

DSA LAB 5

MUHAMMAD QASIM

Muhammad Qasim

[COMPANY NAME] [Company address]

Question 1:

```
public class Queue {
```

```
    int capacity;
```

```
    int end;
```

```
    int[] Q;
```

```
    int size;
```

```
    int front;
```

```
    Queue(int d) {
```

```
        this.capacity = d;
```

```
        this.size = 0;
```

```
        Q = new int[capacity];
```

```
        this.end = -1;
```

```
        this.front = 0;
```

```
    }
```

```
    boolean isFull() {
```

```
        return size == capacity;
```

```
    }
```

```
    boolean isEmpty() {
```

```
        return size == 0;
```

```
    }
```

```
    void enQ(int data) {
```

```
        if (isFull()) {
```

```
        System.out.println("Queue is Full!");
        return;
    }
    end = (end + 1) % capacity;
    Q[end] = data;
    size++;
}
```

```
int deQ() {
    if (isEmpty()) {
        System.out.println("Queue is Empty!");
        return -1;
    }
    int removed = Q[front];
    front = (front + 1) % capacity;
    size--;

    if (size == 0) {
        // reset for fresh start
        front = 0;
        end = -1;
    }
    return removed;
}
```

```
int getFront() {
```

```
    if (isEmpty()) {  
        System.out.println("Queue is Empty!");  
        return -1;  
    }  
    return Q[front];  
}
```

```
int getSize() {  
    return size;  
}
```

```
void printAll() {  
    if (isEmpty()) {  
        System.out.println("Queue is Empty!");  
        return;  
    }  
    System.out.print("Queue (front → rear): ");  
    for (int i = 0; i < size; i++) {  
        System.out.print(" " + Q[(front + i) % capacity] + " ");  
    }  
    System.out.println();  
}
```

```
public static void main(String[] args) {  
    Queue q1 = new Queue(3);
```

```
q1.enQ(5);
```

```
q1.enQ(70);
```

```
q1.enQ(69);
```

```
q1.printAll();
```

```
System.out.println("Dequeued: " + q1.deQ());
```

```
q1.enQ(80);
```

```
q1.printAll();
```

```
System.out.println("Front element: " + q1.getFront());
```

```
System.out.println("Size: " + q1.getSize());
```

```
System.out.println("Is Empty? " + q1.isEmpty());
```

```
System.out.println("Is Full? " + q1.isFull());
```

```
}
```

```
}
```

```
Run Queue x
↑ "C:\Program Files\Java\jdk-24\bin\java.exe" "-javaagent:C:\Program Files\JetBra
↓ Queue (front → rear): [ 5 ] [ 70 ] [ 69 ]
⇌ Dequeued: 5
⇌ Queue (front → rear): [ 70 ] [ 69 ] [ 80 ]
🖨 Front element: 70
🗑 Size: 3
Is Empty? false
Is Full? true

Process finished with exit code 0
```

Question 2:

```
public class QueueLL {
    private static class Node {
        int data;
        Node next;
        Node(int d) {
            data = d;
            next = null;
        }
    }

    private Node front, rear;
    private int size;
```

```
QueueLL() {
```

```
    front = rear = null;
```

```
    size = 0;
```

```
}
```

```
boolean isEmpty() {
```

```
    return size == 0;
```

```
}
```

```
void enQ(int data) {
```

```
    Node newNode = new Node(data);
```

```
    if (rear == null) { // empty queue
```

```
        front = rear = newNode;
```

```
    } else {
```

```
        rear.next = newNode;
```

```
        rear = newNode;
```

```
    }
```

```
    size++;
```

```
}
```

```
int deQ() {
```

```
    if (isEmpty()) {
```

```
        System.out.println("Queue is Empty!");
```

```
        return -1;
```

```
    }
```

```
    int removed = front.data;

    front = front.next;

    if (front == null) {
        rear = null;
    }

    size--;

    return removed;
}
```

```
int getFront() {
    if (isEmpty()) {
        System.out.println("Queue is Empty!");
        return -1;
    }

    return front.data;
}
```

