Exercise Sheet 8: Answers
COMS10018 Algorithms 2024/2025

Reminder: $\log n$ denotes the binary logarithm, i.e., $\log n = \log_2 n$.

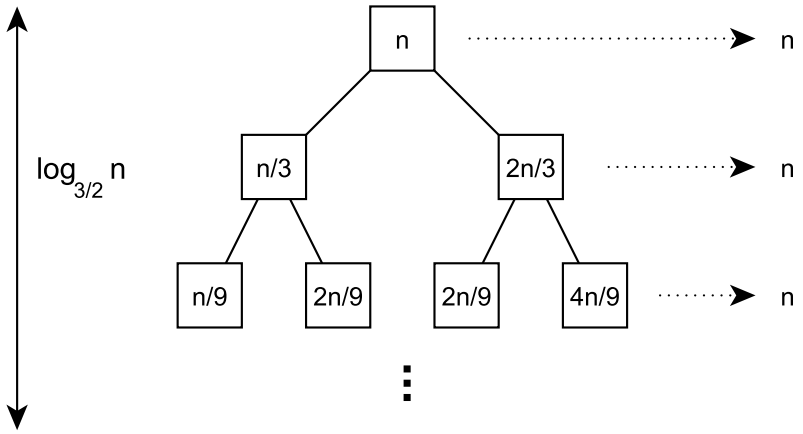
1 Recurrences

Consider the recurrence $T(n) := T(\lfloor \frac{n}{3} \rfloor) + T(\lfloor \frac{2n}{3} \rfloor) + n$, for every $n \geq 3$ and $T(2) = T(1) = 1$.

1. Use the recursion tree method to come up with a guess for an upper bound on the recurrence (in Big-O notation).

   Hint: Ignore the floor operations. Determine the depth of the recursion tree. Determine the "work" that is done in each level of the recursion tree. Sum up the work done in each level to obtain a suitable guess for an upper bound on T.

   Solution. From the recursion tree, we see that the tree has a depth of $\log_{3/2}(n) = O(\log n)$: The right-most path proceeds as $n \to (2/3) \cdot n \to (2/3)^2 n \to \ldots$, which requires $\log_{3/2}(n)$ steps to reach a value $O(1)$. The work done in each level is at most n. Our guess is therefore $O(n \log_{3/2}(n)) = O(n \log n)$.



✓

2. Use the substitution method to prove that the guess obtained in the previous exercise is correct.

   Hint: $0.5 \leq \log(3/2)$

Solution. Our guess is $T(n) = O(n \log n)$. We are thus going to prove $T(n) \leq C \cdot n \log n$, for some constant C that we will determine along the way.

First, we plug the guess into the recurrence:

$$\begin{aligned}
T(n) &= T(\lfloor \frac{n}{3} \rfloor) + T(\lfloor \frac{2n}{3} \rfloor) + n \\
&\leq C\lfloor \frac{n}{3} \rfloor \log(\lfloor \frac{n}{3} \rfloor) + C\lfloor \frac{2n}{3} \rfloor \log(\lfloor \frac{2n}{3} \rfloor) + n \\
&\leq C\frac{n}{3} \log(\frac{n}{3}) + C\frac{2n}{3} \log(\frac{2n}{3}) + n \\
&\leq C\frac{n}{3} \log(\frac{2n}{3}) + C\frac{2n}{3} \log(\frac{2n}{3}) + n \\
&= Cn \log(\frac{2n}{3}) + n \\
&= Cn \log n - Cn \log(3/2) + n \\
&\leq Cn \log n - 0.5Cn + n \leq Cn \log n \ ,
\end{aligned}$$

where the last step holds if $n - 0.5Cn \leq 0$, which implies $C \geq 2$.

Regarding the base case, we pick the case $n = 2$. We have $T(2) = 1$ and $C \cdot 2 \cdot \log(2) = 2C$, which holds for $C \geq \frac{1}{2}$.

We can thus pick $C = 2$, which proves that $T(n) \in O(n \log n)$. ✓

2 Analysis of a Recursive Algorithm

Consider the algorithm ALG listed as Algorithm 1:

Algorithm 1 ALG(n)

