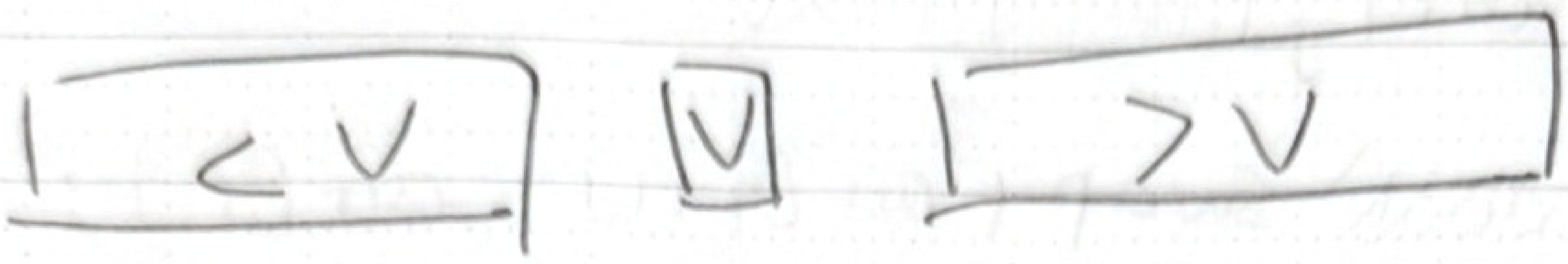
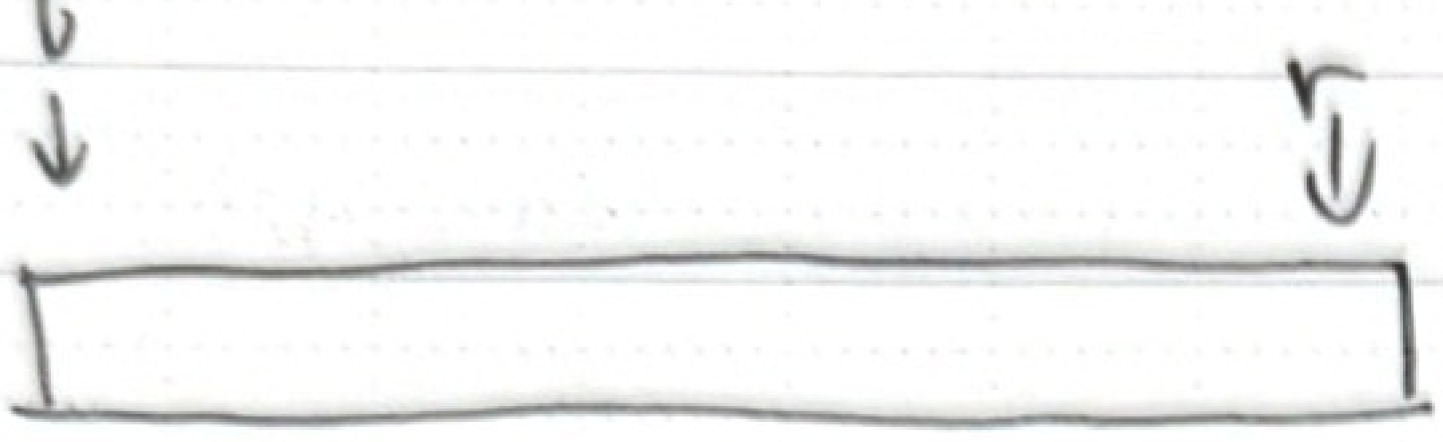
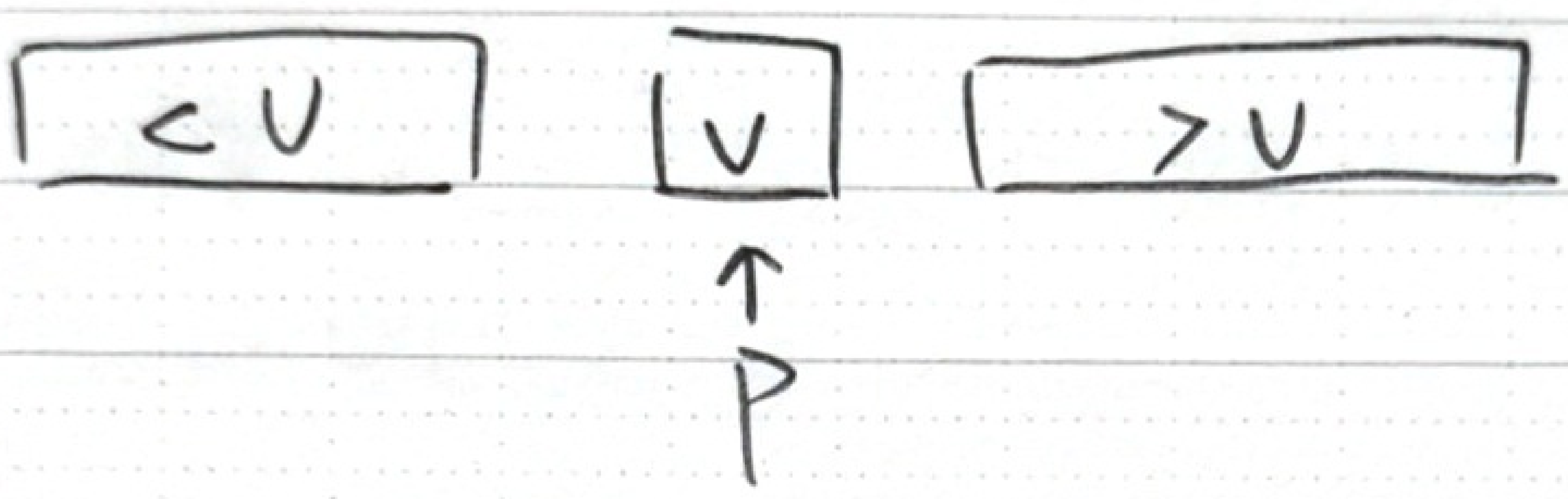
Quick sort