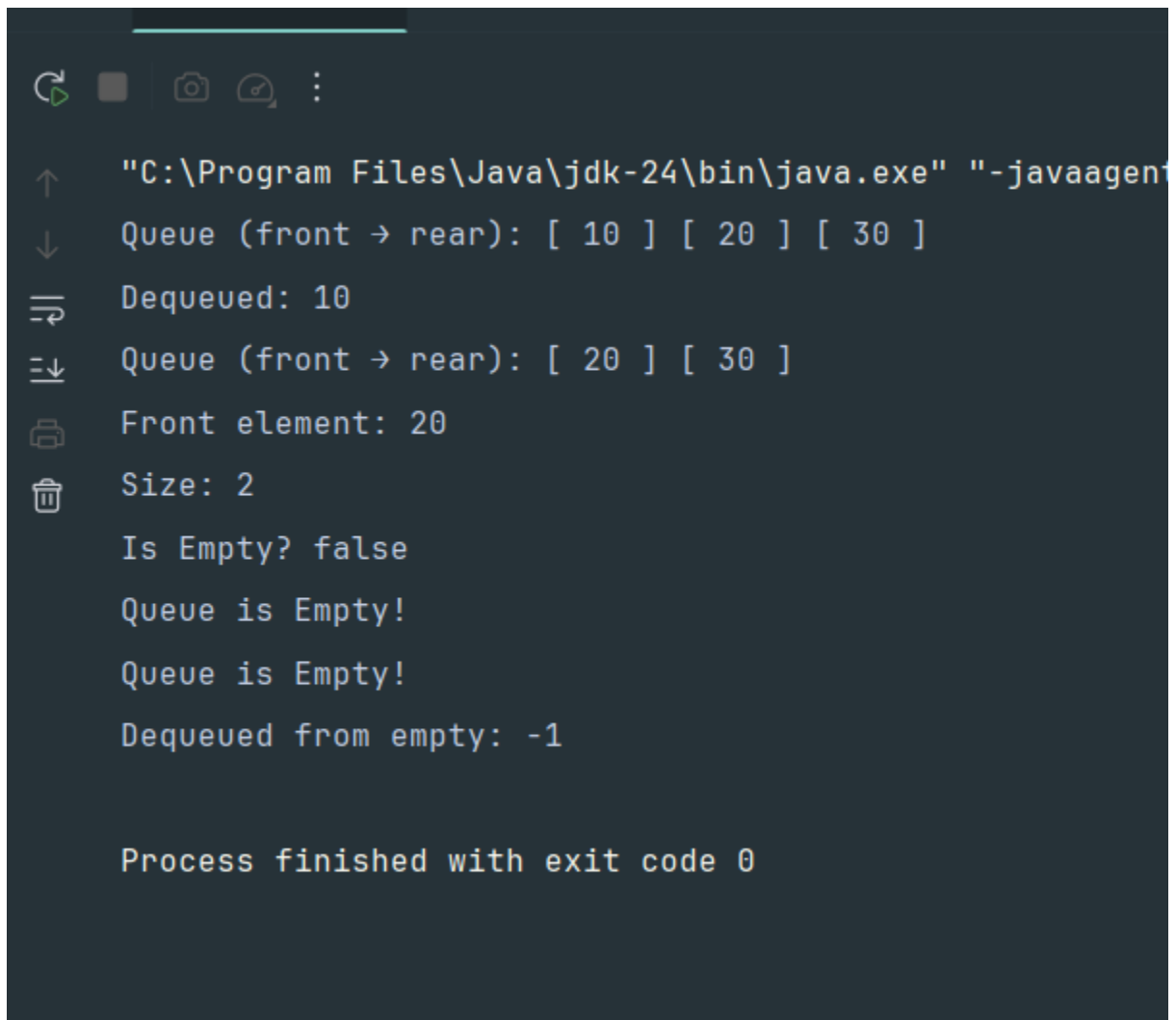
```
int getSize() {
    return size;
}
```

```
void printAll() {
    if (isEmpty()) {
        System.out.println("Queue is Empty!");
        return;
    }
}
```

```
System.out.print("Queue (front → rear): ");  
Node temp = front;  
while (temp != null) {  
    System.out.print("[ " + temp.data + " ] ");  
    temp = temp.next;  
}  
System.out.println();  
}
```

```
public static void main(String[] args) {  
    QueueLL q = new QueueLL();  
  
    q.enQ(10);  
    q.enQ(20);  
    q.enQ(30);  
  
    q.printAll();  
  
    System.out.println("Dequeued: " + q.deQ());  
    q.printAll();  
  
    System.out.println("Front element: " + q.getFront());  
    System.out.println("Size: " + q.getSize());  
    System.out.println("Is Empty? " + q.isEmpty());  
  
    q.deQ();  
}
```

```
q.deQ();  
q.printAll();  
System.out.println("Dequeued from empty: " + q.deQ());  
}  
}
```



