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)
1  | x = A[r]
2  | i = p − 1
3  | c = 0
4  | for j = p to r − 1 do
5  | |  if A[j] ≤ x then
6  | |  |  if A[j] == x then
7  | |  |  |  c = c + 1
8  | |  |  i = i + 1
9  | |  |  exchange A[i] with A[j]
10 | exchange A[i + 1] with A[r]
11 | return (i + 1) − ⌊c/2⌋
```

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.

> First note that counting the number of calls to Random is the same as counting number of calls to Partition.
>
> The worst-case occurs when Partition always gives an $(n-1)$-to-0 split. Note that after the first $n-1$ pivots are selected, the remaining subarray will contain a single element. Since Partition is only called on subarrays of size greater than one, in the worst-case the number of calls to Partition is $n - 1 = \Theta(n)$.
>
> As for the best case, consider the array $A = [1, 2, 3]$. If the element 2 is the first to be selected as a pivot, the subarrays [1] and [3] will not be passed to Partition (both of them has size one) and the number of calls to Partition in this case is 1. In general, in the best-case the number of calls to Partition is $\lfloor n/2 \rfloor = \Theta(n)$.

Section 7.4 – Analysis of quicksort

7.4-1 Show that in the recurrence

$$T(n) = \max_{0 \le q \le n-1}(T(q) + T(n-q-1)) + \Theta(n),$$
$$T(n) = \Omega(n^2).$$

We guess that $T(n) \ge cn^2$ for some constant c. Substituting into the recurrence, yields

$$\begin{aligned} T(n) &\ge \max_{0 \le q \le n-1}(cq^2 + c(n-q-1)^2) + \Theta(n) \\ &= c \cdot \max_{0 \le q \le n-1}(q^2 + (n-q-1)^2) + \Theta(n). \end{aligned}$$

The expression $q^2 + (n-q-1)^2$ achieves a maximum at $q = 0$ (proof on (7.4-3)). Thus, we have

$$\max_{0 \le q \le n-1}(q^2 + (n-q-1)^2) = (n-1)^2,$$

which give us the bound

$$\begin{aligned} T(n) &\ge c(n-1)^2 + \Theta(n) \\ &= cn^2 - 2cn + c + \Theta(n) \\ &= cn^2 - c(2n-1) + \Theta(n) \\ &\ge cn^2, \end{aligned}$$

since we pick the constant c small enough so that the $\Theta(n)$ term dominates the $c(2n-1)$ term, which implies

$$T(n) = \Omega(n^2).$$

7.4-2 Show that quicksort's best-case running time is $\Omega(n \lg n)$.

Let $T(n)$ be the best-case time of QUICKSORT on an input of size n. We have the recurrence

$$T(n) = \min_{0 \le q \le n-1}(T(q) + T(n-q-1)) + \Theta(n).$$

We guess $T(n) \ge cn \lg n$ for some constant c. Substituting into the recurrence yields

$$\begin{aligned} T(n) &\ge \min_{0 \le q \le n-1}(cq \lg q + c(n-q-1)\lg(n-q-1)) + \Theta(n) \\ &= c \cdot \min_{0 \le q \le n-1}(q \lg q + (n-q-1)\lg(n-q-1)) + \Theta(n). \end{aligned}$$

For simplicify, assume that n is odd. The expression $cq \lg q + (n-q-1)\lg(n-q-1)$ achieves a minimum when

$$q = n - q - 1,$$

which implies

$$q = \frac{n-1}{2}.$$

Thus, we have

$$\begin{aligned} T(n) &\ge c\left(\frac{n-1}{2}\right)\lg\left(\frac{n-1}{2}\right) + c\left(n - \frac{n-1}{2} - 1\right)\lg\left(n - \frac{n-1}{2} - 1\right) + \Theta(n) \\ &= c(n-1)\lg\left(\frac{n-1}{2}\right) + \Theta(n) \\ &= c(n-1)\lg(n-1) - c(n-1) + \Theta(n) \\ &= cn\lg(n-1) - c\lg(n-1) - c(n-1) + \Theta(n) \\ &\ge cn\lg\left(\frac{n}{2}\right) - c\lg(n-1) - c(n-1) + \Theta(n) \qquad (n \ge 2) \\ &= cn\lg n - cn - c\lg(n-1) - c(n-1) + \Theta(n) \\ &= cn\lg n - c(2n + \lg(n-1) - 1) + \Theta(n) \\ &\ge cn\lg n, \end{aligned}$$

since we pick the constant c small enough so that the $\Theta(n)$ term dominates the $c(2n + \lg(n-1) - 1)$ term, which implies

$$T(n) = \Omega(n \lg n).$$

7.4-3 Show that the expression $q^2 + (n - q - 1)^2$ achieves a maximum over $q = 0, 1, \ldots, n - 1$ when $q = 0$ or $q = n - 1$.

Let $f(q) = q^2 + (n - q - 1)^2$. We have

$$f'(q) = 2q + 2(n - q - 1) \cdot (-1) = 4q - 2n + 2,$$

and

$$f''(q) = 4.$$

Since the second derivative is positive, $f(q)$ achieves a maximum over $0, 1, \ldots, n - 1$ at either endpoint. But we have

$$f(0) = 0^2 + (n - 1)^2 = (n - 1)^2 + (n - (n - 1) - 1)^2 = f(n - 1),$$

which implies that both endpoints are maximum.

