CS 2500 - ALGORITHMS

Solutions to HW 2

Problem 1

① Selection sort :

```
Selection-Sort (A)
1       for i = 1 to A.length
2           j = minimumindex (A, i)
3           Swap A[i] with A[j]
4       end
```

```
minimumindex (A, i)
    k = i
    for j = (i+1) to A.length
        if A[j] < A[k]
            k = j
        end
    end
    return k
```

Note that minimumindex (A, i) procedure is similar to the maximumelement procedure from HW 1.

⇒ The best-case and worst-case run time for Line 2 in Selection-Sort routine is $\Theta(n)$.

P.T.O.

The for-loop in Line 1 is executed n-times.

$\Rightarrow$ Lines 2 and 3 are executed n-times each.

$\Rightarrow$ Run-time of selection sort (for both best and worst case) is $\Theta(n^2)$.

(2) Quicksort — Python implementation.

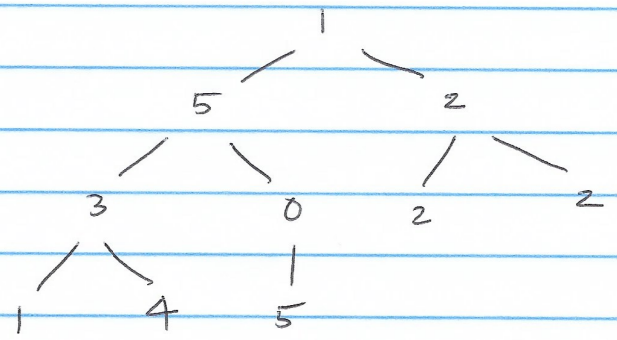
Code will be provided in a GitHub repository.

(3) Input: $A = \{1, 5, 2, 3, 0, 2, 2, 1, 4, 5\}$.

Note: $n = 10$.

(a) Heap sort:

Heap representation of A:

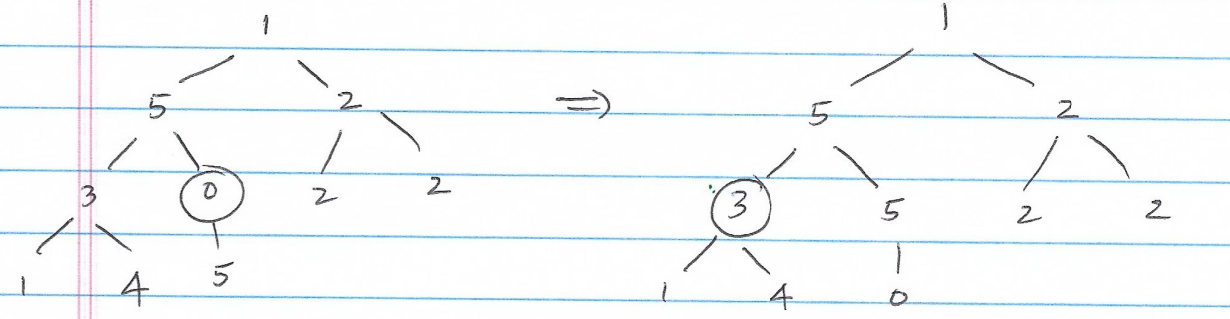


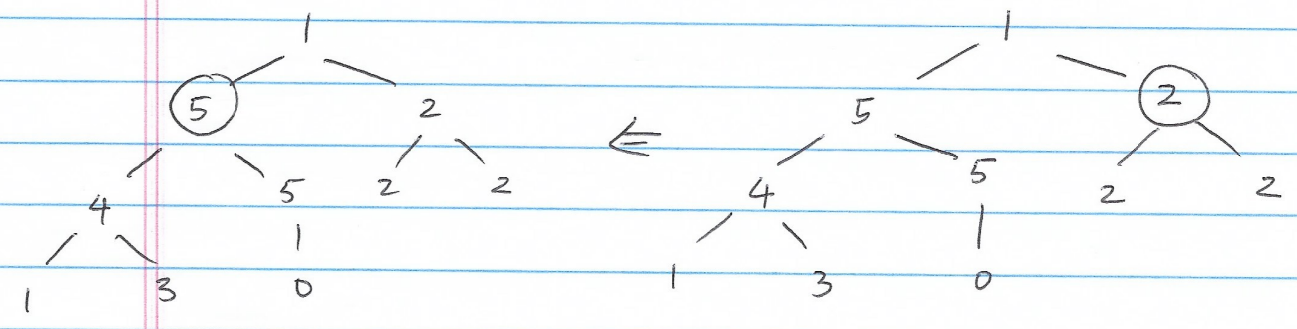
We first build max-heap on this input.

