26 May 2023 10:35

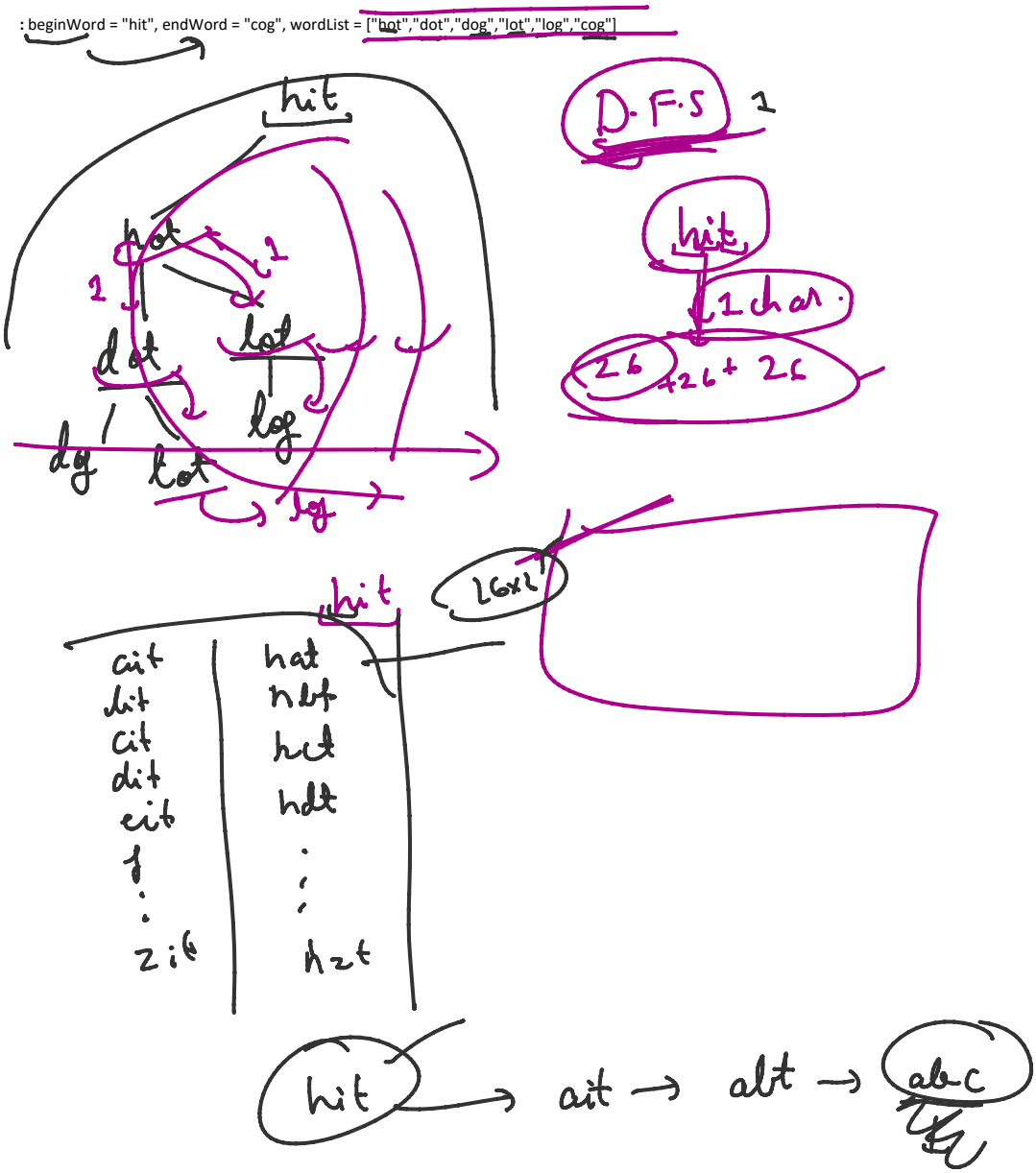
A **transformation sequence** from word beginWord to word endWord using a dictionary wordList is a sequence of words beginWord -> s_1 -> s_2 -> ... -> s_k such that:

- Every adjacent pair of words differs by a single letter.
- Every s_i for 1 <= i <= k is in wordList. Note that beginWord does not need to be in wordList.
- s_k == endWord

Given two words, beginWord and endWord, and a dictionary wordList, return *the* ***number of words*** *in the* ***shortest transformation sequence*** *from* beginWord *to* endWord*, or* 0 *if no such sequence exists.*

Example 1:
Input: beginWord = "hit", endWord = "cog", wordList = ["hot","dot","dog","lot","log","cog"]
Output: 5
Explanation: One shortest transformation sequence is "hit" -> "hot" -> "dot" -> "dog" -> cog", which is 5 words long.
Example 2:
Input: beginWord = "hit", endWord = "cog", wordList = ["hot","dot","dog","lot","log"]
Output: 0
Explanation: The endWord "cog" is not in wordList, therefore there is no valid transformation sequence.

