P7.1

P7.1b

Loop invariant: elements smaller than key are collected at 1 i-1; greater than key are from j to the end. so after exchange j will finally stop at the first half, i will stop at the second half, this will cause return.

P7.1c

j is the last element that is smaller than key, so at most r-1.

P7.1d

In the end of the loop, j stops when $j < i$, meaning the whole sequence is scanned. j will return the first item it met(scan from end to beginning) that is smaller or equal to key.Elements beyond j will be ones greater than the key.

P7.2

P7.2a

$E = \frac{1}{n}$

P7.2b

P7.2c

$X_q = \frac{1}{n}$, sum is symmetry

P7.2d

divide into two parts: 1 through $\lceil \frac{n}{2} \rceil - 1$ and $\lceil \frac{n}{2} \rceil$ through n

$$\sum_{k=1}^{\lceil \frac{n}{2} \rceil - 1} k \lg(k) \leq \lg(\left\lceil \frac{n}{2} \right\rceil - 1) \sum_{k=1}^{\lceil \frac{n}{2} \rceil - 1} k$$

$$\lg(\left\lceil \frac{n}{2} \right\rceil - 1) \leq \lg(\frac{n}{2})$$

so we have

$$\sum_{k=1}^{\lceil \frac{n}{2} \rceil - 1} k \lg(k) \leq (\lg(n) - 1) \sum_{k=1}^{\lceil \frac{n}{2} \rceil - 1} k$$

for the second part:

$$\sum_{k=\lceil \frac{n}{2} \rceil}^{n-1} k \lg(k) \leq \lg(n-1) \sum_{k=\lceil \frac{n}{2} \rceil}^{n-1} k \leq \lg(n) \sum_{k=\lceil \frac{n}{2} \rceil}^{n-1} k$$

Add up these two we have

$$\sum_{k=1}^{n-1} k \lg(k) \leq \lg(n) \sum_{k=1}^{n-1} k - \sum_{k=1}^{\lceil \frac{n}{2} \rceil - 1} k = \frac{n^2}{2} \lg(n) - \frac{n}{2} \lg(n) - \frac{\lceil \frac{n}{2} \rceil (\lceil \frac{n}{2} \rceil - 1)}{2} \leq \frac{n^2}{2} \lg(n) - \frac{n^2}{8}$$

P7.2e

$$E(T(n)) \leq an \lg(n) - bn$$

substitute in,

$$E(T(n)) = \frac{2}{n} \sum_{q=1}^{n-1} E(T(q)) + \Theta(n) \leq \frac{2a}{n} (\frac{n^2}{2} \lg(n) - \frac{n^2}{8}) - \frac{2b}{n} \frac{n(n-1)}{2} = an \lg(n) - \frac{an}{4} - b(n-1) = \Theta(n \lg(n))$$

P7.3

P7.3a

sequence boiled down to 3-element units, compare first two, then second two, then first two will render correct order to this unit. This will be the iteration invariant. From 3 to n, line 6 order the first 2/3, line 7 the second. between i+k to j-k there are j-i-2k+1 elements, larger than or equal to the unoverlapped part between the two 2/3. This assures that all elements are captured.

P7.3b

$$T(n) = 3T(\frac{2n}{3}) + \Omega(1)$$

Use master theorem, running time is $O(n^{\log_{\frac{3}{2}} 3}) = O(n^{2.7})$

P7.3c

No. The running time is worse than all the other sorting algorithms.

P7.4

P7.4a

partitioned sequence is sorted and invariant,and the rest is added into in the next loop until the start meets the end

P7.4b

every time executing partition, key happens to be the greatest and returns r to q; therefore, needs n recursions to make p and r meet(line 1). Each recursion requires O(1).

P7.4c

each time after partition, choose the shorter part to do the sort. if the shorter one is the left one, p = q+1 for the next loop; otherwise, r = r-1 for the next loop

```python
def kwiksort2(A, p, r):
    while p < r:
        q = partition(A, p, r)
        if q - p <= r - q:
            kwiksort2(A, p, q-1)
            p = q + 1
        else:
            kwiksort2(A, q+1, r)
            r = r - 1
```

P7.5

P7.5a

$$pi = \frac{6(i-1)(n-i)}{2n(n-1)(n-2)}$$

P7.5b

$$p_{median} = \frac{3(n-1)}{n(n-2)}$$

$$\lim_{n\to\infty} \frac{6(i-1)(n-i)}{n(n-1)(n-2)} / \frac{1}{n} = \lim_{n\to\infty} \frac{6n(n/2-1)(n/2)}{(n-1)(n-2)} = 1.5)$$

P7.5c

old version: $\frac{1}{3}$
new version: $1 - 3 \times \frac{1}{3}^2 \times \frac{2}{3} \times 2 - \frac{1}{3}^3 \times 2 = \frac{13}{27}$

P7.5d only change the probability of getting a better split. the analysis of ordinary algorithm still applies.

P7.6

P7.6a

Two intervals have intersection, call them equal. Use function to find intersection, then partition the interval set with this pivot intersection into three groups.
First use a, partition into two parts: $a_i \leq a$ and $a_i > a$; return the beginning index of the right-hand part. Then for the left-hand part, partition with b. This will separate ones with $a_i \leq a$ and $b_i < b$ into left-hand part and $a_i < aandb_i \geq b$ into middle part. Ones in middle part can be neglected because they contain the pivot intersection, so they will share all the intersections that [a, b] will share with other intervals. Then do the iteration sort on right-hand and left-hand part of the sequence only. Therefore the longer the pivot intersection is, the less intervals you need to really sort. This will produce the fuzzy-ordered output.

```python
import random as rand
rand.seed(a = 17)
def find_intersection(A, p, s):
    r = rand.randrange(p, s+1)
    x = A[r].copy()
    A[s],A[r] = A[r],A[s]
    for i in range(p, s):
        if x[1] >= A[i][0] and x[0] <= A[i][1]:
            x[0] = max(x[0], A[i][0])
            x[1] = min(x[1], A[i][1])
    return x
def partition_right(A, p, s, x):
    a = x[0]
    i = p - 1
    for j in range(p, s+1):
        if A[j][0] <= a:
            i += 1
            A[i],A[j] = A[j],A[i]
    return i+1
def partition_left(A, p, s, x):
    b = x[1]
    print(b)
    i = p - 1
    for j in range(p, s+1):
        # print(A[j][1])
        if A[j][1] < b:
            i += 1
            A[i],A[j] = A[j],A[i]
            # print(A)
    return i+1
def fuzzy_sort(A, p, s):
    if p < s:
```

```
        pivot = find_intersection(A,p,s)
        print(pivot)
        r = partition_right(A,p,s, pivot)
        q = partition_left(A,p,r, pivot)
        fuzzy_sort(A, p, q-1)
        fuzzy_sort(A, r+1, s)



A = [[2,5],[6,7],[4,6],[1,3],[3,4],[10,12],[23,27],[18,24],[9,19]]
fuzzy_sort(A,0,len(A)-1)
print(A)
```