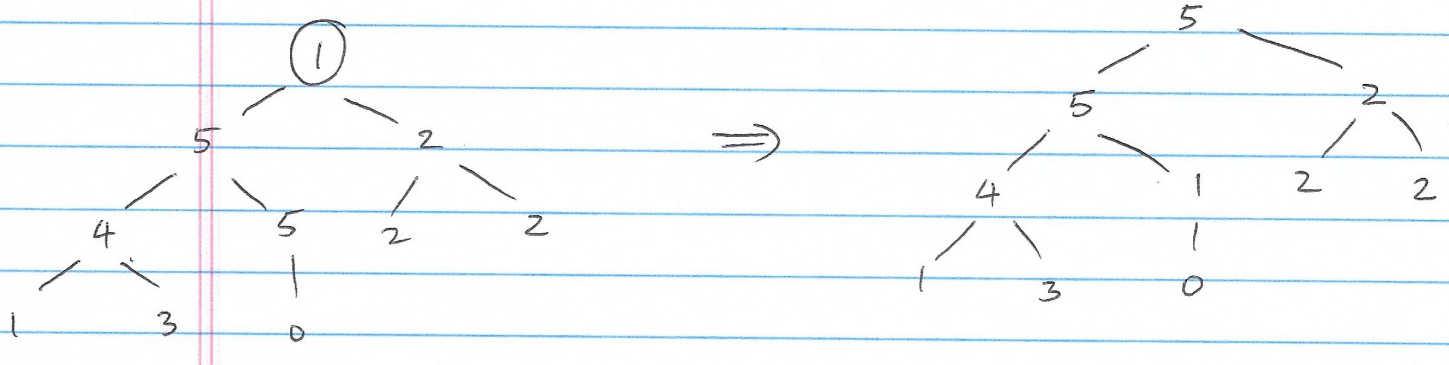
P.T.O.

#3

Build-max heap :