The screenshot shows a terminal window with the following output:

```
"C:\Program Files\Java\jdk-24\bin\java.exe" "-javaagent  
Queue (front → rear): [ 10 ] [ 20 ] [ 30 ]  
Dequeued: 10  
Queue (front → rear): [ 20 ] [ 30 ]  
Front element: 20  
Size: 2  
Is Empty? false  
Queue is Empty!  
Queue is Empty!  
Dequeued from empty: -1  
  
Process finished with exit code 0
```

Question 3:

```
import java.util.Stack;
```

```
public class QueueUsingStacks {
```

```
    private Stack<Integer> stack1;
```

```
    private Stack<Integer> stack2;
```

```
    public QueueUsingStacks() {
```

```
        stack1 = new Stack<>();
```

```
        stack2 = new Stack<>();
```

```
    }
```

```
    public void enQ(int data) {
```

```
        stack1.push(data);
```

```
    }
```

```
    public int deQ() {
```

```
        if (isEmpty()) {
```

```
            System.out.println("Queue is Empty!");
```

```
            return -1;
```

```
        }
```

```
        if (stack2.isEmpty()) {
```

```
            while (!stack1.isEmpty()) {
```

```
                stack2.push(stack1.pop());
```

```
            }
```

```
}
```

```
    return stack2.pop();
```

```
}
```

```
public int getFront() {
```

```
    if (isEmpty()) {
```

```
        System.out.println("Queue is Empty!");
```

```
        return -1;
```

```
    }
```

```
    if (stack2.isEmpty()) {
```

```
        while (!stack1.isEmpty()) {
```

```
            stack2.push(stack1.pop());
```

```
        }
```

```
    }
```

```
    return stack2.peek();
```

```
}
```

```
public int getSize() {
```

```
    return stack1.size() + stack2.size();
```

```
}
```

```
public boolean isEmpty() {
```

```
    return stack1.isEmpty() && stack2.isEmpty();
```

```
}
```

```
public void printAll() {
```

```
    if (isEmpty()) {
```

```
        System.out.println("Queue is Empty!");
```

```
        return;
```

```
    }
```

```
    System.out.print("Queue (front → rear): ");
```

```
    for (int i = stack2.size() - 1; i >= 0; i--) {
```

```
        System.out.print("[" + stack2.get(i) + " ] ");
```

```
    }
```

```
    for (int i = 0; i < stack1.size(); i++) {
```

```
        System.out.print("[" + stack1.get(i) + " ] ");
```

```
    }
```

```
    System.out.println("stack2 (front side): " + stack2);
```

```
    System.out.println("stack1 (rear side): " + stack1);
```

```
}
```

```
public static void main(String[] args) {
```

```
    QueueUsingStacks q = new QueueUsingStacks();
```

```
    q.enQ(10);
```

```
q.enQ(20);
```

```
q.enQ(30);
```

```
q.printAll();
```

```
System.out.println("Dequeued: " + q.deQ());
```

```
q.printAll();
```

```
q.enQ(40);
```

```
q.enQ(50);
```

```
q.printAll();
```

```
System.out.println("Front element: " + q.getFront());
```

```
System.out.println("Size: " + q.getSize());
```

```
System.out.println("Is Empty? " + q.isEmpty());
```

```
}
```

```
}
```

```
Run QueueUsingStacks x
"\"C:\\Program Files\\Java\\jdk-24\\bin\\java.exe\" \"-javaagent:C:\\Program Files\\JetBrains\\IntelliJ I
Queue (front → rear): [ 10 ] [ 20 ] [ 30 ] stack2 (front side): []
stack1 (rear side): [10, 20, 30]
Dequeued: 10
Queue (front → rear): [ 20 ] [ 30 ] stack2 (front side): [30, 20]
stack1 (rear side): []
Queue (front → rear): [ 20 ] [ 30 ] [ 40 ] [ 50 ] stack2 (front side): [30, 20]
stack1 (rear side): [40, 50]
Front element: 20
Size: 4
Is Empty? false

Process finished with exit code 0
```