partition 

partition(arr, l, r)



int p = partition(arr, l, r)

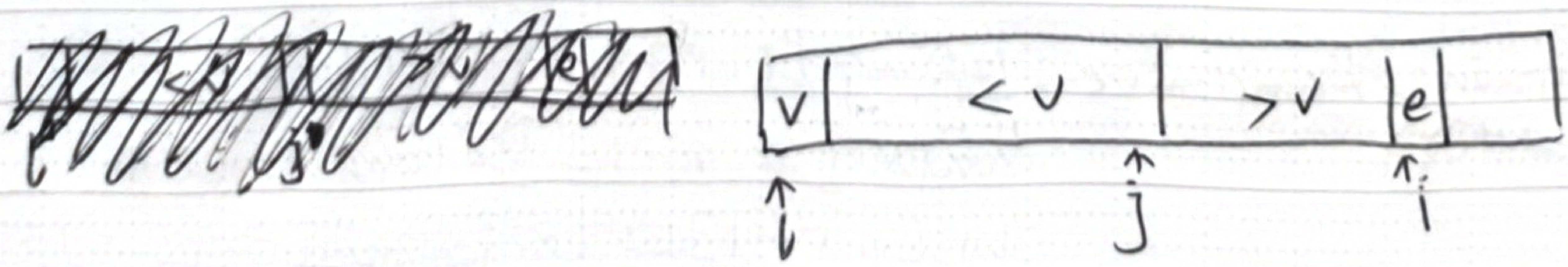


递归:

QuickSort(arr, l, r) ⇒ 对 arr 的[l, r]部分排序

```
QuickSort(arr, l, r){
    if(l>=r) return;            → base case.
    int p = partition(arr, l, r);
  //对arr[l, p-1] 进行排序.
   QuickSort(arr, l, p-1);      → subproblem
  //对arr[p+1, r] 进行排序
   QuickSort(arr, p+1, r);
}
```

partition: 如何进行 sort in place?? (否则易实现，但慢: left... right... } ⇒ [<] [v] [>]



