Experiment No. 2

Title: Implementation of Linear Search

Batch: B-4 **Roll No: 16010422234** **Name: Chandana Ramesh Galgali**

Experiment No.: 2

Aim: To implement a Linear Search algorithm on hackerearth and optimize its solution using suitable data structures.

Resources needed: Web Browser to access Hackerearth platform

Theory:
Competitive Programming involves solving coding problems using data structures and algorithms. Competitive programming helps to improve logical and analytical skills. There are various platforms available for Competitive Programming such as Hackerearth, Codechef, Codeforces etc. On a Competitive Programming platform, one needs to write solutions under various restrictions such as memory limits, execution time, constraints on input size and so on. The Competitive Programming Platform then evaluates the solution against predefined test cases for a problem statement.

Hackerearth is a competitive programming platform which hosts programming challenges and coding competitions. On Hackerearth platform, the problem statement is defined in terms of a real world scenario followed by input format, output format, constraints, sample input, sample output, time limit in seconds and memory limit. Hackerearth supports several programming languages such as C, C++, Java, Python, JavaScript and so on. Any of these supported programming languages can be selected for the implementation of the solution. There is a code editor provided where the code can be written and compiled. When the solution is submitted on the hackerearth platform, it is tested against predefined test cases for the problem statement. And it displays the list of all test cases and the status of the solution against each test case whether the solution has passed that test case or not.

Solving problem on Competitive Programming Platform like hackerearth helps to:

1. Understand problem in terms of real-world scenario
2. Interpret input, output and processing information and constraints from the given real world scenario problem
3. Develop logical and analytical ability for optimization of solution
4. Understand and perform test-case based evaluation of solution

---

Activity:
You have N boxes numbered 1 through N and K candies numbered 1 through K. You put the candies in the boxes in the following order:

first candy in the first box,
second candy in the second box,
....
....
so up to N-th candy in the Nth box,
the next candy in (N - 1)-th box,
the next candy in (N - 2)-th box
....
....
and so on up to the first box,
then the next candy in the second box
....
and so on until there is no candy left.

So you put the candies in the boxes in the following order:

$$1, 2, 3, \ldots, N, N-1, N-2, \ldots, 2, 1, 2, 3, \ldots, N, N-1, \ldots$$

Find the index of the box where you put the K-th candy.

Input format

--The first line contains T denoting the number of test cases. The description of T test cases is as follows:
--Each test case consists of a single line containing two integers N, and K.

Output format

--For each test case, print the index of the box where you put the K-th candy.