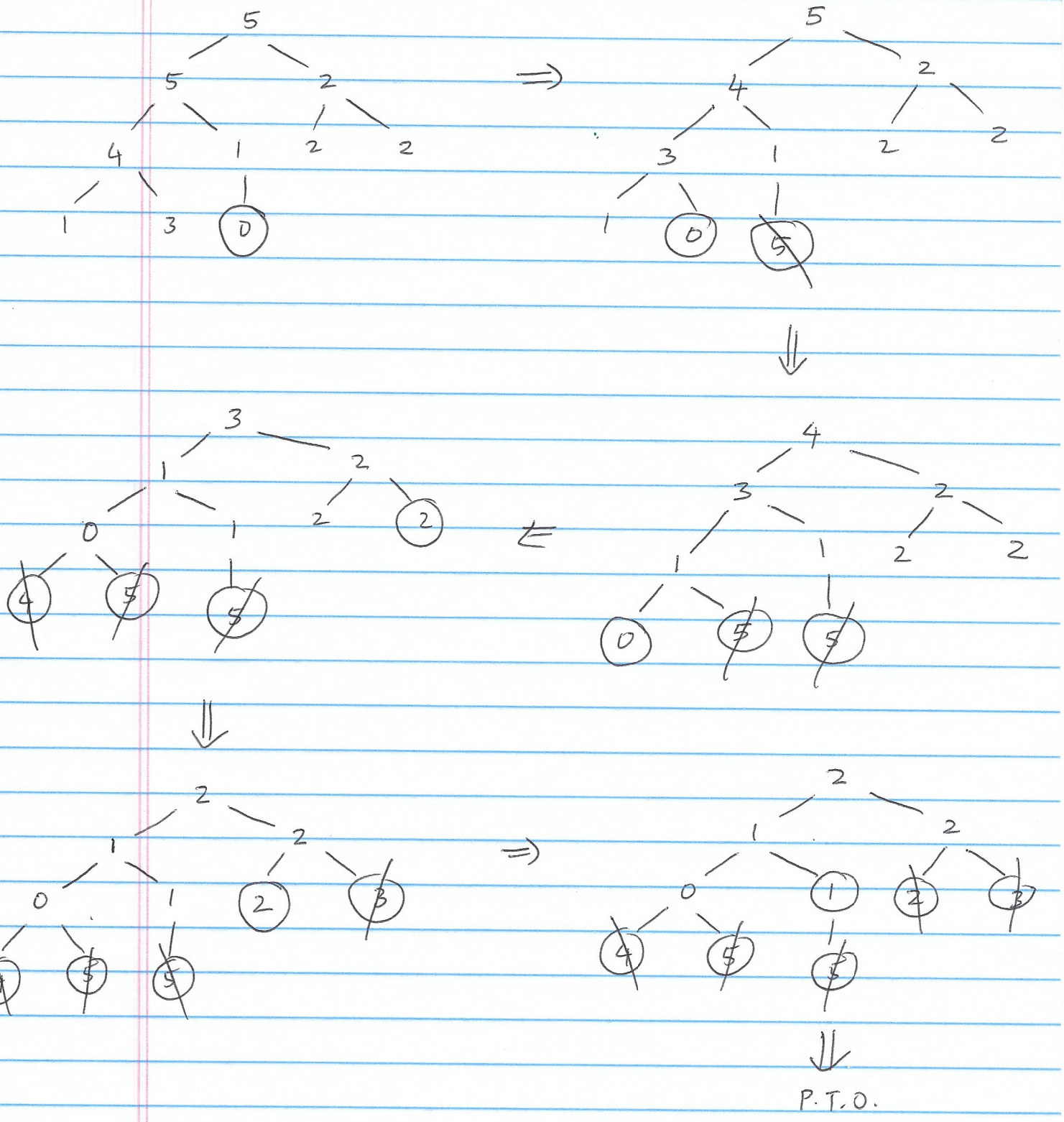






