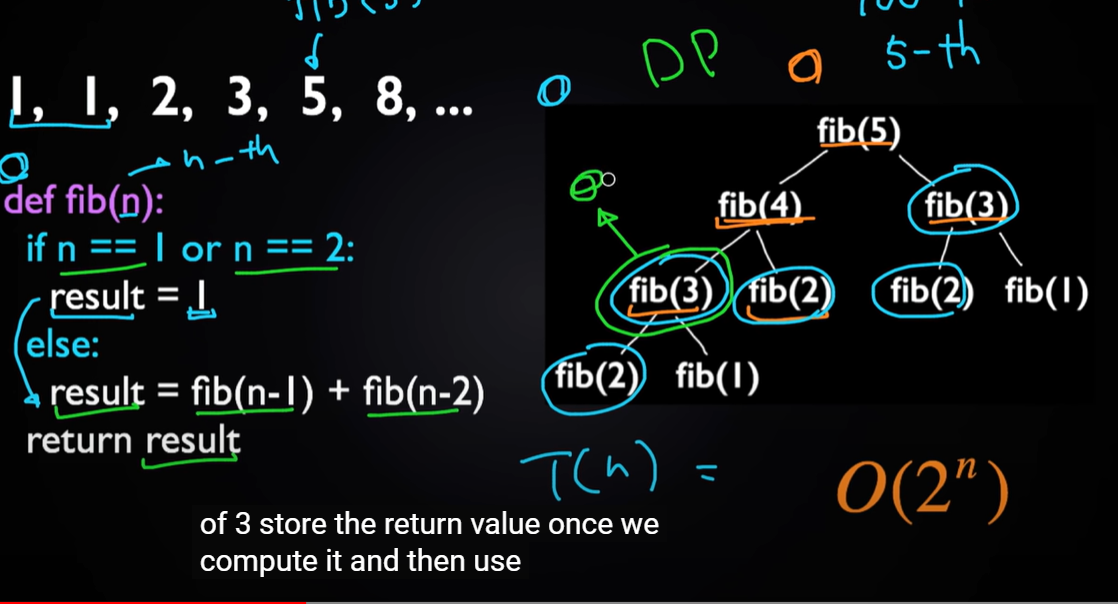


核心就是一个，remembering stuff to save time later

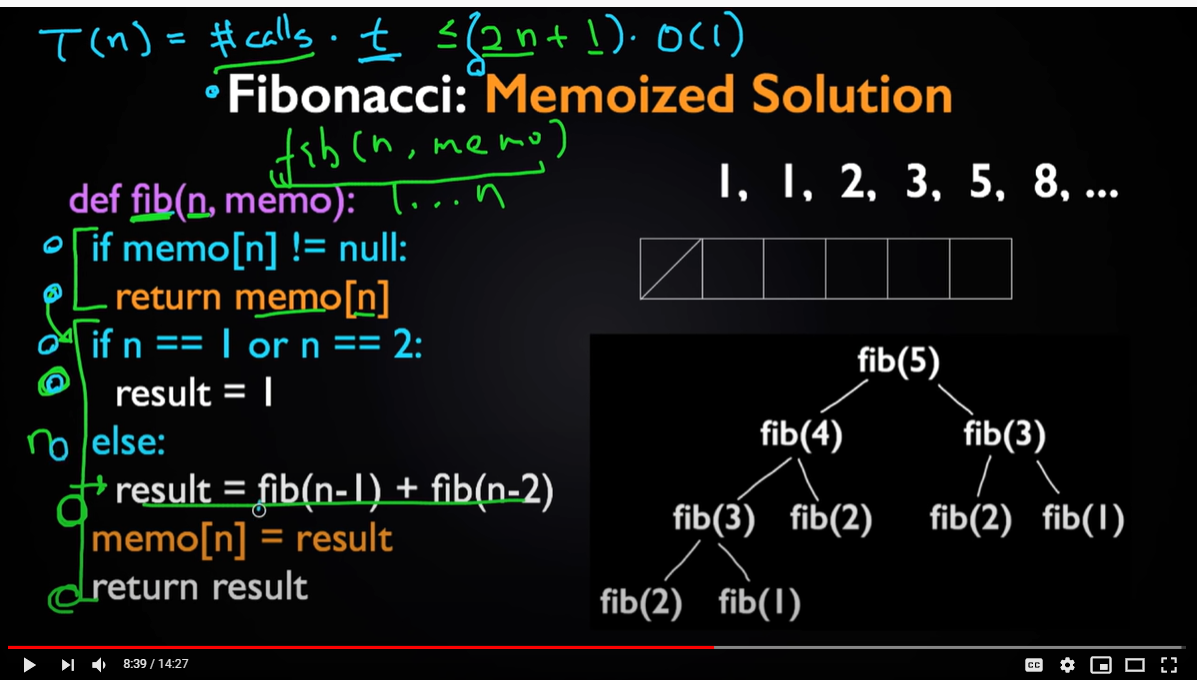
例如计算斐波那契数列



如果每一个都用recursion，那最后的复杂度是2^n

但是我们可以观察到，fib(2),fib(3)实际上是被重复计算的，

那么最简单的动态规划，我们可以创造一个array记录



时间被减少到2n+1，

it is n times because we don't need to call a recursion anymore since we already have the result of fib(n) inside the memo, that's why after the else block we do the operation to input result into memo

什么时候用动态规划

  （1）最优子结构：如果问题的最优解所包含的子问题的解也是最优的，就称该问题具有最优子结构。

          比如要想求棋盘两对角的最短距离，那个每走一格算一格阶段，每一格最优解必定是在上一格的最优解中得来的，也就是说每一个阶段都有最优的一个决策代表最优子解。

   （2）无后效性：某一状态一旦确定，就不受这个状态的以后的决策影响。

           后效性是指，如果上面的例子中的每一个格子的长度是不一致的，那么在当前选择的格子决策背景下，可能会对后面选择决策造成影响，如下图。

                   如果选择了当前选择了值为10的这个节点，则会导致后续只能选择100，甚至200的节点，即在选择10和20节点的时候，会对后续的决策产生影响。

   （3）有重叠子问题：子问题之间不是相互独立的，一个子问题在下一阶段的决策中可能被多次使用到。

