Question 1: Skip A Level

Alex has to complete a multi-level game. Each level has an entry fee that needs to be paid before starting the level. After each level, Alex receives a point. Alex has to play levels in the given order and can skip at most one level.

Given the initial amount in Alex's wallet *k*, the number of levels in the game, *n* and the cost of each level, *costs*. Find the maximum points Alex can collect.

Note: It is not compulsory to complete all the levels

**Example**

*k = 14*

*n = 5*

*costs = [2, 4,1, 8, 6]*

Completing 5 levels without skipping any level, entry fees = 2 + 4 + 1 + 8 + 6 = 21 > *k*

Completing 5 levels while skipping the 4th level, entry fees = 2 + 4 + 1 + 6 = 13 *≤ k,* points collected = 4, as levels 1, 2, 3 and 5 were completed.

It can be proven that you cannot collect more than 4 points. Hence the answer is 4.

**Function Description**

Complete the function *maximumPoints* in the editor below.

*maximumPoints* has the following parameter(s):

*int k:* the initial number of coins in Alex's wallet

*int costs[n]:*  the costs of each level

**Returns**

*int:* the maximum number of points Alex can collect after skipping at most one level

**Constraints**

* *1 ≤ k ≤ 109*
* *1 ≤ n ≤ 105*
* *1 ≤ costs[i] ≤ 109*

**Input Format For Custom Testing**

The first line contains an integer, *k*, the initial number of coins in Alex's wallet.

The second line contains an integer, *n*, the size of the array *costs*.

Each line *i* of the *n* subsequent lines (where 1 *≤ i ≤ n*) contains an integer that describes costs*[i]*.

**Sample Case 0**

**Sample Input For Custom Testing**

STDIN    FUNCTION

-----   --------

10   → k = 10

5 → n = 5

5 → costs = [5, 2, 3, 1, 4]

2

3

1

4

10

5

5

2

3

1

4

**Sample Output**

4

**Explanation**

Completing 5 levels without skipping any level, entry fees = 5 + 2 + 3 + 1 + 4 = 15 > *k*

Completing 5 levels and skipping the 4th level, entry fees = 5 + 2 + 3 + 4 = 14 > *k*

Completing 5 levels and skipping the 4th level, entry fees = 2 + 3 + 1 + 4 = 10 *≤ k,* points collected = 4, as levels 2, 3, 4 and 5 were completed.

It can be proven that you cannot collect more than 4 points. Hence the answer is 4.

**Sample Case 1**

**Sample Input For Custom Testing**

STDIN    FUNCTION

-----   --------

15   → k = 15

6 → n = 6

3 → costs = [3, 2, 6, 4, 6, 1]

2

6

4

6

1

**Sample Output**

4

**Explanation**

Completing 6 levels without skipping any level, entry fees = 3 + 2 + 6 + 4 + 6 + 1 = 22 > *k*

Completing 5 levels and skipping the 3rd level, entry fees = 3 + 2 + 4 + 6 = 15 *≤ k,* points collected = 4, as levels 1, 2, 4 and 5 were completed.

It can be proven that you cannot collect more than 4 points. Hence the answer is 4.

Question 2: Buying Show Tickets

Alex plans on visiting the museum and is at the counter to purchase tickets to get in. Tickets are sold only one at a time. If a visitor needs more than one ticket, he/she has to pass through the queue again to buy each one.