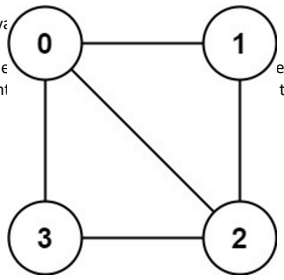
: beginWord = "hit", endWord = "cog", wordList = ["hot","dot","dog","lot","log","cog"]



There is an **undirected** graph with n nodes, where each node is numbered between 0 and n - 1. You are given a 2D array graph, where graph[u] is an array of nodes that node u is adjacent to. More formally, for each v in graph[u], there is an undirected edge between node u and node v. The graph has the following properties:

- There are no self-edges (graph[u] does not contain u).
- There are no parallel edges (graph[u] does not contain duplicate v
- If v is in graph[u], then u is in graph[v] (the graph is undirected).
- The graph may not be connected, meaning there may be two node etween them.

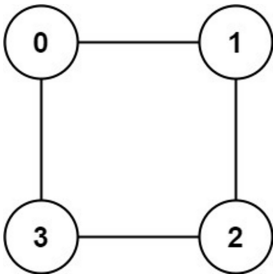
A graph is **bipartite** if the nodes can be partitioned into two independent the graph connects a node in set A and a node in set B.
Return true *if and only if it is* ***bipartite***.



Example 1:
A graph is **bipartite** if the nodes can be partitioned into two independent sets A and B such that **every** edge in the graph connects a node in set A and a node in set B.

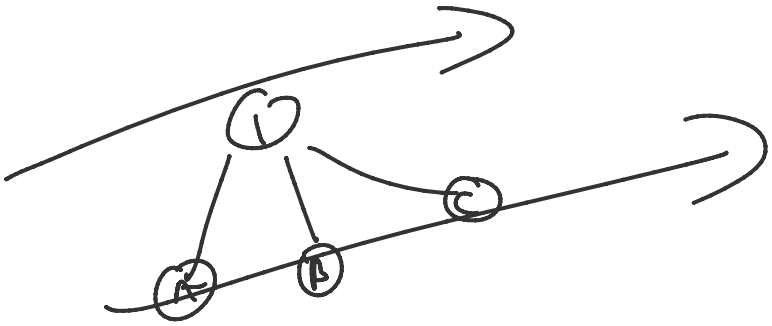
Input: graph = [[1,2,3],[0,2],[0,1,3],[0,2]]
Output: false
Explanation: There is no way to partition the nodes into two independent sets such that every edge connects a node in one and a node in the other.

