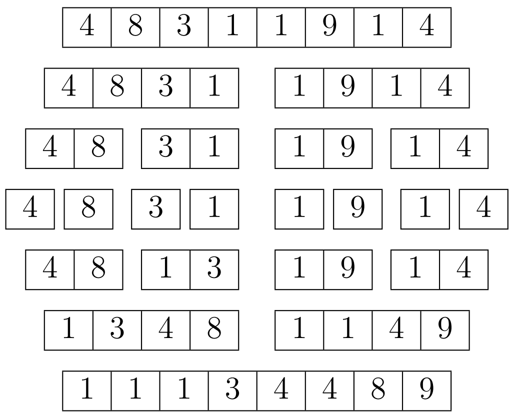
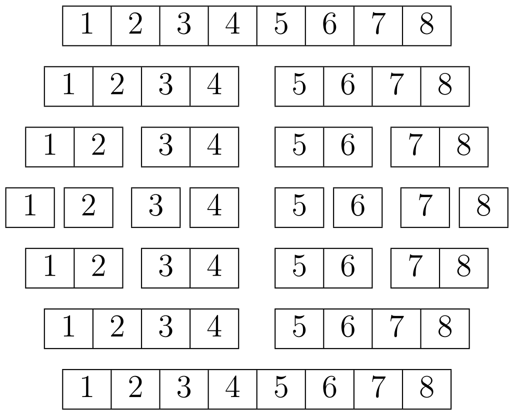
Chapter 2

1. (a)



(b)



3.

$$A = \begin{bmatrix} 1 & 0 & -1 & 2 \\ 3 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 2 & 4 & 7 & 1 \end{bmatrix} \qquad B = \begin{bmatrix} 0 & 1 & 2 & 3 \\ -2 & 1 & -1 & 1 \\ 1 & 0 & 1 & 0 \\ 5 & 1 & -2 & -1 \end{bmatrix}$$

$$S_1 = B_{12} - B_{22} \qquad S_6 = B_{11} + B_{22}$$
$$S_2 = A_{11} + A_{12} \qquad S_7 = A_{12} - A_{22}$$
$$S_3 = A_{21} + A_{22} \qquad S_8 = B_{21} + B_{22}$$
$$S_4 = B_{21} - B_{11} \qquad S_9 = A_{11} - A_{21}$$
$$S_5 = A_{11} + A_{22} \qquad S_{10} = B_{11} + B_{12}$$

$$\begin{aligned}
S_1 &= B_{12} - B_{22} \\
&= \begin{bmatrix} 2 & 3 \\ -1 & 1 \end{bmatrix} - \begin{bmatrix} 1 & 0 \\ -2 & -1 \end{bmatrix} \\
&= \begin{bmatrix} 1 & 0 \\ -2 & -1 \end{bmatrix} \\
S_2 &= A_{11} + A_{12} \\
&= \begin{bmatrix} 1 & 0 \\ 3 & 1 \end{bmatrix} - \begin{bmatrix} -1 & 2 \\ 1 & 1 \end{bmatrix} \\
&= \begin{bmatrix} 0 & 2 \\ 4 & 2 \end{bmatrix} \\
S_3 &= A_{21} + A_{22} \\
&= \begin{bmatrix} 0 & 0 \\ 2 & 4 \end{bmatrix} + \begin{bmatrix} 0 & 1 \\ 7 & 1 \end{bmatrix} \\
&= \begin{bmatrix} 0 & 1 \\ 9 & 5 \end{bmatrix} \\
S_4 &= B_{21} - B_{11} \\
&= \begin{bmatrix} 1 & 0 \\ 5 & 1 \end{bmatrix} - \begin{bmatrix} 0 & 1 \\ -2 & 1 \end{bmatrix} \\
&= \begin{bmatrix} 1 & -1 \\ 7 & 0 \end{bmatrix} \\
S_5 &= A_{11} + A_{22} \\
&= \begin{bmatrix} 1 & 0 \\ 3 & 1 \end{bmatrix} + \begin{bmatrix} 0 & 1 \\ 7 & 1 \end{bmatrix} \\
&= \begin{bmatrix} 1 & 1 \\ 10 & 2 \end{bmatrix} \\
S_6 &= B_{11} + B_{22} \\
&= \begin{bmatrix} 0 & 1 \\ -2 & 1 \end{bmatrix} + \begin{bmatrix} 1 & 0 \\ -2 & -1 \end{bmatrix} \\
&= \begin{bmatrix} 1 & 1 \\ -4 & 0 \end{bmatrix} \\
S_7 &= A_{12} - A_{22} \\
&= \begin{bmatrix} -1 & 2 \\ 1 & 1 \end{bmatrix} - \begin{bmatrix} 0 & 1 \\ 7 & 1 \end{bmatrix} \\
&= \begin{bmatrix} -1 & 1 \\ -6 & 0 \end{bmatrix}
\end{aligned}$$

