

# 课程总览

## -- Data Structure --

Array

Stack / Queue

PriorityQueue (heap)

LinkedList (single / double)

Tree / Binary Tree

Binary Search Tree

HashTable

Disjoint Set

Trie

BloomFilter

LRU Cache

## -- Algorithm --

General Coding

In-order/Pre-order/Post-order traversal

Greedy

Recursion/Backtrace

Breadth-first search

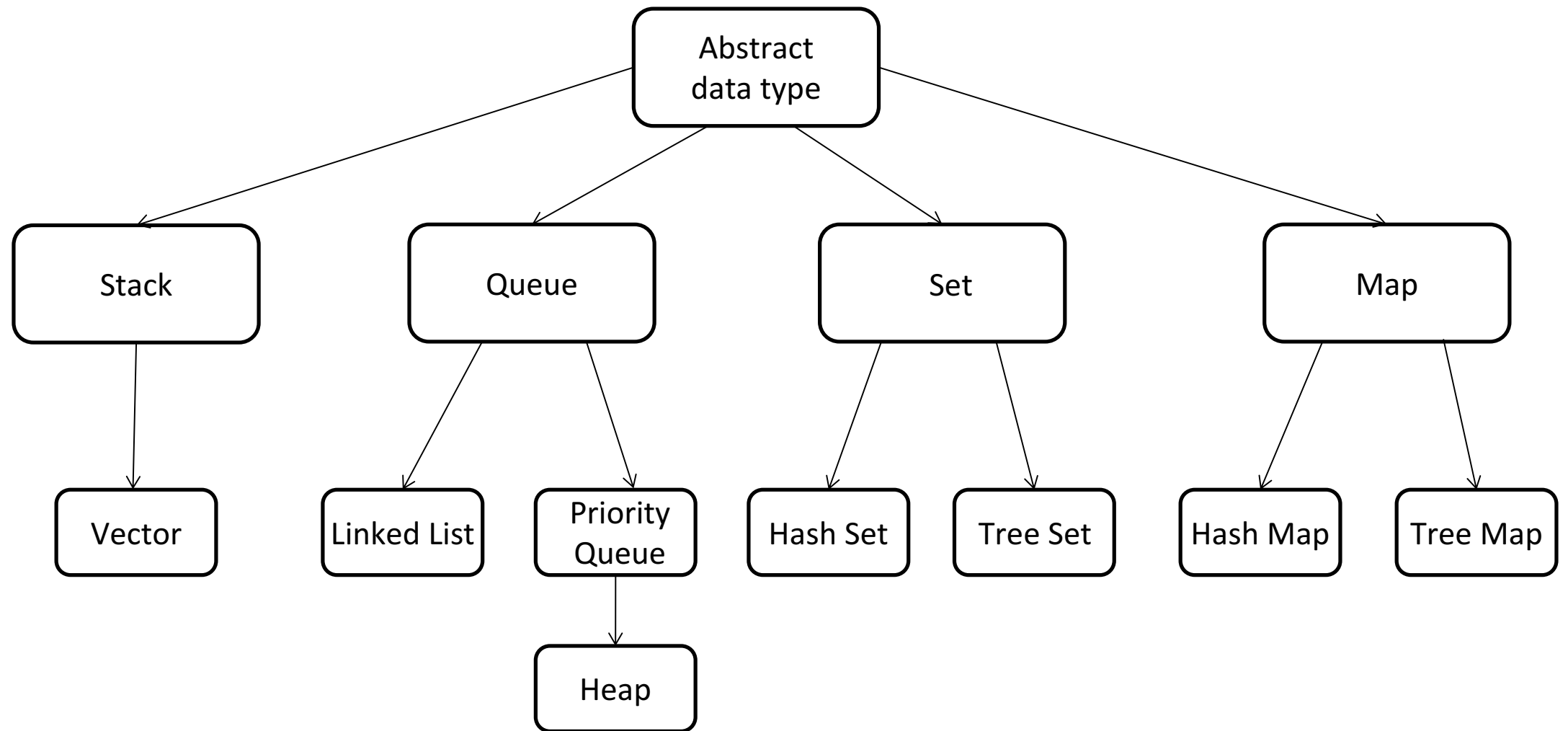
Depth-first search

Divide and Conquer

Dynamic Programming

Binary Search

Graph



时间复杂度

空间复杂度

# Big O notation

-- What is Big O? --

**O(1):** Constant Complexity: Constant 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<sup>2</sup>):** N square Complexity 平方

**O(n<sup>3</sup>):** N square Complexity 立方

**O(2<sup>n</sup>):** Exponential Growth 指数

**O(n!):** Factorial 阶乘

$O(1)$

```
int n = 1000;
```

```
System.out.println("Hey - your input is: " + n);
```