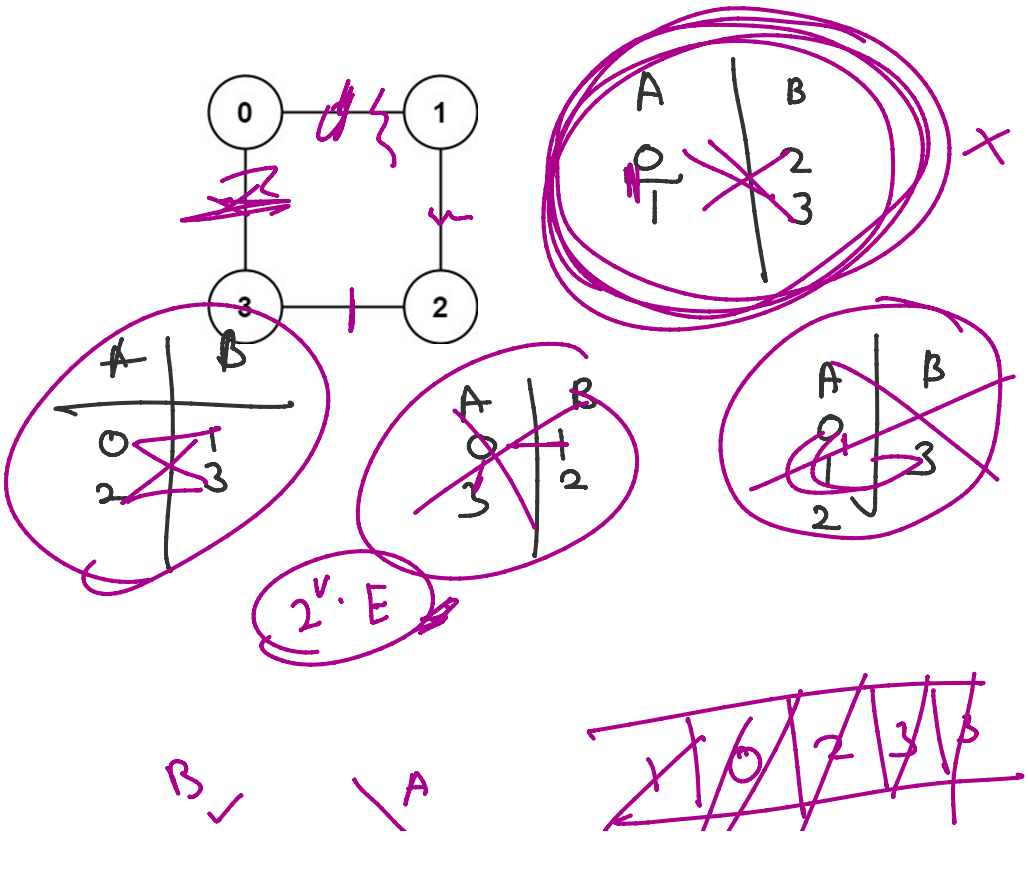
Example 2:

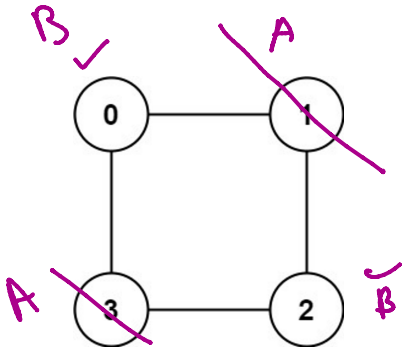


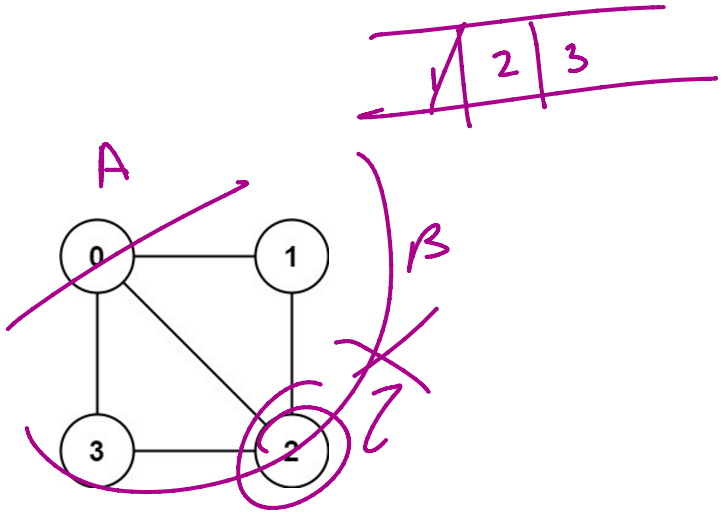
Input: graph = [[1,3],[0,2],[1,3],[0,2]]
Output: true
Explanation: We can partition the nodes into two sets: {0, 2} and {1, 3}.

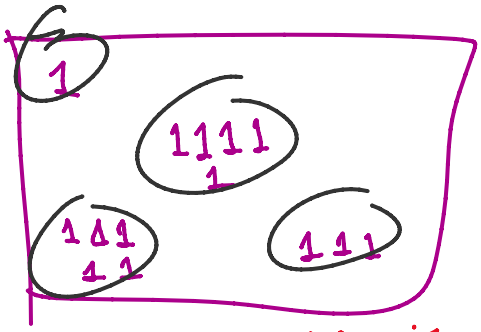
From <https://leetcode.com/problems/is-graph-bipartite/>



