Chapter 7

Michelle Bodnar, Andrew Lohr

September 17, 2017

Exercise 7.1-1

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

Exercise 7.1-2

If all elements in the array have the same value, PARTITION returns r. To make PARTITION return $q = \lfloor (p+r)/2 \rfloor$ when all elements have the same value, modify line 4 of the algorithm to say this: if $A[j] \leq x$ and $j(mod 2) = (p+1)(mod 2)$. This causes the algorithm to treat half of the instances of the same value to count as less than, and the other half to count as greater than.

Exercise 7.1-3

The for loop makes exactly $r - p$ iterations, each of which takes at most constant time. The part outside the for loop takes at most constant time. Since $r - p$ is the size of the subarray, PARTITION takes at most time proportional to the size of the subarray it is called on.

Exercise 7.1-4

To modify QUICKSORT to run in non-increasing order we need only modify line 4 of PARTITION, changing $\leq$ to $\geq$.

Exercise 7.2-1

By definition of Θ, we know that there exists c_1, c_2 so that the $\Theta(n)$ term is between $c_1 n$ and $c_2 n$. We make that inductive hypothesis be that $c_1 m^2 \leq T(m) \leq c_2 m^2$ for all $m < n$, then, for large enough n,

$$c_1 n^2 \leq c_1(n-1)^2 + c_1 n \leq T(n-1) + \Theta(n)$$
$$= T(n) = T(n-1) + \Theta(n) \leq c_2(n-1)^2 + c_2 n \leq c_2 n^2$$

Exercise 7.2-2

The running time of QUICKSORT on an array in which evey element has the same value is n^2. This is because the partition will always occur at the last position of the array (Exercise 7.1-2) so the algorithm exhibits worst-case behavior.

Exercise 7.2-3

If the array is already sorted in decreasing order, then, the pivot element is less than all the other elements. The partition step takes $\Theta(n)$ time, and then leaves you with a subproblem of size $n-1$ and a subproblem of size 0. This gives us the recurrence considered in 7.2-1. Which we showed has a solution that is $\Theta(n^2)$.

Exercise 7.2-4

Let's say that by "almost sorted" we mean that $A[i]$ is at most c positions from its correct place in the sorted array, for some constant c. For INSERTION-SORT, we run the inner-while loop at most c times before we find where to insert $A[j]$ for any particular iteration of the outer for loop. Thus the running time time is $O(cn) = O(n)$, since c is fixed in advance. Now suppose we run QUICK-SORT. The split of PARTITION will be *at best* $n-c$ to c, which leads to $O(n^2)$ running time.

Exercise 7.2-5

The minimum depth corresponds to repeatedly taking the smaller subproblem, that is, the branch whose size is proportional to α. Then, this will fall to 1 in k steps where $1 \approx (\alpha)^k n$. So, $k \approx \log_\alpha(1/n) = -\frac{\lg(n)}{\lg(\alpha)}$. The longest depth corresponds to always taking the larger subproblem. we then have and identical expression, replacing α with $1-\alpha$.

Exercise 7.2-6

Without loss of generality, assume that the entries of the input array are distinct. Since only the relative sizes of the entries matter, we may assume that A contains a random permutation of the numbers 1 through n. Now fix $0 < \alpha \leq 1/2$. Let k denote the number of entries of A which are less than $A[n]$. PARTITION produces a split more balanced than $1-\alpha$ to α if and only if $\alpha n \leq k \leq (1-\alpha)n$. This happens with probability $\frac{(1-\alpha)n-\alpha n+1}{n} = 1-2\alpha+1/n \approx 1-2\alpha$.

Exercise 7.3-1

We analyze the expected run time because it represents the more typical time cost. Also, we are doing the expected run time over the possible randomness used during computation because it can't be produced adversarially, unlike when doing expected run time over all possible inputs to the algorithm.

Exercise 7.3-2

In either case, $\Theta(n)$ calls are made to RANDOM. PARTITION will run faster in the best case because the inputs will generally be smaller, but RANDOM is called every single time RANDOMIZED-PARTITION is called, which happens $\Theta(n)$ times.

