



EPFL



1

Teacher : Michael Kapralov
Algorithms CS-250 - SC
11 Nov 2022
Duration : 110 minutes

Student Sample

SCIPER: **325664**

Do not turn the page before the start of the exam. This document is double-sided, has 20 pages, the last ones possibly blank. Do not unstaple.

- Place your student card on your table.
- You are only allowed to have a handwritten A4 page written on both sides.
- Communication, calculators, cell phones, computers, etc... are not allowed.
- The exam consists of two parts. The first part consists of multiple choice questions and the second part consists of open ended questions.
- Use a **black or dark blue ballpen** and clearly erase with **correction fluid** if necessary.
- If a question is wrong, the teacher may decide to nullify it.

Good luck!

Respectez les consignes suivantes Read these guidelines Beachten Sie bitte die unten stehenden Richtlinien		
choisir une réponse select an answer Antwort auswählen	ne PAS choisir une réponse NOT select an answer NICHT Antwort auswählen	Corriger une réponse Correct an answer Antwort korrigieren
<input checked="" type="checkbox"/> <input checked="" type="checkbox"/> <input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
ce qu'il ne faut PAS faire what should NOT be done was man NICHT tun sollte		



First part: multiple choice questions

For each question, mark the box corresponding to the correct answer. Each question has **exactly one** correct answer.

Question 1 : (4 pts)

Give a tight asymptotic bound for the recurrence $F(n) = 3 \cdot F(n/4) + \Theta(n)$, where we assume $F(n) = \Theta(1)$ for $n \leq 10$.

- $F(n) = \Theta(n^{\log_3 4} \log n)$
- $F(n) = \Theta(n^{\log_3 4})$
- $F(n) = \Theta(n)$
- $F(n) = \Theta(n^{\log_4 3})$
- $F(n) = \Theta(n \log n)$

Question 2 : (4 pts)

Give a tight asymptotic bound for the recurrence $G(n) = 4 \cdot G(n/2) + \Theta(n^2)$, where we assume $G(n) = \Theta(1)$ for $n \leq 10$.

- $G(n) = \Theta(n)$
- $G(n) = \Theta(n^2)$
- $G(n) = \Theta(n^4)$
- $G(n) = \Theta(n^2 \log n)$
- $G(n) = \Theta(n^4 \log n)$

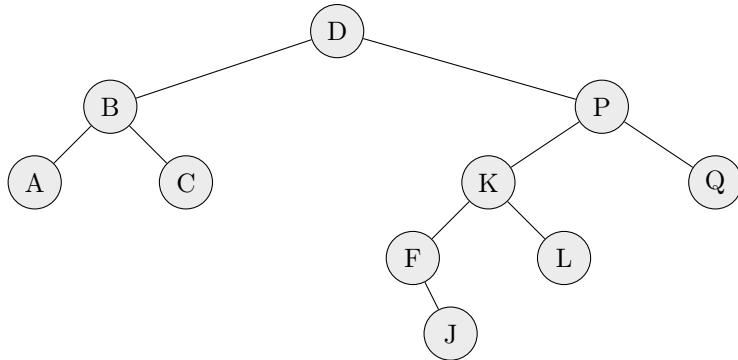
Question 3 : (4 pts)

Give a tight asymptotic bound for the recurrence $H(n) = H(\lfloor n^{1/3} \rfloor) + \Theta(\log n)$, where we assume $H(n) = \Theta(1)$ for $n \leq 10$.

- $H(n) = \Theta(n^{1/3})$
- $H(n) = \Theta(n^{1/3} \log n)$
- $H(n) = \Theta(\log^2 n)$
- $H(n) = \Theta(\log n)$
- $H(n) = \Theta(\log n \log \log n)$

Question 4 : (6 pts)

Let T be the following binary search tree:



Suppose that we delete the root of T using the procedure seen in class. What is the pre-order traversal of the resulting binary search tree:

- $F, A, B, C, P, L, K, Q, J$
- $F, B, A, C, P, J, K, L, Q$
- $F, B, A, C, P, K, J, L, Q$
- $F, A, B, C, P, J, K, L, Q$
- $P, F, A, B, C, L, K, Q, J$
- $P, F, Q, K, L, J, B, C, A$
- $F, P, Q, K, J, L, B, A, C$
- $F, P, Q, K, L, J, B, C, A$

**Question 5 :** (8 pts)

Arrange the following functions in increasing order according to asymptotic growth.

$$2^N, \log N, N^{1/\log \log N}, 2^{\sqrt{\log N}}, (\log \log N)^5, N^3, N \log(N^5 + N)$$

The correct order is:

- $N^{1/\log \log N}, (\log \log N)^5, \log N, 2^{\sqrt{\log N}}, N \log(N^5 + N), N^3, 2^N$
- $\log N, (\log \log N)^5, N^{1/\log \log N}, N^3, N \log(N^5 + N), 2^{\sqrt{\log N}}, 2^N$
- $\log N, (\log \log N)^5, N^{1/\log \log N}, N \log(N^5 + N), N^3, 2^{\sqrt{\log N}}, 2^N$
- $N^{1/\log \log N}, (\log \log N)^5, \log N, N \log(N^5 + N), N^3, 2^{\sqrt{\log N}}, 2^N$
- $(\log \log N)^5, N^{1/\log \log N}, \log N, N \log(N^5 + N), N^3, 2^{\sqrt{\log N}}, 2^N$
- $(\log \log N)^5, N^{1/\log \log N}, 2^{\sqrt{\log N}}, \log N, N \log(N^5 + N), N^3, 2^N$
- $(\log \log N)^5, \log N, 2^{\sqrt{\log N}}, N^{1/\log \log N}, N \log(N^5 + N), N^3, 2^N$
- $N^{1/\log \log N}, \log N, (\log \log N)^5, N^3, N \log(N^5 + N), 2^{\sqrt{\log N}}, 2^N$
- $N^{1/\log \log N}, (\log \log N)^5, N \log(N^5 + N), \log N, N^3, 2^{\sqrt{\log N}}, 2^N$
- $(\log \log N)^5, N^{1/\log \log N}, \log N, 2^{\sqrt{\log N}}, N \log(N^5 + N), N^3, 2^N$
- $(\log \log N)^5, N^{1/\log \log N} \log N, N \log(N^5 + N), N^3, 2^{\sqrt{\log N}}, 2^N$



Heaps. Let $A[1\dots 9] = [17 \ 12 \ 7 \ 11 \ 8 \ 0 \ 3 \ 2 \ 9]$ be a heap consisting of 9 numbers. We remark that in each of the following subproblems we start with a fresh copy of this heap A , i.e., without any modifications.

