# 15.时间复杂度

**编程建议：**

Beautiful is better than ugly

整齐、易读胜过混乱、晦涩

Simple is better than complex

简约胜过复杂

Complex is better than complicated

复杂胜过晦涩

Flat is better than nested

扁平胜过嵌套

Now is better than never.

Although never is often better than right now.

理解一：先行动起来，编写行之有效的代码，不要企图一开始就编写完美无缺的代码

理解二：做比不做要好，但是盲目的不加思考的去做还不如不做

If the implementation is hard to explain, it's a bad idea.

If the implementation is easy to explain, it may be a good idea.

如果方案很难解释，很可能不是有一个好的方案，反之亦然

## 15.1.常见时间复杂度

**求最大值和排序**

**import** **numpy** **as** **np**

x = np.random.randint(100, size=10)

x

array([13, 14, 33, 79, 18, 26, 17, 65, 87, 63])

寻找最大值的时间复杂度为O(n)

选择排序时间复杂度O(n^2)

**代数分析**

**def** one(x):

*"""常数函数"""*

**return** np.ones(len(x))

**def** log(x):

*"""对数函数"""*

**return** np.log(x)

**def** equal(x):

*"""线性函数"""*

**return** x

**def** n\_logn(x):

*"""nlogn函数"""*

**return** x\*np.log(x)

**def** square(x):

*"""平方函数"""*

**return** x\*\*2

**def** exponent(x):

*"""指数函数"""*

**return** 2\*\*x

各时间复杂度对比

**import** **matplotlib.pyplot** **as** **plt**

plt.style.use("seaborn-whitegrid")

t = np.linspace(1, 20, 100)

methods = [one, log, equal, n\_logn, square, exponent]

method\_labels = ["$y = 1$", "$y = log(x)$", "$y = x$", "$y = xlog(x)$", "$y = x^2$", "$y = 2^x$"]

plt.figure(figsize=(12, 6))

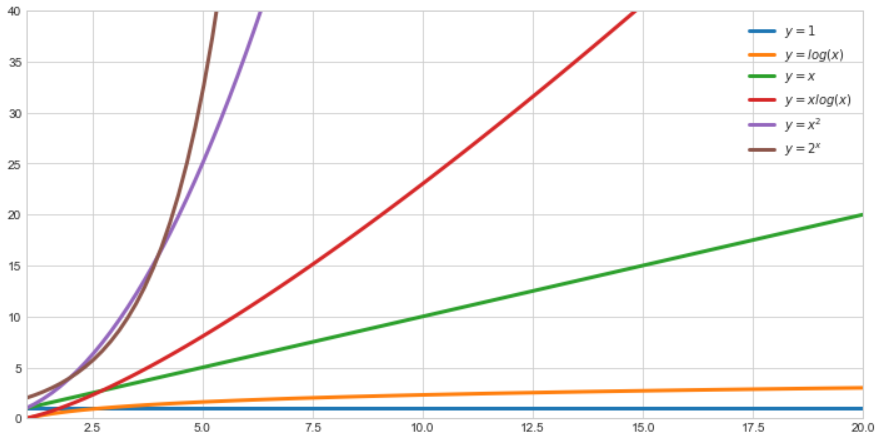
**for** method, method\_label **in** zip(methods, method\_labels):

plt.plot(t, method(t), label=method\_label, lw=3)

plt.xlim(1, 20)

plt.ylim(0, 40)

plt.legend()



我们的最爱：常数函数和对数函数

勉强接受：线性函数和nlogn函数

难以承受：平方函数和指数函数

## 15.2.时间复杂度案例

**【1】三集不相交问题**

问题描述： 假设有A、B、C三个序列，任一序列内部没有重复元素，欲知晓三个序列交集是否为空

**import** **random**

**def** creat\_sequence(n):

A = random.sample(range(1, 1000), k=n)

B = random.sample(range(1000, 2000), k=n)

C = random.sample(range(2000, 3000), k=n)

**return** A, B, C

A, B, C = creat\_sequence(100)