$$\begin{aligned}
S_8 &= B_{21} + B_{22} \\
&= \begin{bmatrix} 1 & 0 \\ 5 & 1 \end{bmatrix} + \begin{bmatrix} 1 & 0 \\ -2 & -1 \end{bmatrix} \\
&= \begin{bmatrix} 2 & 0 \\ 3 & 0 \end{bmatrix} \\
S_9 &= A_{11} - A_{21} \\
&= \begin{bmatrix} 1 & 0 \\ 3 & 1 \end{bmatrix} - \begin{bmatrix} 0 & 0 \\ 2 & 4 \end{bmatrix} \\
&= \begin{bmatrix} 1 & 0 \\ 1 & -3 \end{bmatrix} \\
S_{10} &= B_{11} + B_{12} \\
&= \begin{bmatrix} 0 & 1 \\ -2 & 1 \end{bmatrix} + \begin{bmatrix} 2 & 3 \\ -1 & 1 \end{bmatrix} \\
&= \begin{bmatrix} 2 & 4 \\ -3 & 2 \end{bmatrix}
\end{aligned}$$

$$\begin{aligned}
P_1 &= A_{11}S_1 & P_5 &= S_5S_6 \\
P_2 &= S_2B_{22} & P_6 &= S_7S_8 \\
P_3 &= S_3B_{11} & P_7 &= S_9S_{10} \\
P_4 &= A_{22}S_4
\end{aligned}$$

$$\begin{aligned}
P_1 &= A_{11}S_1 \\
&= \begin{bmatrix} 1 & 0 \\ 3 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ -2 & -1 \end{bmatrix} \\
&= \begin{bmatrix} 1 & 3 \\ 4 & 11 \end{bmatrix} \\
P_2 &= S_2B_{22} \\
&= \begin{bmatrix} 0 & 2 \\ 4 & 2 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ -2 & -1 \end{bmatrix} \\
&= \begin{bmatrix} -4 & -2 \\ 0 & -2 \end{bmatrix}
\end{aligned}$$

$$\begin{aligned}
P_3 &= S_3 B_{11} \\
&= \begin{bmatrix} 0 & 1 \\ 9 & 5 \end{bmatrix} \begin{bmatrix} 0 & 1 \\ -2 & 1 \end{bmatrix} \\
&= \begin{bmatrix} -2 & 1 \\ -10 & 14 \end{bmatrix} \\
P_4 &= A_{22} S_4 \\
&= \begin{bmatrix} 0 & 1 \\ 7 & 1 \end{bmatrix} \begin{bmatrix} 1 & -1 \\ 7 & 0 \end{bmatrix} \\
&= \begin{bmatrix} 7 & 0 \\ 14 & -7 \end{bmatrix} \\
P_5 &= S_5 S_6 \\
&= \begin{bmatrix} 1 & 1 \\ 10 & 2 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ -4 & 0 \end{bmatrix} \\
&= \begin{bmatrix} -3 & 1 \\ 2 & 10 \end{bmatrix} \\
P_6 &= S_7 S_8 \\
&= \begin{bmatrix} -1 & 1 \\ -6 & 0 \end{bmatrix} \begin{bmatrix} 2 & 0 \\ 3 & 0 \end{bmatrix} \\
&= \begin{bmatrix} 1 & 0 \\ -12 & 0 \end{bmatrix} \\
P_7 &= S_9 S_{10} \\
&= \begin{bmatrix} 1 & 0 \\ 1 & -3 \end{bmatrix} \begin{bmatrix} 2 & 4 \\ -3 & 2 \end{bmatrix} \\
&= \begin{bmatrix} 2 & 4 \\ 11 & -2 \end{bmatrix}
\end{aligned}$$