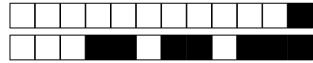
Question 6 : (4 pts)

Illustrate what A looks like after executing the code $\text{HEAP-EXTRACT-MAX}(A, 9)$.

- $A = [12 \ 11 \ 7 \ 9 \ 8 \ 0 \ 3 \ 2]$
- $A = [12 \ 11 \ 9 \ 7 \ 8 \ 0 \ 3 \ 2]$
- $A = [11 \ 2 \ 7 \ 9 \ 8 \ 0 \ 3 \ 12]$
- $A = [12 \ 11 \ 9 \ 8 \ 7 \ 2 \ 3 \ 0]$
- $A = [12 \ 11 \ 7 \ 8 \ 9 \ 0 \ 3 \ 2]$

Question 7 : (4 pts) Illustrate how A looks like after executing the code $\text{HEAP-INCREASE-KEY}(A, 6, 19)$. (6 is the index of the element we increase the key of and 19 is the new key value.)

- $A = [19 \ 12 \ 17 \ 11 \ 9 \ 7 \ 3 \ 2 \ 8]$
- $A = [19 \ 12 \ 17 \ 9 \ 11 \ 7 \ 3 \ 2 \ 8]$
- $A = [19 \ 17 \ 12 \ 9 \ 11 \ 7 \ 3 \ 2 \ 8]$
- $A = [19 \ 12 \ 17 \ 11 \ 8 \ 7 \ 3 \ 2 \ 9]$
- $A = [19 \ 17 \ 12 \ 11 \ 8 \ 7 \ 3 \ 2 \ 9]$



Second part, open questions

This part consists of three questions, each worth 22 points. Please follow the following instructions:

- Your explanations should be clear enough and in sufficient detail that a fellow student can understand them. In particular, do not only give pseudocode without explanations. A good guideline is that a description of an algorithm should be such that a fellow student can easily implement the algorithm following the description.
- You are allowed to refer to material covered in the lectures including algorithms and theorems (without reprobining them). You are however *not* allowed to simply refer to material covered in exercises.
- Please answer all questions within the designated boxes (otherwise your answer may not be accurately scanned). At the end of the exam there are five extra pages if you need additional space for your answers.
- Leave the check-boxes empty, they are used for the grading.

Question 8: Recurrences. (22 pts)

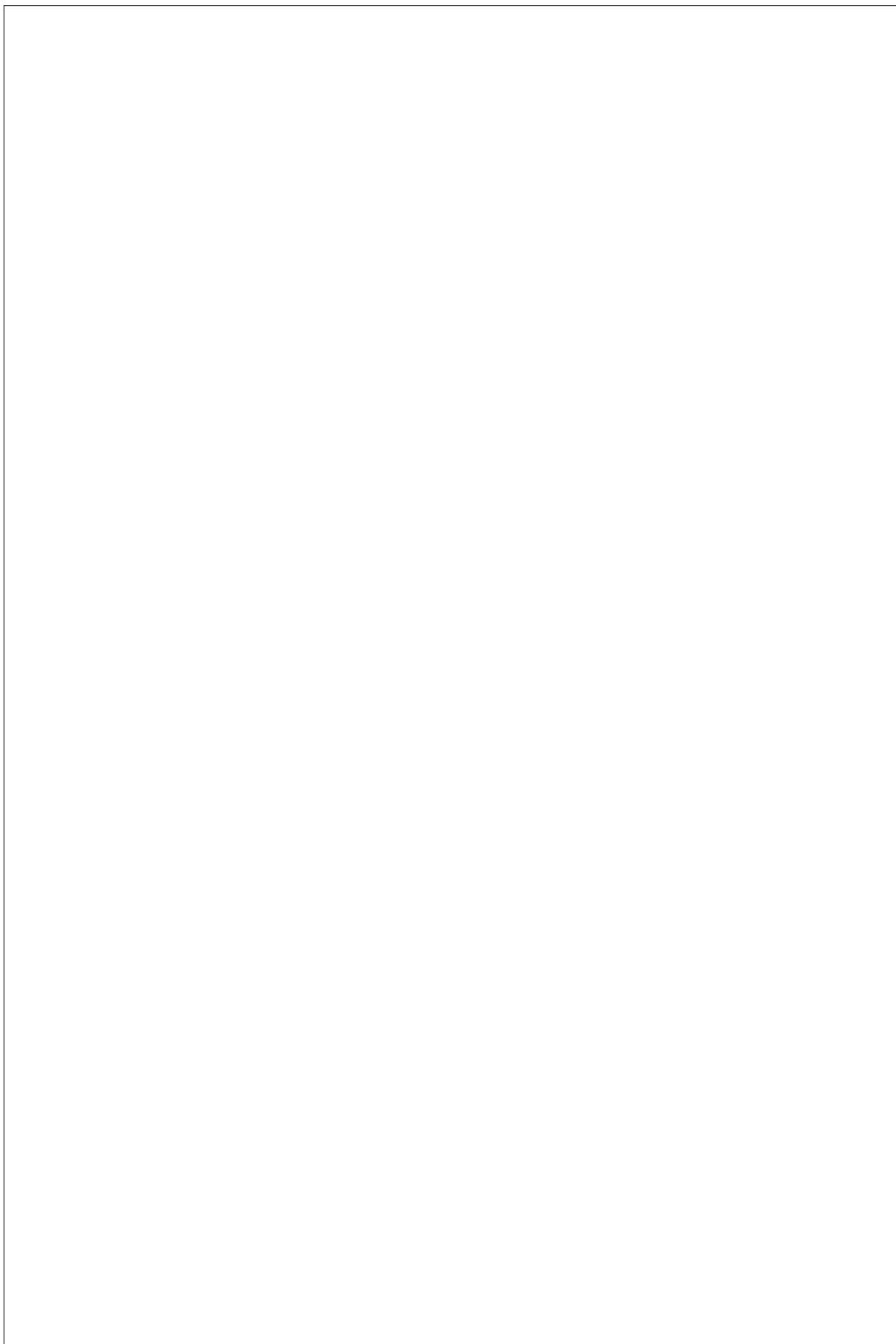
0 1 2 3 4 5 6 7 8 9 10 11
12 13 14 15 16 17 18 19 20 21 22

