7.4-5

Even though it doesn't always mean that the ~~Quick sort~~ $T(n/2)$, But from that, since we stop at time when length reaches to k can we stop the recursion usually stops when $\lg(n/k)$ times of Partition, And the running time would be $O(n \lg n/k)$ so the $\frac{n}{k}$ ~~sec~~ array is not sorted.

Since the average Insertion sort time is $O(n^2)$,

so $\frac{n}{k} O(k^2)$, would be the time for Insertion sort the whole array $\downarrow O(nk)$

These two add together, we have $O(n \lg n/k) + O(nk)$

$= O(nk + n \lg n/k)$.

In Practice, I think we should try to select the best k by doing experiments.

7.4

a. Because it simply replaces the Procedure "Sorting the right array" to $P = q+1$. Next loop while will be

$q_2 =$ PARTITION $(A, q+1, r)$

↑ the right time element

TAIL-Recursive-Quicksort $(A, P = q+1, q_2 - 1)$

~~the It will normally~~ the loop itself helps to sort the right array and the recursion helps to sort the left array

b. One recursive calls will take one, so if the input is already sorted, there will $n-1$ input recursive calls.

c. ~~while $P < r$.~~

C.

```
while p<r:
    q = PARTITION(A, p, r)
    if   q < r/2
        TAIL-Recursive-Quicksort(A, p, q-1)
        p = q+1
    else
        tail_recursive_quicksort(A, q+1, r)
        r = q-1
```

Because the maximum call depends on how many times we need to divide it in two -so stack depth would be $O(\lg n)$

7-6 Fuzzy Sort.

~~To be honest, I searched this on Google. I don't even have any idea what it means at first glimpse, but I still managed to interpret this~~

We firstly define Partition

~~Partition(Interval[], p, r)~~
~~i = p~~
~~j = p~~
~~k = r~~
~~while (j<k)~~
~~if (Int[j] end <= Pivot.begin)~~
~~exchange(Int[i], Int[j])~~
~~i = i+1~~
j = j+1

~~else:~~
~~exchange Int[k], Int[j]~~

7-6 (Inspired by CLRS, ~~III~~ I don't even know what they try to ask If I don't see it)

We can start by fuzzy sort Intervals' Left Point

~~If the left-Point are sorted, with two overlapping Intervals~~

~~[G, G]~~ And the overlapping area don't need to be sorted.

As long as we design algorithm to identify overlap

Find-Intersection (A, P, r) Here A is the sequence of Interval

rand = random(P, r)

exchange A[rand] with A[r]

a = A[r].a

b = ~~A~~ A[r].b

for i = P to r:

if A[i].a ≤ b && A[i].b ≥ a

if A[i].a > a

a = A[i].a

if A[i].b < b

b = A[i].b

return (a, b)

(a, b) it's the initial "First overlapping" of the sequence.

Similarly as Quick-sort, we can design the Fuzzy-sort (A, P, r)

while p < r

(a, b) = Find-Intersection (A, P, r)

t = PARTITION-right (A, a, P, r)

q = PARTITION-LEFT (A, b, P, t)

Fuzzy-sort (A, P, q-1)

Fuzzy-sort (A, t+1, r)

the Partition-right and-Left algorithm, are designed to Partitionize the subarray from p to t with a Pivot-value equals to right end Point b (Find-Intersection) and a.

扫描全能王 创建

PARITION-Right (A, a, p, r)

```
i = p-1
for j = p to r-1   // loop the whole sequence
  if A[j].a ≤ a        Here we use a as Pivot-value
      i = i+1
      exchange A[i] with A[j]
exchange A[i+1] with A[r]
return i+1
```

b. If A[i] have no overlap with each other,
then It's going to be the worst case $O(n\lg n)$.
Cuz it's no different to quick sort
Because It returns [a,b] which is randomized at first
~~(i.e.)~~ And in Fuzzy-sort, Find-Intersection is $\Theta(n)$,

$$T(n) \le 2T(\frac{n}{2}) + \Theta n = O(n\lg n)$$

↑ Because the rand( ) selected random interval

If the intervals of overlap [a,b] are shared by all the elements in A, these recursion will be $O(1)$ and Fuzzy-sort would be $O(n)$