$$\begin{aligned}
C_{11} &= P_5 + P_4 - P_2 + P_6 \\
&= \begin{bmatrix} -3 & 1 \\ 2 & 10 \end{bmatrix} + \begin{bmatrix} 7 & 0 \\ 14 & -7 \end{bmatrix} - \begin{bmatrix} -4 & -2 \\ 0 & -2 \end{bmatrix} + \begin{bmatrix} 1 & 0 \\ -12 & 0 \end{bmatrix} \\
&= \begin{bmatrix} 9 & 3 \\ 4 & 5 \end{bmatrix} \\
C_{12} &= P_1 + P_2 \\
&= \begin{bmatrix} 1 & 3 \\ 4 & 11 \end{bmatrix} + \begin{bmatrix} -4 & -2 \\ 0 & -2 \end{bmatrix} \\
&= \begin{bmatrix} -3 & 1 \\ 4 & 9 \end{bmatrix}
\end{aligned}$$

$$\begin{aligned}
C_{21} &= P_3 + P_4 \\
&= \begin{bmatrix} -2 & 1 \\ -10 & 14 \end{bmatrix} + \begin{bmatrix} 7 & 0 \\ 14 & -7 \end{bmatrix} \\
&= \begin{bmatrix} 5 & 1 \\ 4 & 7 \end{bmatrix} \\
C_{22} &= P_5 + P_1 - P_3 - P_7 \\
&= \begin{bmatrix} -3 & 1 \\ 2 & 10 \end{bmatrix} + \begin{bmatrix} 1 & 3 \\ 4 & 11 \end{bmatrix} - \begin{bmatrix} -2 & 1 \\ -10 & 14 \end{bmatrix} - \begin{bmatrix} 2 & 4 \\ 11 & -2 \end{bmatrix} \\
&= \begin{bmatrix} -2 & -1 \\ 5 & 9 \end{bmatrix}
\end{aligned}$$

$$C = \begin{bmatrix} 9 & 3 & -3 & 1 \\ 4 & 5 & 4 & 9 \\ 5 & 1 & -2 & -1 \\ 4 & 7 & 5 & 9 \end{bmatrix}$$

You can easily check that

$$AB = \begin{bmatrix} 1 & 0 & -1 & 2 \\ 3 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 2 & 4 & 7 & 1 \end{bmatrix} \begin{bmatrix} 0 & 1 & 2 & 3 \\ -2 & 1 & -1 & 1 \\ 1 & 0 & 1 & 0 \\ 5 & 1 & -2 & -1 \end{bmatrix} = \begin{bmatrix} 9 & 3 & -3 & 1 \\ 4 & 5 & 4 & 9 \\ 5 & 1 & -2 & -1 \\ 4 & 7 & 5 & 9 \end{bmatrix}.$$

In this trace, we had to compute seven (2×2)-matrix products. Depending on how we implement Strassen, this could be the base case of the recursive algorithm. If not, we would have to reapply Strassen to compute each of these seven (2×2)-matrix products. Each of these seven (2×2)-matrix products would then be broken down into seven (1×1)-matrix products.

5. (a) We have $a = 2$, $b = 5$ and $d = 1/2$. Since

$$5^{\log_b a} = 5^{\log_5 2} = 2 < \sqrt{5} = 5^d,$$

it follows that $\log_b a < d$, from which we have $T(n) = O(n^d) = O(\sqrt{n})$ by the Master Theorem.

(b) We have $a = 2$, $b = 10/7$ and $d = 1$. Since $10/7 < 2$, it follows that

$$\log_b a = \log_{10/7} 2 > 1 = d,$$

from which we have $T(n) = O(n^{\log_b a}) = O(n^{\log_{10/7} 2})$ by the Master Theorem.

(c) We have $a = 2017$, $b = 2017$ and $d = 1$. Since $\log_b a = d$, we have $T(n) = O(n^d \log n) = O(n \log n)$ by the Master Theorem.