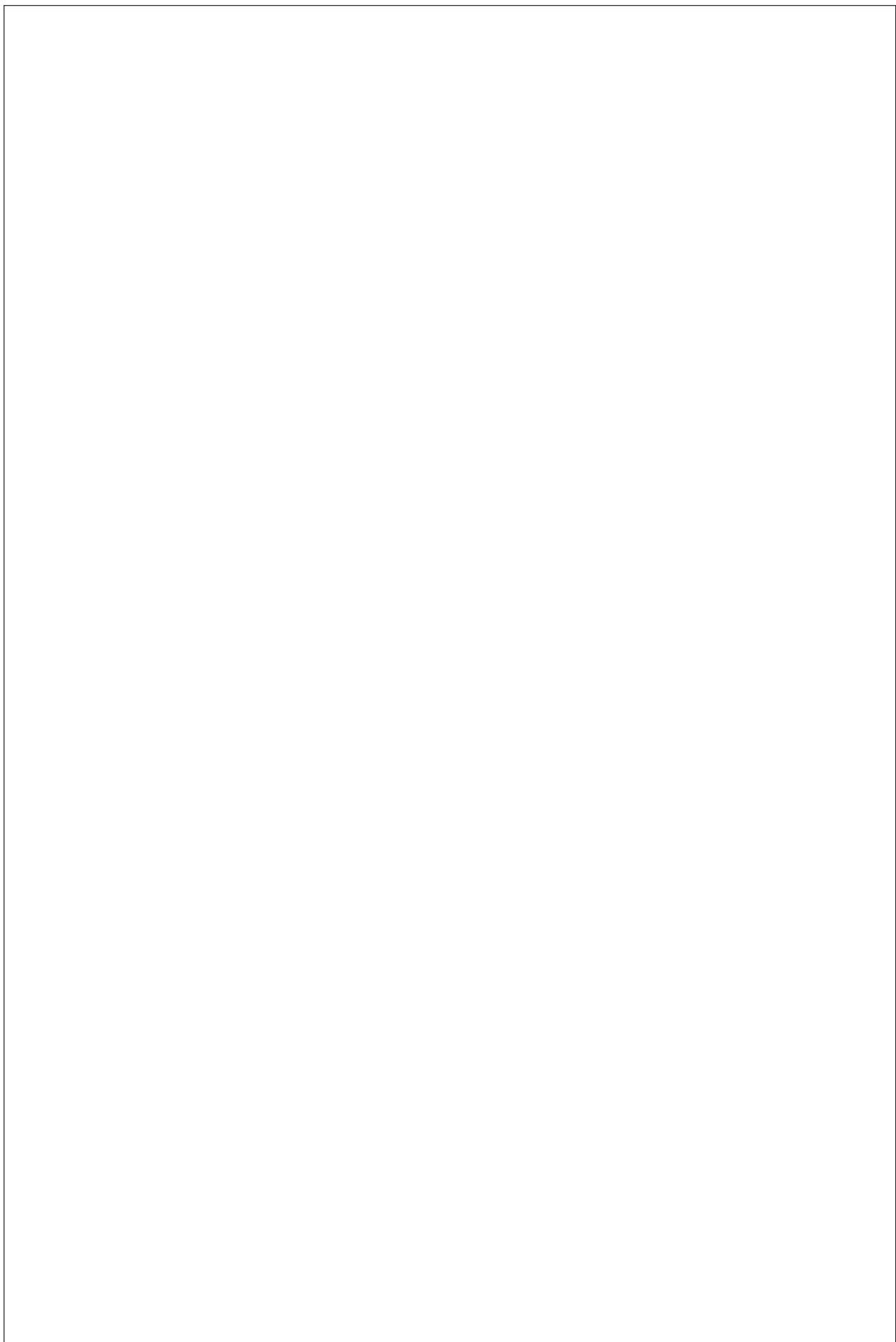
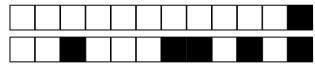
Do not write here.

Consider the recurrence $T(n) = 4T(n/4) + 3T(n/2) + n$, $T(1) = \Theta(1)$. Prove that $T(n) = \Theta(n^2)$.



+1/7/54+





**Question 9: Running a race. (22 pts)**

<input type="text"/> 0	<input type="text"/> 1	<input type="text"/> 2	<input type="text"/> 3	<input type="text"/> 4	<input type="text"/> 5	<input type="text"/> 6	<input type="text"/> 7	<input type="text"/> 8	<input type="text"/> 9	<input type="text"/> 10	<input type="text"/> 11
<input type="text"/> 12	<input type="text"/> 13	<input type="text"/> 14	<input type="text"/> 15	<input type="text"/> 16	<input type="text"/> 17	<input type="text"/> 18	<input type="text"/> 19	<input type="text"/> 20	<input type="text"/> 21	<input type="text"/> 22	

Do not write here.

