Exercise XII, Algorithms 2024-2025

These exercises are for your own benefit. Feel free to collaborate and share your answers with other students. There are many problems on this set, solve as many as you can and ask for help if you get stuck for too long. Problems marked * are more difficult but also more fun :).

Sorting: Quick Sort and Counting Sort

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

Solution: When all elements in the array $A[p \dots r]$ have the same value, the PARTITION algorithm from class returns r, which means that the split is very unbalanced and can thus greatly harm the performance of QUICKSORT.

One way to fix this is to modify PARTITION as follows: run the partitioning first, as described in the lectures; then, check if all the elements in $A[p \dots r]$ are the same; if this is the case, return $q = \lfloor \frac{p+r}{2} \rfloor$, otherwise return $i + 1$ as in the lectures.

A more refined way to achieve this goal is with algorithm PARTITION', which we now describe. We run the partitioning first as described in the lectures. Then, we count the number of occurrences of elements with same value as the pivot. Afterward, we place those elements to be next to the pivot. Observe that after this operation all the elements same as pivot are consecutive, i.e. they form a subarray $A[p' \dots r']$. Finally, we update the pivot and set it as $q = \lfloor \frac{p'+r'}{2} \rfloor$. Here is the algorithm in details.

```
PARTITION' (A, p, r)
1   x ← A[r]
2   i ← p − 1
4   for j = p to r − 1
5       if A[j] ≤ x
6           i ← i + 1
7           exchange A[i] with A[j]
8   exchange A[i + 1] with A[r]
9   k ← i + 1
10  i ← p − 1
11  for j = p to k − 1
12      if A[j] < x
13          i ← i + 1
14          exchange A[i] with A[j]
15  return ⌊(k + i + 1)/2⌋
```

2 (Exercise 7.1-4) How would you modify QUICKSORT to sort in nonincreasing order?

Solution: In the partition procedure, just replace the line

if $A[j] \leq x$

with

if $A[j] \geq x$

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

Solution: Because the worst case is unlikely to happen, since it no longer depends on the input, but instead on the random choices of the algorithm. Therefore, the expected running time is the same for all inputs. Consider quicksort for illustration, where for input size n, the probability of worst case selection of pivots is $\approx 1/n!$ (the worst case is to always select the smallest or biggest pivot).

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

Solution: If element q, where $1 \leq q \leq n$, is picked as the pivot, then the subarrays will have sizes $q-1$ and $n-q$, and with $\Theta(n)$ time to do the partitioning, the recursion is:

$$T(n) = T(q-1) + T(n-q) + \Theta(n)$$

The best case happens for the pivot that minimizes the running time.

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

We will solve the recurrence using the substitution method. As a reminder, the substitution method allows finding the upper/lower bound of a recurrence with the use of induction. First we guess on the solution (usually with the help of the iteration method). Then, we need to show that the base case holds, and finally to show that the inductive case holds with the use of the inductive hypothesis.

We guess that $T(n) = \Omega(n \lg n)$. $T(1) = T(0) = 0$, so for the base case, we have $T(2) \geq 2c$ for which we can always pick the constant c to be small enough.

By inductive hypothesis, we assume that $T(i) \geq ci \lg i$ for all i such that $2 \leq i \leq n-1$. Then we have:

$$T(n) \geq min_{1 \leq q \leq n} \left(c(q-1)\lg(q-1) + c(n-q)\lg(n-q) + \Theta(n)\right)$$

To find the minimum of the expression $(q-1)\lg(q-1) + (n-q)\lg(n-q)$, we need to find its first derivative with respect to q and equal it to zero.

$$\begin{aligned}
&\frac{d}{dq}\left((q-1)\lg(q-1) + (n-q)\lg(n-q)\right) \\
&= \lg(q-1) + (q-1)\frac{1}{(q-1)\ln 2} - \lg(n-q) - (n-q)\frac{1}{(n-q)\ln 2} \\
&= \lg(q-1) - \lg(n-q)
\end{aligned}$$

This is equal to zero for $q-1 = n-q$, or $q = (n+1)/2$. At this point we could also find the second derivative and check whether it is greater than zero to ensure that what we found is indeed a minimum (and not a maximum or an inflection point).

Substituting q in the recurrence, we get:

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

By choosing the constant c small enough that $\Theta(n)$ term dominates $2cn + c\lg(n-1)$, we have $T(n) = \Omega(n\lg n)$.

5 (*, Problem 7-3 **Alternative quicksort analysis**)

An alternative analysis of the running time of randomized quicksort focuses on the expected running time of each individual recursive call to RANDOMIZED-QUICKSORT, rather than on the number of comparisons performed.