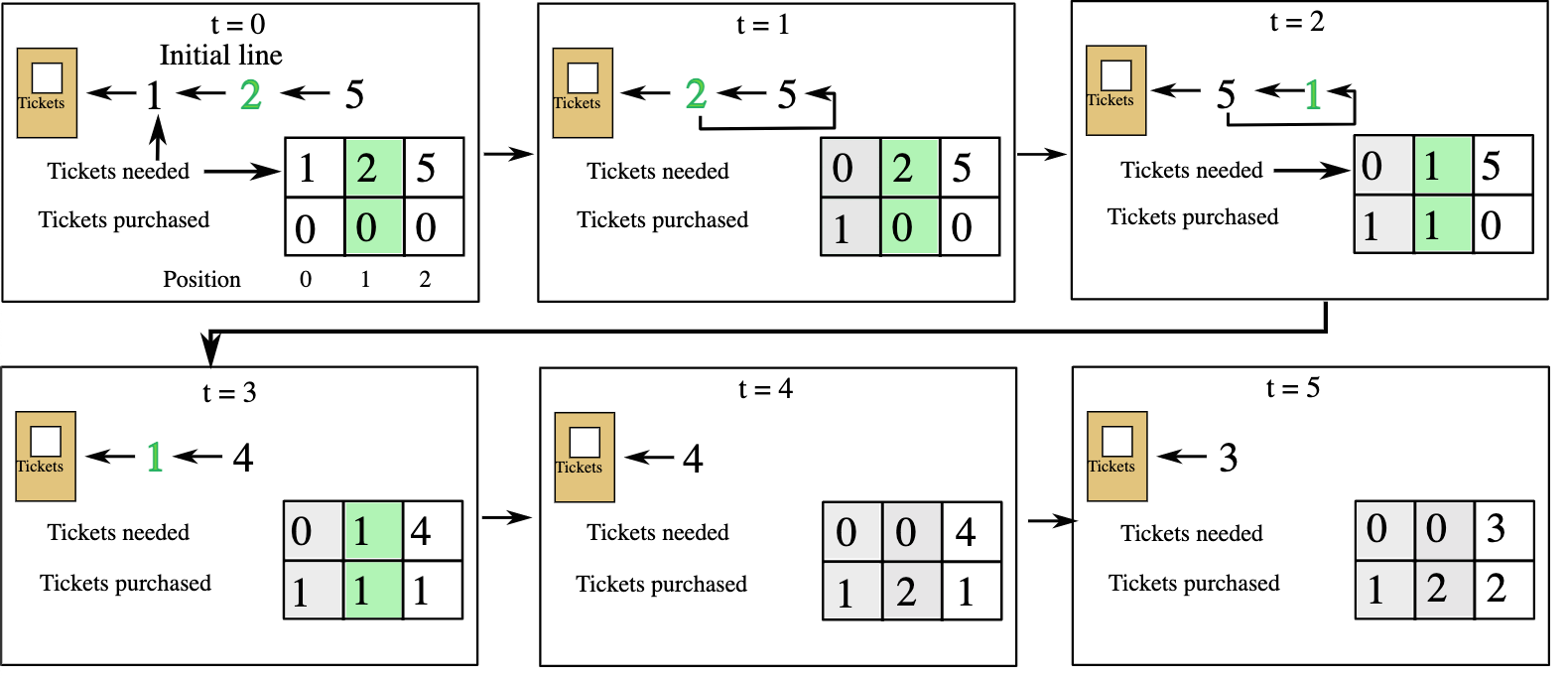
Given the number of visitors and the number of tickets each visitor needs, determine the amount of time Alex requires to buy the number needed. Alex's place in the queue will be given. Each transaction takes 1 unit of time. and the time taken to go to the back of the line can be ignored.

**Example**

*n = 3*

*tickets = [1, 2, 5]*

There are 3 buyers needing 1, 2 and 5 tickets each. Alex is at index *1* and need *2* tickets. The first six units of time, *t = 0* through *t = 5,* are as shown below:



Alex's information is green both in the line to the ticket booth and in the boxes at the bottom.

When a person has purchased all the tickets they need, they leave the line and their boxes are colored gray.  
Alex finishes purchasing *tickets[1] = 2* tickets at the *4th* transaction.

**Function Description**

Complete the function *waitingTime* in the editor below. The function must return an integer representing the units of time it takes Alex to purchase the desired number of tickets.

waitingTime has the following parameter(s):

    int *tickets [n]:*  a zero-indexed array of tickets desired by the person *i* at position *tickets[i]*

    int *p*: Alex's position in line

**Constraints**

* *1 ≤ n ≤ 105*
* *1 ≤ tickets[i] ≤ 109*, where *0 ≤ i < n*.
* *0 ≤ p < n*

**Input Format for Custom Testing**

Input from stdin will be processed as follows and passed to the function.