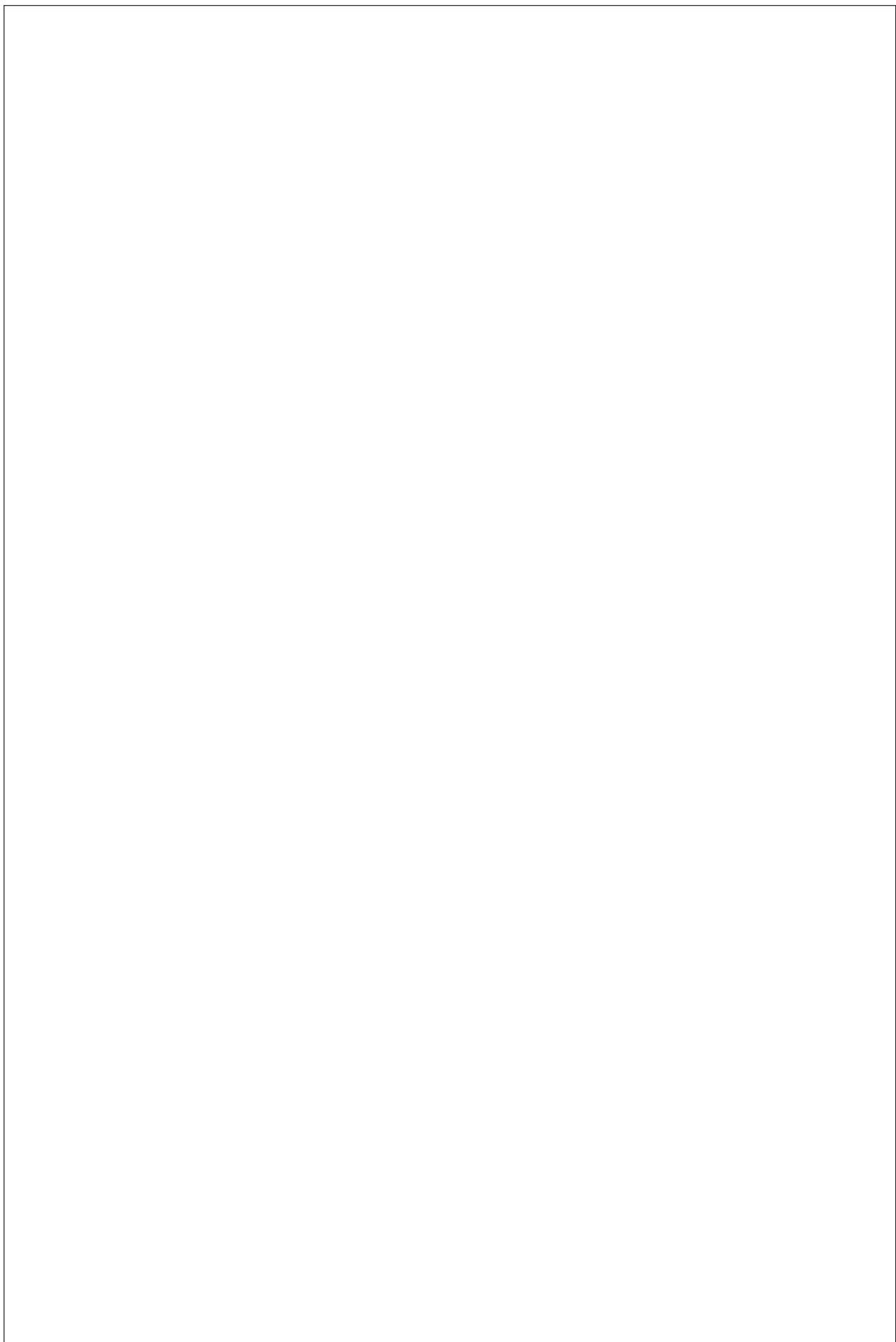
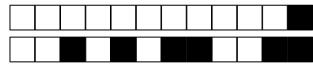
You are managing a team of three runners (Alice, Bob and Charlie) who must run a race consisting of n segments. You have determined a preferred runner for every segment $i = 1, \dots, n$. However, according to the rules of the race the runners can only be swapped in a round-robin fashion, i.e. Bob can only run after Alice, Charlie can only run after Bob and Alice can only run after Charlie (but any given runner can run multiple segments in a row). For example, if $n = 3$, the feasible sequences of runners are ABC, AAB, AAA, ABB, BCA, BBC, BBB, BCC, CAB, CCA, CCC, CAA, where A stands for Alice, B for Bob and C for Charlie. For $n = 4$ feasible sequences include *ABCA* (i.e. note that a runner can run multiple disjoint segments). Formally, a sequence of runners is feasible if A's are followed only by other A's or by B's, B's are followed by B's or C's, and C's are followed by A's or C's. You need to find a feasible sequence of runners such that the maximum possible number of segments is run by your preferred runner for that segment.