$O(?)$

```
int n = 1000;
```

```
System.out.println("Hey - your input is: " + n);
```

```
System.out.println("Hmm.. I'm doing more stuff with: " + n);
```

```
System.out.println("And more: " + n);
```

$O(N)$

```
for (int i = 1; i <= n; i++) {  
    System.out.println("Hey - I'm busy looking at: " + i);  
}
```

$O(N^2)$

```
for (int i = 1; i <= n; i++) {  
    for (int j = 1; j <= n; j++) {  
        System.out.println("Hey - I'm busy looking at: " + i + " and " + j);  
    }  
}
```

$O(\log(n))$

```
for (int i = 1; i < n; i = i * 2) {  
    System.out.println("Hey - I'm busy looking at: " + i);  
}
```

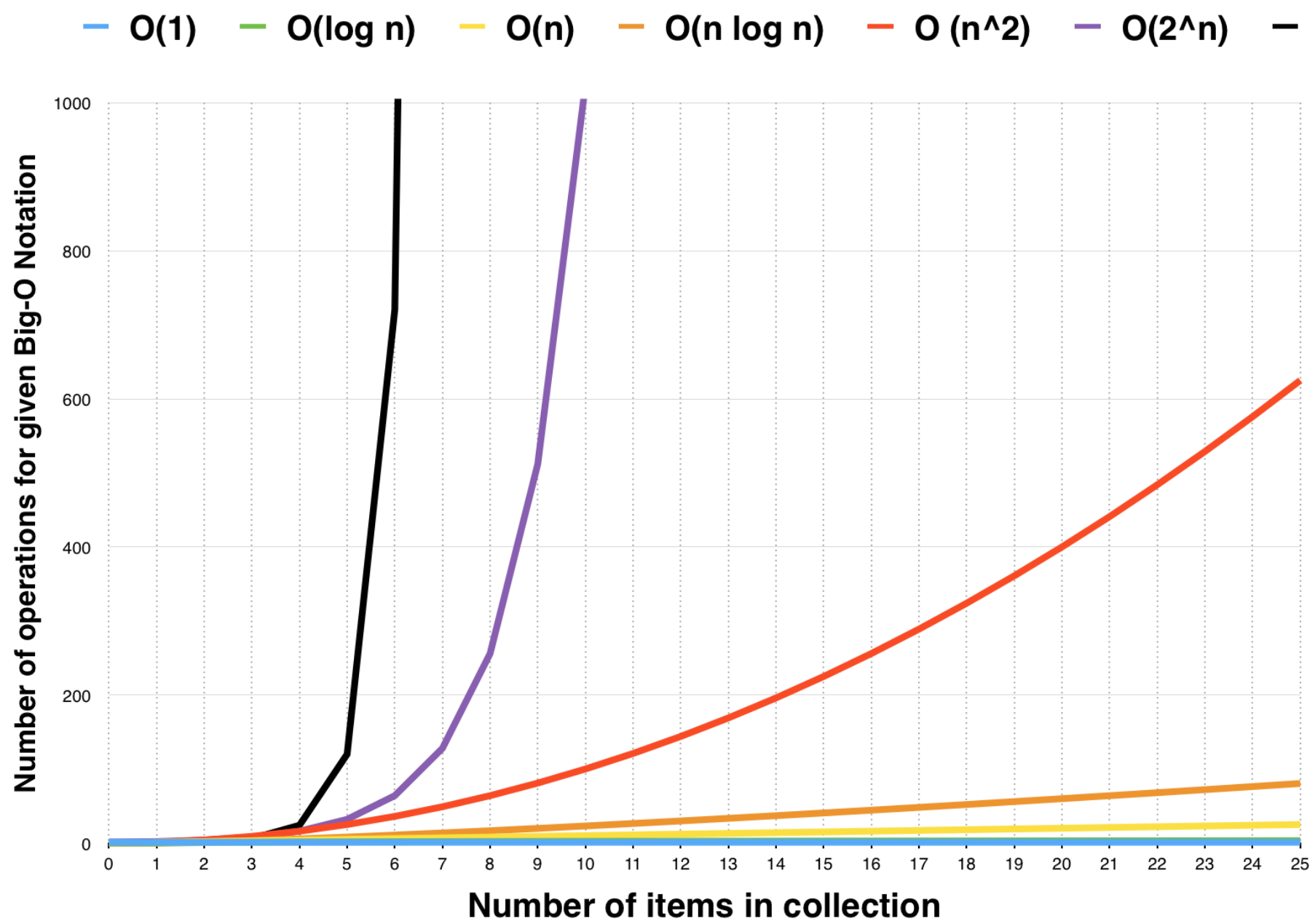
$O(k^n)$

```
for (int i = 1; i <= Math.pow(2, n); i++){  
    System.out.println("Hey - I'm busy looking at: " + i);  
}
```