循环不变量: arr[l+1...j] < v   arr[j+1...i-1] > v

i ⇒ 当前遍历到的元素。

一、

① if $arr[i] > v$: $arr[j+1 \ldots i] > v$

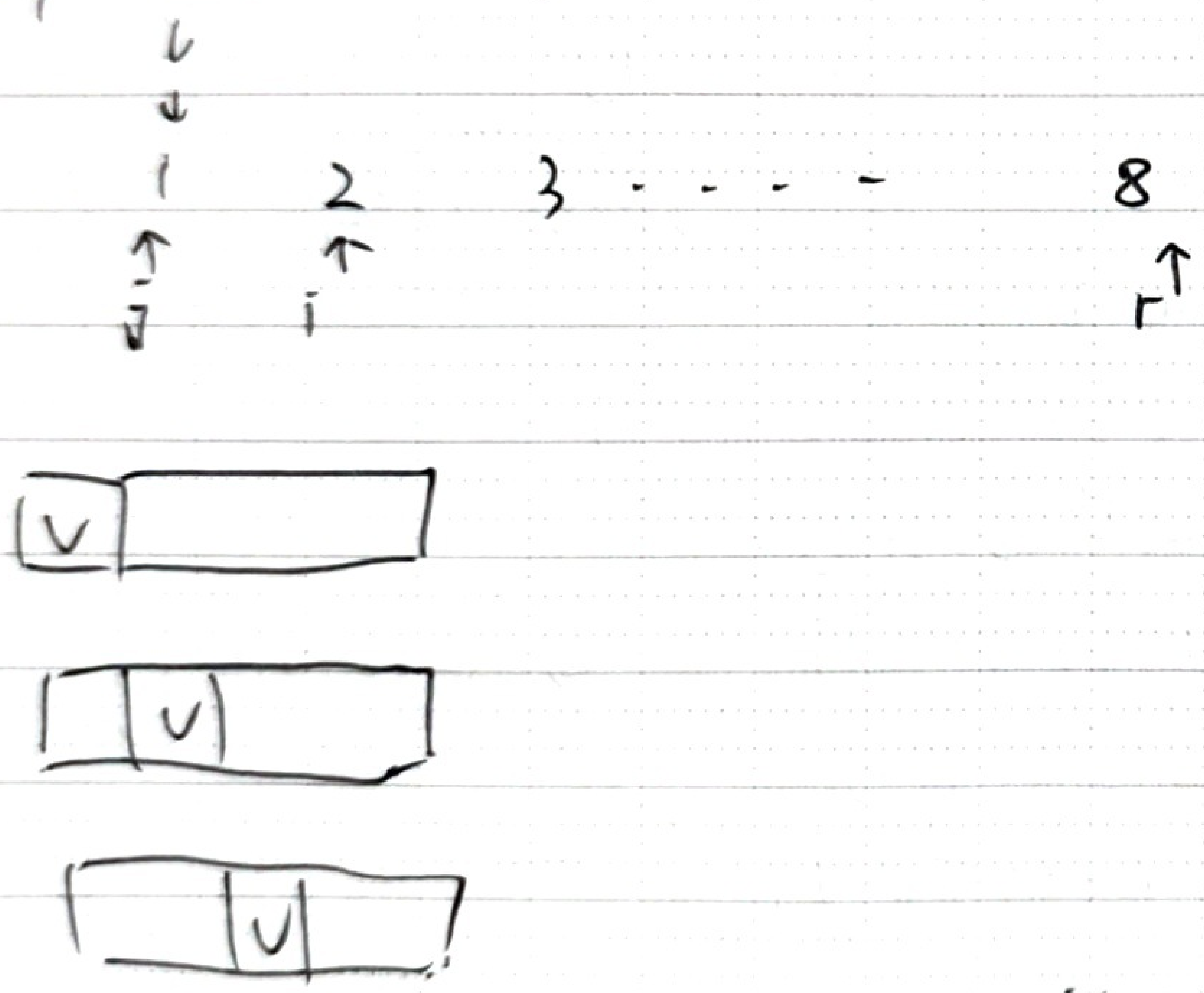
② if $arr[i] < v$: ~~arr~~ swap(arr[j+1], arr[i])