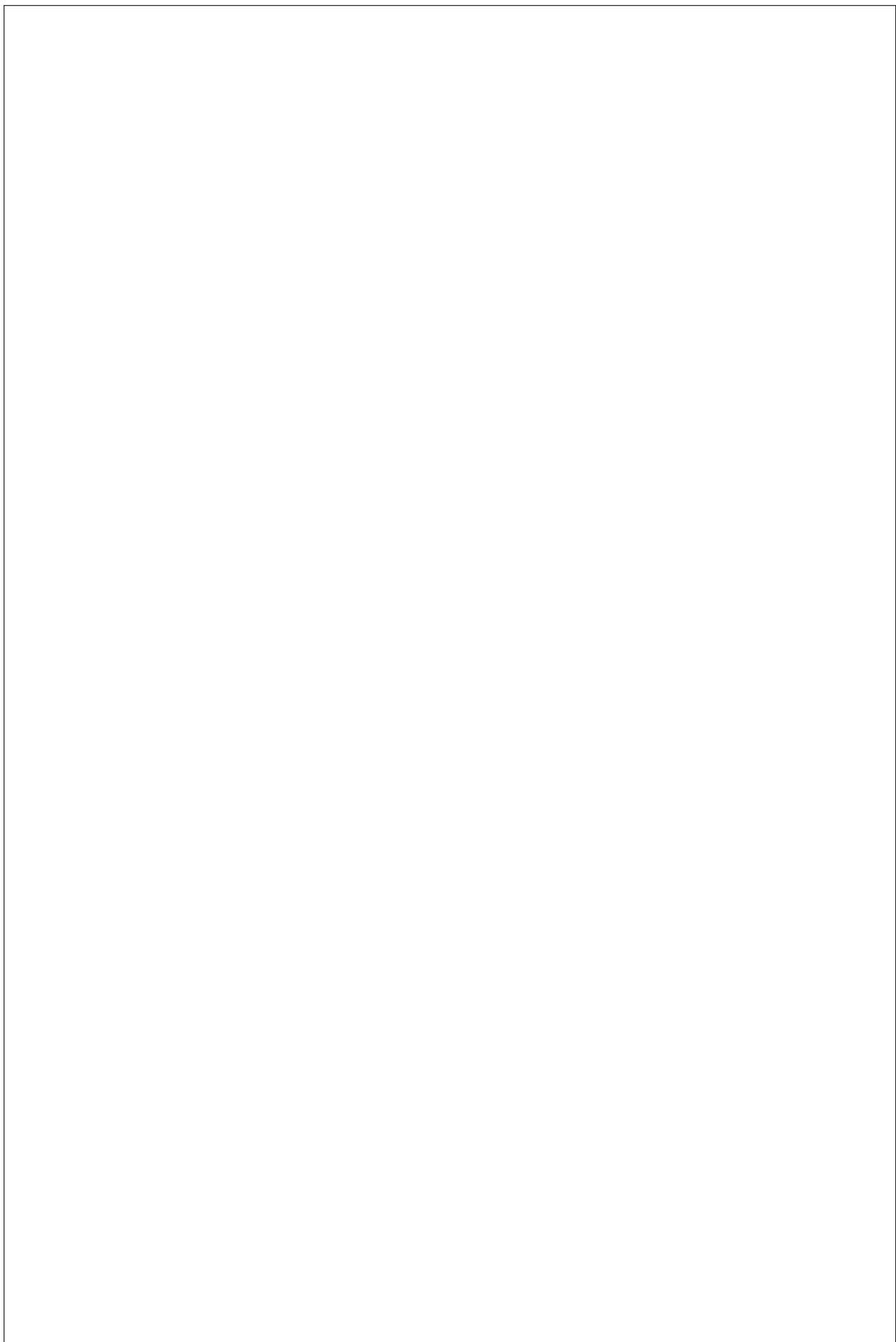
Design and analyze an efficient algorithm for the following problem:

Input: An array $P[1, \dots, n]$ of preferences for the segments, where for every $i \in \{1, 2, \dots, n\}$ one has $P[i] \in \{A, B, C\}$.

Output: A feasible sequence matching P in the largest possible number of segments.

Hint. Use dynamic programming. For every $i \in \{1, 2, \dots, n\}$ and $R \in \{A, B, C\}$ let $m[i, R]$ denote the largest possible overlap that a feasible sequence of runners of length i that finishes with R can have with $P[1 \dots i]$, and show how to compute $m[i, R]$ using a bottom-up approach.





**Question 10: Reconstructing a binary search tree. (22 pts)**

<input type="text"/> 0	<input type="text"/> 1	<input type="text"/> 2	<input type="text"/> 3	<input type="text"/> 4	<input type="text"/> 5	<input type="text"/> 6	<input type="text"/> 7	<input type="text"/> 8	<input type="text"/> 9	<input type="text"/> 10	<input type="text"/> 11
<input type="text"/> 12	<input type="text"/> 13	<input type="text"/> 14	<input type="text"/> 15	<input type="text"/> 16	<input type="text"/> 17	<input type="text"/> 18	<input type="text"/> 19	<input type="text"/> 20	<input type="text"/> 21	<input type="text"/> 22	

Do not write here.

In this problem you will show that a Binary Search Tree T with n keys can be reconstructed uniquely given its in-order traversal and its pre-order traversal. We assume for simplicity that the keys in T are distinct.

Design and analyze an algorithm for the following problem:

Input: An integer n , two length n sequences σ and τ of distinct integers.

Output: A binary search tree T such that σ is its in-order traversal and τ is its pre-order traversal, if such a tree exists, and **NONE** otherwise.

The running time of your algorithm should be polynomial in n . While your algorithm should run in polynomial time, it is not important that you achieve an optimal running time in this question.

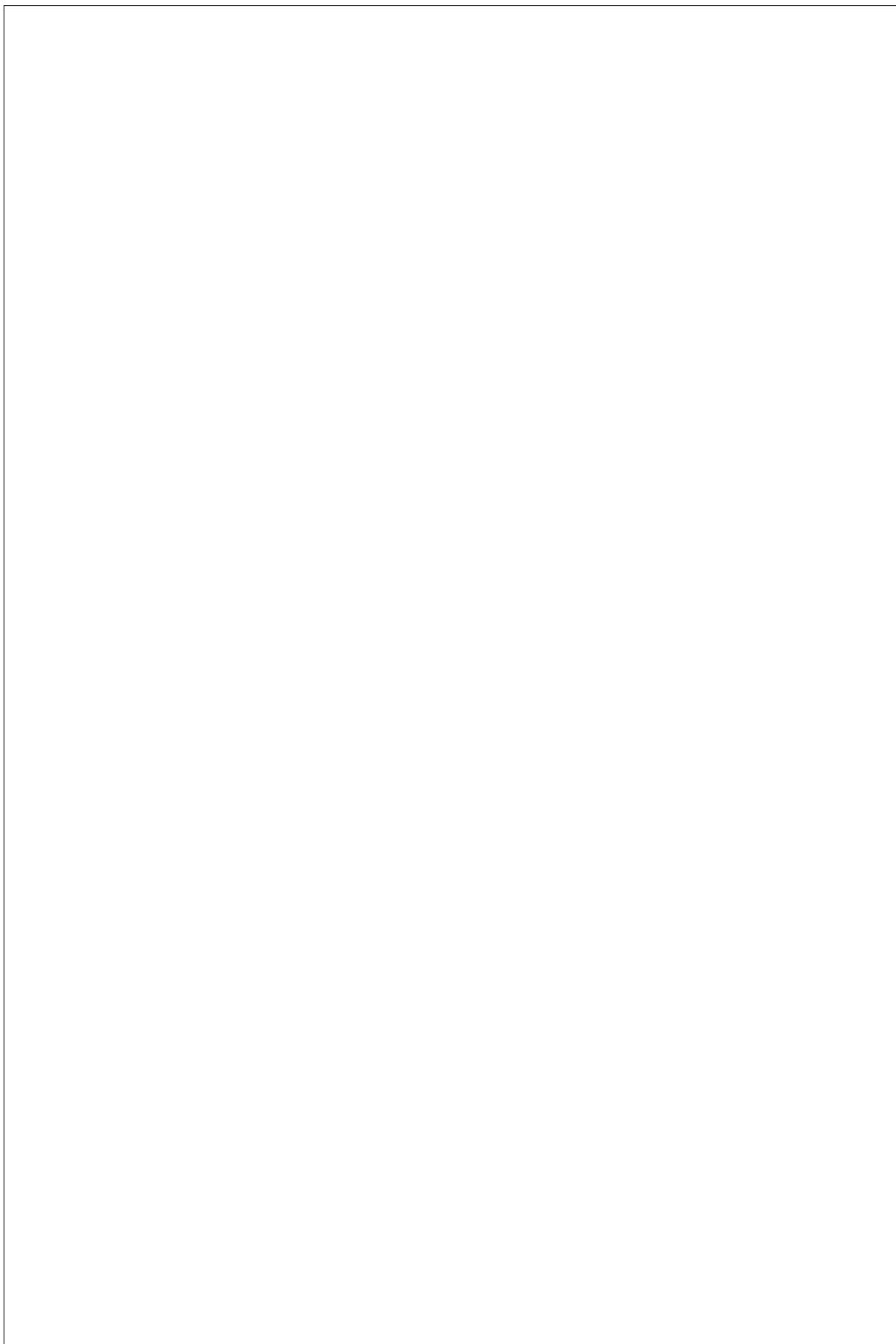
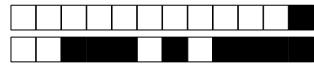
We recall the pseudocode of the in-order tree walk of a tree T rooted at a node x :