5a Argue that, given an array of size n, the probability that any particular element is chosen as the pivot is $1/n$. Use this to define indicator random variables $X_i = I\{i\text{th smallest element is chosen as the pivot}\}$. What is $\mathbb{E}[X_i]$?

Solution: Since we are choosing one element out of n uniformly at random, the probability of choosing any particular element is $1/n$. Then $\mathbb{E}[X_i] = \Pr[\{i\text{th smallest element is chosen as the pivot}\}] = 1/n$.

5b Let $T(n)$ be a random variable denoting the running time of randomized quicksort on an array of size n. Argue that

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

Solution: If element q, where $1 \leq q \leq n$, is picked as the pivot, then the subarrays will have sizes $q-1$ and $n-q$, and with $\Theta(n)$ time to do the partitioning, the recursion is:

$$T(n) = T(q-1) + T(n-q) + \Theta(n)$$

We get the expected running time by summing over running times for all possible pivots proportional to the probability of each pivot.

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

5c Show that we can rewrite equation (1) as

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

Solution: Note that all indicator random variables $X_1, \ldots, X_n$ and $T(1), \ldots, T(n)$ are mutually independent. This lets us change the expectation of the product into a product of expectations, and the expectation of the sum into sum of expectations. We have:

$$\begin{aligned}
\mathbb{E}[T(n)] &= \mathbb{E}\left[\sum_{q=1}^{n} X_q(T(q-1) + T(n-q) + \Theta(n))\right] \\
&= \sum_{q=1}^{n} \mathbb{E}[X_q] \cdot \mathbb{E}[T(q-1) + T(n-q) + \Theta(n)] \\
&= \frac{1}{n}\left(\sum_{q=1}^{n} \mathbb{E}[T(q-1)] + \sum_{q=1}^{n} \mathbb{E}[T(n-q)] + \sum_{q=1}^{n} \Theta(n)\right) \\
&= \frac{1}{n}\left(2\sum_{q=1}^{n} \mathbb{E}[T(q-1)] + n\Theta(n)\right) \\
&= \frac{2}{n}\sum_{q=1}^{n} \mathbb{E}[T(q-1)] + \Theta(n)
\end{aligned}$$

Notice in the above that

$$\sum_{q=1}^{n} \mathbb{E}[T(q-1)] = \sum_{q=1}^{n} \mathbb{E}[T(n-q)] = \mathbb{E}[T(0)] + \cdots + \mathbb{E}[T(n-1)]$$

Also notice that $T(0) = T(1) = 0$, so we can rewrite the above as:

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

5d Show that

$$\sum_{k=2}^{n-1} k \lg k \leq \frac{1}{2}n^2 \lg n - \frac{1}{8}n^2. \tag{3}$$

(Hint: Split the summation into two parts, one for $k = 2, 3, \ldots, \lceil n/2 \rceil - 1$ and one for $k = \lceil n/2 \rceil, \ldots, n-1$.)

Solution:

$$\begin{aligned}
\sum_{k=2}^{n-1} k \lg k &= \sum_{k=2}^{\lceil n/2 \rceil - 1} k \lg k + \sum_{k=\lceil n/2 \rceil}^{n-1} k \lg k \\
&\leq \sum_{k=2}^{\lceil n/2 \rceil - 1} k \lg \frac{n}{2} + \sum_{k=\lceil n/2 \rceil}^{n-1} k \lg n \\
&= \lg \frac{n}{2} \left(\frac{1}{2} \left(\lceil \frac{n}{2} \rceil - 2 \right) \left(\lceil \frac{n}{2} \rceil + 1 \right) \right) + \lg n \left(\frac{1}{2} \left(n - \lceil \frac{n}{2} \rceil \right) \left(n + \lceil \frac{n}{2} \rceil - 1 \right) \right) \\
&\leq \frac{1}{2} (\lg n - 1) \left(\lceil \frac{n}{2} \rceil \right)^2 + \frac{1}{2} \lg n \left(n^2 - \left(\lceil \frac{n}{2} \rceil \right)^2 \right) \\
&= \frac{1}{2} n^2 \lg n - \frac{1}{2} \left(\lceil \frac{n}{2} \rceil \right)^2 \\
&\leq \frac{1}{2} n^2 \lg n - \frac{1}{8} n^2
\end{aligned}$$

5e Using the bound from equation (3), show that the recurrence in equation (2) has the solution $\mathbb{E}[T(n)] = \Theta(n \lg n)$.