(d) We have $a = 8$, $b = 4$ and $d = 3/2$. Since

$$b^d = 4^{3/2} = 2^3 = 8 = a,$$

it follows that $\log_b a = d$, from which we have $T(n) = O(n^d \log n) = O(n^{3/2} \log n)$ by the Master Theorem.

7. If n was a power of two, say $n = 2^k$ for an integer $k \geq 0$, this would be a little easier. Indeed, every time we split the list into two, we get two sublists having the exact same size. Then, we do one comparison to know which sublist to choose. We recurse until we get to a one-element list. Then, we do one last comparison to check if this is the element we are looking for. This makes a total of $\log_2(n) + 1 = k + 1$ comparisons.

   The reasoning is similar when n is not a power of two, except that the two sublists we obtain do not necessarily have the same size. Indeed, we eventually get to a sublist having a size which is an odd number m. Then, when we split this sublist into two, this time we get one sublist of size $\lfloor m/2 \rfloor$ and one sublist of size $\lceil m/2 \rceil$. The worst case happens when the element we are looking for is always in the longer sublist.

   Starting with $n = 10^9$, we successively get to search in a sublist of size $5 \cdot 10^8$, $5^2 \cdot 10^7$, $5^3 \cdot 10^6$, $5^4 \cdot 10^5$, $5^5 \cdot 10^4$, $5^6 \cdot 10^3$, $5^7 \cdot 10^2$, $5^8 \cdot 10^1$, 5^9, 976563, 488282, 244141, 122071, 61036, 30518, 15259, 7630, 3815, 1908, 954, 477, 239, 120, 60, 30, 15, 8, 4, 2 and finally 1. This gives a total of 31 comparisons.

9. Splitting the input into three lists is quite analogous to the standard version: we just use a three-way merge procedure instead. Here is how the algorithm would look like:

```
Input: a list $L = a_0, a_1, \ldots, a_{n-1}$ of $n$ items
Output: a list $M$ containing the items in $L$ in sorted order
  if $n < 2$ then
    return $L$
  end if
  Recurse on $a_0, \ldots, a_{\lfloor n/3 \rfloor - 1}$, obtaining a sorted list $L_0$.
  Recurse on $a_{\lfloor n/3 \rfloor}, \ldots, a_{2\lfloor n/3 \rfloor - 1}$, obtaining a sorted list $L_1$.
  Recurse on $a_{2\lfloor n/3 \rfloor}, \ldots, a_{n-1}$, obtaining a sorted list $L_2$.
  Initialize $M$ as the empty list.
  while $L_0$, $L_1$ or $L_2$ is not empty do
    Take $i \in \{0, 1, 2\}$ such that $L_i$ is non-empty and the first item in $L_i$ is minimum.
    Remove the first element from $L_i$ and append it to $M$.
  end while
  return $M$
```

Outside the while loop and apart from the recursive calls, the algorithm does a constant amount of work. Since, in each iteration of the while loop, one list is depleted of one element, the loop has exactly n iterations and each such iteration takes constant time. Thus, if we ignore the fact that the split may not be perfectly even, a recurrence for the running time of this algorithm is

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

By the master theorem, this recurrence satisfies $T(n) = O(n \log n)$. Could you solve it by unfolding?

11. (a) Split the coins into three groups of three coins. By weighing two of these groups against each other, we can determine which group contains the heavier coin: if they both weigh the same, the heavier coin is in the other group; if one is heavier, the heavier coin is there.

    Similarly, we can find the heavier coin among that group by weighing any two coins: if they weigh the same, the other coin is the heavier one; if one is heavier, that is obviously the heavier coin.

    (b) Assume that $n = 3^k$ for some $k \geq 1$. Split the coins into three groups of 3^{k-1} coins. By weighing two of these groups against each other, we can determine which group contains the heavier coin: if they both weigh the same, the heavier coin is in the other group; if one is heavier, the heavier coin is there. Let S be the heaviest group of 3^{k-1} coins.

    Recursively solve the problem on S.

    When we get to a group of three coins, we can find the heavier coin among that group by weighing any two coins: if they weigh the same, the other coin is the heavier one; if one is heavier, that is obviously the heavier coin.

    The recursion for the number of weighings is

    $$T(n) = T(n/3) + 1,$$

    which solves to $T(n) = \log_3(n) = k$.

