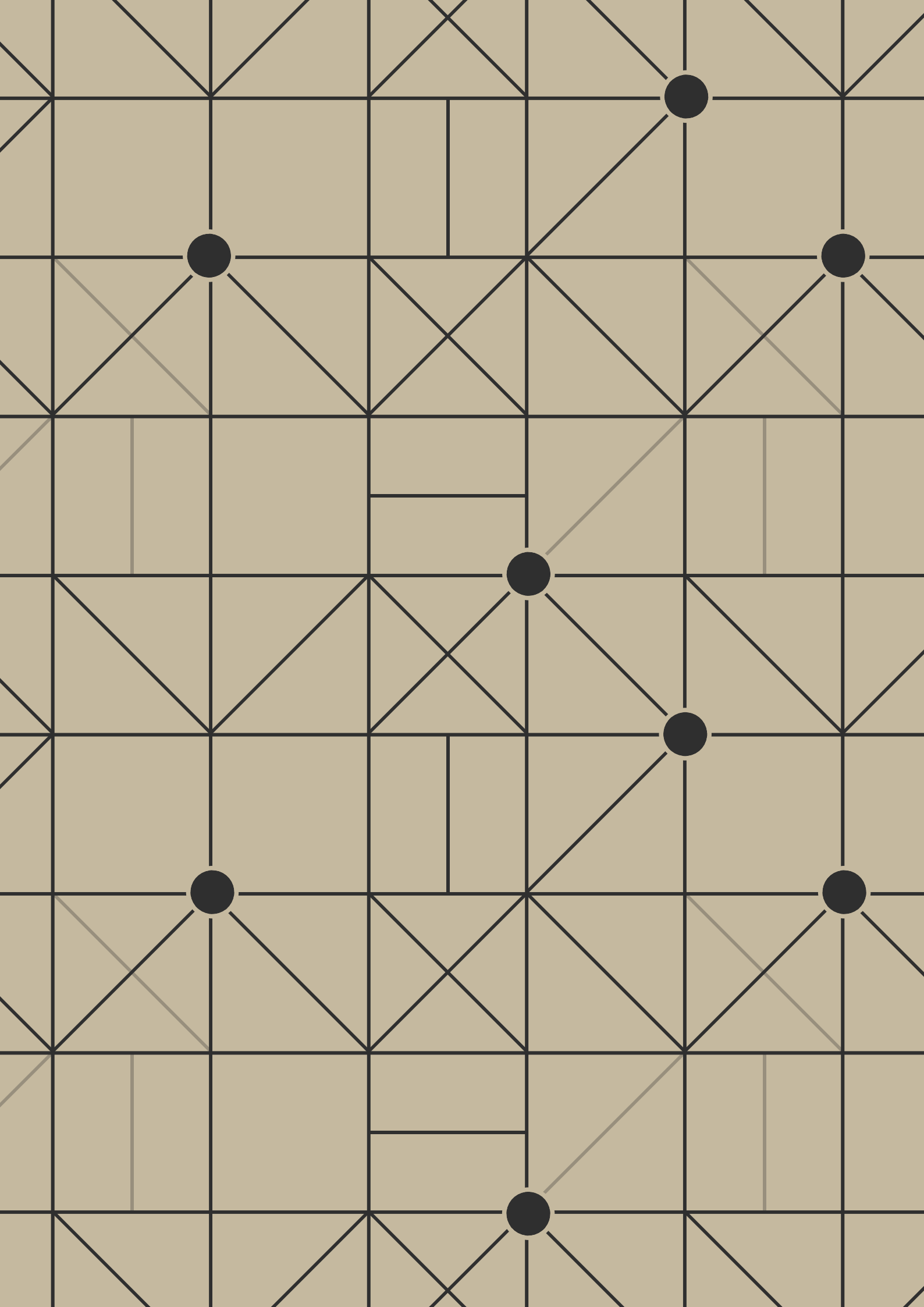

ALGORITHMEN & DATENSTRUKTUREN - ÜBUNGSSTUNDE 4

3.2

(c) Consider an integer $m \in \{0, 1, \ldots, n-2\}$. Using PREFIXTABLE and SUFFIXTABLE, design an algorithm SPANNING(m, k, S) that returns the number of substrings $S[i..j]$ of S that have exactly k ones and such that $i \leq m < j$.

For example, if $S =$ "0110", $k = 2$, and $m = 0$, there exist exactly two such strings: "011" and "0110". Hence, SPANNING$(m, k, S) = 2$.

prefixtable: [1,1,2,0,0]
Suffixtable: [1,1,2,0,0]

Describe the algorithm using pseudocode. Mention and justify the runtime of your algorithm (you don't need to provide a formal proof, but you should state your reasoning).

Hint: *Each substring $S[i..j]$ with $i \leq m < j$ can be obtained by concatenating a string $S[i..m]$ that is a suffix of $S[0..m]$ and a string $S[m+1..j]$ that is a prefix of $S[m+1..n-1]$.*

We are looking for #substrings containing m that have exactly k 1s.
for that we take suffix table of S[0...m] and prefix S[m+1...n-1] values
we multiply the entries of both tables, whenever the indeces add up to k.
this works as we need a total of k. The ideces indicate how many 1's are in the corresponding suffix/prefix. So they need to add up to k. We multiply the values as if there are 3 ways of getting the suffix of x 1's and there are 2 ways of getting a prefix of k-x 1's for each of the 3 ways there are 2 ways:

g) *Computing Powers:*

Provide an algorithm that takes two positive integers a and n as input, and outputs a^n.

You can use addition, subtraction, multiplication and division of integers as basic operations without further explanation. You do not need to proof correctness or analyse the running time of your algorithm. However, for full points the number of basic operations that your algorithm uses needs to be $\mathcal{O}(\log n)$.

```
int power(int a, int n) {
  if (n == 1) {
    return a
  } else {
    if (n % 2 == 1) {
      x ← Power(a, (n - 1)/2)
      return x · x · a
    } else {
      x ← Power(a, n/2)
      return x · x
    }
  }
}
```

c) *Bubblesort*: Bubblesort is implemented as two nested loops. Draw how the following array looks like after the first iteration of bubblesort's outer loop.

19 18 20 9 7 33 1 2 6 5

18 19 9 7 20 1 2 6 5 33

Exercise 5.2 *Sorting algorithms.*

Below you see four sequences of snapshots, each obtained in consecutive steps of the execution of one of the following algorithms: `InsertionSort`, `SelectionSort`, `QuickSort`, `MergeSort`, and `BubbleSort`. For each sequence, write down the corresponding algorithm.

insertion

3	6	5	1	2	4	8	7
3	6	5	1	2	4	8	7
3	5	6	1	2	4	8	7

bubble

3	6	5	1	2	4	8	7
3	5	1	2	4	6	7	8
3	1	2	4	5	6	7	8

merge

3	6	5	1	2	4	8	7
3	6	1	5	2	4	7	8
1	3	5	6	2	4	7	8

selection

3	6	5	1	2	4	8	7
1	6	5	3	2	4	8	7
1	2	5	3	6	4	8	7