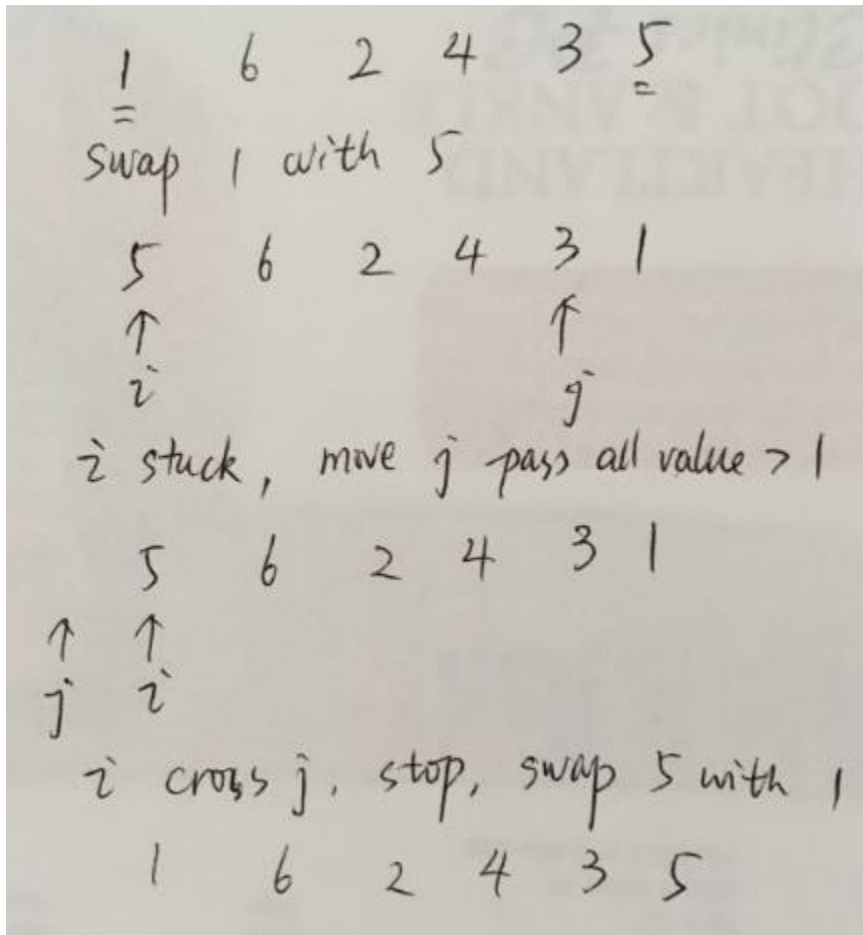


## Lab 5 Solutions

### Problem 2



### Problem 3.

- Good pivots: 2,3,3,4,5
- Yes: 5/9 of the elements are good pivots.

Problem 4.

[22/6] Give an  $o(n)$  (that is, better than  $\Theta(n)$ ) algorithm for determining whether a sorted array  $A$  of distinct integers contains an element  $m$  for which  $A[m] = m$ , and then implement as a Java function

```
int findFixedPoint(int[] A)
```

which returns such an  $m$  if found, or -1 if no such  $m$  is found. You must also provide a proof that your algorithm runs in  $o(n)$  time.

**Step 1: If  $A[0] = 0$ , return 0.**

**Step 2: If  $A[0] > 0$ , return -1.**

**Step 3: Do binary search. The base case will examine  $A[mid]$  to see if  $A[mid] = mid$ , and if so, return  $A[mid]$  (it's the a fixed point). If  $A[mid] > mid$ , search the left side. If  $A[mid] < mid$ , search the right side. If the usual failure signal occurs (lower > upper) return -1. This solves the problem in  $O(\log n)$ .**

5. Devise a pivot-selection strategy for QuickSort that will guarantee that your new QuickSort has a worst-case running time of  $O(n \log n)$ .

**Soln:** Use the super QuickSelect algorithm (with worst case running time  $O(n)$ ) to select pivots each time. This adds  $O(k)$  running time whenever section of the area has length  $k$ , so has the same cost as the partition step. Using this algorithm guarantees that all pivots are good pivots, so the recursion tree has height  $O(\log n)$  and running time is  $O(n \log n)$  in the worst case.