极客时间算法训练营
第八课
二分

李煜东

《算法竞赛进阶指南》作者

目录

二分查找

二分查找的前提

目标函数具有单调性（单调递增或者递减）

存在上下界（bounded）

能够通过索引访问（index accessible)

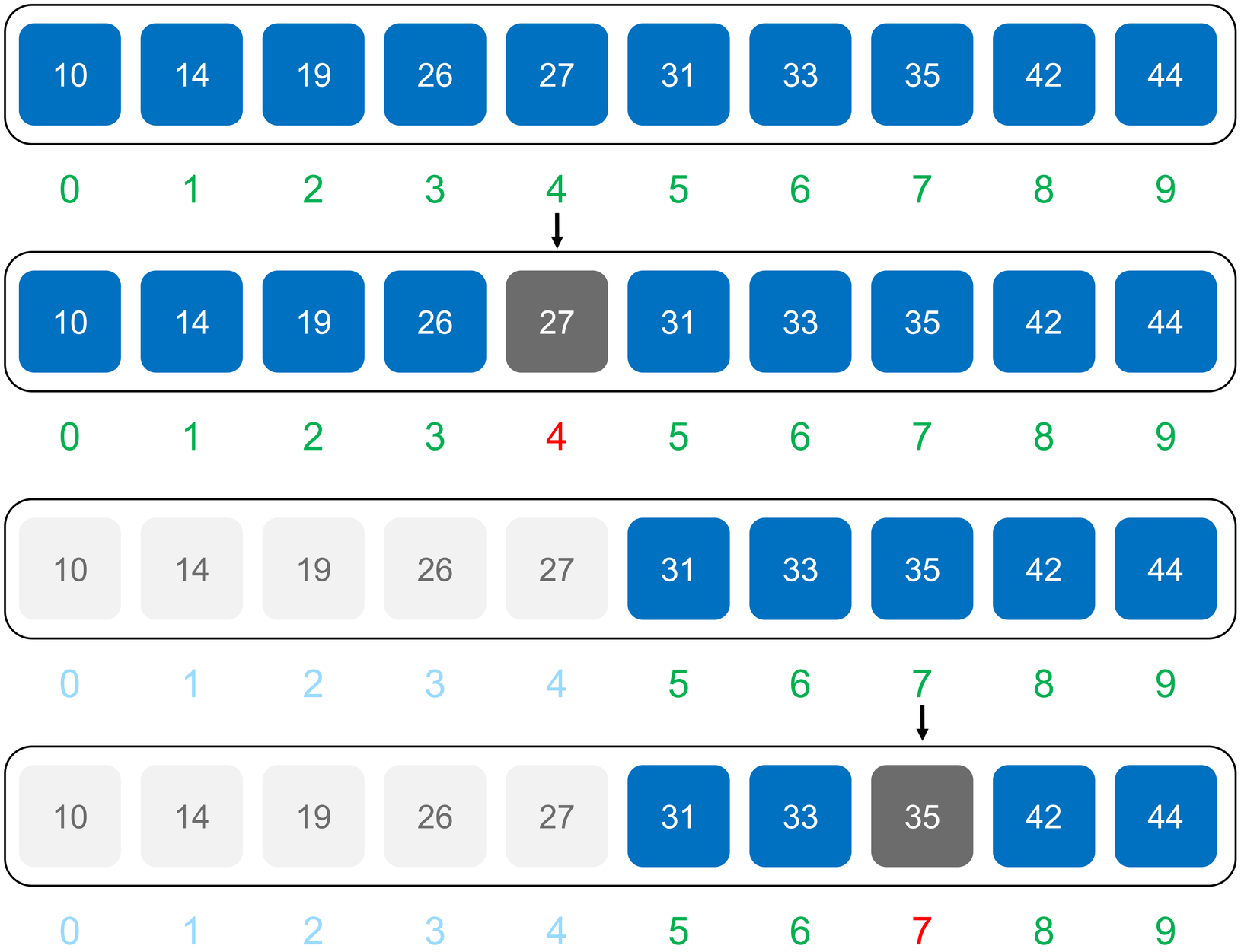
示例

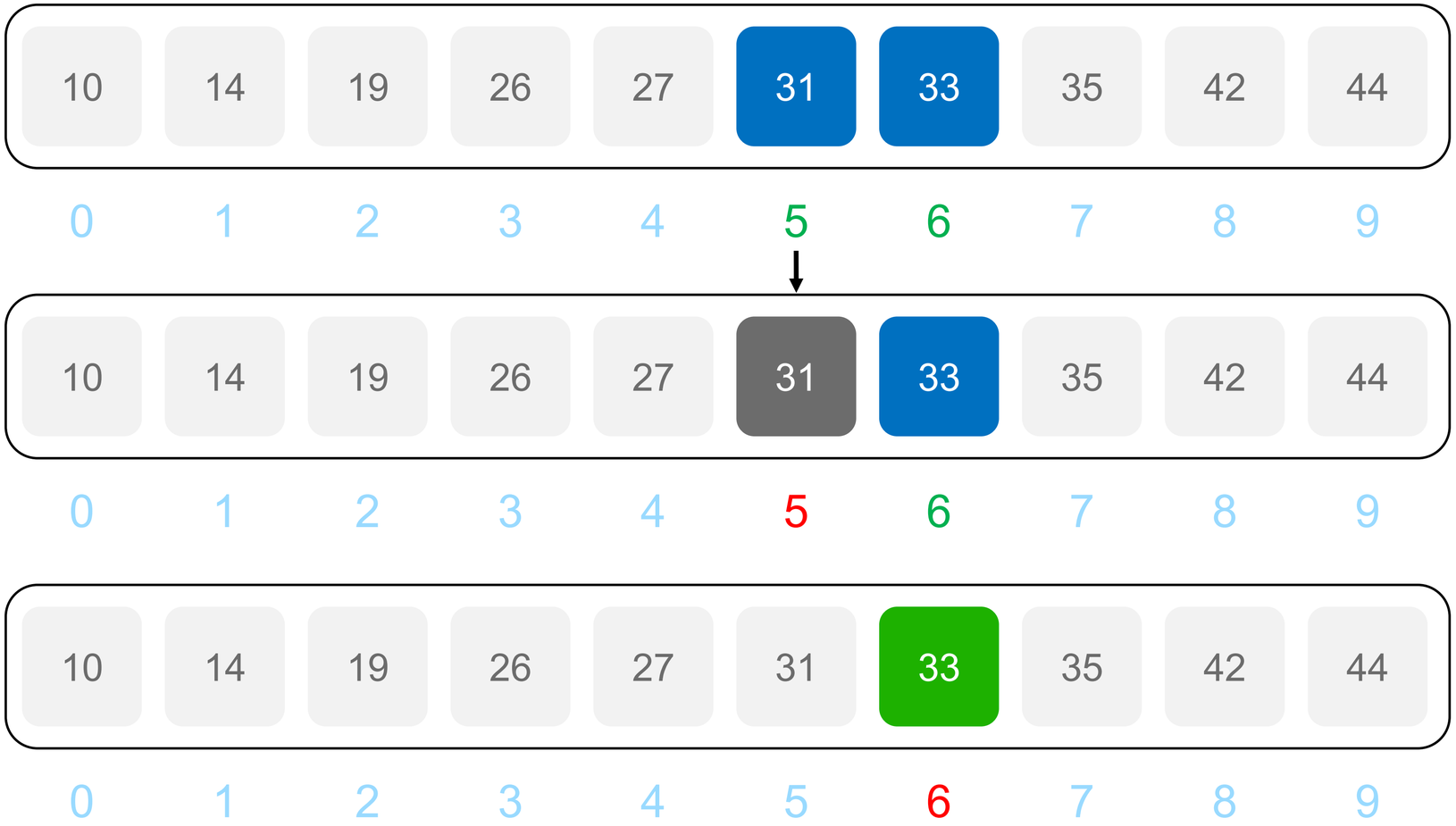
在单调递增数组里

[10, 14, 19, 26, 27, 31, 33, 35, 42, 44]