The first line contains an integer *n*, the size of the array *tickets*.

The next *n* lines each contain an element *tickets[i]* where 0 ≤ i < n.

The next line contains an integer *p*, Alex's position in line.

**Sample Case 0**

**Sample Input 0**

STDIN     Function

-----     --------

5    ⇒    tickets[] size n = 5

2    ⇒    tickets = [2, 6, 3, 4, 5]

6

3

4

5

2    ⇒    p = 2

**Sample Output 0**

12

**Explanation 0**

Given *tickets = [2, 6,* ***3****, 4, 5]*, Alex's position in line is marked in bold. The wait time looks like this:

1. *window ← 2 ← 6 ←****3****← 4 ← 5*
2. *window ← 6 ←****3****← 4 ← 5 ← 1*
3. *window ←****3****← 4 ← 5 ← 1 ← 5*
4. *window ← 4 ← 5 ← 1 ← 5 ←****2***
5. *window ← 5 ← 1 ← 5 ←****2****← 3*
6. *window ← 1 ← 5 ←****2****← 3 ← 4*
7. *window ← 5 ←****2****← 3 ← 4* (the person at the head of the line in the previous step purchased their last ticket and does not re-enter the line)
8. *window ←****2****← 3 ← 4 ← 4*
9. *window ← 3 ← 4 ← 4* ← **1**
10. *window ← 4 ← 4* ← **1** ← 2
11. *window ← 4* ← **1** ← 2 ← 3
12. *window ←****1****← 2 ← 3 ← 3*
13. *window ← 2 ← 3 ← 3* ( Alex purchased the last ticket needed and does not re-enter the line)

It took a total of *12* units of time *(t = 0* through *t = 11)* to purchase *2* tickets.

**Sample Case 1**

**Sample Input 1**

STDIN     Function

-----     --------

4    ⇒    tickets[] size n = 4

1    ⇒    tickets = [1, 1, 1, 1]

1

1

1

0    ⇒    p = 0

**Sample Output 1**

1

**Explanation 1**

Given *tickets = [****1****, 1, 1, 1]*, Alex's wait time looks like this:

1. *window ←****1****← 1 ← 1 ← 1*
2. *window ← 1 ← 1 ← 1* (Alex purchased the necessary ticket and did not re-enter the line)

It took a total of *1* unit of time to purchase *1* ticket.

Sample Case 2

**Sample Input 2**

STDIN     Function

-----     --------

4    ⇒    tickets[] size n = 4

5    ⇒    tickets = [5, 5, 2, 3]

5

2

3

3    ⇒    p = 3

**Sample Output 2**

11

**Explanation 2**

Given *tickets = [5, 5, 2,* ***3****]*, Alex's wait time looks like this:

1. *window ← 5 ← 5 ← 2 ←****3***
2. *window ← 5 ← 2 ←****3****← 4*
3. *window ← 2 ←****3****← 4 ← 4*
4. *window ←****3****← 4 ← 4 ← 1*
5. *window ← 4 ← 4 ← 1 ←****2***
6. *window ← 4 ← 1 ←****2****← 3*
7. *window ← 1 ←****2****← 3 ← 3*
8. *window ←****2****← 3 ← 3* (the person at the head of the line in the previous step purchased their last ticket and did not re-enter the line)
9. *window ← 3 ← 3 ←****1***
10. *window ← 3 ←****1****← 2*
11. *window ←****1****← 2 ← 2*
12. *window ← 2 ← 2* (Alex purchased the last ticket needed and did not re-enter the line)

It took a total of *11* units of time to purchase *3* tickets.

Question 3 : Profit Targets

A financial analyst is responsible for a portfolio of profitable stocks represented in an array. Each item in the array represents the yearly profit of a corresponding stock. The analyst gathers all distinct pairs of stocks that reached the target profit. Distinct pairs are pairs that differ in at least one element. Given the array of profits, find the number of distinct pairs of stocks where the sum of each pair's profits is exactly equal to the target profit.