二、最后, swap(arr[l], arr[j])

初始:

第一版QuickSort的问题:

array ~~small~~ smallest → largese:



TC: $O(n^2)$: $n+(n-1)+\cdots 1 = \frac{(n+1)n}{2} \Rightarrow O(n^2)$

会堆栈溢出是因为递归深度$O(n)$

~~Opm~~ Optimazition: Randomize: 生成一个[l, r]区间的随机值

先生成[0, r-1]的random value

l+[0, r-1]区间的随机值 → [l, r]区间的随机值

```
partition{
    ~~swap(arr, l, r)~~
    int p = l + (new Random().nextInt(r-l+1));
    swap(arr, l, p);
}
```

Date

思考：

① 这样的 randomize 使得不存在特定的测试用例会使算法退化 (所以不是每次取中间值作为标定点.)

② 万一每次还是都取到了最小值？

$P = \frac{1}{n} \times \frac{1}{n-1} \cdots \cdots \frac{1}{1} = \frac{1}{n!}$ 比指数级还大.

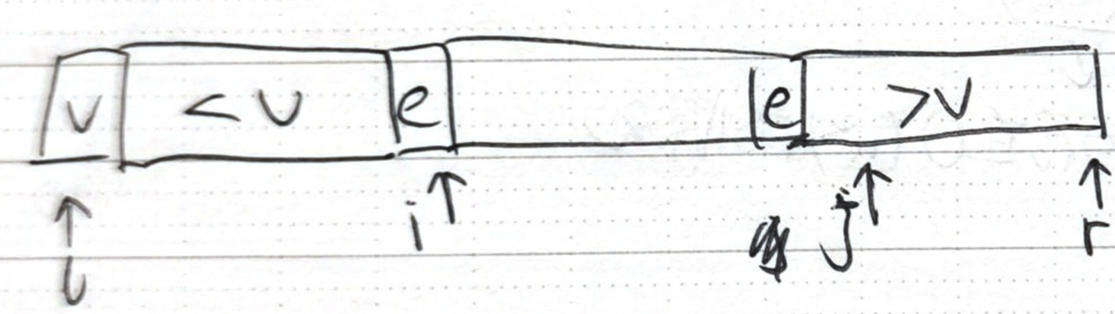
阶乘

Optimization:

一、所有元素都相等.

$TC: O(n^2)$

双路快排：



arr[l+1...i-1] <=v　　arr[j+1...r] >=v

i → 遇到 e>=v，

j 遇到 e<=v，

swap(arr, i, j)

←

ex.

4 6 5 2 3 8 7 1

l i j r

0 0 0 0 0 0 0 0

l i j r

// arr[l+1,... i-1] ⇐ v ; arr[j+1... r] >= v 循环不变量

快排最能证到 $O(n^2)$ 的 worst case 的概率很低，没有实际意义

快排是一个随机算法：pivot的选择随机

所以对快排的TC，我们可以看使用期望

- 层数期望值 $O(\log n)$
- 复杂度期望值 $O(n\log n)$

普通算法：有 worst case → 能够找到一组数据使算法100%悲化的概率很低

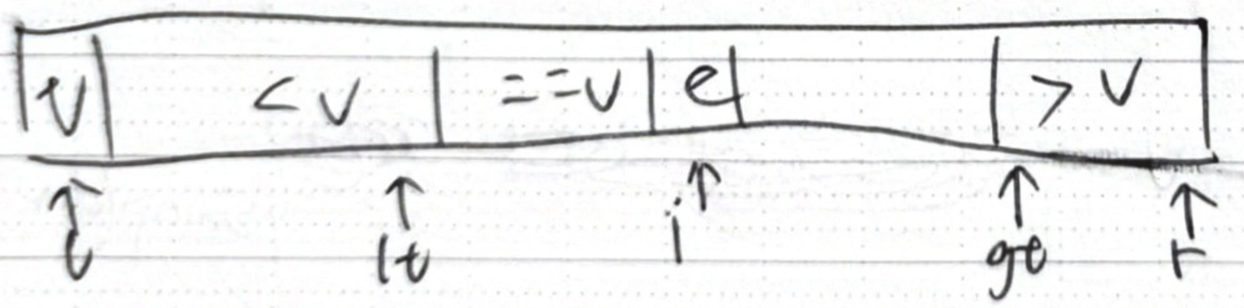
随机~~：有期望 → 找到

(双路，但不是单路)