(Hint: Show, by substitution, that $\mathbb{E}[T(n)] \leq an \lg n$ for sufficiently large n and for some positive constant a.)

Solution: We will again use the substitution method to solve the recurrence. We are going to determine only the upper bound, because the lower bound was shown in the previous problem. The guess of the solution is $O(n \lg n)$. For the base case of $n = 2$, we can always pick the constant a large enough such that $\mathbb{E}[T(2)] \leq 2a$.

By inductive hypothesis, we assume that $\mathbb{E}[T(i)] \leq ai \lg i$ for all i such that $2 \leq i \leq n - 1$. Then we have:

$$\begin{aligned}
\mathbb{E}[T(n)] &= \frac{2}{n} \sum_{q=2}^{n-1} \mathbb{E}[T(q)] + \Theta(n) \\
&\leq \frac{2}{n} \sum_{q=2}^{n-1} aq \lg q + \Theta(n) \\
&\leq \frac{2a}{n} \left(\frac{1}{2} n^2 \lg n - \frac{1}{8} n^2 \right) + \Theta(n) \\
&= an \lg n - \frac{an}{4} + \Theta(n) \\
&\leq an \lg n
\end{aligned}$$

for large enough $a > 0$. Along with the result from the previous problem (that even in the best case quicksort runs in time $\Omega(n \lg n)$, we have $\mathbb{E}[T(n)] = \Theta(n \lg n)$.

6 (Half *, Exercise 8.2-4) Describe an algorithm that, given n integers in the range 0 to k, preprocesses its input and then answers any query about how many of the n integers fall into a range $[a..b]$ in $O(1)$ time. Your algorithm should use $\Theta(n + k)$ preprocessing time.

Solution: Use counting sort to obtain an array $C[]$ of $k+1$ elements where $C[i]$ contains the number of elements with value at most i. This takes $O(n+k)$ time. Assuming the input is in array X:

for $i = 0..k$
  $C[i] = 0$
for $i = 0..n-1$
  $C[X[i]] = C[X[i]] + 1$
for $i = 1..k$
  $C[i] = C[i] + C[i-1]$

The number of integers in the range $[a..b]$ is $C[b] - C[a-1]$, where we interpret $C[-1]$ as 0.

7 *(Exam problem 2013, 20 pts)* **Probabilistic analysis**. We shall analyze a randomized procedure RANDOMIZED-SELECT for the select problem: given an array A consisting of n unique integers and an integer $k \leq n$, output the kth smallest integer of A.

For example, if the input is $A =$ | 89 | 14 | 16 | 28 | 51 | 25 | and $k = 3$ then the correct output is 25.

To simplify the description of RANDOMIZED-SELECT, we let $|A|$ denote the length of the array. The pseudocode is as follows:

RANDOMIZED-SELECT(A, k)

1. Pick *pivot* uniformly at random from the numbers in A.
2. Compare each number in A with *pivot* to obtain arrays S and L:
   S contains all numbers of A strictly smaller than pivot.
   L contains all numbers of A strictly larger than pivot.
3. **if** $|S| = k-1$
4. **return** *pivot*
5. **else if** $|S| \geq k$
5. **return** RANDOMIZED-SELECT(S, k)
6. **else** (we have $|S| < k-1$)
7. **return** RANDOMIZED-SELECT$(L, k-(|S|+1))$

The idea of the algorithm is very similar to RANDOMIZED-QUICKSORT that we saw in class. As in that algorithm, we first select a number *pivot* uniformly at random from the numbers in A. We then partition A into two arrays S and L that contain all numbers of A that are strictly smaller and strictly larger than *pivot*, respectively[1]. The time it takes to execute these steps (Steps 1 and 2) of the algorithm is $\Theta(|A|)$ which is also proportional to the number of "$\leq$"-comparisons we make to find S and L. After that, the algorithm recurses on the array where the kth smallest element can be found or simply returns the kth smallest element if the pivot equals it.

We shall now analyze the running time of RANDOMIZED-SELECT.

7a *(4 pts)* Suppose that we are extremely lucky: every time we select a pivot at random, the pivot that *minimizes* the running time is selected. What is the asymptotic running time of RANDOMIZED-SELECT in this lucky case? Motivate your answer.

[1] As we saw in class, we can do this without using the extra space S and L. We have presented the algorithm in this way to make the description clearer.

Solution: If we have maximum luck the pivot equals the kth smallest element. Therefore the running time of algorithm will be $\Theta(n)$ if $|A| = n$. This is the time it takes to execute Steps 1-4 of the algorithm.

