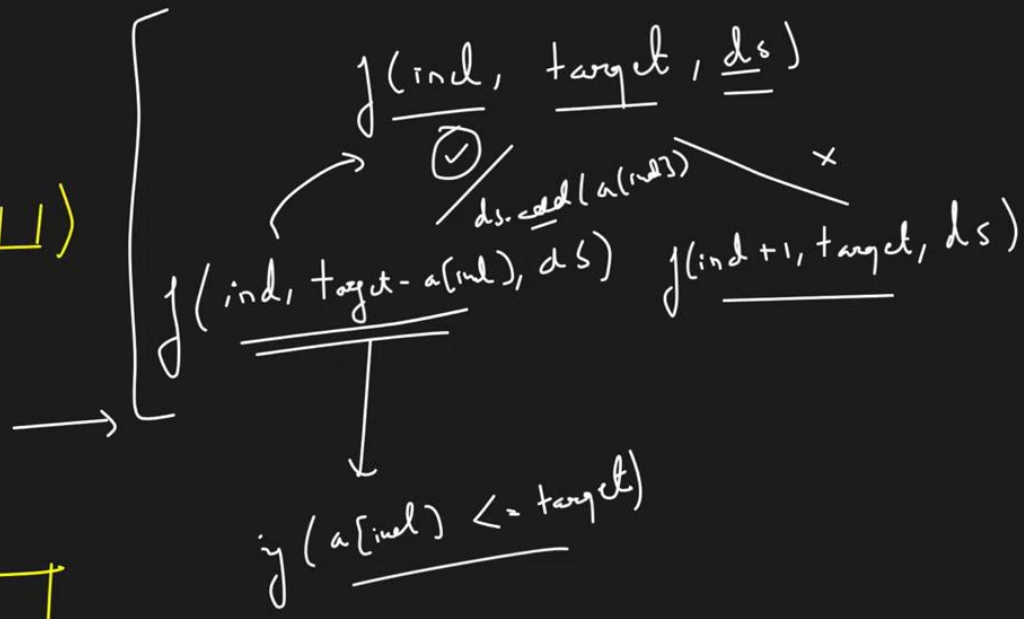

arr[] = [2, 3, 6, 7]
target = 7
f(0, 7, [])
f(0, 5, [2])
f(1, 7, [])
f(0, 3, [2,3])
f(1, 5, [2])
f(1, 3, [2,3])
f(1, 0, [2,2,3])
f(2, 0, [2,2,3])
f(3, 0, [2,2,3])
f(4, 0, [2,2,3])
[2, 2, 3]
TUF

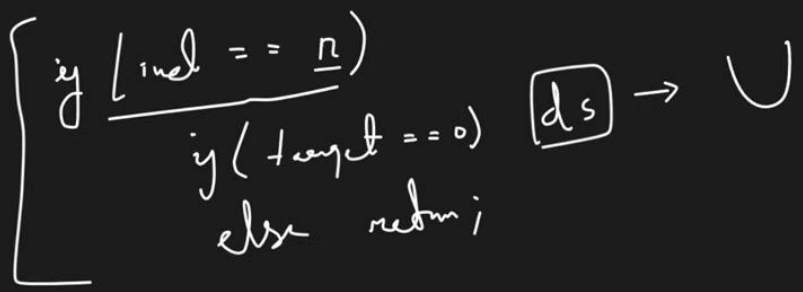

arr[] = [2, 3, 6, 7]
target = 7
f(0, 7, [])
f(0, 5, [2])
f(1, 7, [])
f(0, 3, [2,3])
f(1, 5, [2])
f(1, 3, [3])
f(1, 0, [3,3])
f(2, 0, [2,3])
f(3, 0, [2,3])
f(4, 0, [2,3])
[2, 2, 3)
2 2 2
TUF


f(0, 7, [])
f(1, 7, [])
f(0, 5, [2])
f(0, 3, [2,2])
f(1, 5, [2])
f(1, 3, [2,3])
f(1, 0, [2,3,2])
f(2, 0, [2,3,2])
f(3, 0, [2,3,2])
f(4, 0, [2,3,2])
f(ind, target, ds)
ds.add(a[ind])
f(ind, target - a[ind], ds)
f(ind+1, target, ds)
if(a[ind] <= target)
if(ind == n)
if(target == 0) ds → ∪
else return;
TUF