7.4-4 Show that Randomized-Quicksort's expected running time is $\Omega(n \lg n)$.

Combining equations (7.2) and (7.3), we get

$$\begin{aligned}
\mathrm{E}[X] &= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{2}{j - i + 1} \\
&= \sum_{i=1}^{\lfloor n/2 \rfloor} \sum_{k=1}^{n-i} \frac{2}{k+1} + \sum_{i=\lfloor n/2 \rfloor + 1}^{n-1} \sum_{k=1}^{n-i} \frac{2}{k+1} \\
&\geq \sum_{i=1}^{\lfloor n/2 \rfloor} \sum_{k=1}^{n-i} \frac{2}{k+1} \\
&\geq \sum_{i=1}^{\lfloor n/2 \rfloor} \sum_{k=1}^{n/2} \frac{2}{k+1} \\
&\geq \sum_{i=1}^{\lfloor n/2 \rfloor} \sum_{k=1}^{n/2} \frac{1}{k} && \text{(since } k \geq 1\text{)} \\
&= \left\lfloor \frac{n}{2} \right\rfloor \cdot \left(\lg \left(\frac{n}{2} \right) + O(1) \right) && \text{(approx. of harmonic number)} \\
&= \Omega(n \lg n).
\end{aligned}$$

7.4-5 We can improve the running time of quicksort in practice by taking advantage of the fast running time of insertion sort when its input is "nearly" sorted. Upon calling quicksort on a subarray with fewer than k elements, let it simply return without sorting the subarray. After the top-level call to quicksort returns, run insertion sort on the entire array to finish the sorting process. Argue that this sorting algorithm runs in $O(nk + n \lg(n/k))$ expected time. How should we pick k, both in theory and in practice?

Lets first analyze the modified Quicksort. As in the standard Quicksort, it is easy to see that the worst-case of this modified version is still $O(n^2)$. As for the expected time, we can use a similar argument to the one used on Section 7.2, in which we saw that any split of constant proportionality on Quicksort yields a recursion tree of depth $\Theta(\lg n)$. Assume that Partition on this modified Quicksort always give a 99-to-1 split. The height h of the recursion tree would be

$$\frac{n}{(100/99)^h} = k \to h = \log_{100/99} \frac{n}{k} \to h = \Theta\left(\lg \frac{n}{k} \right).$$

Since each recursion level has cost at most cn, the expected total cost of this modified Quicksort is $O(n \lg \frac{n}{k})$. As for the cost of the Insertion-Sort, note after running the modified Quicksort, every element will be out of order by at most k positions. Thus, each iteration of the outer loop of Insertion-Sort will make at most k swaps, which gives a running time of $O(nk)$. Finally, the cost of the whole algorithm is $O(nk) + O(n \lg(n/k)) = O(nk + n \lg(n/k))$.

7.4-6 ($\star$) Consider modifying the PARTITION procedure by randomly picking three elements from array A and partitioning about their median (the middle value of the three elements). Approximate the probability of getting at worst an α-to-$(1-\alpha)$ split, as a function of α in the range $0 < \alpha < 1$.

First assume $0 < \alpha \leq 1/2$. There four ways to get a split worse than α-to-$(1-\alpha)$:

(a) The index of exactly two elements are smaller than αn

(b) The index of exactly two elements are greater than $n - \alpha n$.

(c) The index of all three elements are smaller than αn.

(d) The index of all three elements are greater than $n - \alpha n$.

Since we want an approximation, assume that we can repeat the same element. The probability of cases (a) and (b) is

$$\Pr\{(a)\} = \Pr\{(b)\} = 3 \cdot \left(\frac{\alpha n}{n} \cdot \frac{\alpha n}{n} \cdot \frac{(1-\alpha)n}{n} \right) = 3\alpha^2 - 3\alpha^3,$$

in which the multiplication on the left is needed since there are $\binom{3}{1} = 3$ ways to pick one of three elements outside the desired range. The probability of cases (c) and (d) is

$$\Pr\{(c)\} = \Pr\{(d)\} = \frac{\alpha n}{n} \cdot \frac{\alpha n}{n} \cdot \frac{\alpha n}{n} = \alpha^3.$$

Thus, the probability of getting a split worse than α-to-$(1-\alpha)$ is

$$\begin{aligned} 1 - \Pr\{(a) + (b) + (c) + (d)\} &= 1 - (\Pr\{(a)\} + \Pr\{(b)\} + \Pr\{(c)\} + \Pr\{(d)\}) \\ &= 1 - \left((3\alpha^2 - 3\alpha^3) + (3\alpha^2 - 3\alpha^3) + \alpha^3 + \alpha^3\right) \\ &= 1 - (6\alpha^2 - 4\alpha^3) \\ &= 1 - 6\alpha^2 + 4\alpha^3. \end{aligned}$$

The proof is similar for $1/2 \leq \alpha < 1$ and the result is the same.