Problem 1

Solution:

- **Subproblem Definition**

  Let's define $f[i]$ as the number of ways to get from (coordinate) 1 to some (coordinate) i.

  We can get to i from i' where $i' = i - j$ and $1 \leq j \leq min(k, i-1)$ by making a step of length j (from i' to $i' + j = i$).

  1. (explanation 1) Then, we have $f[i] = \sum_{j=1}^{min(k,i-1)} f[i-j]$.
     Let's assume that $min(k, i-2) = k$. Then, we have:

     $$f[i] = \sum_{j=1}^{k} f[i-j]$$
     $$f[i-1] = \sum_{j=1}^{k} f[i-j-1]$$

     If we subtract the second equation from the first one, we get :

     $$f[i] - f[i-1] = f[i-1] - f[i-1-k]$$
     $$f[i] = 2f[i-1] - f[i-1-k]$$

  2. (explanation 2) Let's divide all paths from 1 to i into two groups: those that pass through $i-1$ and those that don't. The number of paths from 1 to $i-1$ is $f[i-1]$. We can take each such path and add a step of length 1 to them. Then, each such path will end at coordinate i. Now, let's consider all paths from 1 to i that do not pass through $i-1$. The length of the last step of any of these paths exceeds 1. Let's take the last steps of these paths and decrease their length by 1. Now we have all paths from 1 to $i-1$ such that the last step of any of these paths doesn't exceed $k-1$. The number of such paths is $f[i-1] - f[i-1-k]$, because the number of all paths from 1 to $i-1$, such that the last step has length k, is $f[i-1-k]$. Thus, we have:

     $$f[i] = 2f[i-1] - f[i-1-k]$$

     We can also get the number of paths from 1 to i that do not pass through $i-1$, i.e. f[i - 1] - f[i - 1 - k] by taking all paths from 1 to $i-1$ and extending their last step by 1. Then, we need to subtract all paths that end with a step of length $k+1$.

  As can be seen, we don't need any values $f[i']$ to compute value $f[i]$, such that $i' < i - 1 - k$.

  Therefore, we can optimize the space complexity in the following way:

  $$f[i \bmod (k+2)] = 2f[(i-1) \bmod (k+2)] - f[(i-1-k) \bmod (k+2)]$$

  The answer will be $f[n \bmod (k+2)]$.

- **Recursive Formulation**

$$f[i \bmod (k+2)] = \begin{cases} 1 & \text{if } i = 2 \\ 2f[(i-1) \bmod (k+1)] & \text{else if } i-1-k < 0 \\ 2f[(i-1) \bmod (k+2)] - f[(i-1-k) \bmod (k+2)] & \text{otherwise} \end{cases}$$

- **Pseudocode**

```
procedure compute(n, k)
f[2] = 1
for i = 3 to n:
 f[i mod (k+2)] = 2f[(i-1) mod (k+2)]
 if i-1-k ≥ 0:
  f[i mod (k+2)] -= f[(i-1-k) mod (k+2)]
return f[n mod (k+2)]
```

- **Time and Space Complexity analysis** Our algorithm consists of one loop of size n. The body of the loop consists of a fixed number of operations that work in a constant time. Thus, time complexity is $\mathcal{O}(n)$. As we can see, we only need an array of size $k+2$. Thus, the space complexity is $\mathcal{O}(k)$.

- **Proof of Correctness**
  - Base case: there is only one way to get from 1 to 2. Thus, $f[2] = 1$ is correct.
  - Inductive hypothesis: let's assume that $f[i']$ is the number of ways to get from 1 to i' for all $i' < i$, i.e. $f[i']$ is computed correctly.
  - Inductive step: Let's show that $f[i]$ is the number of ways to get from 1 to i. In order to show this, we can use any of the formulas that we have derived for $f[i]$ (before or after the optimization), as we have shown that they all compute the same values. If we want to use the formulas in the "Recursive Formulation" section, then we can apply the same explanation as in (explanation 2). However, let's use the very first formula, i.e. $f[i] = \sum_{j=1}^{min(k,i-1)} f[i-j]$. We know that we can get to i from i', where $i' = i - j$ and $1 \leq j \leq min(k, i-1)$, by making a step of length j (from i' to $i' + j = i$). According to the inductive hypothesis, the number of ways to get from 1 to such $i - j$ is $f[i-j]$. Thus, we divide all paths from 1 to i by their last step j. Their total number gives the number of ways to get from 1 to i.

Problem 2

Solution:

- **Subproblem Definition** Define $f[i][j]$ as a function which equals 1, if we can represent number j with some numbers from $\{a_1, a_2, ..., a_i\}$, and 0 otherwise.

