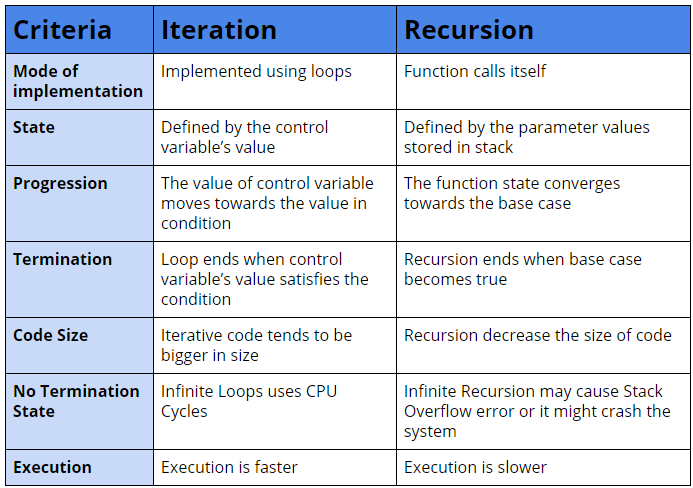
🡪

recursive和iteration不同。recursive(递归)和iteration(迭代)比较：

recursive是将问题从上到下的解决，是top-down风格，也是divide and conquer解决问题的风格。它将不断调用自身，或者说不断调用同一个方法直到基本用例。每一次递归调用，将会将原问题化为多个小问题进行处理。在程序上包含堆栈。

Recursive与堆栈的关系在于，堆栈用于保存当时的变量环境，当程序的控制权返回到函数的时候从堆栈中取出环境继续运行。例如方法A中调用自身方法A称为A1,A1中调用自身方法A称为A2.则程序将执行A,等待A1，A1等待A2.然后A2返回控制权给A1，继续执行A1.A1执行完后返回控制权给A，A继续执行。当A调用A1时候将A的环境入栈，A1调用A2时候将A1环境入栈，最后A2入栈。可以看出程序的执行顺序是A先调用，A1再调用,A2再调用，A2完成，A1完成，A完成。对于栈来说，则是最先入栈的A最后出栈。

iteration是将问题从下到上的解决,是bottom-up风格，也是dynamic programming解决问题的风格。。它不会重复调用某个方法，在程序上也不需要包含堆栈。它只会重复调用某段代码直到条件满足或不满足某些条件。



名称记忆：

Recursive含有re代表recall, 重复调用方法。从上到下分割问题size,并且最后一般需要从小的问题实例，结合归并到原问题的解，为递下去归并回来。为递归。

Iterative的典型代码有通过数组作为载体计算斐波那契数列，从基用例，从下到上逐步计算，一代代计算到所需要n。为迭代。

🡪

Divide and Conquer的步骤为：

1. 将一个问题分成多个更小的instance （即是recursive property形似solution(n) = solution (n-1) + solution(n-4)）

2. 除非instance的size足够小，否则继续使用recursion方式进行调用，划分为更小。当到达问题用例规格到达足够小时候，不再recursive调用直接解决。

3. 如果必要，合并这些instance的solution.作为原instance的答案。

🡪

什么时候不应该采用Divide and Conquer?

当采用Divide and Conquer时候，我们将原size为n的问题，化为多个size为n-1, n-2或者和原size n大小差不多的问题时，我们不应该采用Divide and Conquer. （例如使用recursive方法计算斐波那契数列f(n) = f(n-1) + f(n-2)递归调用。原size为n的问题化为2个size为n-1, n-2的问题时候我们不应该改用recursive方法。）

当采用Divide and Conquer时候，我们将原size为n的问题，化为多个更小问题的解决的时候。如果这些更小的问题需要重复的解决，则不应该使用recursive方法。

🡪

Dynamic Programming的步骤为：

1. 找到一个关系将一个问题分成多个更小的instance （即是recursive property形似solution(n) = solution (n-1) + solution(n-4)）

2. 以bottom up的方式，从基用例开始计算，并且将计算的结果使用数据结构进行记录。逐步从底部向上计算，直到得出原instance的答案.

动态规划从底部向上的计算的过程中，根据不一样的recursive property,会在数据结构中展示出不同形式。

例如：

在计算BinomialCofficent的过程中，recursive property为C(n,k) = C(n-1, k-1) + C(n-1, k)。这种迭代关系不需要取最小值，也不需要分割点。从底部向上的计算过程中，在二维数组中依次按行逐个元素将计算，每个元素计算一次便可得出结果，不需要取多个结果中的最少值作为该点数值。

在使用FloydAlgorithm计算最短路径问题，recursive property为D(k)[i][j] = minimum(D(k-1)[i][j], D(k-1)[i][k] + D(k-1)[k][j])。这种迭代关系存在取最小值（即是否使用k结点不同情况）。从底部向上的计算过程中，需要对二维数组进行多次计算迭代，每次计算时候顺序为按每一行中的每一列计算。而不是沿对角线移动。并且计算某个元素时候，还要取两个值（使用k结点与不使用k结点 ）中的最小值。

在使用ChainMatrixMultiple计算矩阵相乘最优乘法顺序时，recursive property为M[i][j] = minimun(M[i][k] + M[k+1][j] + d(i-1)dkdj for i <= k <= j-1。这种迭代关系存在取最小值（即取i->j-1中哪一点作为分割点）。从底部向上的计算过程中，只需要对二维数组进行一次计算，在计算过程中沿对角线移动，并且计算某个元素时候，还要取所有可能取值中的最小值。

🡪

什么时候问题可以采用Dynamic Programming解决？

如果使用Dynamic Programming解决一个问题。那么该问题必须满足principle of optimality.