**def** no\_intersection\_1(A, B, C):

**for** a **in** A:

**for** b **in** B:

**for** c **in** C:

**if** a == b == c:

**return** **False**

**return** **True**

%**timeit** no\_intersection\_1(A, B, C)

no\_intersection\_1(A, B, C)

36.7 ms ± 2.12 ms per loop (mean ± std. dev. of 7 runs, 10 loops each)

True

**def** no\_intersection\_2(A, B, C):

**for** a **in** A:

**for** b **in** B:

**if** a == b:

**for** c **in** C:

**if** a == c:

**return** **False**

**return** **True**

%**timeit** no\_intersection\_2(A, B, C)

301 µs ± 37.9 µs per loop (mean ± std. dev. of 7 runs, 1000 loops each)

**import** **time**

res\_n\_3 = []

res\_n\_2 = []

**for** n **in** [10, 20, 100]:

A, B, C = creat\_sequence(n)

start\_1 = time.time()

**for** i **in** range(100):

no\_intersection\_1(A, B, C)

end\_1 = time.time()

**for** i **in** range(100):

no\_intersection\_2(A, B, C)

end\_2 = time.time()

res\_n\_3.append(str(round((end\_1 - start\_1)\*1000))+"ms")

res\_n\_2.append(str(round((end\_2 - end\_1)\*1000))+"ms")

print("**{0:<23}{1:<15}{2:<15}{3:<15}**".format("方法", "n=10", "n=20", "n=100"))

print("**{0:<25}{1:<15}{2:<15}{3:<15}**".format("no\_inte rsection\_1", \*res\_n\_3))

print("**{0:<25}{1:<15}{2:<15}{3:<15}**".format("no\_intersection\_2", \*res\_n\_2))

方法 n=10 n=20 n=100

no\_inte rsection\_1 6ms 42ms 4001ms

no\_intersection\_2 0ms 1ms 24ms

**【2】元素唯一性问题**

问题描述：A 中的元素是否唯一

**def** unique\_1(A):

**for** i **in** range(len(A)):

**for** j **in** range(i+1, len(A)):

**if** A[i] == A[j]:

**return** **False**

**return** **True**

**def** unique\_2(A):

A\_sort = sorted(A)

**for** i **in** range(len(A\_sort)-1):

**if** A[i] == A[i+1]:

**return** **False**

**return** **True**

**import** **random**

res\_n\_2 = []

res\_n\_log\_n = []

**for** n **in** [100, 1000]:

A = list(range(n))

random.shuffle(A)

start\_1 = time.time()

**for** i **in** range(100):

unique\_1(A)

end\_1 = time.time()

**for** i **in** range(100):

unique\_2(A)

end\_2 = time.time()

res\_n\_2.append(str(round((end\_1 - start\_1)\*1000))+"ms")

res\_n\_log\_n.append(str(round((end\_2 - end\_1)\*1000))+"ms")

print("**{0:<13}{1:<15}{2:<15}**".format("方法", "n=100", "n=1000"))

print("**{0:<15}{1:<15}{2:<15}**".format("unique\_1", \*res\_n\_2))

print("**{0:<15}{1:<15}{2:<15}**".format("unique\_2", \*res\_n\_log\_n))

方法 n=100 n=1000

unique\_1 49ms 4044ms

unique\_2 1ms 21ms

**【3】第n个斐波那契数**

a(n+2) = a(n+1) + a(n)

**def** bad\_fibonacci(n):

**if** n <= 1:

**return** n

**else**:

**return** bad\_fibonacci(n-2)+ bad\_fibonacci(n-1)

O(2^n)

**def** good\_fibonacci(n):

i, a, b = 0, 0, 1

**while** i < n:

a, b = b, a+b

i += 1

**return** a

O(n)

%**timeit** bad\_fibonacci(10)

20.6 µs ± 1.15 µs per loop (mean ± std. dev. of 7 runs, 10000 loops each)

%**timeit** good\_fibonacci(10)

