Exercise Sheet 8
COMS10017 Algorithms 2020/2021

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

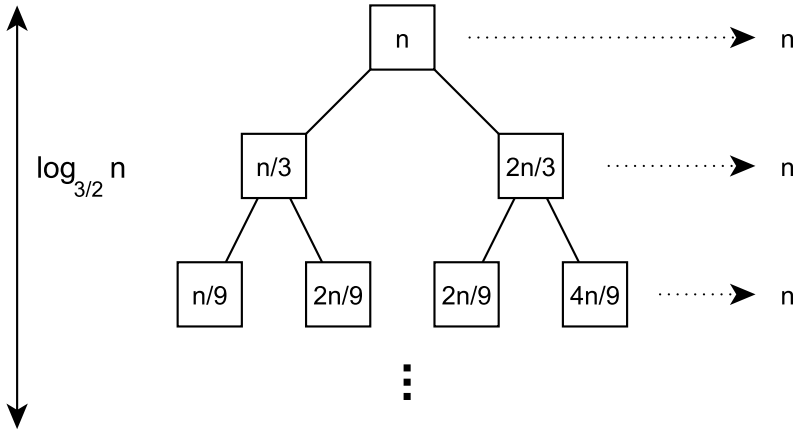
1 Recurrences

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. The product of these quantities serves as 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 n. Our guess is therefore $O(n \log n)$.



✓

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)

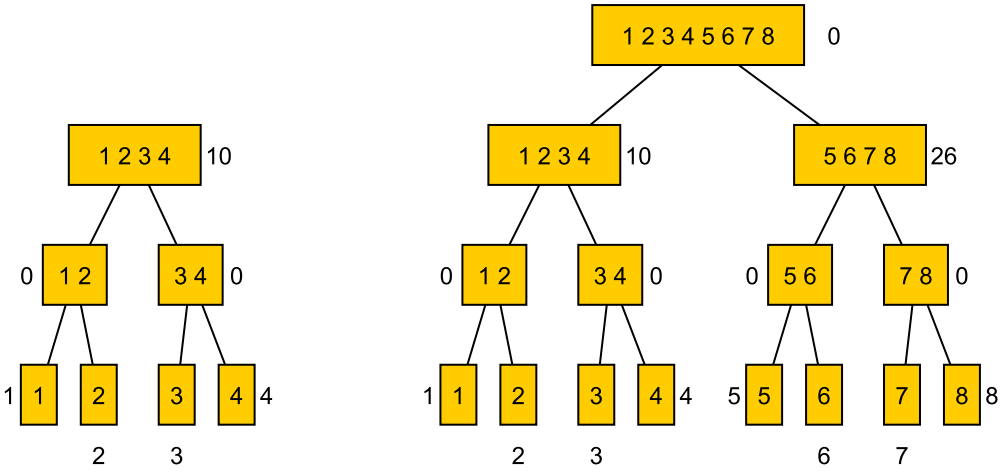
Require: Integer array A of length $n \geq 1$, n is a power of two

```
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 $T(n) \in O(n)$. Use the guess $T(n) \leq C \cdot n - 1$, for a constant C. Give the smallest constant C so that the previous statement is true.

Solution. We first 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? (no justification needed)

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 $\log n \in \{2, 4, 8, 16, \dots\}$ 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 Algorithmic Puzzle: Maxima of Windows of length $n/2$

We are given an array A of n positive integers, where n is even. Give an algorithm that outputs an array B of length $n/2$ such that $B[i] = \max\{A[j]\ ,\ i \leq j \leq i + n/2 - 1\}$. Can you find an algorithm that runs in time $O(n)$?

Solution. Let $C[i]$ and $D[i]$ be new arrays of lengths $n/2$. We first observe that we can rewrite $B[i]$ as the maximum of two maxima:

$$\begin{aligned} B[i] &= \max\{C[i], D[i]\}\ , \text{ where} \\ C[i] &= \max\{A[j]\ :\ i \leq j \leq n/2 - 1\}\ , \text{ and} \\ D[i] &= \max\left(\{A[j]\ :\ n/2 \leq j \leq i + n/2 - 1\} \cup \{0\}\right)\ . \end{aligned}$$

Suppose we already computed the tables C and D. Then in $O(n)$ time, we can compute the table B by computing the maxima $\max\{C[i], D[i]\}$ for every $0 \leq i \leq n/2 - 1$. It thus remains to compute tables C and D. To this end, observe that $C[n/2 - 1] = A[n/2 - 1]$, and for every $k < n/2 - 1$, we have $C[k] = \max\{A[k], C[k + 1]\}$. We thus obtain the following algorithm for computing the table C:

Algorithm 2 Computing table C

```
C[n/2 − 1] ← A[n/2 − 1]
for i = n/2 − 2 . . . 0 do
    C[i] ← max{A[i], C[i + 1]}
end for
```

Similarly, observe that $D[0] = 0$, and for every $k > 0$, we have $D[k] = \max\{D[k - 1], A[k + n/2]\}$. We thus obtain the following algorithm for computing table D:

Algorithm 3 Computing table D

```
D[0] ← 0
for i = 1 . . . n/2 − 1 do
    D[i] ← max{A[i + n/2], D[i − 1]}
end for
```

Computing tables C and D takes $O(n)$ time. The total runtime is therefore $O(n)$. ✓