↓

因为存在 $O(n^2)$ 的情况

三路：



arr[l+1...lt] < v　　arr[gt...r] > v

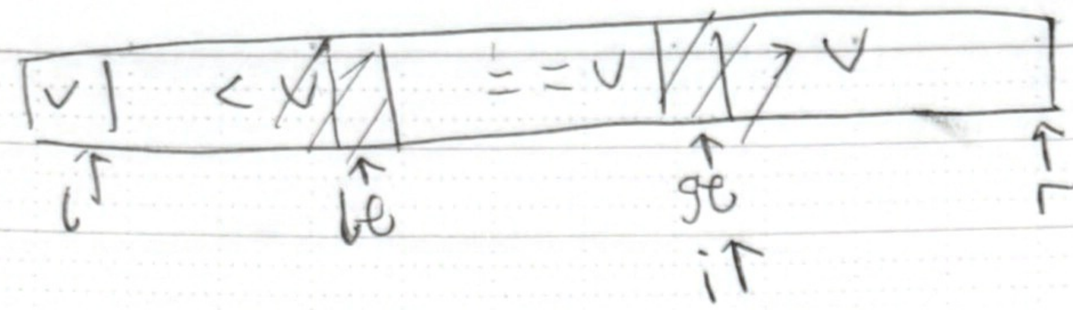
arr[lt+1...i-1] == v

① if e == v, i++

② if e < v, swap(arr, lt+1, i); lt++; i++

③ if e > v, swap(arr, gt-1, i); gt--

当 i >= gt 时，循环结束：

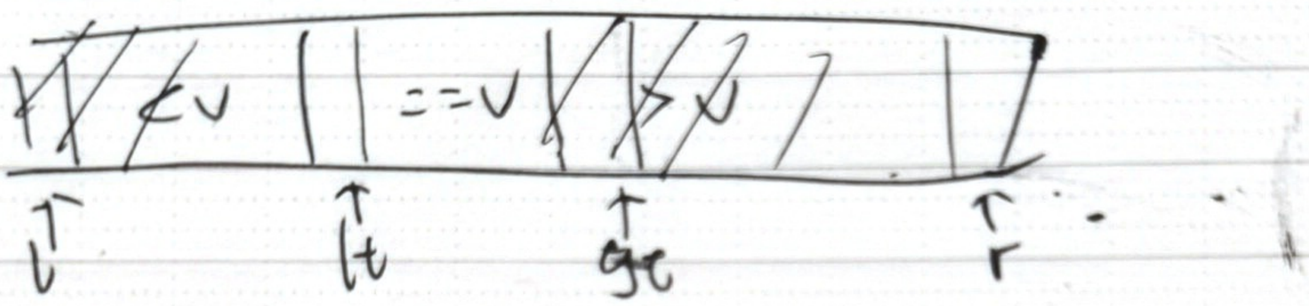


arr[lt+1 ... lt] < v

arr[lt+1 ... i-1] == v

arr[gt ... r] > v

swap(arr, l, lt):



循环不变量：arr[l ... lt-1] < v

arr[lt, gt-1] == v

arr[gt, r] > v

此题递归 < v 和 > v 的部分，Best case: O(n)

总结：一、两路快排：

完全有序的数据退化成 $O(n^2)$

⇒引入随机化

⇒所有元素都有一样，退化成 $O(n^2)$

二、双路快排：O(nlogn) 解决（可以把pivot swap到中间）

Best case O(nlogn)

三、三路快排：可以不用递归 ✓ arr[lt+1 ... i-1] == v

Best case: O(n)