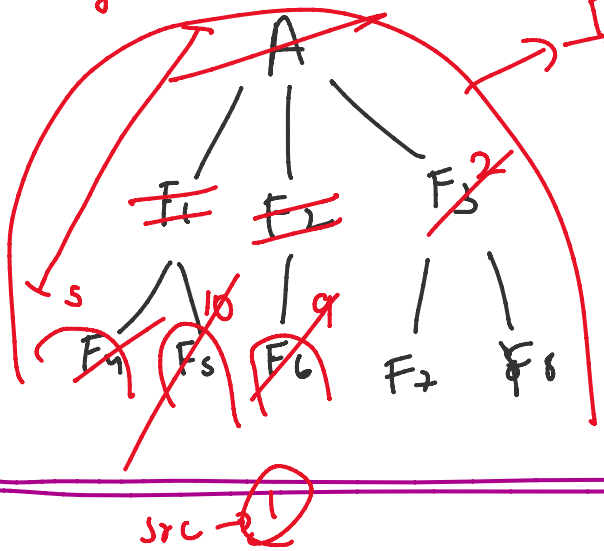




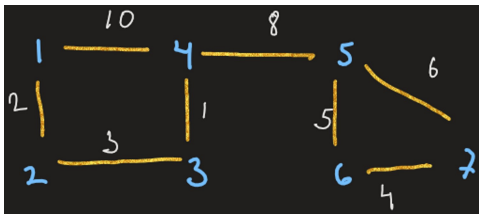


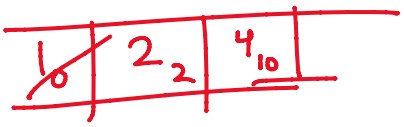


Single source shortest Algo :- Dijkstra

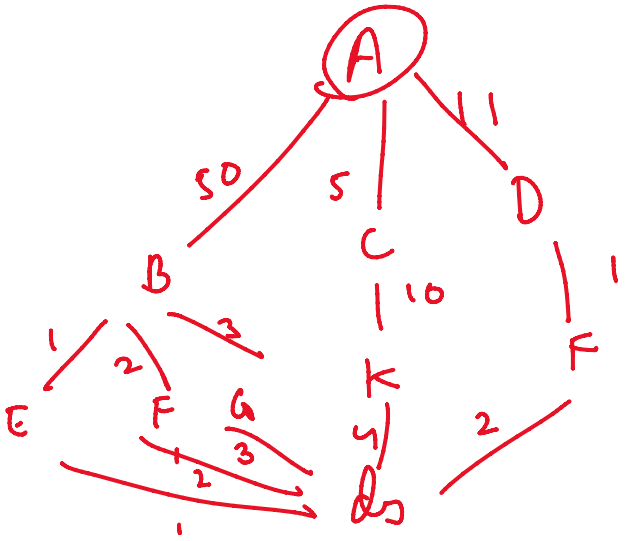


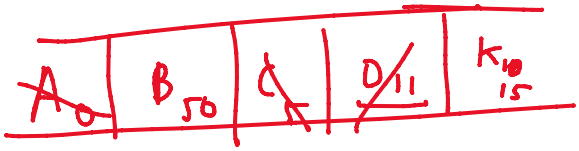
Src → 1



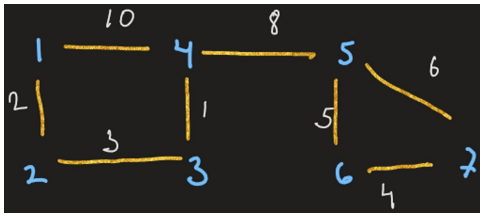


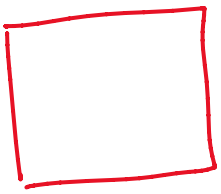
2





Q V_{dst}





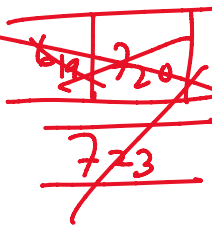
$\{1 \mapsto 0, 2 \to 2, 3 \to 5$

$4 \to 6$

$5 \to 4$

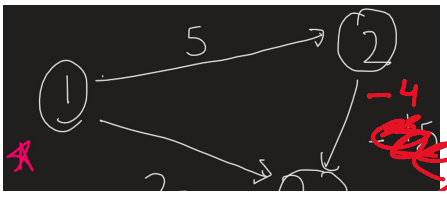
$6 \to 14$

$7 \to 10$

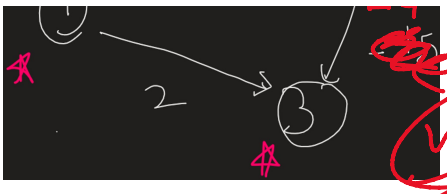


V 3

Bellman Ford



V+E



E log E

Dijkst Edge

Fib(0) → 0

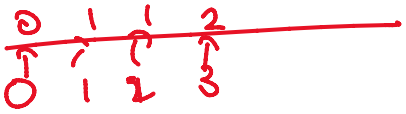