Question 4:

```
import java.util.LinkedList;
```

```
import java.util.Queue;
```

```
public class SortQueue {
```

```
    public static void sortQueue(Queue<Integer> q) {
```

```
        int n = q.size();
```

```
        for (int i = 1; i <= n; i++) {
```

```
            int minIndex = getMinIndex(q, n - i);
```

```
            insertMinToRear(q, minIndex);
```

```
        }
```

```
    }
```

```
private static int getMinIndex(Queue<Integer> q, int sortedIndex) {  
    int minIndex = -1;  
    int minValue = Integer.MAX_VALUE;  
    int n = q.size();  
  
    for (int i = 0; i < n; i++) {  
        int curr = q.remove();  
  
        if (curr < minValue && i <= sortedIndex) {  
            minValue = curr;  
            minIndex = i;  
        }  
  
        q.add(curr);  
    }  
  
    return minIndex;  
}
```

```
private static void insertMinToRear(Queue<Integer> q, int minIndex) {  
    int minValue = 0;  
    int n = q.size();  
  
    for (int i = 0; i < n; i++) {  
        int curr = q.remove();
```

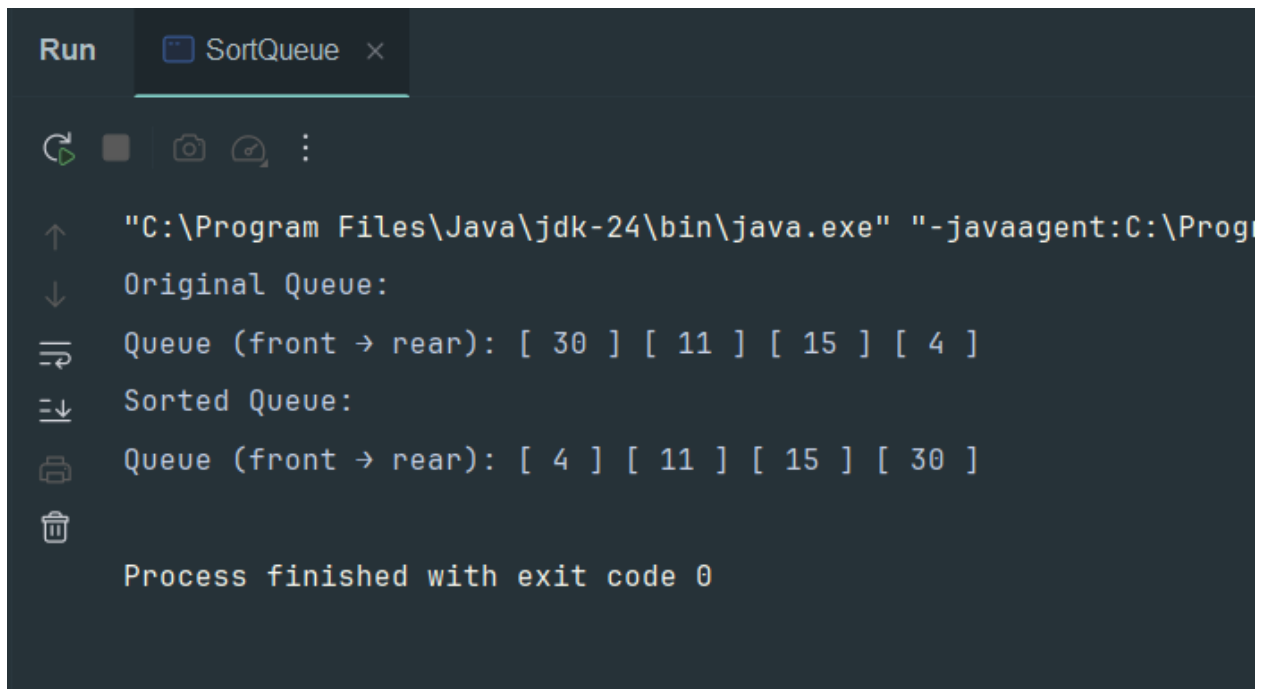
```
    if (i == minIndex) {  
        minValue = curr;  
    } else {  
        q.add(curr);  
    }  
}
```

```
q.add(minValue);  
}
```

```
public static void printQueue(Queue<Integer> q) {  
    System.out.print("Queue (front → rear): ");  
    for (int val : q) {  
        System.out.print(" " + val + " ");  
    }  
    System.out.println();  
}
```

```
public static void main(String[] args) {  
    Queue<Integer> q = new LinkedList<>();  
  
    q.add(30);  
    q.add(11);  
    q.add(15);  
    q.add(4);  
}
```

```
System.out.println("Original Queue:");  
printQueue(q);  
  
sortQueue(q);  
  
System.out.println("Sorted Queue:");  
printQueue(q);  
}  
}
```