Exercise 7.4-1

By definition of Θ, we know that there exists c_1, c_2 so that the $\Theta(n)$ term is between $c_1 n$ and $c_2 n$. We make that inductive hypothesis be that $c_1 m^2 \leq T(m) \leq c_2 m^2$ for all $m < n$, then, for large enough n,

$$\begin{aligned}
c_1 n^2 &\leq c_1 \max_{q\in[n]} n^2 - 2n(q+2) + (q+1)^2 + (q+1)^2 + n \\
&= \max_{q\in[n]} c_1(n-q-2)^2 + c_1(q+1)^2 + c_1 n \\
&\leq \max_{q\in[n]} T(n-q-2) + T(q+1) + \Theta(n) \\
&= T(n)
\end{aligned}$$

Similarly for the other direction

Exercise 7.4-2

We'll use the substitution method to show that the best-case running time is $\Omega(n \lg n)$. Let $T(n)$ be the best-case time for the procedure QUICKSORT on an input of size n. We have the recurrence

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

Suppose that $T(n) \geq c(n \lg n + 2n)$ for some constant c. Substituting this guess into the recurrence gives

$$\begin{aligned}
T(n) &\geq \min_{1\leq q\leq n-1}(cq\lg q + 2cq + c(n-q-1)\lg(n-q-1) + 2c(n-q-1)) + \Theta(n)\\
&= \frac{cn}{2}\lg(n/2) + cn + c(n/2-1)\lg(n/2-1) + cn - 2c + \Theta(n)\\
&\geq (cn/2)\lg n - cn/2 + c(n/2-1)(\lg n - 2) + 2cn - 2c\Theta(n)\\
&= (cn/2)\lg n - cn/2 + (cn/2)\lg n - cn - \lg n + 2 + 2cn - 2c\Theta(n)\\
&= cn\lg n + cn/2 - \lg n + 2 - 2c + \Theta(n)
\end{aligned}$$

Taking a derivative with respect to q shows that the minimum is obtained when $q = n/2$. Taking c large enough to dominate the $-\lg n + 2 - 2c + \Theta(n)$ term makes this greater than $cn\lg n$, proving the bound.

Exercise 7.4-3

We will treat the given expression to be continuous in q, and then, any extremal values must be either adjacent to a critical point, or one of the endpoints. The second derivative with respect to q is 4, So, we have that any critical points we find will be minima. The expression has a derivative with respect to q of $2q - 2(n-q-2) = -2n + 4q + 4$ which is zero when we have $2q + 2 = n$. So, there will be a minima at $q = \frac{n-2}{2}$. So, the maximal values must only be the endpoints. We can see that the endpoints are equally large because at $q = 0$, it is $(n-1)^2$, and at $q = n-1$, it is $(n-1)^2 + (n-n+1-1)^2 = (n-1)^2$.

Exercise 7.4-4

We'll use the lower bound (A.13) for the expected running time given in the section:

$$\begin{aligned}
E[X] &= \sum_{i=1}^{n-1}\sum_{j=i+1}^{n}\frac{2}{j-i+1}\\
&= \sum_{i=1}^{n-1}\sum_{k=1}^{n-i}\frac{2}{k}\\
&\geq \sum_{i=1}^{n-1} 2\ln(n-i+1)\\
&= 2\ln\left(\prod_{i=1}^{n-1} n-i+1\right)\\
&= 2\ln(n!)\\
&= \frac{2}{\lg e}\lg(n!) \qquad\qquad \geq cn\lg n
\end{aligned}$$

for some constant c since $\lg(n!) = \Theta(n\lg n)$ by Exercise 3.2-3. Therefore RANDOMIZED-QUICKSORT's expected running time is $\Omega(n\lg n)$.

Exercise 7.4-5

If we are only doing quick-sort until the problem size becomes $\leq k$, then, we will have to take $\lg(n/k)$ 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 $k \sim \frac{1}{n^2}$. The constant of proportionality will depend on the relative size of the constants in the nk term and in the $n\lg(n/k)$ 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.