arr[] = [2, 3, 6, 7] target = 7

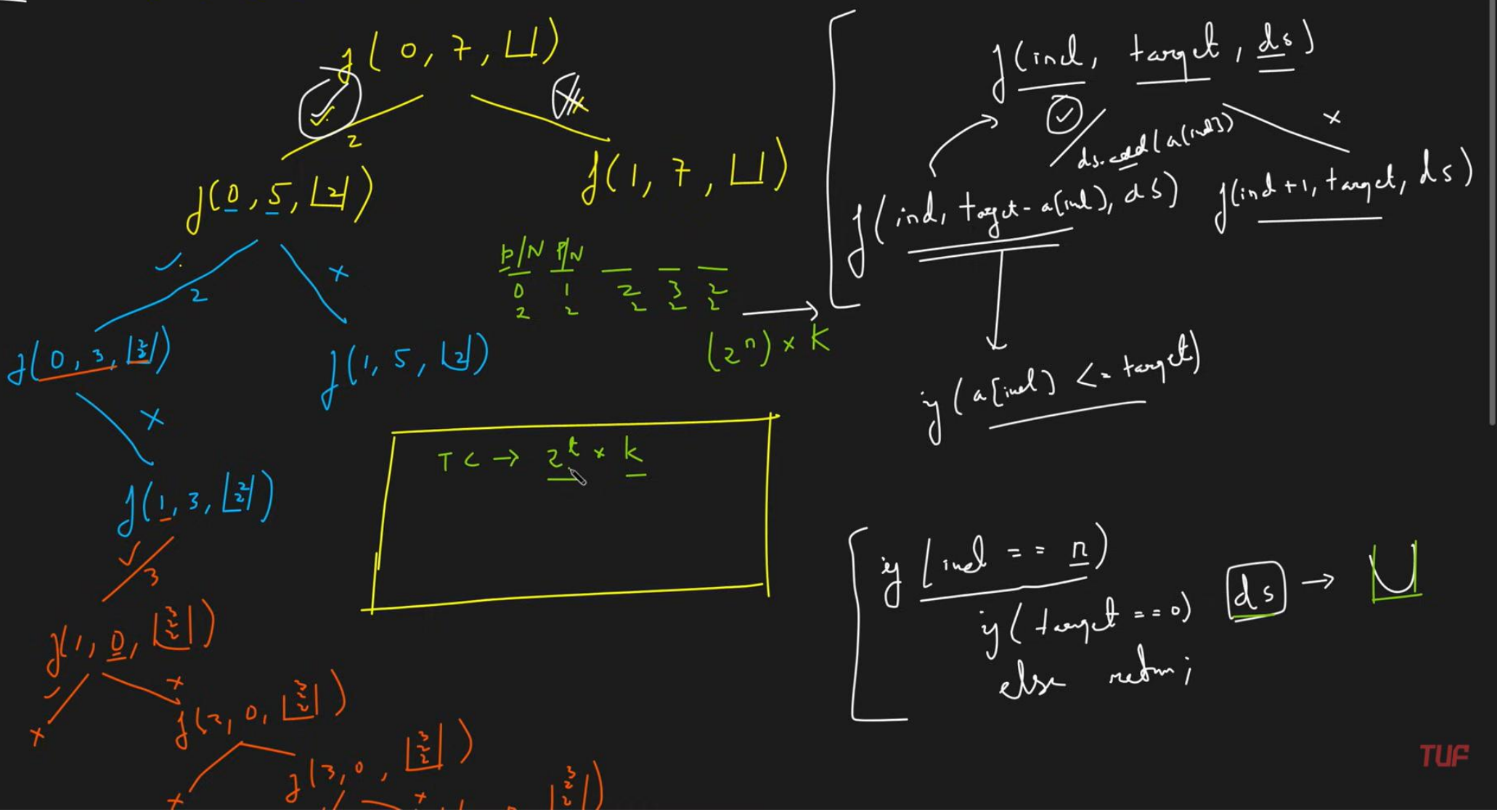




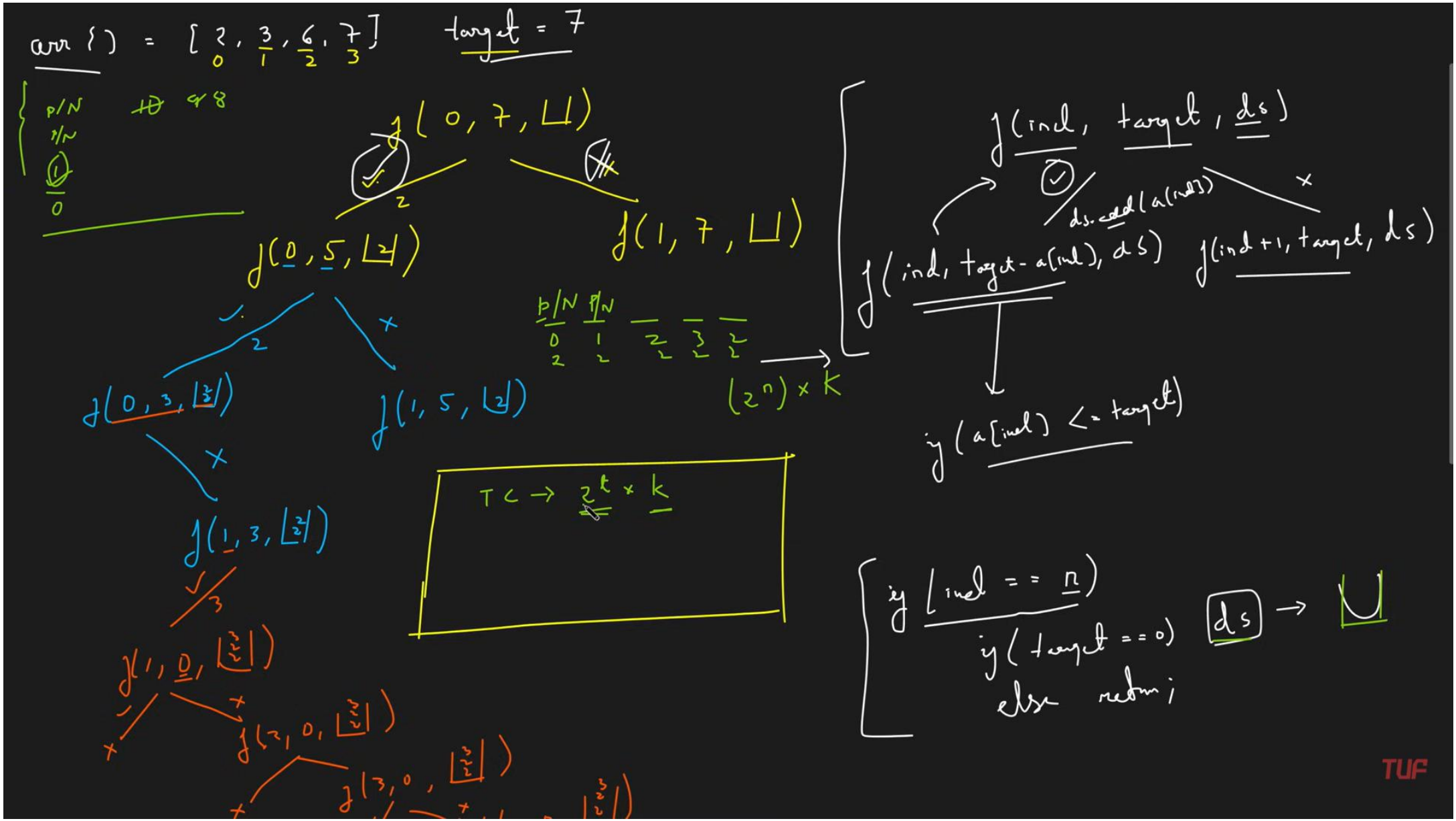


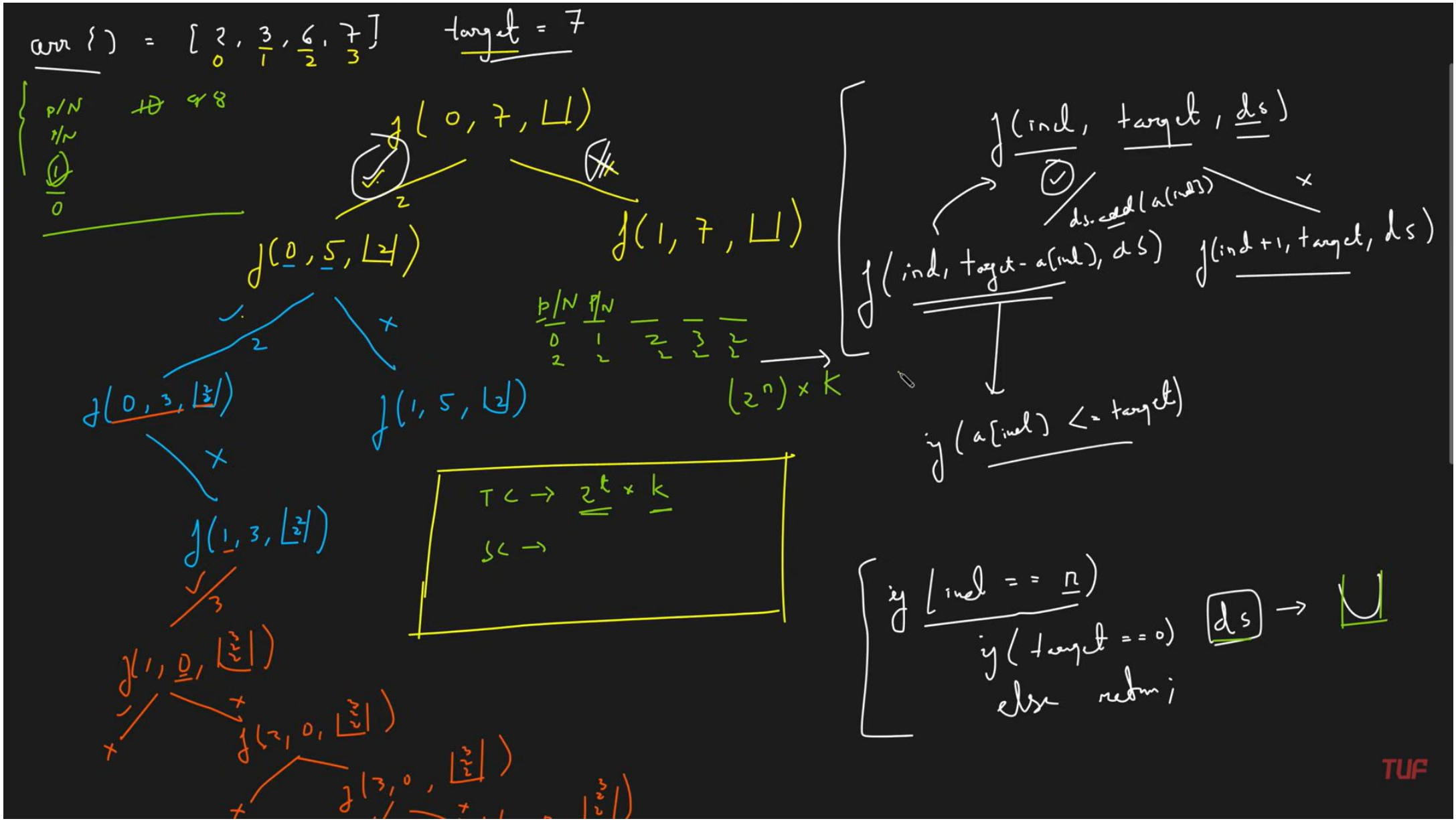
TUF

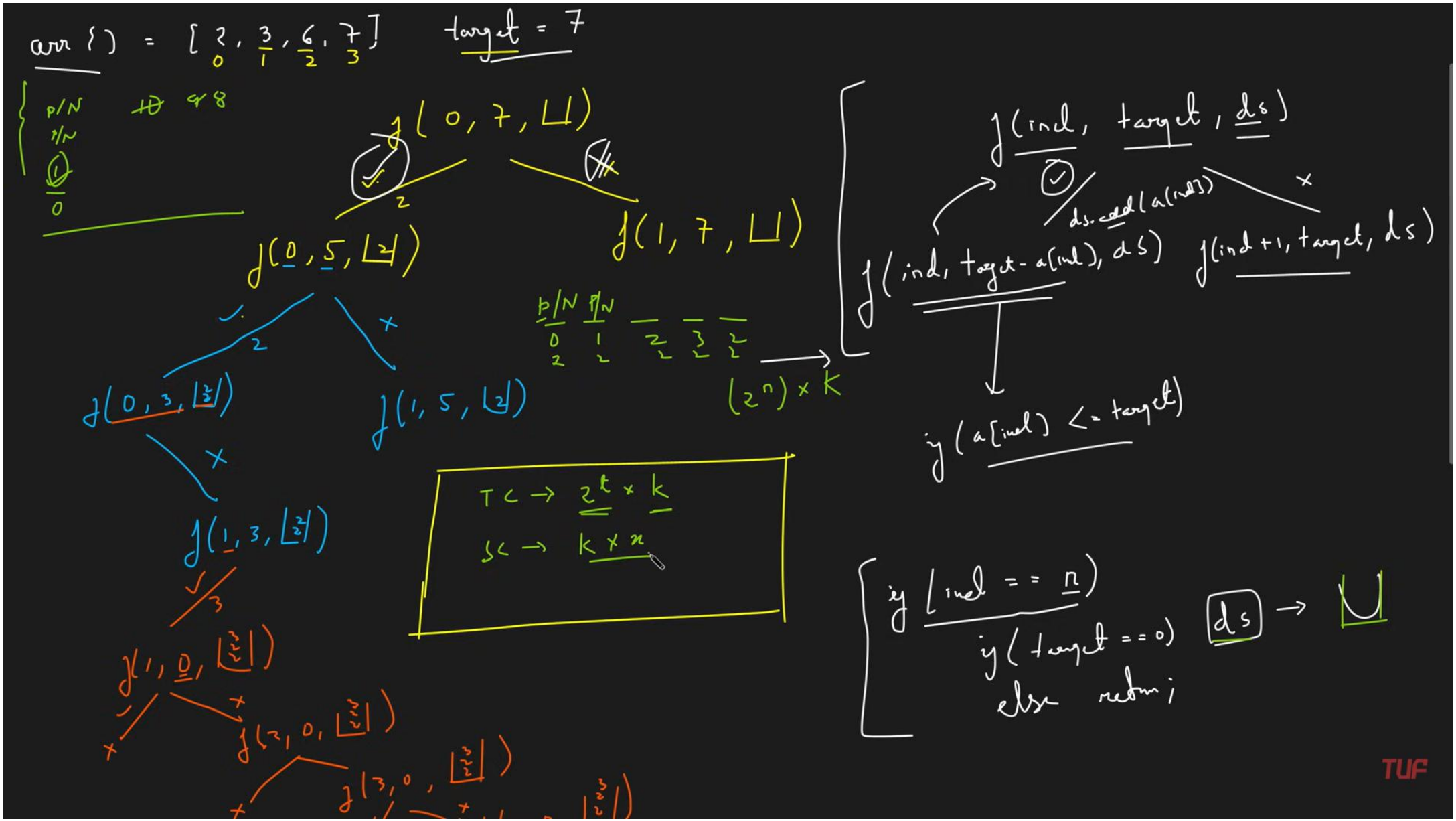
arr[] = [2, 3, 6, 7] target = 7



TUF