7b *(6 pts)* Suppose that we are extremely unlucky: every time we select a pivot at random, the pivot that *maximizes* the running time is selected. What is the asymptotic running time of RANDOMIZED-SELECT in this unlucky case? Motivate your answer.

Solution:

Let $|A| = n$. Suppose $k \leq n/2$ (the argument is symmetric if $k > n/2$). Now if we have bad luck, Step 1 always selects the pivot that equals the largest number. It will then continue to look for the kth smallest element in an array S of size $n-1$. Thus we are stuck with the following recursion $T(n) = T(n-1) + \Theta(n)$ and $T(k) = 1$. When $k \leq n/2$ this is $\Theta(n^2)$. Note that the maximum bad luck can only be worse so that has also running time $\Omega(n^2)$. That it is $O(n^2)$ follows from that one can see that the running time is always upper bounded by $T(n) = T(n-1) + O(n)$.

7c We shall now analyze the *expected* running time of RANDOMIZED-SELECT on an array of length n. Similarly to RANDOMIZED-QUICKSORT, the running time is proportional to the total number of comparisons. As we saw in class, if we let X be the random variable that equals the total number of comparisons then

$$\mathbb{E}[X] = \sum_{i=1}^{n} \sum_{j=i+1}^{n} \mathbb{E}[X_{ij}],$$

where X_{ij} is the random indicator variable that takes value 1 if the ith smallest number was compared to the jth smallest number of the array, and 0 otherwise.

Give a tight asymptotic analysis of the *expected* running time of RANDOMIZED-SELECT by analyzing the above expression.

(Hint: Distinguish between the following three cases: $i < j \leq k$, $k \leq i < j$, and $i < k < j$.)

Solution:

We start by noting that

$$\mathbb{E}[X_{i,j}] = \Pr[i\text{th smallest number was compared to the } j\text{th smallest number}]$$

and that two numbers are compared only if one is a pivot. We now do case distinction as in the hint.

Case $i < j \leq k$: Suppose that the ℓth smallest number was the first pivot such that $i \leq \ell \leq k$. i and j are compared if $\ell = i$ or $\ell = j$. We shall show that they will never be compared if $\ell \neq i$ and $\ell \neq j$. To see this suppose first that $i < \ell < j$ but then i will be put in S and j in L so they will never be compared. On the other hand, if $j < \ell \leq k$ then $i, j \in S$ but $k \in L$ so the algorithm will recurse on L. Therefore, the probability that i and j will be compared in this case is $\frac{2}{k-i+1}$.

Case $k \leq i < j$: By the same arguments as in the case above, i and j will be compared with probability $\frac{2}{j-k+1}$.

Case $i < k < j$: In this case we only compare i and j if no pivot is chosen between them before anyone of them is chosen as a pivot (same as in the quicksort analysis). The probability that they are compared is thus $\frac{2}{j-i+1}$.

Having analyzed $\mathbb{E}[X_{ij}]$, we turn our attention to the sum. Note that it can be written as

$$\begin{aligned}
\mathbb{E}[X] &= \sum_{i=1}^{k-1}\sum_{j=i+1}^{k}\mathbb{E}[X_{ij}] + \sum_{j=k+1}^{n}\sum_{i=k}^{j-1}\mathbb{E}[X_{ij}] + \sum_{i=1}^{k-1}\sum_{j=k+1}^{n}\mathbb{E}[X_{ij}] \\
&= \sum_{i=1}^{k-1}\sum_{j=i+1}^{k}\frac{2}{k-i+1} + \sum_{j=k+1}^{n}\sum_{i=k}^{j-1}\frac{2}{j-k+1} + \sum_{i=1}^{k-1}\sum_{j=k+1}^{n}\frac{2}{j-i+1} \\
&= \sum_{i=1}^{k-1}(k-i)\frac{2}{k-i+1} + \sum_{j=k+1}^{n}(j-k)\frac{2}{j-k+1} + \sum_{i=1}^{k-1}\sum_{j=k+1}^{n}\frac{2}{j-i+1} \\
&\leq 2n + \sum_{i=1}^{k-1}\sum_{j=k+1}^{n}\frac{2}{j-i+1} \\
&\leq 2n + \sum_{l=1}^{n-1}\sum_{i=k-\ell+1}^{k-1}\frac{2}{\ell+1} \\
&\leq 2n + 2n = 4n.
\end{aligned}$$

We have thus proved that in expectation $4n$ comparisons are made. Therefore, RANDOM-SELECT runs in expected linear time.