

HackerRank Username: s_maisiba

The screenshot shows the HackerRank challenge page for "Queue using Two Stacks". At the top, a notification states: "You have earned 30.00 points! You are now 34 points away from the 2nd star for your problem solving badge." A progress bar indicates 51% completion (66/100 points). Below this, there are two main sections: "Congratulations" with a "Next Challenge" button, and "Earn a certificate in Problem Solving" with a "Get Certified" button. The "Test case 0" section is expanded, showing a "Compiler Message" of "Success" and an "Input (stdin)" section with the following data:

```
1 10
2 1 42
3 2
4 1 14
5 3
6 1 28
7 3
8 1 60
9 1 78
```

The Windows taskbar at the bottom shows the time as 14:37 on 14/02/2021.

The screenshot shows the HackerRank challenge page for "Queue using Two Stacks". The top navigation bar includes "PRACTICE", "CERTIFICATION", "COMPETE", "JOBS", and "LEADERBOARD". The user's profile "s_maisiba" is visible. The challenge title "Queue using Two Stacks" is displayed with a star icon. A notification states: "Your Queue using Two Stacks submission got 30.00 points. You are now 34 points away from the 2nd star for your problem solving badge." Below this, there are tabs for "Problem", "Submissions", "Leaderboard", "Discussions", and "Editorial". The "Problem" tab is selected, showing the problem description: "A queue is an abstract data type that maintains the order in which elements were added to it, allowing the oldest elements to be removed from the front and new elements to be added to the rear. This is called a First-In-First-Out (FIFO) data structure because the first element added to the queue (i.e., the one that has been waiting the longest) is always the first one to be removed. A basic queue has the following operations: Enqueue: add a new element to the end of the queue. Dequeue: remove the element from the front of the queue and return it. In this challenge, you must first implement a queue using two stacks. Then process q queries, where each query is one of the following types: 1. enqueue x: enqueue an element x to the queue. 2. dequeue: dequeue an element from the queue and return it. 3. front: return the front element of the queue. 4. back: return the back element of the queue. 5. is_empty: return true if the queue is empty, false otherwise. 6. is_full: return true if the queue is full, false otherwise. The queue is considered full if the number of elements in the queue is equal to the value of k . The queue is considered empty if the number of elements in the queue is equal to 0." The "Submissions" tab is also visible, showing the author "saikiran9194", difficulty "Medium", max score "30", and submitted by "47957". The "Problem" tab is selected, showing the problem description. The "Submissions" tab is also visible, showing the author "saikiran9194", difficulty "Medium", max score "30", and submitted by "47957". The "Problem" tab is selected, showing the problem description. The "Submissions" tab is also visible, showing the author "saikiran9194", difficulty "Medium", max score "30", and submitted by "47957".

```
1 # Starting with stacks class
2
3 class Stack:
4     def __init__(self):
5         self.stack = []
6
7     def push(self, x):
8         self.stack.append(x)
9
10    def pop(self):
11        return self.stack.pop()
12
13    def __len__(self):
14        return len(self.stack)
15
16    def top(self):
17        if self.stack:
18            return self.stack[-1]
19        return None
20
21
22 # To be linked to stacks class
23
24 class Queue:
25     def __init__(self):
26         self.front = Stack()
27         self.end = Stack()
28
29
```

Line: 68 Col: 5

Upload Code as File Test against custom input Run Code Submit Code

```
# Starting with stacks class

class Stack:

    def __init__(self):
        self.stack = []

    def push(self, x):
        self.stack.append(x)

    def pop(self):
        return self.stack.pop()

    def __len__(self):
        return len(self.stack)

    def top(self):
        if self.stack:
            return self.stack[-1]
        return None
```

```
# To be linked to stacks class
```

```
class Queue:
```

```
    def __init__(self):
```

```
        self.front = Stack()
```

```
        self.end = Stack()
```

```
    def enqueue(self, x):
```

```
        self.end.push(x)
```

```
    def dequeue(self):
```

```
        if self.front:
```

```
            return self.front.pop()
```

```
        return self.replace_front().pop()
```

```
    def peek(self):
```

```
        if self.front:
```

```
            return self.front.top()
```

```
        return self.replace_front().top()
```

```
    def replace_front(self):
```

```
        while self.end:
```

```
            self.front.push(self.end.pop())
```

```
        return self.front
```

```
# Function to read from standard input
```

```
def from_stdin():
```

```
    lines = input().strip()
```

```
    line = lines.split()
```

```
    the_type = int(line[0])
```

```
    if len(line) == 1:
        return (the_type, None)

    num = int(line[1])

    return the_type, num


# Calling all functions


def main():
    q = Queue()
    queries = int(input().strip())

    for z in range(queries):
        the_type, num = from_stdin()

        if the_type == 1:
            q.enqueue(num)

        elif the_type == 2:
            q.dequeue()

        elif the_type == 3:
            print(q.peak())


if __name__ == '__main__':
    main()
```