Exercise 7.4-6

For simplicity of analysis, we will assume that the elements of the array A are the numbers 1 to n. If we let k denote the median value, the probability of getting at worst an α to $1-\alpha$ split is the probability that $\alpha n \leq k \leq (1-\alpha)n$. The number of "bad" triples is equal to the number of triples such that at least two of the numbers come from $[1, \alpha n]$ or at least two of the numbers come from $[(1-\alpha)n, n]$. Since both intervals have the same size, the probability of a bad triple is $2(\alpha^3 + 3\alpha^2(1-\alpha))$. Thus the probability of selecting a "good" triple, and thus getting at worst an α to $1-\alpha$ split, is $1-2(\alpha^3+3\alpha^2(1-\alpha)) = 1+4\alpha^3-6\alpha^2$.

Problem 7-1

a. We will be calling with the parameters $p = 1$, $r = |A| = 12$. So, throughout, $x = 13$.

												i	j
13	19	9	5	12	8	7	4	11	2	6	21	0	13
6	19	9	5	12	8	7	4	11	2	13	21	1	11
6	13	9	5	12	8	7	4	11	2	19	21	2	10
6	13	9	5	12	8	7	4	11	2	19	21	10	2

And we do indeed see that partition has moved the two elements that are bigger than the pivot, 19 and 21, to the two final positions in the array.

b. We know that at the beginning of the loop, we have that $i < j$, because it is true initially so long as $|A| \geq 2$. And if it were to be untrue at some iteration, then we would of left the loop in the prior iteration. To show that we won't access outside of the array, we need to show that at the beginning of every run of the loop, there is a $k > i$ so that $A[k] \geq x$, and a $k' < j$ so that $A[j'] \leq x$. This is clearly true because initially i and j are outside the bounds of the array, and so the element x must be between the two. Since

$i < j$, we can pick $k = j$, and $k' = i$. The elements k satisfies the desired relation to x, because the element at position j was the element at position i in the prior iteration of the loop, prior to doing the exchange on line 12. Similarly, for k'.

c. If there is more than one run of the main loop, we have that $j < r$ because it decreases by at least one with every iteration.

   Note that at line 11 in the first run of the loop, we have that $i = 1$ because $A[p] = x \geq x$. So, if we were to terminate after a single iteration of the main loop, we must also have that $j = 1 < p$.

d. We will show the loop invariant that all the elements in $A[p..i]$ are less than or equal to x which is less than or equal to all the elements of $A[j..r]$. It is trivially true prior to the first iteration because both of these sets of elements are empty. Suppose we just finished an iteration of the loop during which j went from j_1 to j_2, and i went from i_1 to i_2. All the elements in $A[i_1+1..i_2-1]$ were $< x$, because they didn't cause the loop on lines $8-10$ to terminate. Similarly, we have that all the elements in $A[j_2+1..j_1-1]$ were $> x$. We have also that $A[i_2] \leq x \leq A[j_2]$ after the exchange on line 12. Lastly, by induction, we had that all the elements in $A[p..i_1]$ are less than or equal to x, and all the elements in $A[j_1..r]$ are greater than or equal to x. Then, putting it all together, since $A[p..i_2] = A[p..i_1] \cup A[i_1+1..i_2-1] \cup \{A[i_2]\}$ and $A[j_2..r] = \cup\{A[j_2]\} \cup A[j_2+1..j_1-1] \cup A[j_1..r]$, we have the desired inequality. Since at termination, we have that $i \geq j$, we know that $A[p..j] \subseteq A[p..i]$, and so, every element of $A[p..j]$ is less than or equal to x, which is less than or equal to every element of $A[j+1..r] \subseteq A[j..r]$.

e. After running Hoare-partition, we don't have the guarantee that the pivot value will be in the position j, so, we will scan through the list to find the pivot value, place it between the two subarrays, and recurse

Problem 7-2

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. See the PARTITION' algorithm for modifications.

c. See the RANDOMIZED-PARTITION' algorithm for modifications.