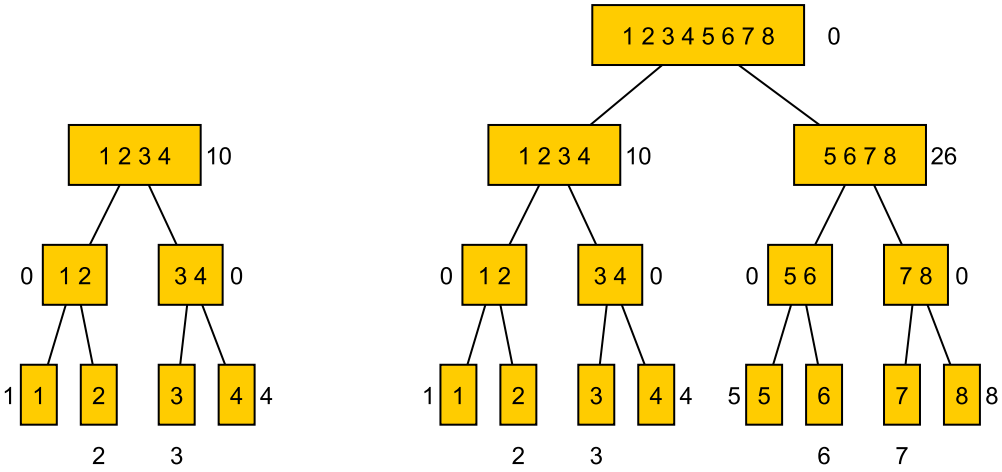
Require: Integer array A of length $n \geq 1$, n is a power of two

```
S ← 0
for i ← 0 ... n − 1 do
    S ← S + A[i]
end for
if n ≤ 1 then
    return S
else
    return S − ALG(A[0, n/2 − 1]) − ALG(A[n/2, n − 1])
end if
```

We assume that the length n of the input array in ALG is always a power of two, i.e., $n \in \{1, 2, 4, 8, 16, \dots\}$.

1. Let $A = 1, 2, 3, 4$ and let $B = 1, 2, 3, 4, 5, 6, 7, 8$. Draw the recursion trees of the calls ALG(A) and ALG(B). For both trees, annotate each node with the value that is returned by the function call that corresponds to this node.

   Solution.



✓

2. Recall that n is a power of two. Let $T(n)$ be the number of times the function ALG (listed in Algorithm 1) is executed when invoked on an input array of length n (including the initial invocation on the array of length n). Give a recursive definition of $T(n)$.

Solution.

$$T(1) = 1 \text{ and } T(n) = 2T(\frac{n}{2}) + 1, \text{ for every } n \geq 2 \ .$$

✓

3. Let $T(n)$ be the function defined in the previous exercise. Use the substitution method to show that $T(n) \in O(n)$.

Solution. We will use the guess $T(n) \leq C \cdot n - 1$, for some constant C. Similar to the lectures, it can be seen that the guess $T(n) \leq C \cdot n$ does not work.

We now plug the guess into the recurrence:

$$T(n) = 2T(\frac{n}{2}) + 1 \leq 2(C\frac{n}{2} - 1) + 1 = Cn - 2 + 1 = Cn - 1 \ .$$

Next, we verify the base case $n = 1$. We see that $T(1) = 1$ and $C \cdot 1 - 1 = C - 1$. We thus need to chose $C \geq 2$ and we pick $C = 2$. ✓

4. What is the runtime of Alg?

Solution. The runtime is $O(n \log n)$. ✓

5. Recall that n is a power of two. Describe an algorithm with best-case runtime $\Theta(1)$ and worst-case runtime $\Theta(n)$ that computes the exact same output as Alg.

Solution. If $\log n \in \{1, 3, 5, 7, \dots\}$ then we output 0. If $\log n \in \{2, 4, 6, 8, \dots\}$ then we compute the sum of the elements of A and output this sum. Observe that we can compute the sum of n element in time $O(n)$. ✓

3 Divide-and-Conquer and the Number of Subproblems

In this exercise, we assume that n is a power of two. We consider a divide-and-conquer algorithm that, on an input of length $n \geq 2$, executes k recursive calls, each on inputs of lengths $n/2$, for some integer $k \geq 1$. The divide and combine phases of the divide-and-conquer algorithm on an instance of size n have a runtime of $O(n)$. We also assume that the runtime on an instance of length 1 is $O(1)$.

1. What is the runtime of the algorithm if $k = 1$?

   Solution. We see that the recursion tree is a path and consists of $\log(n)+1$ levels. The work done in level i is $O(\frac{n}{2^{i-1}})$. Summing up the work done in all levels yields:

   $$\sum_{i=1}^{\log(n)+1} O(\frac{n}{2^{i-1}}) = O(n) \cdot \sum_{i=0}^{\log n} \frac{1}{2^i} \leq O(n) \cdot 2 = O(n) \ .$$

   ✓

2. What is the runtime of the algorithm if $k = 2$?

   Solution. This is exactly the recurrence obtained when analyzing Mergesort (see lectures). The runtime is $O(n \log n)$. ✓