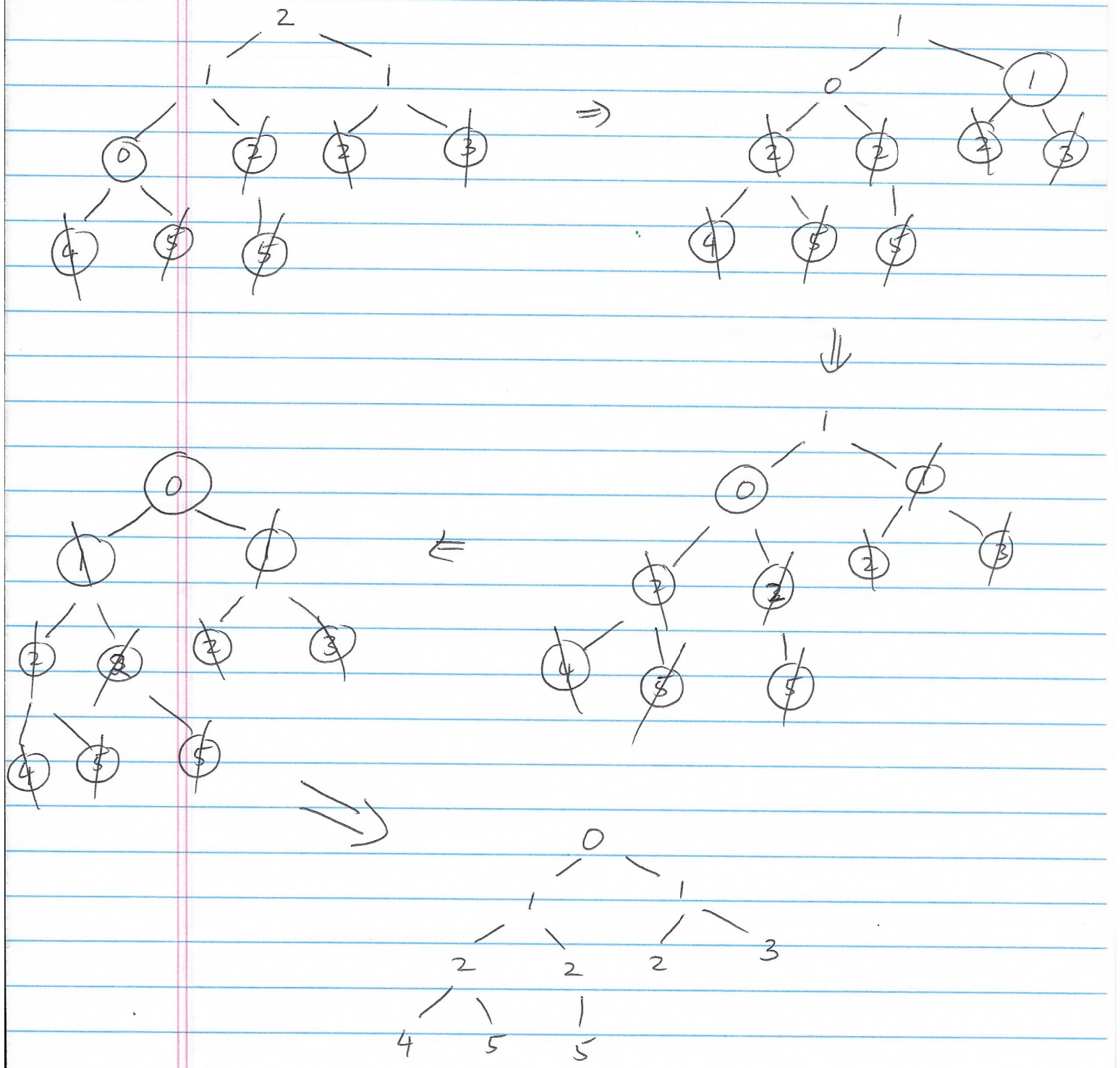
#4

Heap-sort :