13. We are given an $n \times n$ array with a missing square such that n is a power of two and we need to tile it with trominos. If $n = 1$, the missing square is the whole array and we simply need to place no trominos.

    If n is larger than 1, we can divide the array into four quadrants. One of them already has a missing square and we can take the square closest to the center of the array from each of the other three quadrants so that each quadrant has exactly one missing square. Now we can recursively tile all quadrants and, conveniently enough, the additional squares we removed form a tromino (refer to Figure 1), so we can just insert one tromino there and then recurse.

15. **Input:** a value x and an $m \times n$ matrix $A[0..m-1][0..n-1]$ whose rows are sorted from left to right and whose columns are sorted from top to bottom

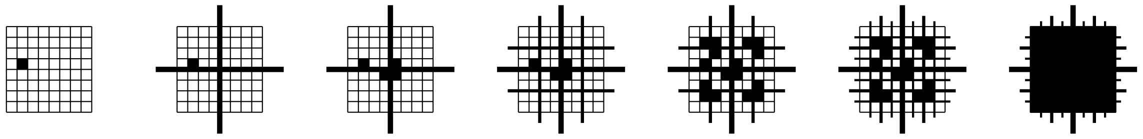


Figure 1: Illustration of the solution to Question 13.

```
Output: indices $i$ and $j$ such that $A[i][j] = x$ or an indication that they do not exist
  $i \leftarrow 0$
  $j \leftarrow n - 1$
  while $i < m$ and $j \geq 0$ do
    if $x < A[i][j]$ then
      $j \leftarrow j - 1$
    else if $x = A[i][j]$ then
      return $(i, j)$
    else
      $i \leftarrow i + 1$
    end if
  end while
  return "not found"
```

In plain English: we start at the top right corner of the matrix. Let y be the current value in the matrix. If $x < y$, we go left by one column. If $x = y$, we return the indices of y. Otherwise, we go down by one row. We repeat this whole process until we find x or until we get out of the matrix, in which case we return "not found".

Each iteration of the loop takes constant time and either increments i or decrements j. We start at $(i, j) = (0, n - 1)$ and in the worst case, the value we are looking for is at $(i, j) = (m - 1, 0)$. Therefore, the algorithm takes $O(m + n)$ time.

17. Assume that n is even to simplify the discussion and let $C[1..(2n)]$ be the array resulting from merging A and B. By definition, the median of C is $C[n]$. Since we can only afford $O(\log(n))$ time, we cannot build C. Let $m_1 = A[n/2]$ and $m_2 = B[n/2]$ be the medians of A and B, respectively (so m_1 and m_2 can be computed in $O(1)$ time).

Assume that $m_1 < m_2$. Then, all elements in $A[1..(n/2 - 1)]$ are smaller than or equal to m_1, which is smaller than m_2, which is smaller than or equal to all elements in $B[(n/2 + 1)..n]$. Therefore, the median of C is in $A[(n/2 + 1)..n]$ or in $B[1..(n/2 - 1)]$. So we recurse on $A[(n/2 + 1)..n]$ and $B[1..(n/2 - 1)]$.

Assume that $m_1 = m_2$. In $C[1..(n - 2)]$, we find all elements from $A[1..(n/2 - 1)]$ and all elements from $B[1..(n/2 - 1)]$. We have $C[n - 1] = C[n] = m_1 = m_2$. Then, in $C[(n + 1)..2n]$, we find all elements from $A[(n/2 + 1)..n]$ and all elements from $B[(n/2 + 1)..n]$. Therefore, the median of C is $m_1 = m_2$. We return m_1.

Assume that $m_1 > m_2$. Then, all elements in $A[(n/2 + 1)..n]$ are larger than or equal to m_1, which is larger than m_2, which is larger than or equal to all elements in

$B[1..(n/2-1)]$. Therefore, the median of C is in $A[1..(n/2-1)]$ or in $B[(n/2+1)..n]$. So we recurse on $A[1..(n/2-1)]$ and $B[(n/2+1)..n]$.

Then, the running time $T(n)$ of our algorithm becomes

$$T(n) = T\left(\frac{n}{2}\right) + 1,$$

which solves to $T(n) = O(\log(n))$ by the Master Theorem.