  The answer to this problem then will be $f[n][k]$.

  In order to find $f[i][j]$, let's look into two cases: when we must use number a_i in the representation of j and when we don't use it.

  If we don't use number a_i, then we must determine whether we can represent j with some numbers from $\{a_1, a_2, ..., a_{i-1}\}$. We can find the answer to this using $f[i-1][j]$.

  If we must use number a_i, then we should represent number $j - a_i$ with some numbers from $\{a_1, a_2, ..., a_{i-1}\}$ (we can use each number only once). We can find the answer to this using $f[i-1][j-a_i]$. Obviously, this case makes sense only if $j - a_i \geq 0$.

- **Recursive Formulation**

$$f[i][j] = \begin{cases} 1 & \text{if } j = 0 \\ 0 & \text{else if } i = 0 \\ f[i-1][j] & \text{if } j - a_i < 0 \\ max(f[i-1][j], f[i-1][j-a_i]) & \text{otherwise} \end{cases}$$

- **Pseudocode**

```
procedure compute(i, j)
if f[i][j] = −1:
 if j = 0:
  f[i][j] = 1
 else if i = 0:
  f[i][j] = 0
 else if j − a_i < 0:
  f[i][j] = compute(i − 1, j)
 else:
  f[i][j] = max(compute(i − 1, j), compute(i − 1, j − a_i))
return f[i][j]
```

- **Time and Space Complexity analysis** There are $\mathcal{O}(nk)$ subproblems. Each subproblem will be computed only once. Finding $f[i][k]$ takes a constant time (assigning a number or computing a maximum of two numbers). Thus, the overall time complexity is $\mathcal{O}(nk)$. As we store values for all subproblems, the space complexity is $\mathcal{O}(nk)$ as well. Note that the space complexity can be optimized to $\mathcal{O}(k)$, but then we would have to use the tabulation method.

- **Proof of Correctness**

  – Base case:

  1. $j = 0$ we can represent number 0 as a sum of an empty set. Thus, $f[i][j]$ for $j = 0$ is computed correctly.
  2. $i = 0$ and $j > 0$, we can't represent a positive number with an empty set. Thus, $f[i][j]$ for $i = 0$ and $j > 0$ is computed correctly.

- Inductive hypothesis: Let's assume that $f[i'][j']$ equals 1, if we can represent number j' with some numbers from $\{a_1, a_2, ..., a_{i'}\}$, and 0 otherwise, for all $i' \leq i$, $j' \leq j$ and $(i', j') \neq (i, j)$.
- Inductive step: Let's show that $f[i][j]$ is computed correctly. As has been mentioned before, we can divide all possibilities into two cases. According to the inductive hypothesis, a case where a_i is included to the representation will be correctly defined by $f[i-1][j-a_i]$ (if $j - a_i \geq 0$). Similarly, according to the inductive hypothesis, a case where a_i is not included to the answer, will be correctly defined by $f[i-1][j]$. Thus, if, at least, one of them equals 1, i.e. if we can represent j with some of the first i numbers, at least, in one case, then $f[i][j]$ will also be equal to 1. If both of them equal 0, then we can't represent j using some of the first i numbers. In this case, $f[i][j]$ will be equal 0.

Problem 3

Solution:

- **Subproblem Definition** Let's define $f[i][j]$ as the optimal answer to the subproblem $A(i,j) = (a_i, a_{i+1}, ..., a_j)$, i.e. the minimum cost to erase all numbers in $A(i,j)$ except for a_i and a_j.

  When we erase numbers in the interval (i,j), some number in the interval will be erased last. Let that number be k. If a_k is the last number to be erased in the interval, then it means we need to erase all numbers in intervals (i,k) and (k,j) first. In this case, the total cost would be $f[i][k] + f[k][j] + a_i * a_j$. The first term is the cost of erasing all the elements in the interval (i,k), the second term is the cost of erasing all numbers in the interval (k,j), and the last term is the cost of deleting a_k. We don't know which case will result in the minimum cost, so we iterate through all possibilities.