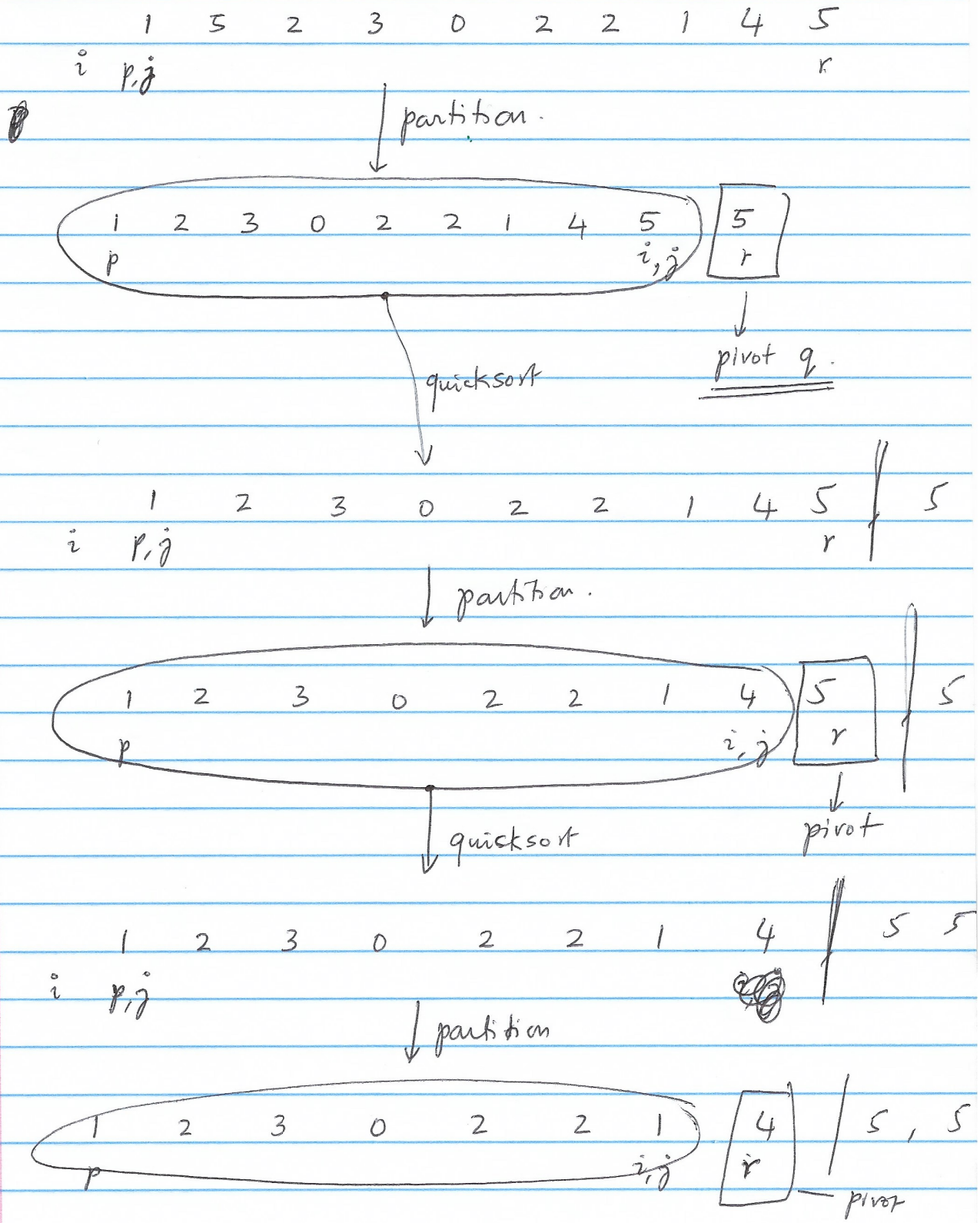
P.T.O.