查找：33

返回 33 所在的下标


10 14 19 26 27 31 33 35 42 44
0 1 2 3 4 5 6 7 8 9
10 14 19 26 27 31 33 35 42 44
0 1 2 3 4 5 6 7 8 9
10 14 19 26 27 31 33 35 42 44
0 1 2 3 4 5 6 7 8 9
10 14 19 26 27 31 33 35 42 44
0 1 2 3 4 5 6 7 8 9


10 14 19 26 27 31 33 35 42 44
0 1 2 3 4 5 6 7 8 9
10 14 19 26 27 31 33 35 42 44
0 1 2 3 4 5 6 7 8 9
10 14 19 26 27 31 33 35 42 44
0 1 2 3 4 5 6 7 8 9

C++ / Java 代码模板

```
int left = 0, right = n - 1;
while (left <= right) {
    int mid = (left + right) / 2;
    if (array[mid] == target)
        // find the target!
        break or return mid;
    if (array[mid] < target)
        left = mid + 1;
    else
        right = mid - 1;
}
```

Python 代码模板

```
left, right = 0, len(array) - 1
while left <= right:
      mid = (left + right) // 2
      if array[mid] == target:
            # find the target!
            break or return mid
      elif array[mid] < target:
            left = mid + 1
      else:
            right = mid - 1
```

实战

二分查找

https://leetcode-cn.com/problems/binary-search/

lower_bound

在单调递增数组里

[10, 14, 19, 25, 27, 31, 33, 35, 42, 44]

查找第一个 >= 31 的数（返回下标）

不存在返回 array.length

upper_ bound

在单调递增数组里

[10, 14, 19, 25, 27, 31, 33, 35, 42, 44]

