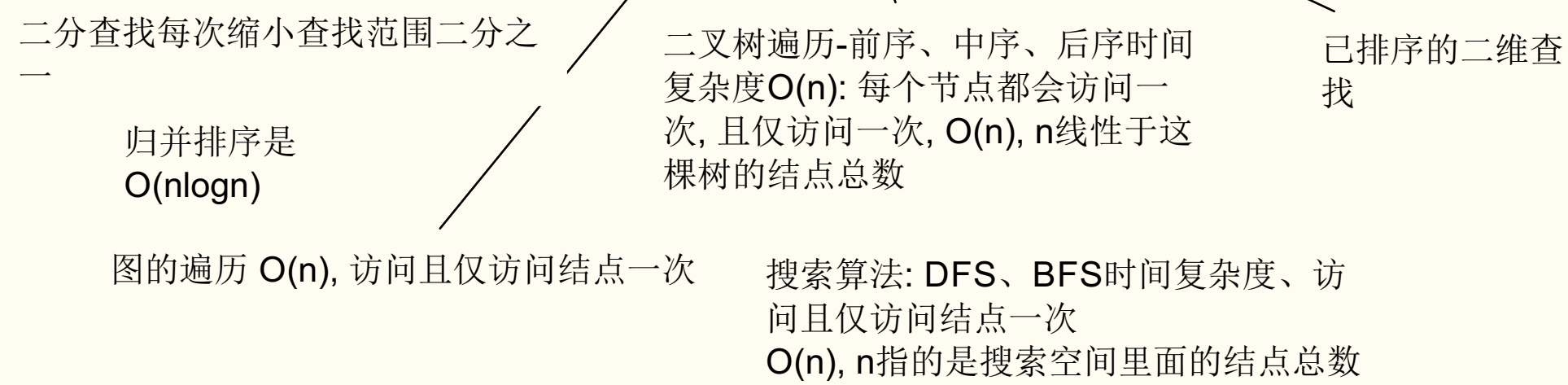


主定理: 用于算出递归执行次数

以下是四种常用递归复杂度

Application to common algorithms [edit]			
Algorithm	Recurrence relationship	Run time	Comment
Binary search	$T(n) = T\left(\frac{n}{2}\right) + O(1)$	$O(\log n)$	Apply Master theorem case $c = \log_b a$, where $a = 1, b = 2, c = 0, k = 0$ ^[5]
Binary tree traversal	$T(n) = 2T\left(\frac{n}{2}\right) + O(1)$	$O(n)$	Apply Master theorem case $c < \log_b a$ where $a = 2, b = 2, c = 0$ ^[5]
Optimal sorted matrix search	$T(n) = 2T\left(\frac{n}{2}\right) + O(\log n)$	$O(n)$	Apply the Akra–Bazzi theorem for $p = 1$ and $g(u) = \log(u)$ to get $\Theta(2n - \log n)$
Merge sort	$T(n) = 2T\left(\frac{n}{2}\right) + O(n)$	$O(n \log n)$	Apply Master theorem case $c = \log_b a$, where $a = 2, b = 2, c = 1, k = 0$



时间复杂度

- $O(1)$: Constant Complexity 常数复杂度
 - $O(\log n)$: Logarithmic Complexity 对数复杂度
 - $O(n)$: Linear Complexity 线性时间复杂度
 - $O(n^2)$: N square Complexity 平方
 - $O(n^3)$: N square Complexity 立方
 - $O(2^n)$: Exponential Growth 指数
 - $O(n!)$: Factorial 阶乘
- for循环嵌套几层, 就是 $O(n^{\text{几层}})$
- 斐波那契 $F(n) = F(n - 1) + F(n - 2)$, 常规递归复杂度 $O(2^n)$
- 注意: 只看最高复杂度的运算, 不考虑系数

时间复杂度曲线