Principle of optimality性质为如果对size为n的问题的最优解，同时也是包含对所有子问题，size<n的子问题的最优解则该问题满足Principle of optimality性质。可以使用Dynamic Programming解决。

例如在最短路径问题中，寻找v1 -> v5的最短路径。找到一条最后解路径从v1 -> v5后，假设该路径会途经v4节点。则我们可以确定在该路径中v1->v4,v4 -> v5路径，同时也是v1->v4,以及v4->v5的最优解（子问题，原问题是v1->v5, 子问题是v1->v4, v4->v5）。那么最短路径问题满足Principle of Optimality原则，所以可以使用Dynamic Programming解决。

🡪

T(n)为every case time complexity.意思是算法对于n大小的input,无论什么情况算法复杂度都一样。基于此的算法复杂度。如果算法在某些情况下复杂度高，某些情况下复杂度度低，则不存在T(n)概念. 若T(n)存在的前提下，T(n) = B(n)= A（n） = W(n)

W（n）为worst case time complexity.意思是算法对于n大小的input,最差情况下的算法复杂度。

A(n)为average case time complexity。意思是算法对于n大小的input,平均情况下的算法复杂度。它基于input分布的概率，由概率加权平均数获得。

B(n)为best case time complexity. 意思是算法对于n大小的input,最好情况下的算法复杂度。

🡪

常见的排序（sorting）算法:

归并排序MergeSort，采用recursive的divide and conquer. 将数组从中间进行分隔成两半，然后合并。此过程一直递归调用到分数组只剩1个元素，然后每次合并时候进行排序。其W(n)为Order(nlgn)，即使合并的时候每次分别从子数组取一个元素，合并过程一直需要比较.

QuickSort快速排序。采用recursive的divide and conquer,将数组中某个元素（通常为第一个元素）作为指标，将所有比其小的元素放在指标左边，所有比其大的元素放在指标的右边。然后对左右两边继续递归调用。直到指标左右两边的分数组只有一个元素。其W(n)为n2，即数组原本已经排好序，每次分组的其中一个分数组为空。A(n)为Order(nlgn).

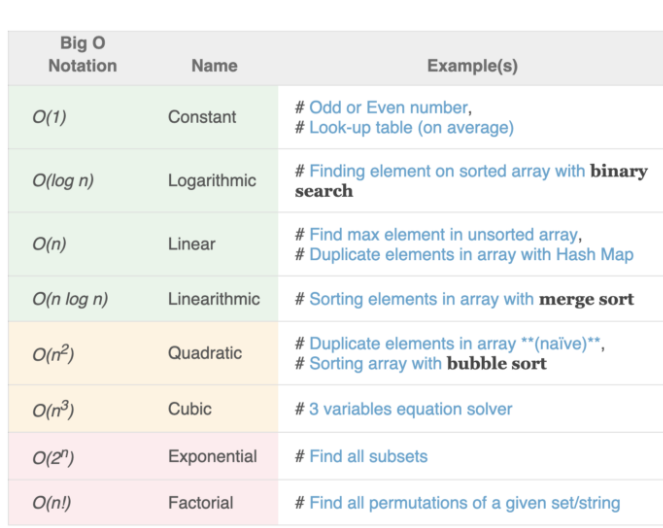
ExchangeSort交换排序。即是对数组的每个元素于后面的所有元素进行对比，每发现更小的元素将其放在首位。T(n)=n2

🡪

有向图无向图可以在程序中可以使用二维数组（临接矩阵adjacency matrix）进行表示。如果图中有n个节点，则将二维数组为n X n. 在此数据结构上进行最短路径计算, TravelSalesMan问题，最小生成树问题等。例如数组名[3][4] = 10, 表示3号节点有边连接到4号节点，花费为10. 数组名[3][4] = +oo 表示3号节点没有直接连接4号节点。

🡪

算法复杂度分类：



在Cubic以及以下的复杂度的都可以称为polynomial(多项式)时间复杂度。

🡪

一个全集含有n个元素的全集。{n1, n2, n3, n4, n5…}

则子集的数量为2n. 且子集中元素个数最少为0,为空集。子集中含元素个数最多为n,为全集。

在含有n个节点的连通无向图中，边的数目最少为n-1条，最多为n(n-1)/2

🡪经典算法：

**Divide and Conquer(分治法):**

MergeSort排序,W(n) = nlgn

QuickSort排序,W(n) = n2 , A(n) = nlgn

**Dynamic Programming(动态规划)：**

Floyd ->寻找图中任意一点到图中另外任意一点的最短路径, T(n) = n3

**Greedy Approach(贪心算法)：**

Prim’s （普里姆算法）🡪 求最小生成树， T(n) = n2

Kruskal’s (克鲁斯卡尔算法) 🡪 求最小生成树， W(n) = n2lgn