Fib(1) → 1

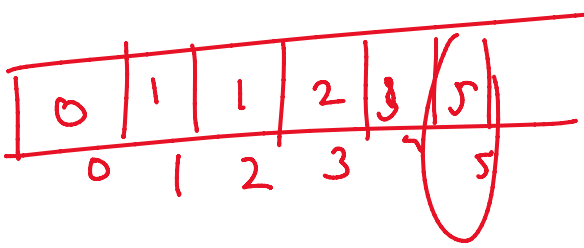
Fib(2) → 1

Fib(3) → 2

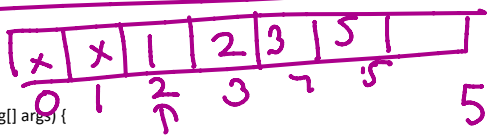
Fib(4) → 3

Fib(5) → 5



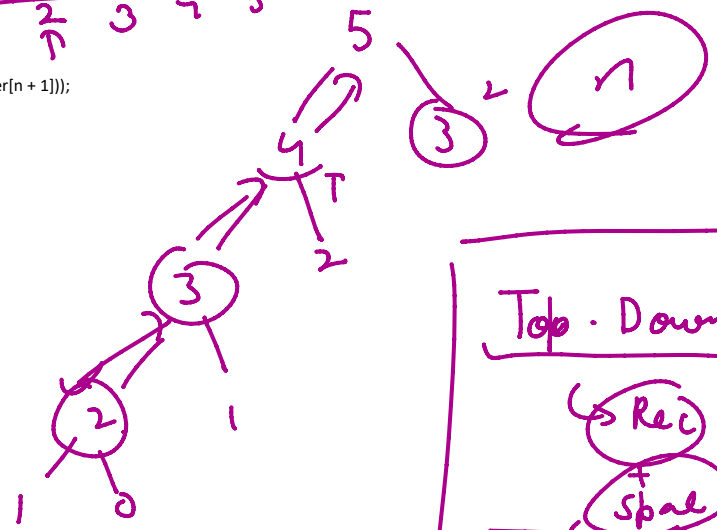


T.D



```
public static void main(String[] args) {
        int n = 5;
        System.out.println(Fibo(n, new Integer[n + 1]));
}

public static int Fibo(int n, Integer[] dp) {
        if (n <= 1) {
                return n;
        }
        if (dp[n] != null) {
                return dp[n];
        }
        int sp1 = Fibo(n - 1, dp);
        int sp2 = Fibo(n - 2, dp);
//          Fibo (n) = > index?!
        dp[n] = sp1 + sp2;
        return sp1 + sp2;
}
```