**Example**

*stocksProfit = [5, 7, 9, 13, 11, 6, 6, 3, 3]*

*target = 12 profit's target*

* There are *4* pairs of stocks that have the sum of their profits equals to the target 12 . Note that because there are two instances of *3* in *stocksProfit*there are two pairs matching (9, 3): *stocksProfits* indices 2 and 7, and indices 2 and 8, but only one can be included.
* There are *3* distinct pairs of stocks: *(5, 7), (3, 9),*and*(6, 6)*and the return value is*3.*

**Function Description**

Complete the function *stockPairs* in the editor below.

*stockPairs* has the following parameter(s):

*int stocksProfit[n]:* an array of integers representing the stocks profits

*target:* an integer representing the yearly target profit

Returns:

*int*: the total number of pairs determined

**Constraints**

* *1 ≤ n ≤ 5 × 105*
* *0 ≤ stocksProfit[i] ≤ 109*
* *0 ≤* *target ≤ 5 × 109*

**Input Format for Custom Testing**

Input from stdin will be processed as follows and passed to the function.

The first line contains an integer *n*, the size of the array *stocksProfit*.

The next *n* lines each contain an element *stocksProfit[i]* where *0 ≤ i < n.*

The next line contains an integer  *target*, the target value.

**Sample Case 0**

**Sample Input 0**

STDIN   Function

**-----      --------**

6      **→** *stocksProfit*[] size n = 6

1      **→** *stocksProfit* = [1, 3, 46, 1, 3, 9]

3

46

1

3

9

47     **→** target = 47

**Sample Output 0**

1

**Explanation 0**

There are *4* pairs where *stocksProfit[i] + stocksProfit[j] =* 47

1. *(stocksProfit0] = 1, stocksProfit[2] = 46)*
2. *(stocksProfit[2] = 46, stocksProfit[0] = 1)*
3. *(stocksProfit[2] = 46, stocksProfit[3] = 1)*
4. *(stocksProfit[3] = 1, stocksProfit[2] = 46)*

Since all four pairs contain the same values, there is only *1* *distinct* pair of stocks : *(1, 46)*.

**Sample Case 1**

**Sample Input 1**

STDIN      Function

-----      --------

7      →   *stocksProfit*[] size n = 7

6      **→** *stocksProfit* = [6, 6, 3, 9, 3, 5, 1]

6

3

9

3

5

1

12     **→** target = 12

**Sample Output 1**

2

**Explanation 1**

There are *5* pairs where *stocksProfit[i] + stocksProfit[j] =* 12:

1. *(stocksProfit[0] = 6, stocksProfit[1] = 6)*
2. *(stocksProfit[1] = 6, stocksProfit[0] = 6)*
3. *(stocksProfit[2] = 3, stocksProfit[3] = 9)*
4. *(stocksProfit[3] = 9, stocksProfit[2] = 3)*
5. *(stocksProfit[3] = 9, stocksProfit[4] = 3)*
6. *(stocksProfit[4] = 3, stocksProfit[3] = 9)*

The first *2* pairs are the same, as are the last *4.* There are only *2* *distinct* pairs of stocks: *(3, 9)* and *(6, 6)*.

Question 4 : Slot Machine 2.0

A slot machine has multiple wheels that are spun n times. In each spin, each wheel may have multiple stops from 1 to 9 and shows one random number on the machine's dashboard.

Given the number of spins n, determine the minimum number of stops on each wheel to produce the numbers displayed on the dashboard for each spin. Then, calculate the total stops.

**Example**

*n = 4*

*spins[] = [ '712', '246', '365', '312' ]*

the spins on a slot machine with *3* wheels are recorded as an array, *history*:

7 1 2

2 4 6

3 6 5

3 1 2

One wheel needs to have at least *7* stops to produce the numbers displayed on the dashboard for *1st* spin. Since *7* is the highest value in any row, remove the highest value from each of the rows:

1 2

2 4

3 5

1 2

Now the highest value is *5*, so another wheel must have *5* stops to produce the numbers displayed on the dashboard for 3*rd* spin. Using the same logic, the final wheel needs *3* stops. Total stops are *7 + 5 + 3 = 15*.