arr[] = [2, 3, 6, 7]
target = 7
0 1 2 3
P/N
1/N
1
0
f(0, 7, [])
2
f(0, 5, [2])
f(1, 7, [])
f(0, 3, [2,3])
f(1, 5, [2])
f(1, 3, [2,3])
3
f(1, 0, [2,3])
f(2, 0, [2,3])
f(3, 0, [2,3])
P/N P/N
0 1 2 3
2 2 2 2
(2^n) x k
T.C -> 2^t x k
f(ind, target, ds)
ds.add(a[ind])
f(ind, target - a[ind], ds)
f(ind+1, target, ds)
if(a[ind] <= target)
if(ind == n)
if(target == 0) ds
else return;
TUF


arr[] = [2, 3, 6, 7]
target = 7
P/N
N/N
1
0
f(0, 7, [])
2
f(0, 5, [2])
f(1, 7, [])
f(0, 3, [2,3])
f(1, 5, [2])
f(1, 3, [2,3])
3
f(1, 0, [2,3])
f(2, 0, [2,3])
f(3, 0, [2,3])
P/N P/N
0 1 2 3 2
2 2 2 2 2
(2^n) x k
T <-> 2^t x k
S <->
f(ind, target, ds)
ds.add(a[ind])
f(ind, target - a[ind], ds)
f(ind+1, target, ds)
f(a[ind] <= target)
if(ind == n)
if(target == 0) ds
else return;
TUF