          即当前的最优解的计算，我们只需要记录下来，在后续的比较中使用到，而不需要再次计算，增加时间复杂度。

动态规划的基本过程描述：

      每次决策依赖于当前状态，又引起状态的转移，一个决策序列就是在变化的状态中产生出来的，所以，动态规划可以看成是多阶段最优化决策来解决问题的过程。

使用动态规划解决问题，最重要的就是确定动态的三要素：

   （1）问题的阶段。

   （2）每个阶段的状态。

   （3）从前一个阶段到后一个阶段的递推关系。

1，最优子结构

母问题的最优解包含bai其子问题的最优解，我们就称du此问题具有最优子结构。即也就是说，子问题最优时，母问题通过优化一定能求得最优解

2，子问题重叠

子问题本质上是和母问题一样的，只是问题的输入参数不一样，就可以称之为子问题重叠，这是动态规划解决问题的高效的本质所在，我们可以利用很多子问题具有相同的输入参数这一个性质，来减少计算量。

3，问题存在边界

子问题在一定情况下就不存在子问题了， 我们称这种情况为问题存在边界，对于自顶向上和自底向下的方法，边界分别是问题的出口和入口。

4，子问题相互独立

个子问题在求解最优解时事相互独立的，即本自问题的求解和其他平行子问题是不相干的。当平行子问题解决后，选择权交给母问题时，它才会考虑各子问题之间的关系，是求最大值还是最小值，还是要做相关的运算得到母问题的最优解。

动态规划常规结构

class Solution {

public int rob(int[] nums) {

if (nums.length==0) return 0;

int[] dp=new int[nums.length+1];

dp[0]=0; //第一个通常设为0

dp[1]=nums[0];

for(int i=1;i<dp.length-1;i++){

}

return dp[dp.length-1]; //永远return 最后一个

}

}