19. (a) The following recurrence follows directly from the problem statement:

$$T(k) = \begin{cases} 0, & k = 0 \\ 1 + T(k-1), & k > 0 \end{cases}$$

(b) It can be shown by induction that $T(k) = k$.

(c) As $64 = 2^6$, there are $T(6) = 6$ rounds.

21. (a) For $n = 1$, the algorithm makes only a constant amount of work. For larger n, the algorithm divides the input list into two lists of sizes $\lfloor n/2 \rfloor$ and $\lceil n/2 \rceil$, calls itself recursively on them and finally merges them together in linear time. Therefore we get a recurrence of the given format for the running time of the algorithm.

(b) This is a difficult exercise.

For the base case, we have $T(1) = c = c \cdot 1 + 3c \cdot 1 \cdot \log_2(1)$. We also have $T(2) = 2c + c + c = 4c \leq c \cdot 2 + 3c \cdot 2 \cdot \log_2(2)$. Assume that $T(n') \leq c\,n' + 3c\,n' \log(n')$ for all $1 \leq n' < n$. Then we have

$$\begin{aligned}
T(n) &\leq c\,n + T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) \\
&\leq c\,n + \Big(c\lfloor n/2 \rfloor + 3c\lfloor n/2 \rfloor \log_2(\lfloor n/2 \rfloor)\Big) + \Big(c\lceil n/2 \rceil + 3c\lceil n/2 \rceil \log_2(\lceil n/2 \rceil)\Big) \\
&\qquad\qquad \text{by the induction hypothesis,} \\
&\leq c\,n + \Big(c\lfloor n/2 \rfloor + 3c\lfloor n/2 \rfloor \log_2(n/2)\Big) + \Big(c\lceil n/2 \rceil + 3c\lceil n/2 \rceil \log_2(n)\Big) \\
&\qquad\qquad \text{since } \lfloor n/2 \rfloor \leq n/2 \text{ for all } n \geq 1 \\
&\qquad\qquad \text{and } \lceil n/2 \rceil \leq n \text{ for all } n \geq 1, \\
&= c\,n + \Big(c\lfloor n/2 \rfloor + 3c\lfloor n/2 \rfloor(\log_2(n) - \log_2(2))\Big) + \Big(c\lceil n/2 \rceil + 3c\lceil n/2 \rceil \log_2(n)\Big) \\
&= c\,n + \Big(c\lfloor n/2 \rfloor + 3c\lfloor n/2 \rfloor(\log_2(n) - 1)\Big) + \Big(c\lceil n/2 \rceil + 3c\lceil n/2 \rceil \log_2(n)\Big) \\
&= c\Big(n + \lfloor n/2 \rfloor - 3\lfloor n/2 \rfloor + \lceil n/2 \rceil\Big) + 3c\Big(\lfloor n/2 \rfloor + \lceil n/2 \rceil\Big)\log_2(n) \\
&= c\Big(2n - 3\lfloor n/2 \rfloor\Big) + 3c\,n\,\log_2(n) \\
&\qquad\qquad \text{since } \lfloor n/2 \rfloor + \lceil n/2 \rceil = n \text{ for all } n \geq 1, \\
&\leq c\,n + 3c\,n\,\log_2(n) \\
&\qquad\qquad \text{since } 2n - 3\lfloor n/2 \rfloor \leq n \text{ for all } n \geq 2.
\end{aligned}$$

(c) Since $3c$ is a constant and $cn \leq 3cn\log_2(n)$ for all $n \geq 2$, we have

$$T(n) \leq cn + 3cn\log_2(n) = O(n\log_2(n)) = O\left(n\frac{\log(n)}{\log(2)}\right) = O(n\log(n)).$$

You can also show it from the definition of O or using the limit criterion.

23. By the master theorem:

- If $d < 1$, then $T(n) = \Theta(n^{\log_b a}) = \Theta(n)$;
- If $d = 1$, then $T(n) = \Theta(n^d \log n) = \Theta(n\log n)$; and
- If $d > 1$, then $T(n) = \Theta(n^d)$.

Therefore:

(a) $d < 1$

(b) $d \leq 1$

(c) $d \leq 2$