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

Quicksort(A,p,r)

```
if $p < r$ then
    $x = A[p]$
    $q = HoarePartition(A, p, r)$
    $i = 0$
    while $A[i] \neq x$ do
        $i = i + 1$
    end while
    if $i \leq q$ then
        exchange $A[i]$ and $A[q]$
    else
        exchange $A[i]$ and $A[q + 1]$
        $q = q + 1$
    end if
    Quicksort(A,p,q-1)
    Quicksort(A,q+1,r)
end if
```

Algorithm 1 PARTITION'(A,p,r)

```
$x = A[r]$
exchange $A[r]$ with $A[p]$
$i = p - 1$
$k = p$
for $j = p + 1$ to $r - 1$ do
    if $A[j] < x$ then
        $i = i + 1$
        $k = i + 2$
        exchange $A[i]$ with $A[j]$
        exchange $A[k]$ with $A[j]$
    end if
    if $A[j] = x$ then
        $k = k + 1$
        exchange $A[k]$ with $A[j]$
    end if
end for
exchange $A[i + 1]$ with $A[r]$
return $i + 1$ and $k + 1$
```

Algorithm 2 RANDOMIZED-PARTITION'

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

Algorithm 3 QUICKSORT'(A,p,r)

```
1: if $p < r$ then
2:     $(q, t) = RANDOMIZED - PARTITION'$
3:     QUICKSORT'(A,p,q-1)
4:     QUICKSORT'(A,t+1,r)
5: end if
```

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 $z_i \le x \le z_j$, 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.

Problem 7-3

a. Since the pivot is selected as a random element in the array, which has size n, the probabilities of any particular element being selected are all equal, and add to one, so, are all $\frac{1}{n}$. As such, $E[X_i] = \Pr[\text{i smallest is picked }] = \frac{1}{n}$.

b. We can apply linearity of expectation over all of the events X_i. Suppose we have a particular X_i be true, then, we will have one of the sub arrays be length $i - 1$, and the other be $n - i$, and will of course still need linear time to run the partition procedure. This corresponds exactly to the summand in equation (7.5).

c.

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

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

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

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

$$= \Theta(n) + \frac{2}{n}\sum_{q=0}^{n-1} T(q) = \Theta(n) + \frac{2}{n}\sum_{q=2}^{n-1} T(q)$$

d. We will prove this inequality in a different way than suggested by the hint. If we let $f(k) = k\lg(k)$ treated as a continuous function, then $f'(k) = \lg(k)+1$. Note now that the summation written out is the left hand approximation of the integral of $f(k)$ from 2 to n with step size 1. By integration by parts, the anti-derivative of $k\lg k$ is

$$\frac{1}{\ln(2)}\left(\frac{k^2}{2}\ln(k) - \frac{k^2}{4}\right)$$

So, plugging in the bounds and subtracting, we get $\frac{n^2\lg(n)}{2} - \frac{n^2}{4\ln(2)} - 1$. Since f has a positive derivative over the entire interval that the integral is being evaluated over, the left hand rule provides a underapproximation of the integral, so, we have that

$$\sum_{k=2}^{n-1} k\lg(k) \le \frac{n^2\lg(n)}{2} - \frac{n^2}{4\ln(2)} - 1 \le \frac{n^2\lg(n)}{2} - \frac{n^2}{8}$$

where the last inequality uses the fact that $\ln(2) > 1/2$.

e. Assume by induction that $T(q) \le q\lg(q) + \Theta(n)$. Combining (7.6) and (7.7), we have

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

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

$$\le \frac{2}{n}(\frac{1}{2}n^2\lg(n) - \frac{1}{8}n^2) + \Theta(n) = n\lg(n) - \frac{1}{4}n + \Theta(n) = n\lg(n) + \Theta(n)$$

Problem 7-4

a. We'll proceed by induction. For the base case, if A contains 1 element then $p = r$ so the algorithm terminates immediately, leave a single sorted element. Now suppose that for $1 \le k \le n-1$, TAIL-RECURSIVE-QUICKSORT correctly sorts an array A containing k elements. Let A have size n. We set q equal to the pivot and by the induction hypothesis, TAIL-RECURSIVE-QUICKSORT correctly sorts the left subarray which is of strictly smaller size. Next, p is updated to $q+1$ and the exact same sequence of steps follows as if we had originally called TAIL-RECURSIVE-QUICKSORT(A,q+1,n). Again, this array is of strictly smaller size, so by the induction hypothesis it correctly sorts $A[q+1...n]$ as desired.

