EL9343 Homework 5

1. Exercise 7.4-5 in CLRS Textbook.

Solution:

If we are only doing quick-sort until the problem size becomes ≤ k, then, we will have to take lg($\frac{n}{k}$) steps, since, as in the original analysis of randomized quick sort, we expect there to be lg(n) levels to the recursion tree. Since we then just call quicksort on the entire array, we know that each element is within k of its final position. This means that an insertion sort will take the shifting of at most k elements for every element that needed to change position. This gets us the running time described.

In theory, we should pick k to minimize this expression, that is, taking a derivative with respect to k, we want it to be evaluating to zero. So, n − $\frac{n}{k}$ = 0, so k = $\frac{1}{n^2}$. The constant of proportionality will depend on the relative size of the constants in the nk term and in the n lg($\frac{n}{k}$) term. In practice, we would try it with a large number of input sizes for various values of k, because there are gritty properties of the machine not considered here such as cache line size.

2. Problem 7-2 in CLRS Textbook.

Solution:

a. Since all elements are the same, the initial random choice of index and swap change nothing. Thus, randomized quicksort's running time will be the same as that of quicksort. Since all elements are equal, PARTITION(A, P, r) will always return r − 1. This is worst-case partitioning, so the runtime is $\Theta(n^2)$.

b.

Algorithm 1 PARTITION'(A,p,r)

```
x = A[p]
i = p − 1
k = p
for j = p + 1 to r do
    if A[j] < x then
        i++
        k++
        exchange (A[i], A[j])
        exchange (A[k], A[j])
        end if
    end
    if A[j] == x then
        k++
        exchange (A[k], A[j])
    end
end
return i + 1, k
```

c.

```
RANDOMIZED-PARTITION'
i = RANDOM(p, r)
exchange A[r] with A[i]
return PARTITION' (A,p,r)

QUICKSORT' (A,p,r)
if p < r then
    (q, t) = RANDOMIZED − P ART IT ION0
    QUICKSORT' (A,p,q-1)
    QUICKSORT' (A,t+1,r)
end if
```

d.Let d be the number of distinct elements in A. The running time is dominated by the time spent in the PARTITION procedure, and there can be at most d calls to PARTITION. If X is the number of comparisons performed in line 4 of PARTITION over the entire execution of QUICKSORT', then the running time is O(d + X). It remains true that each pair of elements is compared at most once. If z_i is the i th smallest element, we need to compute the probability that z_i is compared to z_j . This time, once a pivot x is chosen with $z_i \leqslant x \leqslant z_j$, we know that z_i and z_j cannot be compared at any subsequent time. This is where the analysis differs, because there could be many elements of the array equal to z_i or z_j , so the probability that z_i and z_j are compared decreases. However, the expected percentage of distinct elements in a random array tends to $1 - \frac{1}{e}$, so asymptotically the expected number of comparisons is the same.

3. Similar to Figure 8.2, illustrate the operation of COUNTING-SORT on

A = [5, 7, 3, 1, 3, 6, 2, 1, 3, 5]

Solution:

	1	2	3	4	5	6	7	8	9	10
A	5	7	3	1	3	6	2	1	3	5

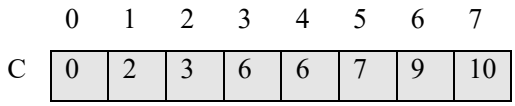
	0	1	2	3	4	5	6	7
C	0	2	1	3	0	2	1	1

(a)

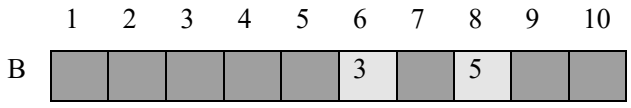
	0	1	2	3	4	5	6	7
C	0	2	3	6	6	8	9	10

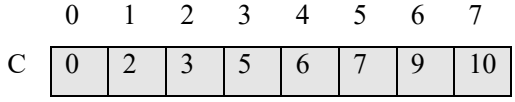
(b)

	1	2	3	4	5	6	7	8	9	10
B								5		

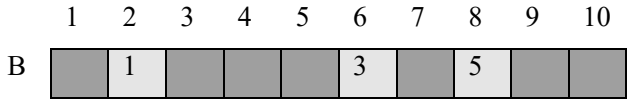


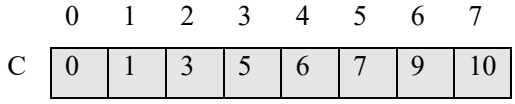
(c)





(d)



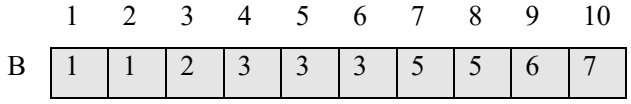


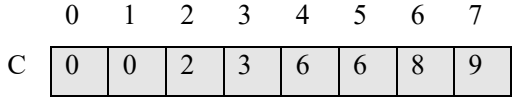
(e)

.

.

.





(f,final)

4. Exercise 9.1-2 in CLRS Textbook.

Solution:

If n is odd, there are

$$\begin{aligned} 1 + \frac{3(n-3)}{2} + 2 &= \frac{3n}{2} - \frac{3}{2} \\ &= (\left\lceil \frac{3n}{2} \right\rceil - \frac{1}{2}) - \frac{3}{2} \\ &= \left\lceil \frac{3n}{2} \right\rceil - 2 \end{aligned}$$

comparisons.

If n is even, there are

$$\begin{aligned} 1 + \frac{3(n-2)}{2} &= \frac{3n}{2} - 2 \\ &= \left\lceil \frac{3n}{2} \right\rceil - 2 \end{aligned}$$

comparisons.

5. Problem 9-1 in CLRS Textbook.

Solution:

a. The running time of sorting the numbers is $O(n \lg n)$, and the running time of listing the i largest is $O(i)$. Therefore, the total running time is $O(n \lg n + i)$.

b. The running time of building a max-priority queue (using a heap) from the numbers is $O(n)$, and the running time of each call EXTRACT-MAX is $O(\lg n)$. Therefore, the total running time is $O(n + i \lg n)$.

c. The running time of finding and partitioning around the ith largest number is $O(n)$, and the running time of sorting the i largest numbers is $O(i \lg i)$. Therefore, the total running time is $O(n + i \lg i)$.