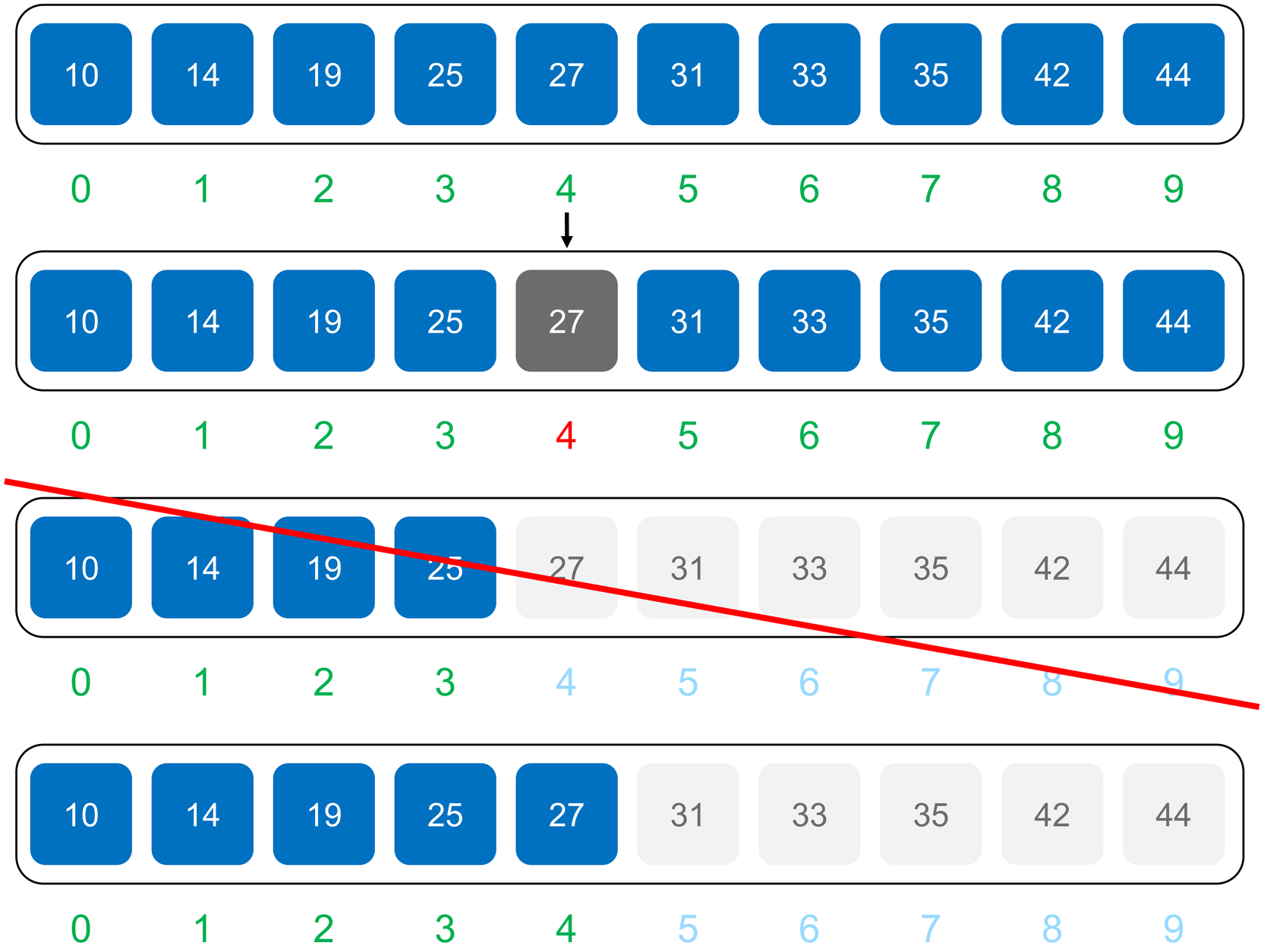
查找第一个 > 26 的数（返回下标）

不存在返回 array.length

lower_bound和upper_bound的问题是：

给定的 target 不一定在数组中存在

array[mid] 即使不等于 target，也可能就是最后的答案，不能随便排除在外


10 14 19 25 27 31 33 35 42 44
0 1 2 3 4 5 6 7 8 9
10 14 19 25 27 31 33 35 42 44
0 1 2 3 4 5 6 7 8 9
10 14 19 25 27 31 33 35 42 44
0 1 2 3 4 5 6 7 8 9
10 14 19 25 27 31 33 35 42 44
0 1 2 3 4 5 6 7 8 9

解决方案

根据个人喜好，掌握三种二分写法中的一种

- (1.1) + (1.2)：最严谨的划分，一侧包含，一侧不包含，终止于 left == right
- (2)：双侧都不包含，用 ans 维护答案，终止于 left > right
- (3)：双侧都包含，终止于 left + 1 == right，最后再检查答案

“90%的程序员都写不对二分”

所以必须熟记这三种之一

适用性更广的二分模板 (1.1 - 后继型)

查找 lower_bound（第一个 >= target 的数），不存在返回 n

```
int left = 0, right = n;
while (left < right) {
    int mid = (left + right) >> 1;
    if (array[mid] >= target)  // condition satisfied, should be included
        right = mid;
    else
        left = mid + 1;
}
return right;
```

改为 `array[mid] > target` 就是 upper_bound

适用性更广的二分模板 (1.2 - 前驱型)

查找最后一个 <= target 的数，不存在返回 -1

```
int left = -1, right = n - 1;
while (left < right) {
    int mid = (left + right + 1) >> 1;
    if (array[mid] <= target)  // condition satisfied, should be included
        left = mid;
    else
        right = mid - 1;
}
return right;
```

适用性更广的二分模板 (2)

也可以这样写

```
int left = 0, right = n - 1;
int ans = -1;
while (left <= right) {
    int mid = (left + right) / 2;
    if (array[mid] <= target) {
            // update ans using mid
            ans = max(ans, mid);
        left = mid + 1;
    } else {
        right = mid - 1;
      }
}
```

适用性更广的二分模板 (3)

还可以这样写

```
int left = 0, right = n - 1;
while (left + 1 < right) {
    int mid = (left + right) / 2;
    if (array[mid] <= target)
        left = mid;
    else
        right = mid;
}
// 答案要么是 left，要么是 right，要么不存在
// 此处检查 left 和 right，返回一个合适的结果
```