Top · Down / Memoized

Rec + Space

Memoization

1) Rewrite

2) D.S → Sol^n ?? } confuse

1) Recursive

2) D.S → sol^n ?? } confuse

3) D.S use

4) D.S ka Use

BU ← TD → D.S mein

D.S

1) smallest to Biggest Prob → loop

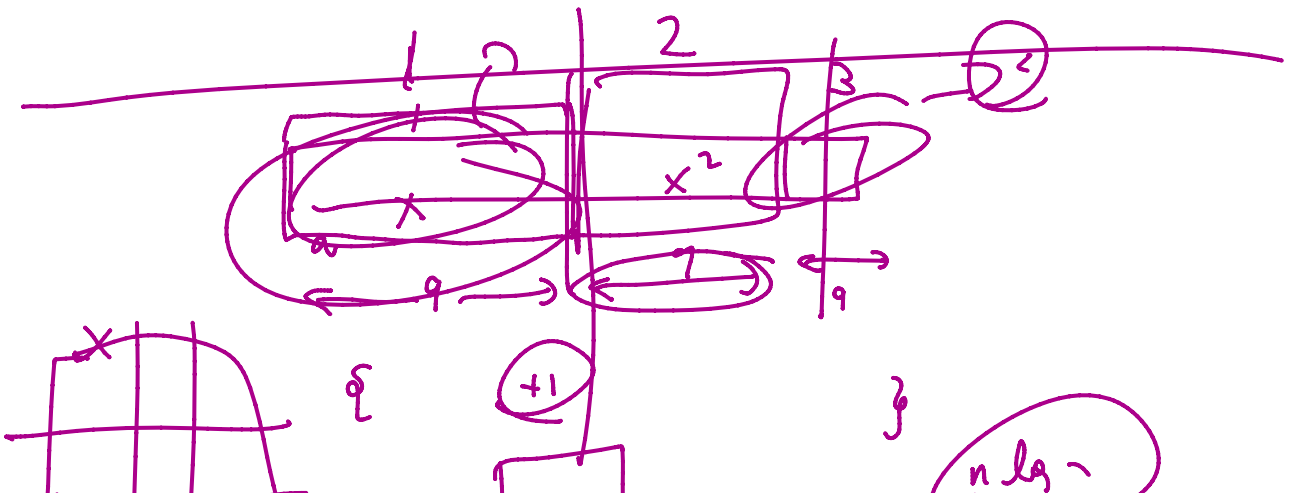
2) ~~Return~~ curr sol^n ← T.D. copy → recursive calls

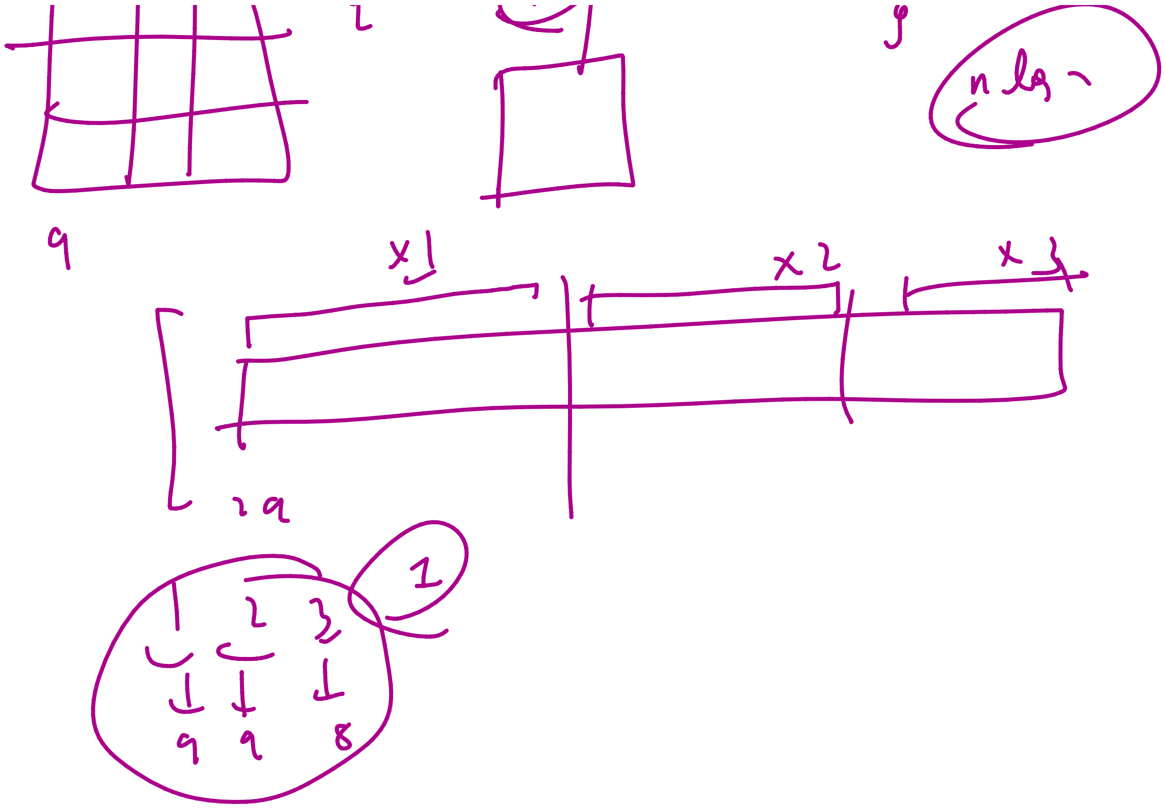
D.S ki form.

3) Base Case

4) Biggest Problem

~~5~~) D.S ka size




n lg n
9
x1
x2
x3
29
1
2
3
9
9
8