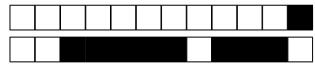
```
IN-ORDER-TREE-WALK( $x$ )
If  $x \neq \text{NIL}$  then
  IN-ORDER-TREE-WALK( $x.left$ )
  print( $key[x]$ )
  IN-ORDER-TREE-WALK( $x.right$ )
```

The pseudocode of the pre-order tree walk of a tree T rooted at a node x is:

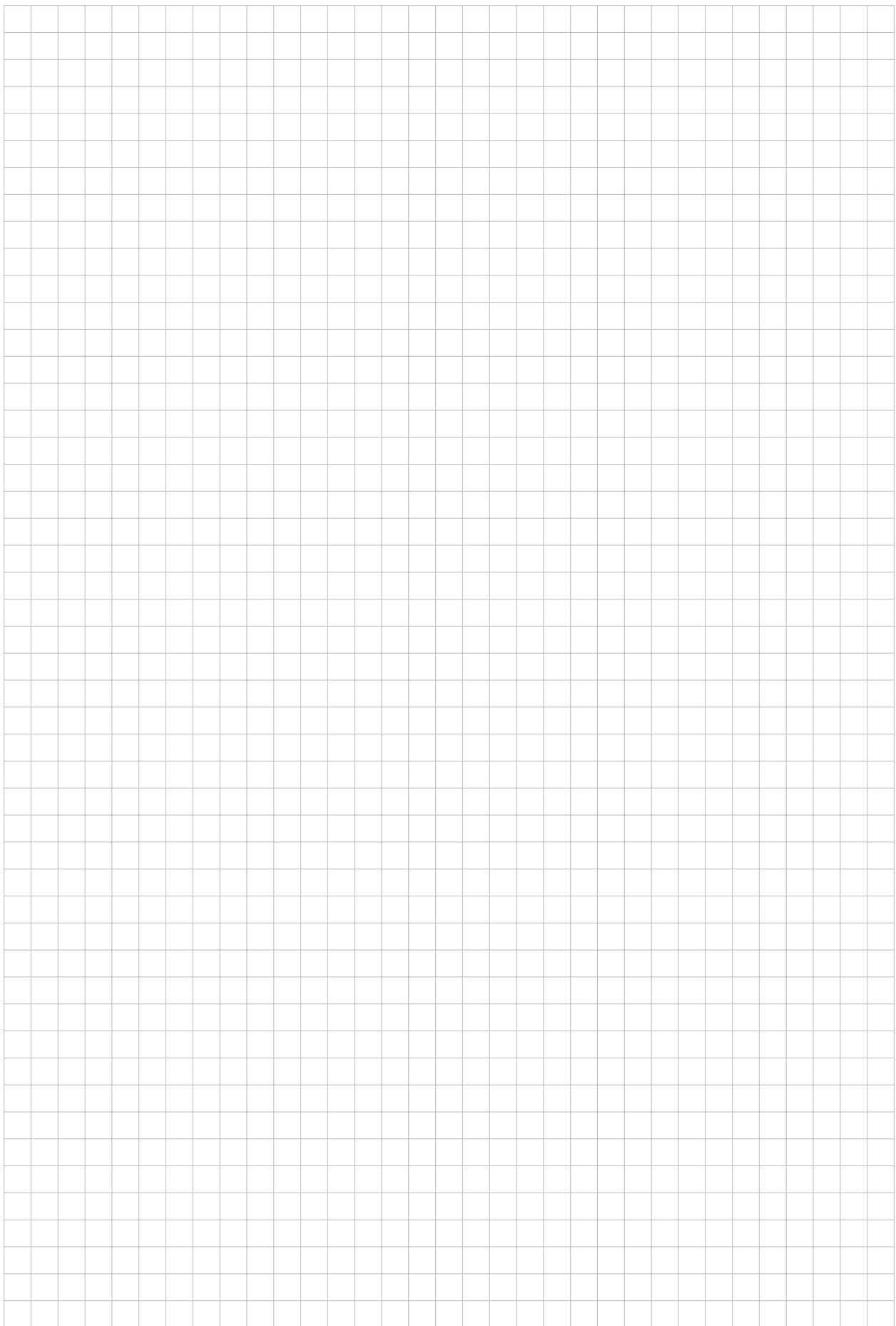
```
PRE-ORDER-TREE-WALK( $x$ )
If  $x \neq \text{NIL}$  then
  print( $key[x]$ )
  PRE-ORDER-TREE-WALK( $x.left$ )
  PRE-ORDER-TREE-WALK( $x.right$ )
```