实战

寻找旋转排序数组中的最小值

https://leetcode-cn.com/problems/find-minimum-in-rotated-sorted-array/

寻找旋转排序数组中的最小值 II（Homework）

https://leetcode-cn.com/problems/find-minimum-in-rotated-sorted-array-ii/

推荐使用1.1+1.2

只要“条件”单调，二分就适用

- 旋转排序数组中的最小值，序列本身并不单调
- 但“≤ 结尾”这个条件，把序列分成两半，一半不满足（>结尾），一半满足（≤ 结尾）

可以把“条件满足”看作 1，不满足看作 0，这就是一个 0/1 分段函数，二分查找分界点

写出正确的二分代码“三步走”：

1. 写出二分的条件（一般是一个不等式，例如 upper_bound：> val 的数中最小的）
2. 把条件放到 if (...) 里，并确定满足条件时要小的（right = mid）还是要大的（left = mid）
3. 另一半放到 else 里（left = mid + 1 或 right = mid - 1），如果是后者，求 mid 时补 +1

如果题目有无解的情况，上界增加1或下界减小1，用于表示无解。

实战

在排序数组中查找元素的第一个和最后一个位置

https://leetcode-cn.com/problems/find-first-and-last-position-of-element-in-sorted-array/

x 的平方根

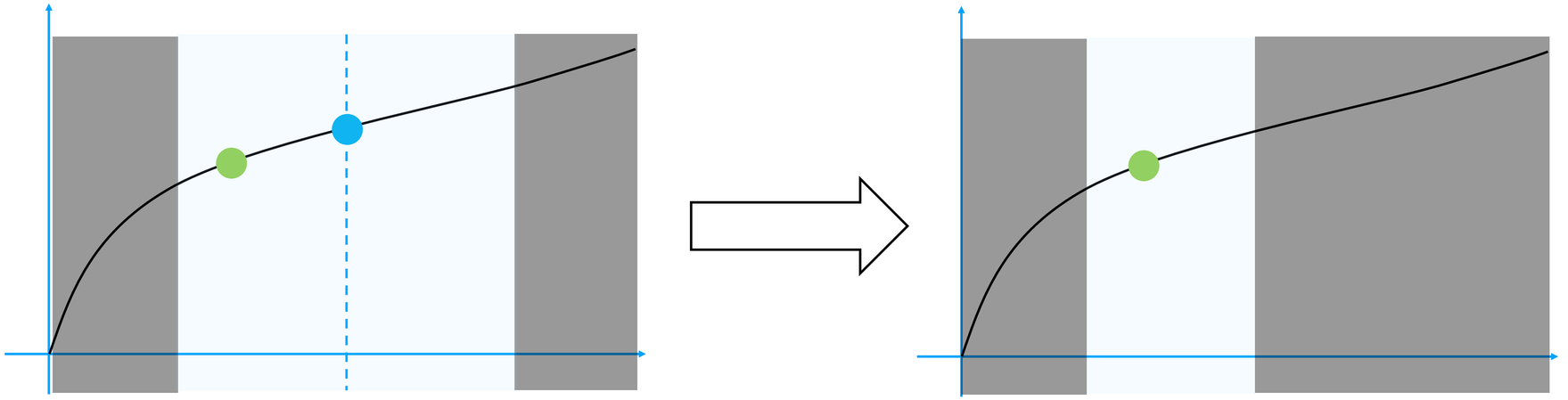
https://leetcode-cn.com/problems/sqrtx/

搜索二维矩阵（Homework）

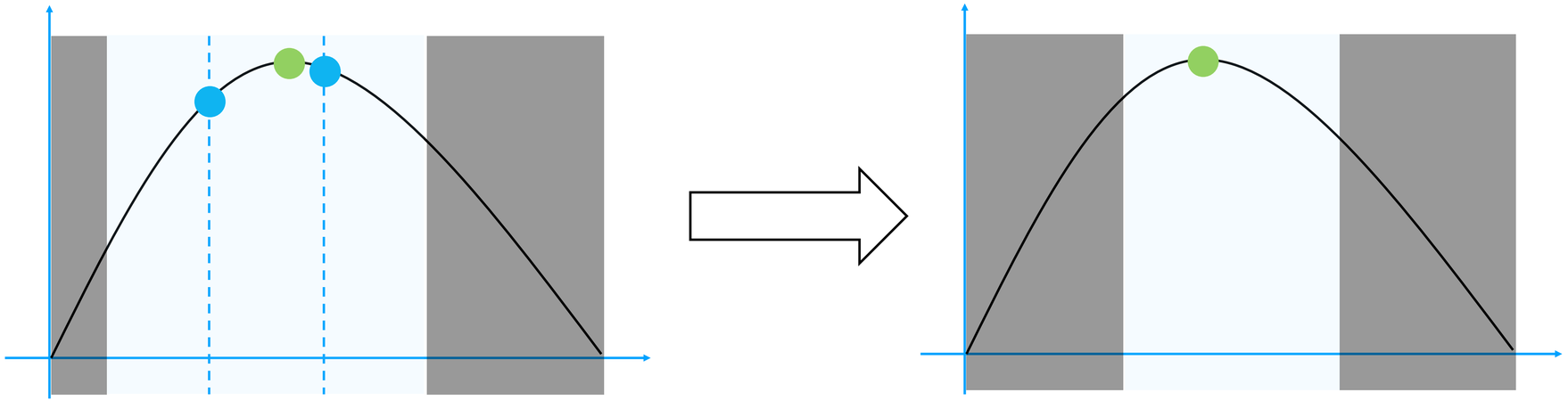
https://leetcode-cn.com/problems/search-a-2d-matrix/

三分查找

三分查找



二分：用于在单调函数上寻找特定值



三分：用于在单峰函数上寻找极大值

三分查找

三分法用于求单峰函数的极大值（或单谷函数的极小值）

三分法也可用于求函数的局部极大/极小值

要求：函数是分段严格单调递增/递减的（不能出现一段平的情况）

以求单峰函数 f 的极大值为例，可在定义域 [l,r] 上取任意两点 lmid, rmid

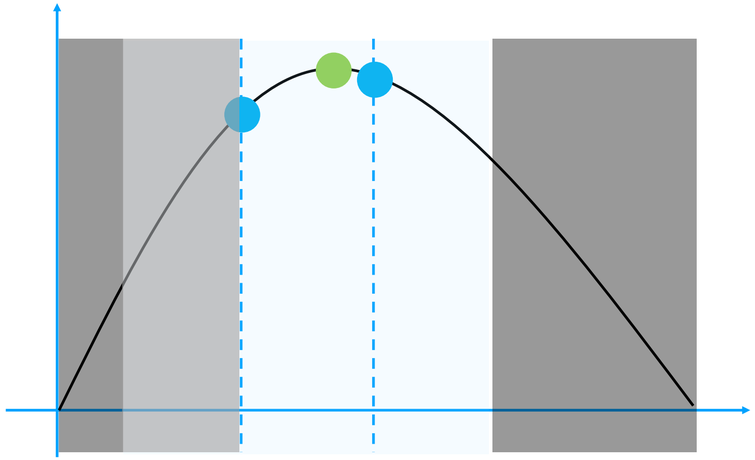
- 若 f(lmid) <= f(rmid)，则函数必然在 lmid 处单调递增，极值在 [lmid, r] 上
- 若 f(lmid) > f(rmid)，则函数必然在 rmid 处单调递减，极值在 [l, rmid] 上