$O(n!)$

```
for (int i = 1; i <= factorial(n); i++){  
    System.out.println("Hey - I'm busy looking at: " + i);  
}
```





# To calculate: $1 + 2 + 3 + \dots + n$

- $1 + 2 + 3 + \dots + n$  (总共累加 $n$ 次)

$y = 0$

for  $i = 1$  to  $n$ :

$y = i + y$

- 求和公式:  $n(n+1)/2$

$y = n * (n + 1) / 2$

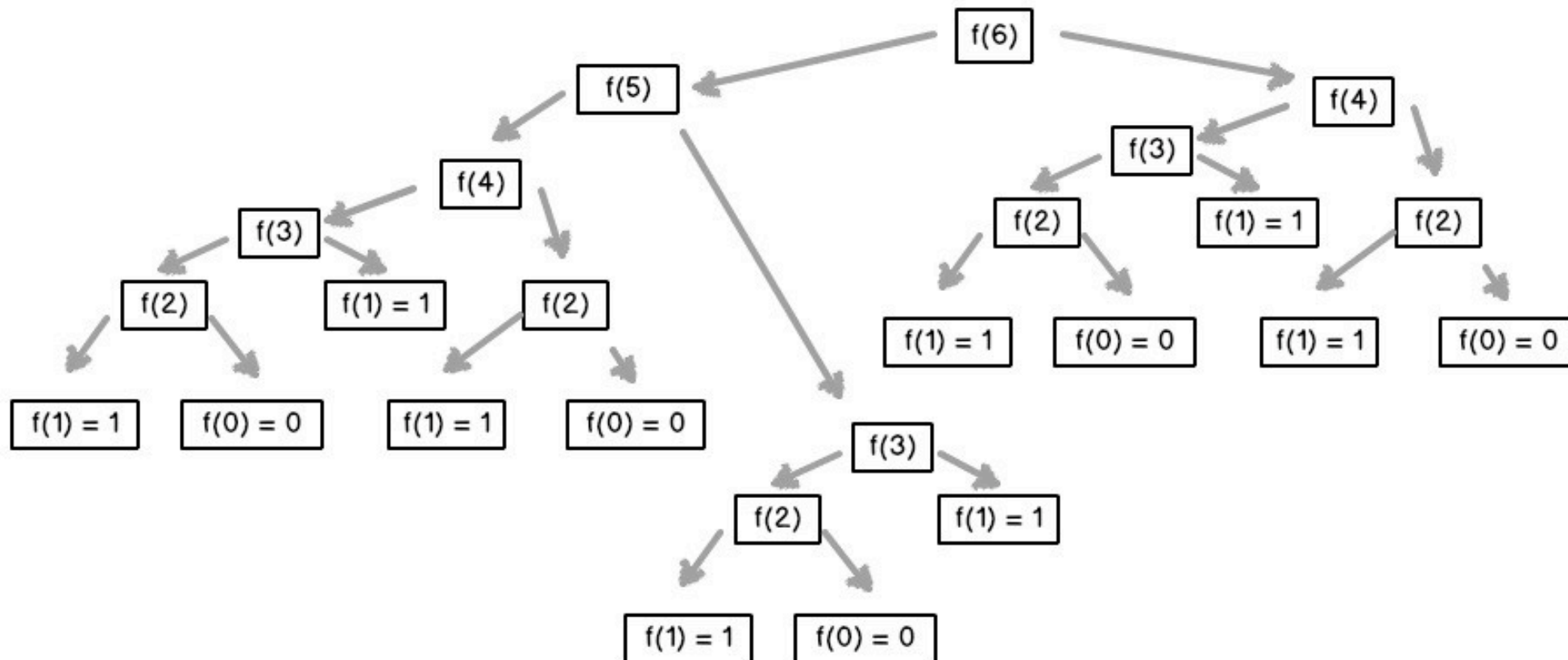
# What if recursion ?

- Fibonacci array: 1, 1, 2, 3, 5, 8, 13, 21, 34, ...

$$F(n) = F(n-1) + F(n-2)$$

```
def fib(n):  
    if n == 0 or n == 1:  
        return n  
    return fib(n - 1) + fib(n - 2)
```

# Fib(6)



# Master Theorem

[https://en.wikipedia.org/wiki/Master\\_theorem\\_\(analysis\\_of\\_algorithms\)](https://en.wikipedia.org/wiki/Master_theorem_(analysis_of_algorithms))

<https://zh.wikipedia.org/wiki/%E4%B8%BB%E5%AE%9A%E7%90%86>

## Application to common algorithms [\[ edit \]](#)

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