Chapter 7. Quicksort

7.1-1.
Swap $9-13, 5-19, 8-13, 7-19, 4-12, 2-13, 6-19, 11-12$.

7.1-2.
It returns r.
We can count the number in which $A[j] = x$ and subtract the half of the number from the returning value to get desired answer.

7.1-3.
The for loop executes $\Theta(n)$ times and the operation inside the loop takes constant time.

7.1-4.
Fix inequality $A[j] \leq x$ to $A[j] \geq x$.

7.2-1.
$T(n) = T(n-1) + \Theta(n) \leq c_1(n-1)^2 + c_2 n = c_1 n^2 + (c_2 - 2c_1)n + c_1$. For $2c_1 > c_2$ and $n \geq c_1/(2c_1 - c2)$, we have $T(n) \leq c_1 n^2$.

7.2-2.
$\Theta(n^2)$, because PARTITION decreases the size of subproblem only by 1.

7.2-3.
If the array is sorted in decreasing order, PARTITION decrease the size of subproblem only by 1, hence $T(n) = \sum_{i=1}^{n} \Theta(i) = \Theta(n^2)$.

7.2-4.
If the array is almost-sorted, insertion sort will be close to linear, because the running time of insertion sort is the sum of number of inversion in the

array and the length of the array. In contrast, the number of subproblems decreased by each PARTITION is number of inversions that includes the pivot element, so if the array is almost-sorted, quicksort will performance worse.

7.2-5.
$\alpha^k n = \Theta(1) \Rightarrow k = -\lg n/\lg \alpha + c_1$.
$(1-\alpha)^l n = \Theta(1) \Rightarrow l = -\lg n/\lg(1-\alpha) + c_2$.

7.2-6.
Set an ordering to elements of the array. If PARTITION picks a pivot that is in the smallest or the largest αn element, the partition will produce a split less balanced than $1-\alpha$ to α. Therefore the probability of producing a split more balanced is $1-2\alpha$.

7.3-1.
Because the randomized algorithm represents the average running time better.

7.3-2.
Worst case: $T(n) = T(n-1) + 1 = \Theta(n)$.
Best case: $T(n) = 2T(n/2) + 1 = \sum_{i=1}^{\lg n} 2^i = \Theta(n)$.

7.4-1.
Let $T(k) \geq (m/2)k^2 + (m/2)k$ for $k < n$, where m is the coefficient of $\Theta(n)$ term.
$T(n) \geq \max_{0 \leq q \leq n-1} (m/2)(q^2 + (n-1-q)^2) + (m/2)(n-1) + mn$.
Since $q^2 + (n-1-q)^2$ attains its maximum at $q = 0, n-1$ and its maximum value is $(n-1)^2$, $T(n) \geq (m/2)(n-1)^2 + (m/2)(n-1) + mn = (m/2)n^2 + (m/2)n$.
Therefore $T(n) = \Omega(n^2)$.

7.4-2.
The best case running time for quicksort has the recurrence $T(n) = \min_{0 \leq q \leq n-1}(T(q) + T(n-q-1)) + \Theta(n)$. Let $T(k) \geq ck\lg k + ck$ for $k < n$. Then $T(n) \geq c\min_{0 \leq q \leq n-1}(q\lg q + (n-q-1)\lg(n-q-1)) + c(n-1) + \Theta(n)$.
Since $q\lg q + (n-q-1)\lg(n-q-1)$ attains its minimum at $q = (n-1)/2$ and its minimum value is $(n-1)\lg((n-1)/2)$, $T(n) \geq c(n-1)(\lg((n-$

$1)/2) + 1) + \Theta(n) = c(n-1)\lg(n-1) + \Theta(n) \geq c(n-1)(\lg n - 1) + \Theta(n) = cn\lg n - c\lg n + c(n-1) + \Theta(n) \geq cn\lg n + cn$. Therefore $T(n) = \Omega(n\lg n)$.

7.4-3.
Let $f(q) = q^2 + (n-q-1)^2$. $f(0) = f(n-1) = (n-1)^2$. $f(q) = (n-1)^2 - 2q(n-1) + 2q^2 = (n-1)^2 - 2q(n-q-1) \leq (n-1)^2$.

7.4-4.

$$\mathbb{E}[X] = \sum_{i=1}^{n-1}\sum_{j=i+1}^{n}\frac{2}{j-i+1} = \sum_{i=1}^{n-1}\sum_{k=1}^{n-i}\frac{2}{k+1} > \sum_{i=1}^{n-1}\sum_{k=1}^{n-i}\frac{2}{2k} > \sum_{i=1}^{n-1}\ln(n-i+1) = \ln(n!) > (n/2)\ln($$

7.4-5.
The running time of quicksort part is $O(n\lg(n/k))$, and the running time of insertion sort part is $(n/k)O(k^2) = O(nk)$, therefore the total expected running time is $O(nk + n\lg(n/k))$.
Since the expected running time of normal quicksort is $O(n\lg n)$, if we choose k that the algorithm is improved, then $c_1 n\lg n > c_2 nk + c_3 n\lg(n/k) \Rightarrow (c_1 - c_3)\lg n > c_2 k - c_3\lg k$. The threshold k should be experimentally determined.