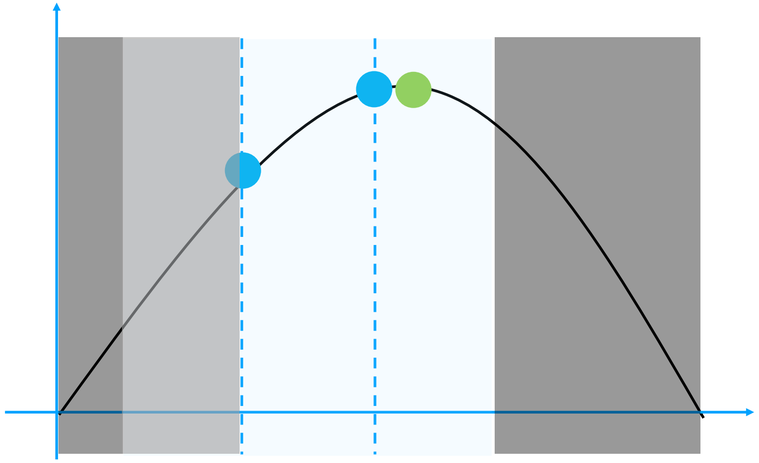
lmid, rmid 可取三等分点

也可取 lmid 为二等分点，rmid 为 lmid 稍加一点偏移量

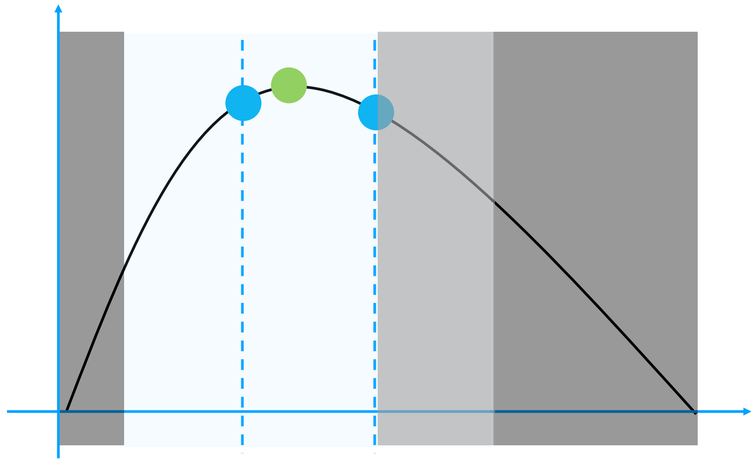
~~取黄金分割点最快*~~

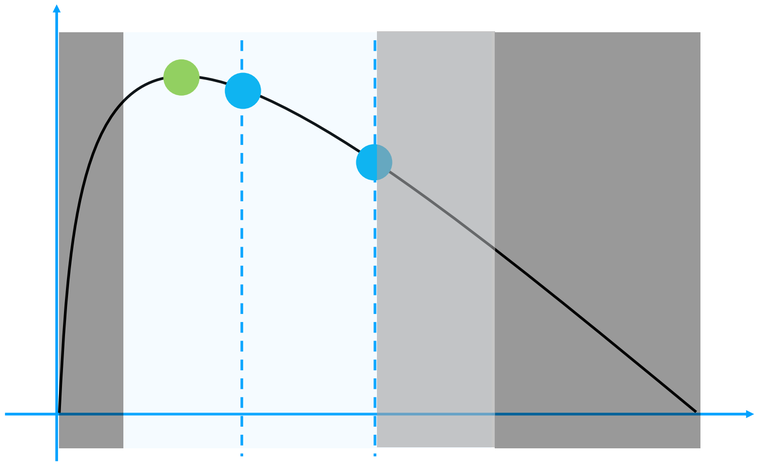
三分查找





f(lmid) < f(rmid)





f(lmid) > f(rmid)

