Section 7.1 – Description of quicksort

7.1-1 Using Figure 7.1 as a model, illustrate the operation of PARTITION on the array $A = \langle 13, 19, 9, 5, 12, 8, 7, 4, 21, 2, 6, 11 \rangle$.

13	19	9	5	12	8	7	4	21	2	6	11
13	19	9	5	12	8	7	4	21	2	6	11
13	19	9	5	12	8	7	4	21	2	6	11
9	19	13	5	12	8	7	4	21	2	6	11
9	5	13	19	12	8	7	4	21	2	6	11
9	5	13	19	12	8	7	4	21	2	6	11
9	5	8	19	12	13	7	4	21	2	6	11
9	5	8	7	12	13	19	4	21	2	6	11
9	5	8	7	4	13	19	12	21	2	6	11
9	5	8	7	4	13	19	12	21	2	6	11
9	5	8	7	4	2	19	12	21	13	6	11
9	5	8	7	4	2	6	12	21	13	19	11
9	5	8	7	4	2	6	11	21	13	19	12

7.1-2 What value of q does PARTITION return when all elements in the array $A = [p, \ldots, r]$ have the same value? Modify PARTITION so that $q = \lfloor (p + r)/2 \rfloor$ when all elements in the array $A[p, \ldots, r]$ have the same value.

It will return $q = r$. We can update PARTITION to split elements that are equal to the pivot on both sides as follows:

(a) Count the number of elements y such that $y = x$ and set this value to c;

(b) Subtract the final pivot index by $\lfloor c/2 \rfloor$.

The updated pseucode is stated below.

```
Partition-Improved(A, p, r)
    $x = A[r]$
    $i = p - 1$
    $c = 0$
    for $j = p$ to $r - 1$ do
        if $A[j] \leq x$ then
            if $A[j] == x$ then
                $c = c + 1$
            $i = i + 1$
            exchange $A[i]$ with $A[j]$
    exchange $A[i + 1]$ with $A[r]$
    return $(i + 1) - \lfloor c/2 \rfloor$
```

7.1-3 Give a brief argument that the running time of PARTITION on a subarray of size n if $\Theta(n)$.

The **for** loop of lines 3–6 iterates $n - 1$ times and each iteration does a constant amount of work. Thus, it is $O(n)$.

7.1-4 How would you modify QUICKSORT to sort into nonincreasing order?

We just need to update the condition

$$A[j] \leq x,$$

to

$$A[j] \geq x.$$

Section 7.2 – Performance of quicksort

7.2-1 Use the substitution method to prove the recurrence $T(n) = T(n-1) + \Theta(n)$ has the solution $T(n) = \Theta(n^2)$, as claimed at the beginning of Section 7.2.

Our guess is

$$T(n) \leq cn^2 - dn \ \forall\, n \geq n_0,$$

where c, d, and n_0 are positive constants. Substituting into the recurrence, yields

$$\begin{aligned} T(n) &\leq c(n-1)^2 - d(n-1) + en \\ &= cn^2 - 2cn + c - d(n-1) + en \quad (c = 1, d = 2e) \\ &\leq cn^2, \end{aligned}$$

where the last step holds as long as $n_0 \geq 2$.

7.2-2 What is the running time of Quicksort when all elements of array A have the same value?

As discussed in (7.1-2), when all elements are the same, q will always be equal to r, which gives the worst-case split. Thus, Quicksort as implemented in Section 7.1, will run in $\Theta(n^2)$ in this case.

7.2-3 Show that the running time of Quicksort is $\Theta(n^2)$ when the array A contains distinct elements and is sorted in decreasing order.

The pivot index q will always be 1, which gives a 0 to $n-1$ split. The recurrence will be $T(n) = T(n-1) + \Theta(n) = \Theta(n^2)$.

7.2-4 Banks often record transactions on an account in order of the times of the transactions, but many people like to receive their bank statements with checks listed in order by check number. People usually write checks in order by check number, and merchants usually cash them with reasonable dispatch. The problem of converting time-of-transaction ordering to check-number ordering is therefore the problem of sorting almost-sorted input. Argue that the procedure Insertion-Sort would tend to beat the procedure Quicksort on this problem.

Lets assume that each item is out of order by no more than k positions. Note that in the above scenario, k usually can be bounded by a constant. In this case, Insertion-Sort runs in $O(kn)$ (it will make at most k swaps for each item of the array), which is close to linear for small k. On the other hand, *most* splits given by the Partition procedure will be no better than a $k-1$ to $n-k$ split. Assuming that it always give an $k-1$ to $n-k$ split, the recurrence of Quicksort will be $T(n) = T(k) + T(n-k) + \Theta(n)$, which is close to quadratic for small k.

7.2-5 Suppose that the splits at every level of quicksort are in the proportion $1-\alpha$ to α, where $0 < \alpha \leq 1/2$ is a constant. Show that the minimum depth of a leaf in the recursion tree is approximately $-\lg n / \lg \alpha$ and the maximum depth is approximately $-\lg n / \lg(1-\alpha)$. (Don't worry about integer round-off.)

Note that

$$\alpha \leq \frac{1}{2} \leq 1 - \alpha,$$

which implies $\alpha n \leq (1-\alpha)n$. Thus, the minimum depth occurs on the path from which the problem size is always divided by $1/\alpha$. This depth is the number of divisions of n by $(1/\alpha)$ until reaching a value less than of equal to one, which is

$$\log_{1/\alpha} n = \frac{\lg n}{\lg(1/\alpha)} = \frac{\lg n}{-\lg \alpha} = -\frac{\lg n}{\lg \alpha}.$$

The maximum depth occurs on the path from which the problem size is always divided by $1/(1-\alpha)$. This depth is the number of divisions of n by $1/(1-\alpha)$ until reaching a value less than or equal to one, which is

$$\log_{1/(1-\alpha)} n = \frac{\lg n}{\lg(1/(1-\alpha))} = \frac{\lg n}{-\lg(1-\alpha)} = -\frac{\lg n}{\lg(1-\alpha)}.$$

7.2-6 ($\star$) Argue that for any constant $0 < \alpha \leq 1/2$, the probability is approximately $1 - 2\alpha$ that on a random input array, PARTITION produces a split more balanced than $1 - \alpha$ to α.

Note that α denotes the proportion of the smallest split. Since the input array is random, the possible proportions for the smallest split forms a uniform probability distribution, such that

$$\Pr\left\{\left[0, \frac{1}{2}\right]\right\} = 1.$$

Thus, the probability of getting a more balanced split is

$$\begin{aligned}\Pr\left\{\left(\alpha, \frac{1}{2}\right]\right\} &= \Pr\left\{\left[\alpha, \frac{1}{2}\right]\right\} \\ &= \frac{1/2 - \alpha}{1/2 - 0} \\ &= \frac{1/2}{1/2} - \frac{\alpha}{1/2} \\ &= 1 - 2\alpha.\end{aligned}$$

Section 7.3 – A randomized version of quicksort

7.3-1 Why do we analyze the expected running time of a randomized algorithm and not its worst-case running time?

> We can analyze the worst-case. However, due to the randomization, it is not very useful since we can not associate a specific input to a specific running time. On the other hand, we can calculate the expected running time, which takes into account all the possible inputs.

7.3-2 When RANDOMIZED-QUICKSORT runs, how many calls are made to the random-number generator RANDOM in the worst case? How about in the best case? Give your answer in terms of Θ-notation.

> Every element is selected as a pivot exactly one time and the number of calls to RANDOM will always be n. Thus, both the best-case and the worst-case takes $\Theta(n)$, where n denotes the number of calls to RANDOM.