7.4-6.
Let $\beta = \min(\alpha, 1-\alpha)$. We have $0 < \beta < 1/2$. To get the partition that is worse than $1-\alpha$ to α, the picked element should be in the smallest βn or the largest βn elements. This means that two of three elements need to be in the smallest βn or the largest βn elements. Hence the probability to get the bad partition is $3 \cdot 4\beta^2(1-2\beta) + 8\beta^3 = 12\beta^2 - 16\beta^3$. The probability to get the good partition is $1 - 12\beta^2 + 16\beta^3$.

7-1.
a.
$x = 13$
$[6, 19, 9, 5, 12, 8, 7, 4, 11, 2, 13, 21], j = 11, i = 1$
$[6, 2, 9, 5, 12, 8, 7, 4, 11, 19, 13, 21], j = 10, i = 2$
$[6, 2, 9, 5, 12, 8, 7, 4, 11, 19, 13, 21], j = 9, i = 10$
b. We have the following loop invariant: Before comparing i and j, $A[p, \cdots, i-1] \leq x$ and $A[j+1, \cdots, r] \geq x$.

Initialization: Trivial.
Maintenance: By exchanging i and j, we have $A[p, \cdots, i] \leq x$ and $A[j, \cdots, r] \geq x$. Incrementing i and decrementing j maintain this invariant.
Termination: The loop terminates when $i \geq j$, and this ensures the returning value to be a valid partition.
By this loop invariant, we can conclude that i and j always access valid element of the array.
c. By the loop invariant discovered in b, the returning value has $p \leq j < r$.
d. This is the termination criteria of the loop invariant.

e.

Algorithm 1: QUICKSORT(A, p, r)

```
if $p < r$ then
    q = HOARE-PARTITION(A, p, r);
    QUICKSORT(A, p, q);
    QUICKSORT(A, q + 1, r);
end
```

7-2.
a. Each iteration reduces the size of subproblem by 1, giving the running time $\Theta(n^2)$.
b.

Algorithm 2: PARTITION'(A, p, r)

```
x = A[p];
l = p;
m = p;
h = r;
while $m \leq h$ do
    if $A[m] < x$ then
        exchange $A[l]$ and $A[m]$;
        l += 1;
        m += 1;
    else if $A[m] > x$ then
        exchange $A[m]$ with $A[h]$;
        h -= 1;
    else
        m += 1;
    end
end
return (l, m - 1);
```

c.

Algorithm 3: QUICKSORT'(A, p, r)

```
if $p < r$ then
    (l, m) = PARTITION'(A, p, r);
    QUICKSORT'(A, p, l - 1);
    QUICKSORT'(A, m + 1, r);
end
```

d.
Since QUICKSORT' reduces the length of ranges on search by at least 2 each time, the elements need not be all distinct.

7-3.
a. $\mathbb{E}[X_i] = 1/n$.
b. The expected running time of quicksort can be calculated by summing over events that an arbitrary element is picked as a pivot. If the element q is selected as a pivot, the partition will produce two subarrays with length

$q-1$ and $n-q$, running in the linear time. This concludes

$$\mathbb{E}[T(n)] = \mathbb{E}[\sum_{q=1}^{n} X_q(T(q-1)+T(n-q)+\Theta(n))].$$

c.

$$\mathbb{E}[T(n)] = \mathbb{E}[\sum_{q=1}^{n} X_q(T(q-1)+T(n-q)+\Theta(n))] = \sum_{q=1}^{n} \mathbb{E}[X_q(T(q-1)+T(n-q)+\Theta(n))]$$

$$= \sum_{q=1}^{n}(T(q-1)+T(n-q)+\Theta(n))/n = \Theta(n) + \frac{1}{n}\sum_{q=1}^{n}(T(q-1)+T(n-q))$$

$$= \Theta(n) + \frac{2}{n}\sum_{q=2}^{n-1} T(q).$$

d.