- **Recursive Formulation**

$$f[i][j] = \begin{cases} 0 & \text{if } j - i \leq 1 \\ min_{i<k<j}\{f[i][k] + f[k][j] + a_i * a_j\} & \text{if } j - i > 1 \end{cases}$$

- **Pseudocode**

```
procedure compute(i, j)
if f[i][j] = −1:
 if j − i ≤ 1:
  f[i][j] = 0
 else:
  for k = i to j − 1:
   f[i][j] = min(f[i][j], compute(i, k) + compute(k, j) + a_i * a_j)
return f[i][j]
```

- **Time and Space Complexity analysis** There are $\mathcal{O}(n^2)$ subproblems. Each subproblem is calculated only once. In order to find $f[i][j]$ using the values for its subproblems, we need to go through a loop of $\mathcal{O}(n)$ steps. Each step (computing a maximum of a fixed number of values) takes a constant time. Thus, the overall time complexity is $\mathcal{O}(n^3)$. We store the value for each subproblem, thus the space complexity is $\mathcal{O}(n^2)$.

- **Proof of Correctness**

  - Base case: if $j - i <= 1$, then there are no elements to be erased. Thus, the cost is zero. Therefore, $f[i][j]$ for all $j - i \leq 1$ is computed correctly.
  - Let's assume that $f[i'][j']$ is the minimum cost to erase all numbers in the interval (i', j'), where $i \leq i' \leq j' \leq j$ and $(i,j) \neq (i', j')$.
  - Inductive step: As we iterate through all possible values for k and as we know how to erase all numbers in intervals (i,k) and (k,j) optimally (according to the inductive hypothesis), we can find the minimum cost to erase all the numbers in the interval (i,j), by taking the minimum of all possibilities.

Problem 4

Solution:

- **Subproblem Definition** Let's define $f[i]$ as the length of the longest interesting subsequence that ends with element a_i.

  If the length of the longest increasing subsequence that ends with a_i is larger than 1, then there exists an element a_j, such that $j < i$ and $|a_j - a_i| < K$, and a_j is the second last element in the longest interesting subsequence that ends with a_i. We will iterate through all possible elements a_j to find the value for $f[i]$.

- **Recursive Formulation**

$$f[i] = \begin{cases} max_{1 \le j < i}\{f[j] + 1\} & \text{for all } a_j : |a_j - a_i| < K \\ 1 & \text{if no such } a_j \text{ exists} \end{cases}$$

- **Pseudocode**

```
procedure compute(n, K)
ans = 0
for i = 1 to n:
 f[i] = 1
 for j = 1 to i − 1:
  if |a_j − a_i| < K:
   f[i] = max(f[i], f[j] + 1)
 ans = max(ans, f[i])
return ans
```

- **Time and Space Complexity analysis** Both loops work in $\mathcal{O}(n)$ time. The body of the inner loop works in a constant time. Thus, overall time complexity is $\mathcal{O}(n^2)$. We need $\mathcal{O}(n)$ to store values for $f[i]$. Thus, the overall space complexity is $\mathcal{O}(n)$.

- **Proof of Correctness**

  – Base case: when there are no elements a_j, such that $|a_j - a_i| < K$, then $f[i] = 1$, which means we are only taking a_i itself. Thus, $f[1]$ will be computed correctly.

  – Inductive hypothesis: Let's assume that $f[i']$ is the length of the longest interesting subsequence that ends with element $a_{i'}$, for all $i' < i$.

  – Inductive step: if the length of the longest increasing subsequence that ends with a_i is 1, then $f[i]$ will also be equal to 1, which means it is computed correctly in this case. Otherwise, as has been mentioned before, there will be some element a_j, such that $j < i$ and $|a_j - a_i| < K$, and a_j is the second last element in the longest interesting subsequence that ends with a_i. According to the inductive hypothesis, we know the length of the longest interesting subsequence that ends with a_j – $f[j]$. By adding a_i to the end of this subsequence (adding 1 to $f[j]$) and by iterating through all possible values for a_j, we will find the optimal value for $f[i]$ as well.

Problem 5

Solution:

- **Subproblem Definition** Let's define $f[i][j]$ as the optimal answer (the sum of the optimal sequence), if we only have to use the first i columns and if the last element (d_i) in our sequence equals $A[j][i]$ (the jth element in the ith column). As we can't take any neighboring elements in terms of columns in the matrix that are also in the same row, the previous element in the sequence must be in $\{A[1][i-1], ..., A[j-1][i-1], A[j+1][i-1], ..., A[k][i-1]\}$. We can iterate through all possible values for the previous element in the sequence and take the one which gives the best result:

$$f[i][j] = max_{1 \le x \le k \text{ and } x \ne j}\{f[i-1][x]\} + A[j][i]$$
$$f[i][j] = A[j][i], \text{ if } i = 1$$

Let's rewrite the equation in the following way:

$$f[i][j] = max(max_{1 \le x < j}\{f[i-1][x]\}, max_{j < x \le k}\{f[i-1][x]\}) + A[j][i]$$

Let's fix column i. And let's define:

$$left[j] = max_{1 \le x \le j}\{f[i-1][x]\}$$
$$right[j] = max_{j \le x \le k}\{f[i-1][x]\}$$

Now, we have:

$$f[i][j] = max(left[j-1], right[j+1]) + A[j][i]$$

These two arrays are computed for each column separately (when we are about to find the values for the column). They can be computed in $\mathcal{O}(k)$ time. Then, with their help we can compute $f[i][j]$ for all $1 \le j \le k$ in $\mathcal{O}(k)$ time.