b. The stack depth will be $\Theta(n)$ if the input array is already sorted. The right subarray will always have size 0 so there will be $n-1$ recursive calls before the while-condition $p < r$ is violated.

c. We modify the algorithm to make the recursive call on the smaller subarray to avoid building pushing too much on the stack:

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

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

Problem 7-5

a. p_i is the probability that a randomly selected subset of size three has the $A'[i]$ as it's middle element. There are 6 possible orderings of the three elements selected. So, suppose that S' is the set of three elements selected. We will compute the probability that the second element of S' is $A'[i]$ among all possible 3-sets we can pick, since there are exactly six ordered 3-sets corresponding to each 3-set, these probabilities will be equal. We will compute the probability that $S'[2] = A'[i]$. For any such S', we would need to select the first element from $[i-1]$ and the third from $\{i+1, \dots, n\}$. So, there are $(i-1)(n-i)$ such 3-sets. The total number of 3-sets is $\binom{n}{3} = \frac{n(n-1)(n-2)}{6}$. So,

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

b. If we let $i = \lfloor \frac{n+1}{2} \rfloor$, the previous result gets us an increase of

$$\frac{6(\lfloor \frac{n-1}{2} \rfloor)(n - \lfloor \frac{n+1}{2} \rfloor)}{n(n-1)(n-2)} - \frac{1}{n}$$

in the limit n going to infinity, we get

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

c. To save the messiness, suppose n is a multiple of 3. We will approximate the sum as an integral, so,

$$\sum_{i=n/3}^{2n/3} p_i \approx \int_{n/3}^{2n/3} \frac{6(-x^2+nx+x-n)}{n(n-1)(n-2)}dx = \frac{6(-7n^3/81+3n^3/18+3n^2/18-n^2/3)}{n(n-1)(n-2)}$$

Which, in the limit n goes to infinity, is $\frac{13}{27}$ which is a constant that is larger than 1/3 as it was in the original randomized quicksort implementation.

d. Since the new algorithm always has a "bad" choice that is within a constant factor of the original quicksort, it will still have a reasonable probability that the randomness leads us into a bad situation, so, it will still be $n \lg n$.

Problem 7-6

a. Our algorithm will be essentially the same as the modified randomized quicksort written in problem 2. We sort the a_i's, but we replace the comparison operators to check for overlapping intervals. The only modifications needed are made in PARTITION, so we will just rewrite that here. For a given element x in position i, we'll use $x.a$ and $x.b$ to denote a_i and b_i respectively.

Algorithm 5 FUZZY-PARTITION(A,p,r)

```
 1: $x = A[r]$
 2: exchange $A[r]$ with $A[p]$
 3: $i = p - 1$
 4: $k = p$
 5: for $j = p + 1$ to $r - 1$ do
 6:     if $b_j < x.a$ then
 7:         $i = i + 1$
 8:         $k = i + 2$
 9:         exchange $A[i]$ with $A[j]$
10:         exchange $A[k]$ with $A[j]$
11:     end if
12:     if $b_j \geq x.a$ or $a_j \leq x.b$ then
13:         $x.a = \max(a_j, x.a)$ and $x.b = \min(b_j, x.b)$
14:         $k = k + 1$
15:         exchange $A[k]$ with $A[j]$
16:     end if
17: end for
18: exchange $A[i + 1]$ with $A[r]$
19: return $i + 1$ and $k + 1$
```

When intervals overlap we treat them as equal elements, thus cutting down on the time required to sort.

---

b. For distinct intervals the algorithm runs exactly as regular quicksort does, so its expected runtime will be $\Theta(n \lg n)$ in general. If all of the intervals overlap then the condition on line 12 will be satisfied for every iteration of the for loop. Thus the algorithm returns p and r, so only empty arrays remain to be sorted. FUZZY-PARTITION will only be called a single time, and since its runtime remains $\Theta(n)$, the total expected runtime is $\Theta(n)$.