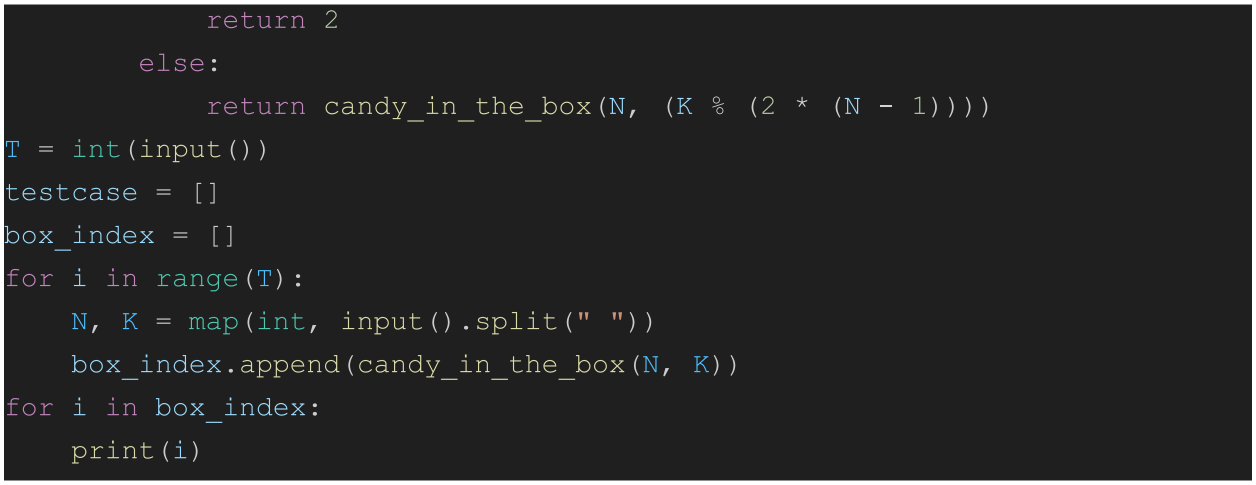
Constraints

$1 \leq T \leq 10^5$
$2 \leq N \leq 10^9$
$1 \leq K \leq 10^9$

https://www.hackerearth.com/practice/algorithms/searching/linear-search/practice-problems/algorithm/candy-in-the-box-75b63839/

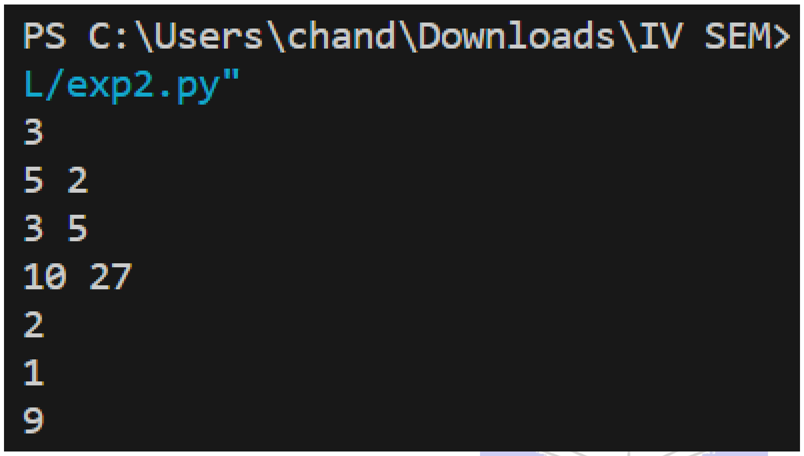
Program:

```
def candy_in_the_box(N, K):
    if N > K:
        return K
    elif 2 * N > K:
        return 2 * N - K
    else:
        if candy_in_the_box(N, (K % (2 * (N - 1)))) == 0:
```



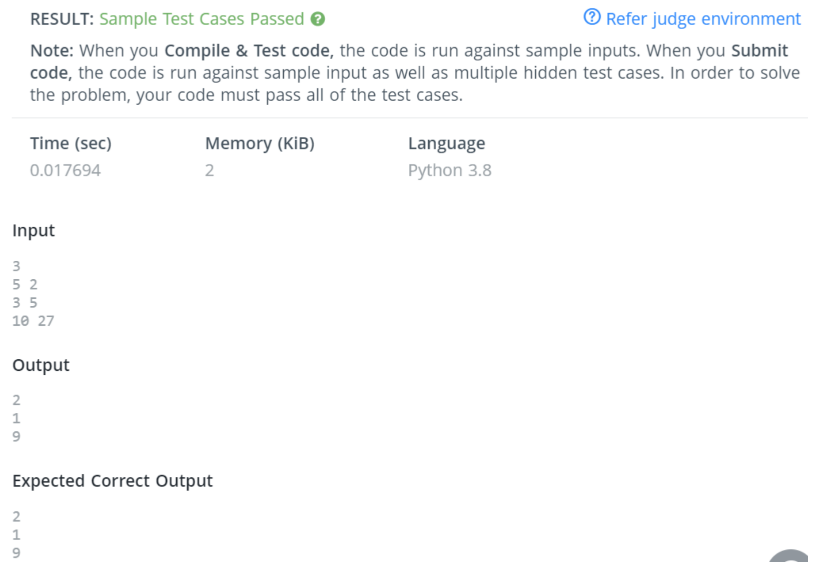
```
            return 2
        else:
            return candy_in_the_box(N, (K % (2 * (N - 1))))
T = int(input())
testcase = []
box_index = []
for i in range(T):
    N, K = map(int, input().split(" "))
    box_index.append(candy_in_the_box(N, K))
for i in box_index:
    print(i)
```

Output:



```
PS C:\Users\chand\Downloads\IV SEM>
L/exp2.py"
3
5 2
3 5
10 27
2
1
9
```

Test Result:



RESULT: Sample Test Cases Passed

Refer judge environment

Note: When you **Compile & Test code**, the code is run against sample inputs. When you **Submit code,** the code is run against sample input as well as multiple hidden test cases. In order to solve the problem, your code must pass all of the test cases.