$$\sum_{k=2}^{n-1} k\lg k = \sum_{k=2}^{\lceil n/2\rceil -1} k\lg k + \sum_{k=\lceil n/2\rceil}^{n-1} k\lg k$$

$$\le \sum_{k=2}^{\lceil n/2\rceil -1} k\lg k + \sum_{k=n/2+1}^{n} k\lg k \le \sum_{k=2}^{\lceil n/2\rceil -1} k\lg\frac{n}{2} + \sum_{k=n/2+1}^{n} k\lg n$$

$$= (\lg n-1)\sum_{k=2}^{n/2} k + \lg n \sum_{k=n/2+1}^{n} k\lg n = \frac{1}{2}n^2\lg n - \frac{1}{8}n^2 - \frac{64n\lg n + 32\lg n - 16n - 1}{64}$$

$$\le \frac{1}{2}n^2\lg n - \frac{1}{8}n^2$$

e.

$$\mathbb{E}[T(n)] = \frac{2}{n}\sum_{q=2}^{n-1}\mathbb{E}[T(q)] + \Theta(n) \le \frac{2}{n}\sum_{q=2}^{n-1} cn\lg n + \Theta(n)$$

$$\le \frac{2c}{n}(\frac{1}{2}n^2\lg n - \frac{1}{8}n^2) + \Theta(n) \le cn\lg n.$$

7-4.

a. TAIL-RECURSIVE-QUICKSORT differs from QUICKSORT by only one thing: it just changes $p = q+1$ and restart the loop. It generates the same

procedure.
b. This happens when $q = r$, that is, when the array is already sorted.
c.

Algorithm 4: TAIL-RECURSIVE-QUICKSORT'(A, p, r)

```
while $p < r$ do
    q = PARTITION(A, p, r);
    m = $\lfloor (p + r)/2 \rfloor$;
    if $q < m$ then
        TAIL-RECURSIVE-QUICKSORT(A, p, q - 1);
        p = q + 1;
    else
        TAIL-RECURSIVE-QUICKSORT(A, q + 1, r);
        r = q - 1;
    end
end
```

7-5.
a. The number of combinations is $n(n-1)(n-2)/6$. The probability that the i-th element to be pivot is that we pick three elements such that one is smaller than the i-th element, one is larger, and one is the i-th element. There are $(i-1)(n-i)$ ways to pick three elements in that way. Hence the probability is

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

b.

$$\lim_{n\to\infty} \frac{6(\lfloor \frac{n+1}{2} \rfloor - 1)(n - \lfloor \frac{n+1}{2} \rfloor)}{n(n-1)(n-2)} \Big/ \frac{1}{n} = \frac{3}{2}.$$

c.

$$\lim_{n\to\infty} \sum_{i=n/3}^{2n/3} \frac{6((n+1)/2 - 1)(n - (n+1)/2)}{n(n-1)(n-2)} = \lim_{n\to\infty} \int_{\frac{n}{3}}^{\frac{2n}{3}} \frac{6(n-x)(x-1)}{n(n-1)(n-2)} dx = \frac{13}{27}.$$

d. Even if we have the best case, that is, always the median of the whole subarray is chosen as the pivot, the depth of the recursion tree is still $\Theta(\lg n)$. Therefore the algorithm remains $\Omega(n \lg n)$.

7-6.

a.

Algorithm 5: INTERSECTION(A, p, r)

```
i = RANDOM(p, r);
exchange A[i] with A[r];
b = A[r].b;
e = A[r].e;
for $i = p$ to $r - 1$ do
    if $A[i].b \leq e$ and $A[i].e \geq b$ then
        b = max($A[i].b, b$);
        e = min($A[i].e, e$);
    end
end
return (b, e);
```

Algorithm 6: PARTITION-RIGHT(A, b, p, r)

```
i = p - 1;
for $j = p$ to $r - 1$ do
    if $A[j].b \leq b$ then
        i += 1;
        exchange $A[i]$ with $A[j]$;
    end
    exchange $A[i + 1]$ with $A[j]$;
    return i + 1;
end
```

Algorithm 7: PARTITION-LEFT(A, e, p, t)

```
i = p - 1;
for $j = p$ to $t - 1$ do
    if $A[j].e < b$ then
        i += 1;
        exchange $A[i]$ with $A[j]$;
    end
    exchange $A[i + 1]$ with $A[t]$;
    return i + 1;
end
```

Algorithm 8: FUZZY-SORT(A, p, r)

```
if $p < r$ then
    (b, e) = INTERSECTION(A, p, r);
    q = PARTITION-RIGHT(A, b, p, r);
    s = PARTITION-LEFT(A, e, p, t);
    FUZZY-SORT(A, p, q - 1);
    FUZZY-SORT(A, s + 1, r);
end
```

b. If all intervals are disjoint, the algorithm become identical to quicksort, providing the running time $\Theta(n \lg n)$. If all intervals overlap, INTERSECTION will return the common intersection in a single pass, and the algorithm terminates at that iteration, providing the running time $\Theta(n)$.