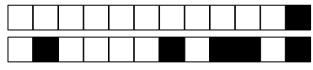




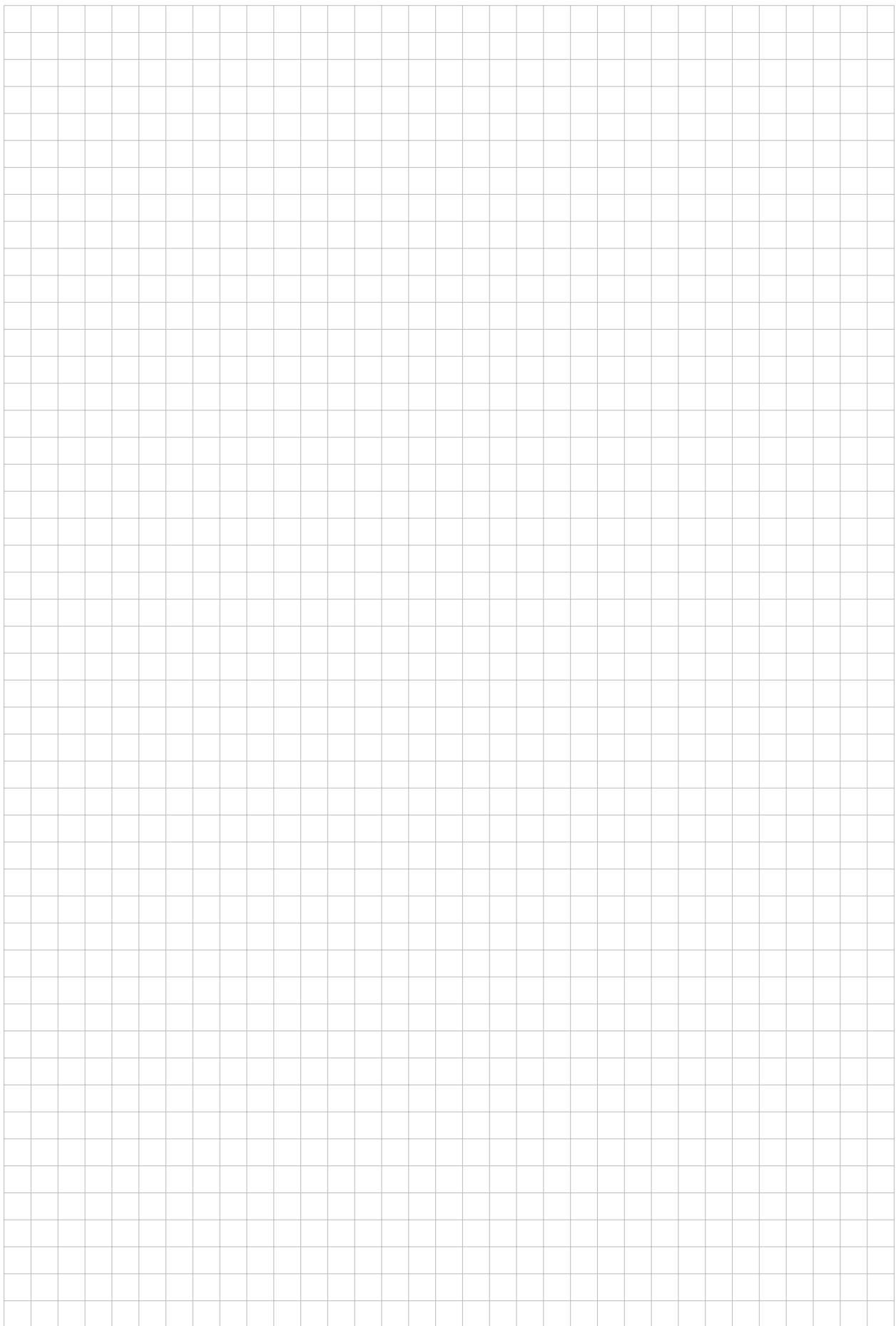


+1/15/46+



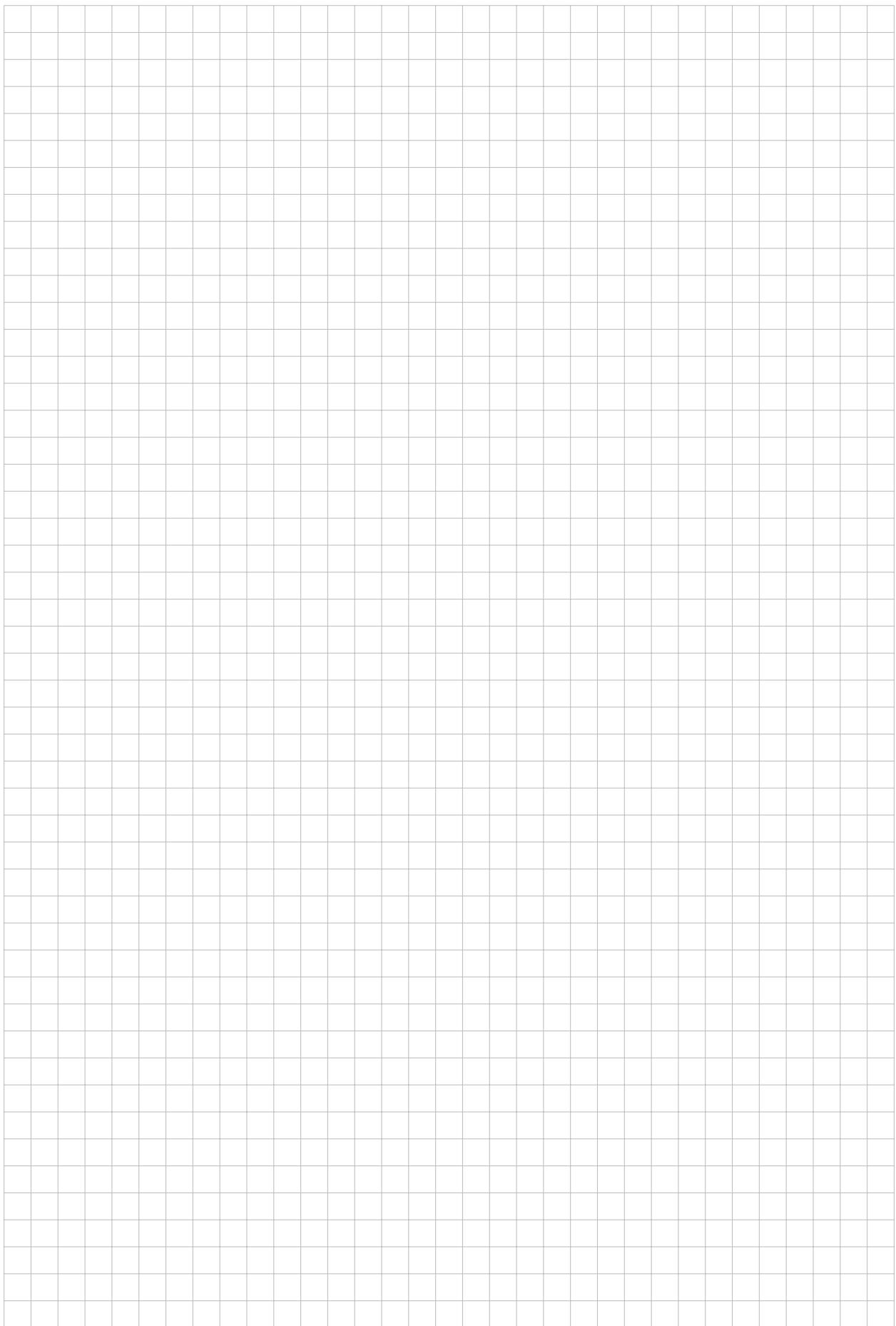


