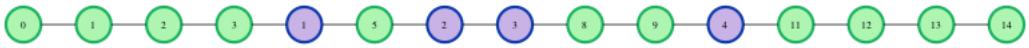
☆ Long Break

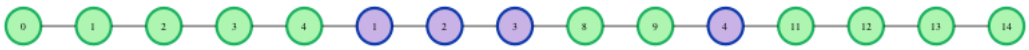
You are organizing an event where there will be a number of presenters. The event starts at time *0* and time be allocated for networking at any time during the event when there is not a presentation being made. The presentations may not overlap as they are in the same room, but this allows them to run consecutively, without breaks. While the order of speeches cannot be changed, there is a maximum number given that indicates how many speeches may be rescheduled. Your goal is to maximize the length of the longest networking period you can arrange.

For example, there are $n = 4$ presenters scheduled for the course of the event which begins at time *0* and ends at time $t = 15$. The meetings start at times $start = [4, 6, 7, 10]$ and end at times $finish = [5, 7, 8, 11]$. You can rearrange up to $k = 2$ meetings. Green cells are free, marked with the hour number, and blue cells have a presentation scheduled, marked with presentation number.

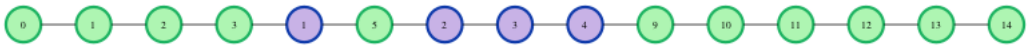


In this case, we have *4* periods without speakers scheduled: *[(0-3), (5), (8-9), (11-14)]*. The meeting ends after hour *14*. If the first meeting is shifted to an hour later, a break is created from *0* to *5*, *5 hours*. If the last speech is moved up to *8*, it will end at *9* leaving a break from *9* to *15*. There is no point to moving the middle two speeches in this case. The longest break that can be achieved is $15 - 9 = 6$ hours by moving the last speech to two hours earlier. The two options are illustrated below.

Option 1:



Option 2:



Function Description

Complete the function *findBreakDuration* in the editor below. The function must return an integer that denotes the length of the maximum break that can be created.

findBreakDuration has the following parameter(s):

n: an integer, the number of presentations
k: an integer, the number of presentations that can be moved
t: an integer, the end time for the event
start[start[0],...start[n-1]]: an array of integers, start times for each event
finish[finish[0],...finish[n-1]]: an array of integers, ending times for each event

Constraints

- $1 \le n \le 10^5$
- $0 \le k \le n$
- $1 \le t \le 10^9$
- $0 \le start[i] < finish[i] \le t$
- $finish[i] \le start[i + 1]$ if $i \ne n - 1$

Input Format For Custom Testing

The first line contains an integer, *n*, the number of presenters and the number of elements in the arrays *start* and *finish*.
The second line contains an integer, *k*, the number of meetings that can be rescheduled.
The next line contains an integer, *t*, the time the event ends.
The following line repeats *n*.
Each of the next *n* lines contains an integer, *start[i]*.
The next line repeats *n*.
Each of the next *n* lines contains an integer, *finish[i]*.

Sample Case 0

```
15
3
0
6
7
3
5
7
8
```

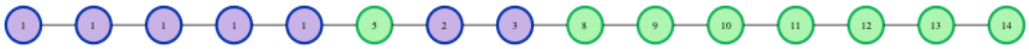
Sample Output 0

```
8
```

Explanation 0

```
n = 3
k = 2
t = 15
start = {0, 6, 7}
finish = {5, 7, 8}
```

An illustration of the original schedule is:



Move the *[6, 7]* to *[5, 6]* and *[7, 8]* to *[6, 7]*. Then *[7, 15]* is a break and its duration is *8:*

YOUR ANSWER

We recommend you take a quick tour of our editor before you proceed. The timer will pause up to 90 seconds for the tour.

Start tour

For help on how to read input and write output in Python 3, click here.

Draft saved 02:59 pm | Original Code | Python 3

```
50          #print("homany=%s start=%s,end=%s"%(self.howmany,self.start_t, self.end_t))
51
52      def __str__(self):
53          return "p%s" % self.howmany
54
55      def __repr__(self):
56          return "p%s(%s-%s)" % (self.howmany, self.start_t, self.end_t)
57
58      def add(self, end_t):
59          assert end_t > self.end_t
60          self.howmany += 1
61          self.end_t = end_t
62          #print("> homany=%s end=%s"%(self.howmany,self.end_t))
63
64  # ============================================================================ #
65
66  class TimeLine(object):
67      first: Element = None
68      last: Element = None
69      first_break: Break = None
70      last_break: Break = None
71
72      k = 0
73
74      total_breaks = 0
75      total_breaks_time = 0
```

Test against custom input

Run Code | Submit code & Continue

(You can submit any number of times)

Download sample test cases *The input/output files have Unix line endings. Do not use Notepad to edit them on windows.*

1 2 3 4 5 6

Compiled successfully. 11/14 test cases passed.

Tip: Debug your code against custom input

Test Case #1:	✔	Test Case #6:	✔	Test Case #11:	⏱
Test Case #2:	✔	Test Case #7:	✔	Test Case #12:	⏱
Test Case #3:	✔	Test Case #8:	✔	Test Case #13:	✔
Test Case #4:	✔	Test Case #9:	⏱	Test Case #14:	✔
Test Case #5:	✔	Test Case #10:	✔		

Testcase 1: *Success*

Input [Download]

```
3
2
15
3
0
6
7
3
5
7
8
```

Your Output

```
8
```

Expected Output [Download]

```
8
```

Testcase 2: *Success*

Input [Download]

```
1
0
1000000000
1
607641287
1
607641288
```

Your Output

```
607641287
```

Expected Output [Download]

```
607641287
```

Testcase 3: *Success*

Your Output

1
2
3
4
5
6

Testcase 4: *Success*

Your Output

```
Output hidden
```

Testcase 5: *Success*

Your Output

```
Output hidden
```

Testcase 6: *Success*

Your Output

```
Output hidden
```

Testcase 7: *Success*

Your Output

```
Output hidden
```

Testcase 8: *Success*

Your Output

```
Output hidden
```

Testcase 9: *Terminated due to timeout*

Your Output

```
Output hidden
```

Testcase 10: *Success*

Your Output

```
Output hidden
```

Testcase 11: *Terminated due to timeout*

Your Output

```
Output hidden
```

Testcase 12: *Terminated due to timeout*

Your Output

```
Output hidden
```

Testcase 13: *Success*

Your Output

```
Output hidden
```

Testcase 14: *Success*

Input [Download]

```
4
4
6
7
10
4
5
7
8
11
```

Your Output

```
6
```

Expected Output [Download]

```
6
```