875 ns ± 24.5 ns per loop (mean ± std. dev. of 7 runs, 1000000 loops each)

**【4】最大盛水容器（leetcode第11题）**

原题： 给定 n 个非负整数 ，每个数代表坐标中的一个点。在坐标内画 n 条垂直线，垂直线 i 的两个端点分别为和 。找出其中的两条线，使得它们与 x 轴共同构成的容器可以容纳最多的水。

说明：你不能倾斜容器，且 n 的值至少为 2。



暴力求解——双循环

**def** max\_area\_double\_cycle(height):

*"""暴力穷举双循环"""*

i\_left, i\_right, max\_area = 0,0,0

**for** i **in** range(len(height)-1):

**for** j **in** range(i+1, len(height)):

area = (j-i) \* min(height[j], height[i])

**if** area > max\_area:

i\_left, i\_right, max\_area = i, j, area

**return** i\_left, i\_right, max\_area

height = np.random.randint(1, 50, size=10)

print(height)

max\_area\_double\_cycle(height)

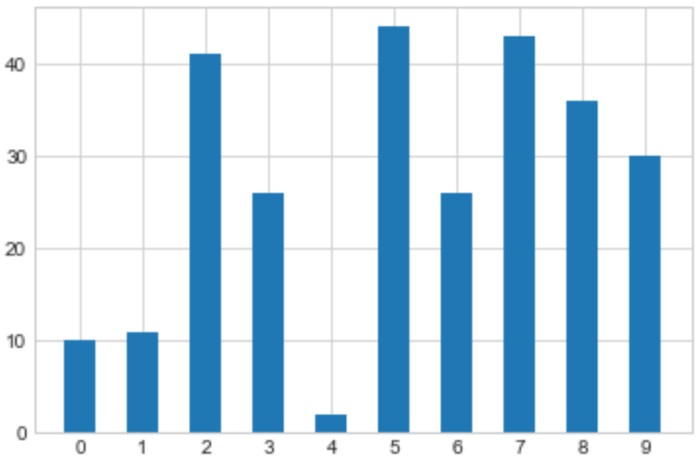
[10 11 41 26 2 44 26 43 36 30]

(2, 8, 216)

**import** **matplotlib.pyplot** **as** **plt**

plt.bar(range(10), height, width=0.5)

plt.xticks(range(0, 10, 1))



双向指针

**def** max\_area\_bothway\_points(height):

*"""双向指针法"""*

i = 0

j = len(height)-1

i\_left, j\_right, max\_area=0, 0, 0

**while** i < j:

area = (j-i) \* min(height[i], height[j])

**if** area > max\_area:

i\_left, j\_right, max\_area = i, j, area

**if** height[i] == min(height[i], height[j]):

i += 1

**else**:

j -= 1

**return** i\_left, j\_right, max\_area

max\_area\_bothway\_points(height)

(2, 8, 216)

double\_cycle = []

bothway\_points = []

**for** n **in** [5, 50, 500]:

height = np.random.randint(1, 50, size=n)

start\_1 = time.time()

**for** i **in** range(100):

max\_area\_double\_cycle(height)

end\_1 = time.time()

**for** i **in** range(100):

max\_area\_bothway\_points(height)

end\_2 = time.time()

double\_cycle.append(str(round((end\_1 - start\_1)\*1000))+"ms")

bothway\_points.append(str(round((end\_2 - end\_1)\*1000))+"ms")

print("**{0:<15}{1:<15}{2:<15}{3:<15}**".format("方法", "n=5", "n=50", "n=500"))

print("**{0:<13}{1:<15}{2:<15}{3:<15}**".format("暴力循环", \*double\_cycle))

print("**{0:<13}{1:<15}{2:<15}{3:<15}**".format("双向指针", \*bothway\_points))

方法 n=5 n=50 n=500

暴力循环 3ms 97ms 7842ms

双向指针 2ms 8ms 56ms

**【5】思考**

是不是时间复杂度低就一定好?

100000n VS 0.00001n^2

影响运算速度的因素

硬件

软件

算法