3. What is the runtime of the algorithm if $k = 3$?

   Solution. As in the case $k = 1$ (and $k = 2$), the recursion tree has $\log(n) + 1$ levels. In level i, we have 3^{i-1} nodes, and each node is assigned a work of $O(\frac{n}{2^{i-1}})$. Hence, the total work done in level i is

   $$3^{i-1} \cdot O(\frac{n}{2^{i-1}}) = O(n \cdot (\frac{3}{2})^{i-1}) \ .$$

   Summing up the work done in all levels, we obtain:

   $$\begin{aligned}\sum_{i=1}^{\log(n)+1} O(n \cdot (\frac{3}{2})^{i-1}) = O(n) \sum_{i=0}^{\log(n)} O((\frac{3}{2})^i) &= O(n) \cdot O(\frac{(\frac{3}{2})^{\log(n)+1} - 1}{\frac{1}{2}}) \\ &= O(n \cdot (\frac{3}{2})^{\log(n)}) = O(n n^{\log(\frac{3}{2})}) \approx O(n^{1.5849\ldots}) \ ,\end{aligned}$$

   where we used the formula $\sum_{i=0}^{k} x^i = \frac{x^{k+1}-1}{x-1}$.

   ✓

4 Optional and Difficult Questions

Exercises in this section are intentionally more difficult and are there to challenge yourself.

4.1 Search in a Sorted Matrix (Difficult!)

We assume in this exercise that n is a power-of-two.

We are given an n-by-n integer matrix A that is sorted both row- and column-wise, i.e., every row is sorted in non-decreasing order from left to right, and every column is sorted in non-decreasing order from top to bottom. Give a divide-and-conquer algorithm that answers the question:

"Given an integer x, does A contain x?"

What is the runtime of your algorithm?

Solution. We define the following submatrices of matrix A:

$$\begin{aligned} A_{11} &= A[0 \dots \frac{n}{2} - 1, 0 \dots \frac{n}{2} - 1] \\ A_{21} &= A[\frac{n}{2} \dots n - 1, 0 \dots \frac{n}{2} - 1] \\ A_{12} &= A[0 \dots \frac{n}{2} - 1, \frac{n}{2} \dots n - 1] \\ A_{22} &= A[\frac{n}{2} \dots n - 1, \frac{n}{2} \dots n - 1] \end{aligned}$$

Observe that the dimensions of all submatrices are $n/2 \times n/2$.

We first check whether $A_{\frac{n}{2}-1,\frac{n}{2}-1} = x$. If this is the case then we have found x and we are done. Otherwise, we distinguish the following two cases:

1. Suppose that $A_{\frac{n}{2}-1,\frac{n}{2}-1} < x$ holds. Then, since A is sorted in both column and row order, it is not hard to see that x is not contained in A_{11}. We then invoke our algorithm recursively and search for x in the three submatrices A_{12}, A_{21}, A_{22}.

2. Suppose that $A_{\frac{n}{2}-1,\frac{n}{2}-1} > x$ holds. Then, similar as before, it is not hard to see that x is not contained in A_{22}. We then invoke our algorithm recursively and search for x in the three submatrices A_{11}, A_{12}, A_{21}.

Observe that the proposed algorithm is a recursive algorithm. We thus need to decide what to do if the input to a recursive call is a 1×1 matrix. In this case we simply check whether the single element in the matrix equals x in $O(1)$ time.

Let $T(n)$ be the runtime of the algorithm when executed on an input array of dimension $n \times n$. We thus obtain the following recurrence:

$$T(n) = \begin{cases} O(1) \ , & \text{if } n = 1, \\ 3T(n/2) + O(1) \ , & \text{otherwise.} \end{cases}$$

It remains to solve the recurrence $T(n)$. First, we eliminate the $O(1)$ terms and replace them with a large enough constant C:

$$T(n) = \begin{cases} C \ , & \text{if } n = 1, \\ 3T(n/2) + C \ , & \text{otherwise.} \end{cases}$$

Our recursion is simple enough to obtain a solution via the recursion tree method. In the lecture, we used the recursion tree method in order to obtain a guess the we then verified using

the substitution method. The recursion here is however simple enough to conduct a complete analysis using the recursion tree.

From the recursion tree, we see that the tree has $\log(n)+1$ levels. Denoting the root of the tree as level 0, we see that level i has 3^i nodes. Furthermore, every node is labeled by C. The total work therefore is:

$$\begin{aligned}
\sum_{i=0}^{\log n} 3^i C &= C \cdot \sum_{i=0}^{\log n} 3^i = C \cdot \frac{3^{\log(n)+1}-1}{3-1} \\
&= \frac{C}{2} \cdot \left(2^{\log(3)\log(n)+\log(3)} - 1\right) \leq \frac{C}{2} \cdot \left(2^{\log(3)\log(n)+\log(3)}\right) \\
&= \frac{C}{2} \cdot \left(n^{\log 3} \cdot 3\right) = O(n^{\log 3}) \approx O(n^{1.5849\ldots}) \ .
\end{aligned}$$

We used the formula $\sum_{i=0}^{k} x^i = \frac{x^{k+1}-1}{x-1}$ in this calculation.

Last, I would like to mention that there exists a solution to this problem that runs in time $O(n)$. Can you think of such a solution? ✓