arr[] = [2, 3, 6, 7]
target = 7
0 1 2 3
P/N
1/N
1
0
f(0, 7, ⌊⌋)
2
f(0, 5, ⌊2⌋)
f(1, 7, ⌊⌋)
f(0, 3, ⌊3/2⌋)
f(1, 5, ⌊2⌋)
f(1, 3, ⌊3/2⌋)
3
f(1, 0, ⌊3/2⌋)
f(2, 0, ⌊3/2⌋)
f(3, 0, ⌊3/2⌋)
P/N P/N
0 1 2 3 2
2 2 2 2
(2^n) x k
T.C → 2^t x k
S.C → k x n
f(ind, target, ds)
ds.add(a[ind])
f(ind, target - a[ind], ds)
f(ind+1, target, ds)
if(a[ind] <= target)
if(ind == n)
if(target == 0) ds → ⌊⌋
else return;
TUF

Java Autocomplete

```java
class Solution {

    private void findCombinations(int ind, int[] arr, int target, List<List<Integer>> ans, List<Integer> ds) {
        if(ind == arr.length) {
            if(target == 0) {
                ans.add(new ArrayList<>(ds));
            }
            return;
        }

        if(arr[ind] <= target) {
            ds.add(arr[ind]);
            findCombinations(ind, arr, target - arr[ind], ans, ds);
            ds.remove(ds.size() - 1);
        }
        findCombinations(ind + 1, arr, target, ans, ds);
    }
    public List<List<Integer>> combinationSum(int[] candidates, int target) {
        List<List<Integer>> ans = new ArrayList<>();
        findCombinations(0, candidates, target, ans, new ArrayList<>());
        return ans;
    }
}
```

TUF

Java

Autocomplete

```java
class Solution {

    private void findCombinations(int ind, int[] arr, int target, List<List<Integer>> ans, List<Integer> ds) {
        if(ind == arr.length) {
            if(target == 0) {
                ans.add(new ArrayList<>(ds));
            }
            return;
        }

        if(arr[ind] <= target) {
            ds.add(arr[ind]);
            findCombinations(ind, arr, target - arr[ind], ans, ds);
            ds.remove(ds.size() - 1);
        }
        findCombinations(ind + 1, arr, target, ans, ds);
    }
    public List<List<Integer>> combinationSum(int[] candidates, int target) {
        List<List<Integer>> ans = new ArrayList<>();
        findCombinations(0, candidates, target, ans, new ArrayList<>());
        return ans;
    }
}
```

TUF

Java Autocomplete

```java
class Solution {

    private void findCombinations(int ind, int[] arr, int target, List<List<Integer>> ans, List<Integer> ds) {
        if(ind == arr.length) {
            if(target == 0) {
                ans.add(new ArrayList<>(ds));
            }
            return;
        }

        if(arr[ind] <= target) {
            ds.add(arr[ind]);
            findCombinations(ind, arr, target - arr[ind], ans, ds);
            ds.remove(ds.size() - 1);
        }
        findCombinations(ind + 1, arr, target, ans, ds);
    }
    public List<List<Integer>> combinationSum(int[] candidates, int target) {
        List<List<Integer>> ans = new ArrayList<>();
        findCombinations(0, candidates, target, ans, new ArrayList<>());
        return ans;
    }
}
```

TUF

C++ Autocomplete

```cpp
class Solution {
public:
    void findCombination(int ind, int target, vector<int> &arr, vector<vector<int>> &ans, vector<int>&ds) {
        if(ind == arr.size()) {
            if(target == 0) {
                ans.push_back(ds);
            }
            return;
        }
        // pick up the element
        if(arr[ind] <= target) {
            ds.push_back(arr[ind]);
            findCombination(ind, target - arr[ind], arr, ans, ds);
            ds.pop_back();
        }

        findCombination(ind+1, target, arr, ans, ds);

    }
public:
    vector<vector<int>> combinationSum(vector<int>& candidates, int target) {
        vector<vector<int>> ans;
        vector<int> ds;
        findCombination(0, target, candidates, ans, ds);
        return ans;
    }
};
```

TUF