Time (sec)	Memory (KiB)	Language
0.017694	2	Python 3.8

Input

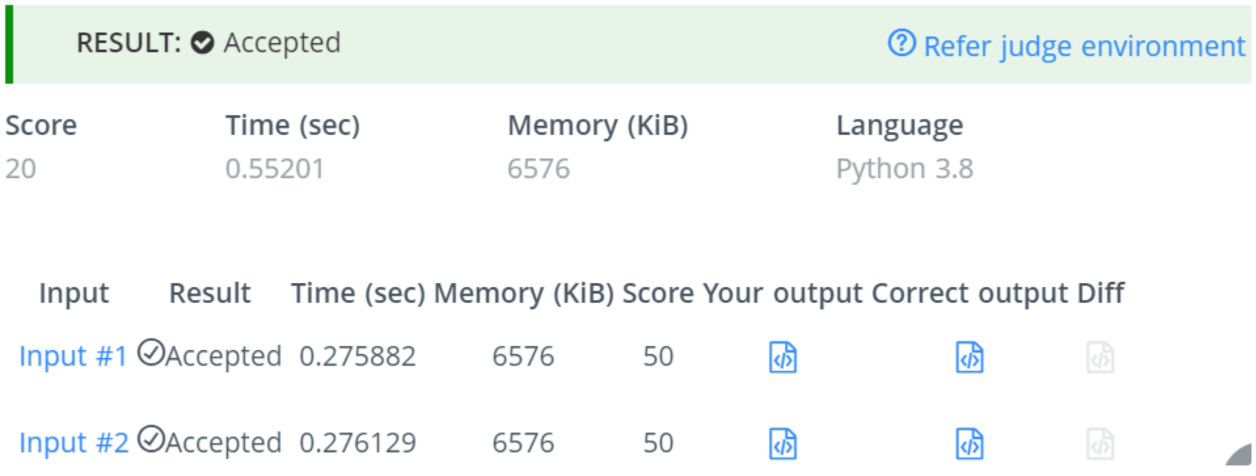
```
3
5 2
3 5
10 27
```

Output

```
2
1
9
```

Expected Correct Output

```
2
1
9
```


RESULT: Accepted

Refer judge environment

Score	Time (sec)	Memory (KiB)	Language
20	0.55201	6576	Python 3.8

Input	Result	Time (sec)	Memory (KiB)	Score	Your output	Correct output	Diff
Input #1	Accepted	0.275882	6576	50			
Input #2	Accepted	0.276129	6576	50			

Outcomes: Understand the fundamental concepts for managing the data using different data structures such as lists, queues, trees etc.

Conclusion: (Conclusion to be based on the objectives and outcomes achieved)
The experiment successfully met its objectives of implementing a Linear Search algorithm on HackerEarth and optimizing its solution using suitable data structures. The experience not only enhanced the understanding of basic search algorithms but also highlighted the importance of selecting the right data structure for optimizing algorithmic efficiency. The knowledge gained from this experiment can be applied to further explore and optimize algorithms in various programming contexts.

References:

1. https://www.hackerearth.com/practice/data-structures/arrays/1-d/practice-problems/algorithm/sum-as-per-frequency-88b00c1f/
2. T.H. Coreman ,C.E. Leiserson,R.L. Rivest, and C. Stein, " Introduction to algorithms", 3rd Edition 2009, Prentice Hall India Publication
3. Antti Laaksonen, “Guide to Competitive Programming”,Springer,2018
4. Gayle Laakmann McDowell,” Cracking the Coding Interview”,CareerCup LLC,2015
5. Steven S. Skiena Miguel A. Revilla,”Programming challenges, The Programming Contest Training Manual”, Springer, 2006
6. Antti Laaksonen, “Competitive Programmer’s Handbook”, Hand book, 2018
7. Steven Halim and Felix Halim, “Competitive Programming 3: The Lower Bounds of Programming Contests”, Handbook for ACM ICPC