实战

寻找峰值

https://leetcode-cn.com/problems/find-peak-element/

二分答案

猜数字大小

https://leetcode-cn.com/problems/guess-number-higher-or-lower/

猜数游戏的规则如下：

- 每轮游戏，我都会从 1 到 n 随机选择一个数。
- 请你猜选出的是哪个数。
- 如果你猜错了，我会告诉你，你猜测的数比我选出的数是大了还是小了。

二分答案
——最优性问题转化为判定问题的基本技巧

二分答案

对于一个最优化问题

求解：求一个最优解（最大值 / 最小值）

判定：给一个解，判断它是否合法（是否能够实现）

“判定”通常要比“求解”简单很多

如果我们有了一个判定算法，那把解空间枚举+判定一遍，就得到解了

当解空间具有单调性时，就可以用二分代替枚举，利用二分+判定的方法快速求出最优解，这种方法称为二分答案法

例如：求解——猜数；判定——大了还是小了；

低效算法：1 到 n 挨个猜一遍；高效算法：二分

二分答案

分割数组的最大值

https://leetcode-cn.com/problems/split-array-largest-sum/

给定一个非负整数数组 nums 和一个整数 m ，你需要将这个数组分成 m 个非空的连续子数组。
设计一个算法使得这 m 个子数组各自和的最大值最小。

求解：最小化 “m 个子数组各自和的最大值”
判定：给一个数值 T，“m 个子数组各自和的最大值 <= T” 是否合法
换一种说法：“能否将 nums 分成 m 个连续子数组，每组的和 <= T”

二分答案

为什么是 “<= T”？

nums = [7,2,5,10,8]，一共有四种方法将 nums 分割为 2 个子数组

[7] [2,5,10,8]，sum = [7, 25]，max = 25

[7,2] [5,10,8]，sum = [9, 23]，max = 23

[7,2,5] [10,8]，sum = [14, 18]，max = 18

[7,2,5,10] [8]，sum = [24, 8]，max = 24

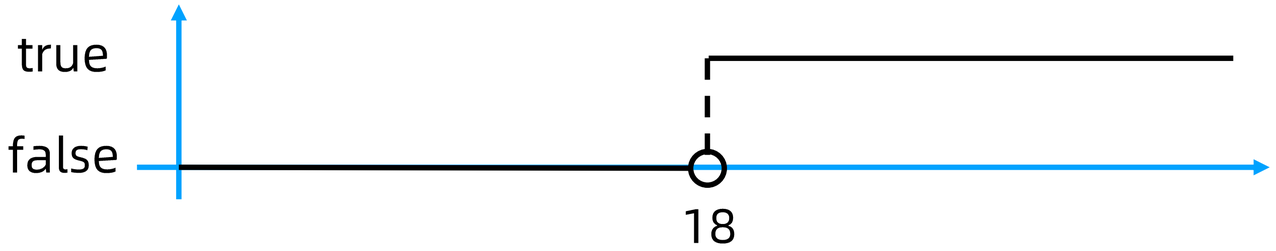
“能否将 nums 分成 m 个连续子数组，每组的和 = T” —— 只有18, 23, 24, 25 合法

“能否将 nums 分成 m 个连续子数组，每组的和 <= T” —— 17之前不合法，18之后合法

二分答案

“能否将 nums 分成 m 个连续子数组，每组的和 <= T”

这个解空间具有特殊的单调性 ——单调分段 0/1 函数



直接求出 18 比较困难，但可以通过猜测一个值 T，判断T是否合法（true or false），从而得知答案是在 T 左侧还是右侧

最高效的猜测方法当然就是二分

二分答案

通常用于最优化问题的求解

- 尤其是在出现 “最大值最小” “最小值最大” 这类字眼的题目上
- “最大值最小” 中的 “最小” 是一个最优化目标，“最大” 一般是一个限制条件（例如：限制划分出的子数组的和）

对应的判定问题的条件通常是一个不等式

- 不等式就反映了上述限制条件

关于这个条件的合法情况具有特殊单调性

此时就可以用二分答案把求解转化为判定的技巧

二分答案的本质是建立一个单调分段 0/1 函数
定义域为解空间（答案），值域为 0 或 1，
在这个函数上二分查找分界点

实战

制作 m 束花所需的最少天数

https://leetcode-cn.com/problems/minimum-number-of-days-to-make-m-bouquets/

Homework

在 D 天内送达包裹的能力

https://leetcode-cn.com/problems/capacity-to-ship-packages-within-d-days/

在线选举

https://leetcode-cn.com/problems/online-election/

爱吃香蕉的珂珂

https://leetcode-cn.com/problems/koko-eating-bananas/

THANKS