Finally, as can be seen, we only use values $f[i-1][x]$ to find values for $f[i][j]$. So we don't need to store values for $f[y][x]$, where $y < i - 1$.

At last, we have:

$$left[j] = max(left[j-1], f[(i-1) \bmod 2][j]))$$
$$right[j] = max(right[j+1], f[(i-1) \bmod 2][j])$$
$$f[i \bmod 2][j] = max(left[j-1], right[j+1]) + A[j][i]$$

- **Recursive Formulation** $left$, $right$ are computed for each $i > 1$ separately.

$$left[j] = \begin{cases} -\infty & \text{if } j = 0 \\ max(left[j-1], f[(i-1) \bmod 2][j]) & \text{otherwise} \end{cases}$$

$$right[j] = \begin{cases} -\infty & \text{if } j = k+1 \\ max(right[j+1], f[(i-1) \bmod 2][j]) & \text{otherwise} \end{cases}$$

$$f[i \bmod 2][j] = \begin{cases} A[j][i] & \text{if } i = 1 \\ max(left[j-1], right[j+1]) + A[j][i] & \text{otherwise} \end{cases}$$

- **Pseudocode**

```
procedure compute(n, k)
for i = 1 to n:
 left[0] = -∞
 for j = 1 to k:
  left[j] = max(f[(i − 1) mod 2][j], left[j − 1])

 right[k + 1] = −∞
 for j = k to 1:
  right[j] = max(f[(i − 1) mod 2][j], right[j + 1])

 for j = 1 to k:
  f[i mod 2][j] = −∞
  if i > 1:
   f[i mod 2][j] = max(f[i mod 2][j], max(left[j − 1], right[j + 1]) + A[j][i])
  else:
   f[i mod 2][j] = A[j][i]

ans = -∞
for j = 1 to k:
 ans = max(ans, f[n mod 2][j])
return ans
```

- **Time and Space Complexity analysis** There are $\mathcal{O}(n)$ steps in the outer loop. Each of the three inner loops run in $\mathcal{O}(k)$ time. The time complexity of the last loop, where we find the answer, is also $\mathcal{O}(k)$. Thus, the overall time complexity is $\mathcal{O}(nk)$.

  As can be seen, we only need $\mathcal{O}(k)$ space for arrays $left$ and $right$. We also need only two columns of k elements each for $f[i][j]$. Thus, the overall space complexity is $\mathcal{O}(k)$.

- **Proof of Correctness** Similar to question 1, we can prove the correctness of any of the obtained formulas, as they all compute the same values. For simplicity, let's use:

$$f[i][j] = max_{1 \le x \le k \text{ and } x \ne j}\{f[i-1][x]\} + A[j][i]$$

  - Base case: when $i = 1$, we can only pick one number, as we have only one column. Thus, $f[i][j] = A[j][i]$ will be the optimal answer, if the sequence has to end with $A[j][i]$.
  - Inductive hypothesis: Let's assume that for all $1 \le j \le k$ $f[i-1][j]$ is the sum of the optimal sequence if we use the first $i-1$ columns and the last element in the sequence, i.e. d_{i-1}, is $A[j][i-1]$.
  - Inductive step: Let's show that $f[i][j]$ is computed correctly. As has been mentioned before, the second last element in the optimal sequence, when we use the first i columns, must belong

to $\{A[1][i-1], ..., A[j-1][i-1], A[j+1][i-1], ..., A[k][i-1]\}$. According to the inductive hypothesis, we know the sums of optimal sequences that end with all the elements from the mentioned set. Their values are $f[i-1][1], ..., f[i-1][j-1], f[i-1][j+1], ..., f[i-1][k]$ respectively. As they are all optimal, finding their maximum and adding the value of $A[j][i]$ to the result will give us the sum of the optimal sequence. Then, when we find all values for $f[i][j]$, we take the maximum of $f[n][1], f[n][2], ..., f[n][k]$. This is correct, as the optimal sequence must end with one of the elements in the last column.