**Function Description**

Complete the function *slotSpins* in the editor below.

slotWheels has the following parameter(s):

*string spins [n]:*  an array of equal length strings of digits spun

Returns:

*int:* an integer that represents the sum of the minimum number of stops on all of the wheels.

**Constraints:**

* *1 ≤ |spins| ≤ 50*
* *1 ≤ spins[i] ≤ 50*
* All *spins[i]* in a test case will be of equal length.
* All characters in each *spins[i] ∈ [0-9]*

**Input Format for Custom Testing**

Input from stdin will be processed as follows and passed to the function.

The first line contains an integer *n*, the size of the array *history*.

Each of the next *n* lines contains a string *spins[i]* where *0 ≤ i < n*.

**Sample Case 0**

**Sample Input 0**

STDIN    Function Parameters

-----    -------------------

4 →  *spins*[] Size n = 4

137 →  *spins*[] = ['137', '364', '115', '724']

364

115

724

**Sample Output 0**

14

**Explanation 0**

the spins on a slot machine with *4* wheels are recorded as an array, *spins*:

1 3 7

3 6 4

1 1 5

7 2 4

One wheel needs to have at least *7* stops to produce the numbers displayed on the dashboard for *1st* spin and *4th*  spin. Since *7* is the highest value in any row, remove the highest value from each of the rows:

1 3

3 4

1 1

2 4

One wheel needs to have at least *4* stops to produce the numbers displayed on the dashboard for 2*nd* spin and *4th*  spin. Since *4* is the highest value in any row, remove the highest value from each of the rows. Using the same logic, the final wheel needs *3* stops. Total stops are *7 + 4 + 3 = 14*.

**Sample Case 1**

**Sample Input 1**

STDIN    Function Parameters

-----    -------------------

4     →  *spins*[] Size = 4

1112 →  *spins*[] = ['1112', '1111', '1211', '1111']

1111

1211

1111

**Sample Output 1**

5

**Explanation 1**

the spins on a slot machine with *4* wheels are recorded as an array, *spins*:

1 1 1 2

1 1 1 1

1 2 1 1

1 1 1 1

One wheel needs to have at least *2* stops to produce the numbers displayed on the dashboard for *1st* spin and 3*th*  spin. Since *2* is the highest value in any row, remove the highest value from each of the rows:

1 1 1

1 1 1

1 1 1

1 1 1

Using the same logic, 3 wheels need to have 1 stop each. Total stops are 2*+ 1 + 1 + 1 = 5*.

Question 5 : REST Server Response

Which of the following is (are) valid server response formats? (Multiple Choice)

1. XML
2. JSON
3. CSV
4. None of these

Question 6 : Good URI Design

Which of the following rules should be followed to design a good *URI*? (Multiple Choice)

1. URIs should never be changed
2. URIs must be constructed by the client
3. URIs should be short in length
4. URIs should be case-sensitive
5. HTTP verbs should be used instead of operation names in URIs
6. Use spaces when designing a URI
7. Redirection must be used if a change in URI is required

Question 7 :

Given a full binary tree with n internal nodes, how many leaf nodes does it have? (Multiple Choice)

1. n + 1
2. 2n + 1
3. n -1
4. n2

Question 8 :

A Priority-Queue is implemented as a Max-Heap. Initially, it has 5 elements. The level-order traversal of the heap is given below: 10, 8, 5, 3, 2 Two new elements ”1‘ and ”7‘ are inserted in the heap in that order.

The level-order traversal of the heap after the insertion of the elements is: (Multiple Choice)

1. 10, 8, 7, 5, 3, 2, 1
2. 10, 8, 7, 2, 3, 1, 5
3. 10, 8, 7, 1, 2, 3, 5
4. 10, 8, 7, 3, 2, 1, 5