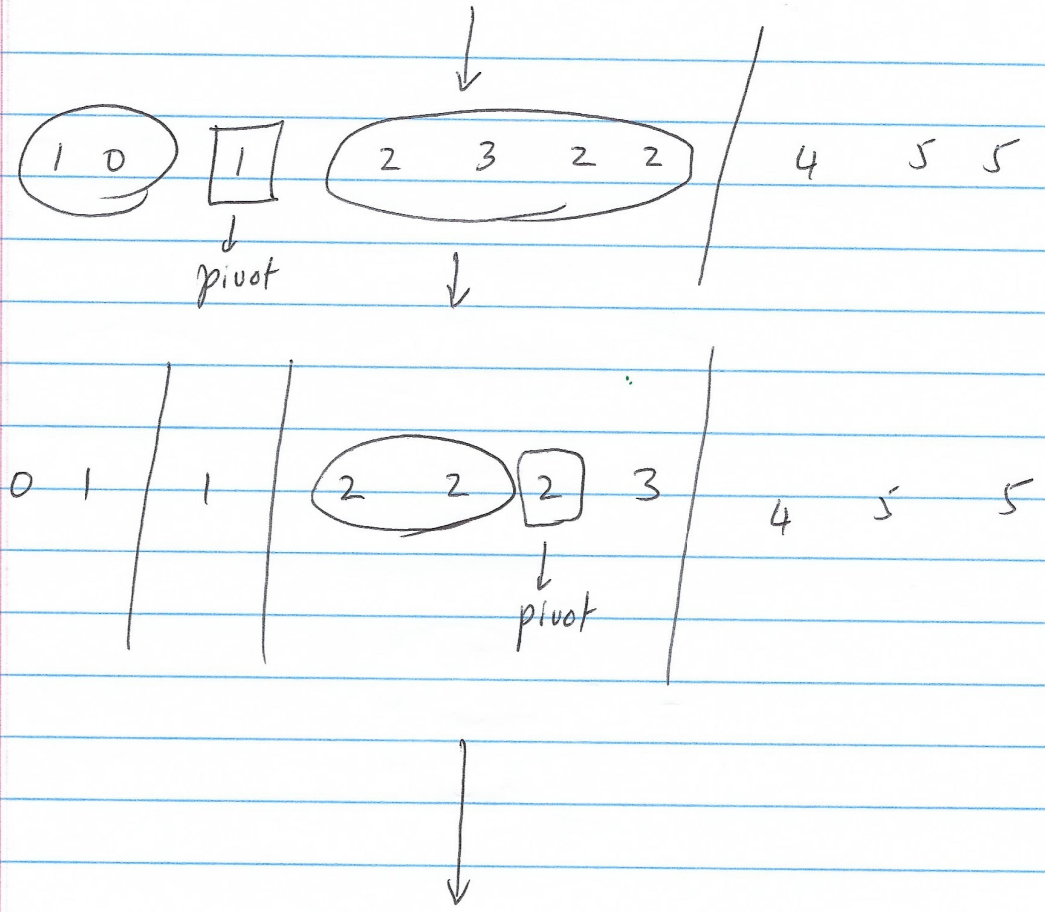
#5


2
1
1
0
2
2
3
4
5
5
⇒
1
0
1
2
2
2
3
4
5
5
⇓
1
0
1
2
2
2
3
4
5
5
⇐
0
1
1
2
2
2
3
4
5
5
⇘
0
1
1
2
2
2
3
4
5
5

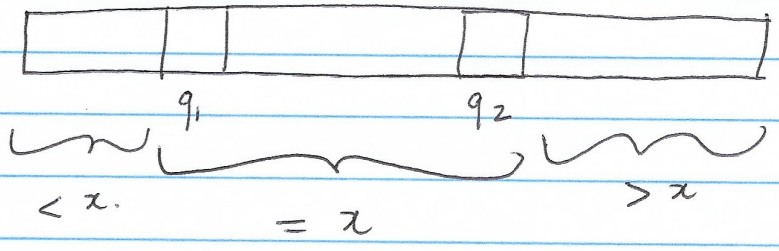
P.T.O.

(b) Quicksort :


1 5 2 3 0 2 2 1 4 5
i p,j
r
partition.
1 2 3 0 2 2 1 4 5
p
i,j
5
r
pivot q.
quicksort
1 2 3 0 2 2 1 4 5
i p,j
r
5
partition.
1 2 3 0 2 2 1 4
p
i,j
5
r
pivot
quicksort
1 2 3 0 2 2 1 4
i p,j
5 5
partition
1 2 3 0 2 2 1
p
i,j
4
r
5, 5
pivot



④ 3-way Quicksort:



$\Rightarrow T(n) = T(q_1 - 1) + T(n - q_2) + \Theta(n)$

Note that the new partition also takes $\Theta(n)$ steps.

$\Rightarrow$ the worst-case run time is

$$T_{WC}(n) = \underset{q_1, q_2}{\text{maximize}} \left[T(q_1 - 1) + T(n - q_2) \right] + \Theta(n)$$

such that $1 \leq q_1 \leq q_2 \leq n$.

If $q_1 = q_2 = n-1$, we obtain the same as traditional quicksort.

$\Rightarrow T_{WC}(n) = \Theta(n^2)$.

However, one can find a significant gain in average run-time if repetitions are allowed in the input.

For instance, if $q_1 = 1$, $q_2 = n$,

Then $T_{BC}(n) = 2 \cdot T(0) + \Theta(n)$

$T(0)$ = 0

$\Theta(n)$ ↓ time needed to detect that all terms are indeed equal.

$\therefore T_{BC}(n) = \Theta(n)$.