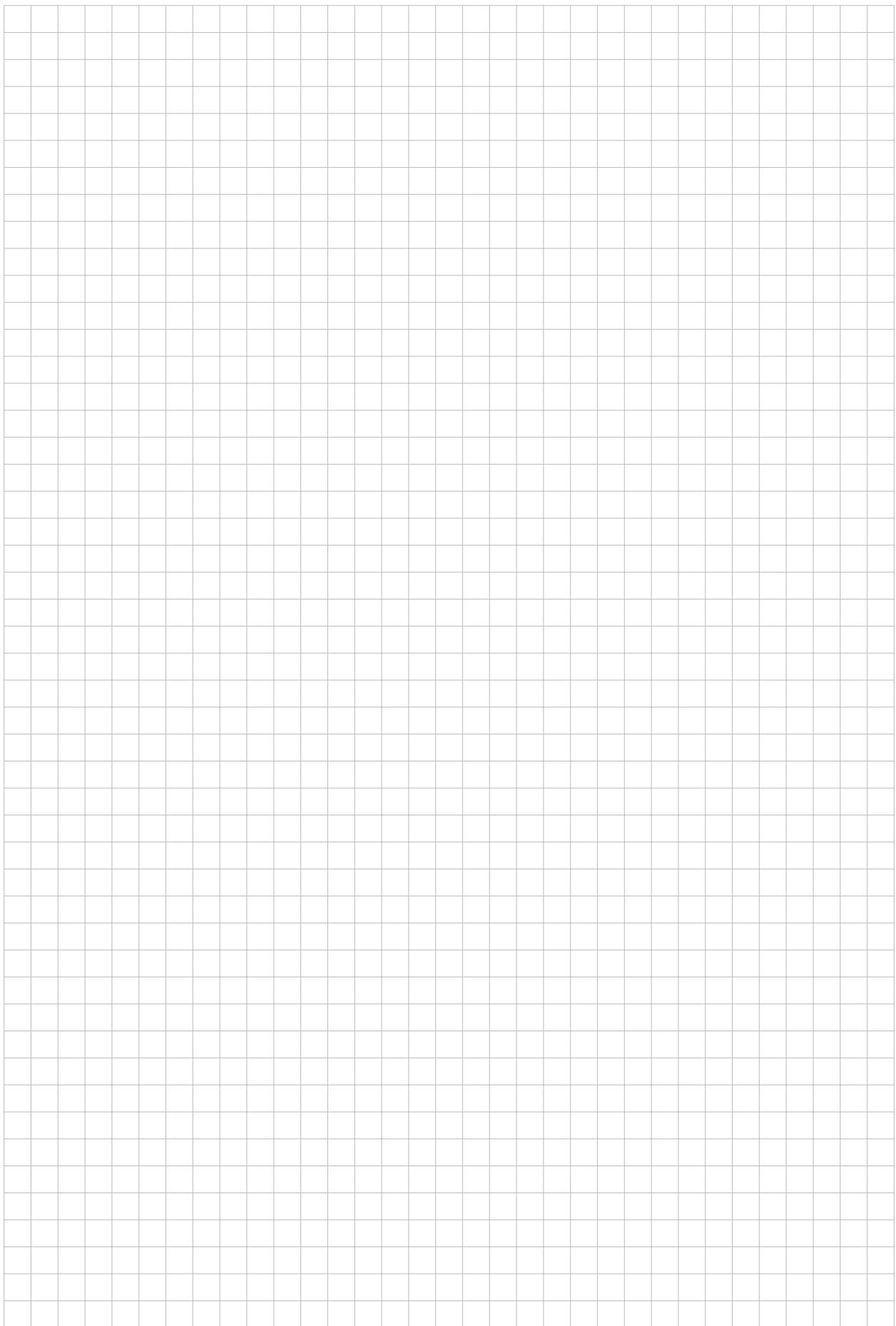
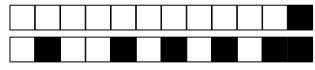
+1/16/45+





+1/17/44+







+1/19/42+