The screenshot shows the 'Run' console of an IDE with a tab for 'SortQueue'. The console output is as follows:

```
"C:\Program Files\Java\jdk-24\bin\java.exe" "-javaagent:C:\Progr  
Original Queue:  
Queue (front → rear): [ 30 ] [ 11 ] [ 15 ] [ 4 ]  
Sorted Queue:  
Queue (front → rear): [ 4 ] [ 11 ] [ 15 ] [ 30 ]  
Process finished with exit code 0
```

Question 5:

933. Number of Recent Calls

Easy Topics Companies

You have a `RecentCounter` class which counts the number of recent requests within a certain time frame.

Implement the `RecentCounter` class:

- `RecentCounter()` Initializes the counter with zero recent requests.
- `int ping(int t)` Adds a new request at time `t`, where `t` represents some time in milliseconds, and returns the number of requests that has happened in the past `3000` milliseconds (including the new request). Specifically, return the number of requests that have happened in the inclusive range `[t - 3000, t]`.

It is **guaranteed** that every call to `ping` uses a strictly larger value of `t` than the previous call.

Example 1:

```
Input
["RecentCounter", "ping", "ping", "ping", "ping"]
[[], [1], [100], [3001], [3002]]


Output
[null, 1, 2, 3, 3]
```


Java Auto

```
1 class RecentCounter {
2     Queue<Integer> q;
3
4     public RecentCounter() {
5         q = new LinkedList<>();
6     }
7
8     public int ping(int t) {
9         q.add(t);
10        while (q.peek() < t - 3000) {
11            q.poll();
12        }
13        return q.size();
14    }
15 }
16
17
18 /**
19  * Your RecentCounter object will be instantiated and called as such:
20  * RecentCounter obj = new RecentCounter();
21  * int param_1 = obj.ping(t);
22  */
```

Accepted 68 / 68 testcases passed

 kas-sim submitted at Sep 18, 2025 11:51

 Editorial

 Solution


⌚ Runtime

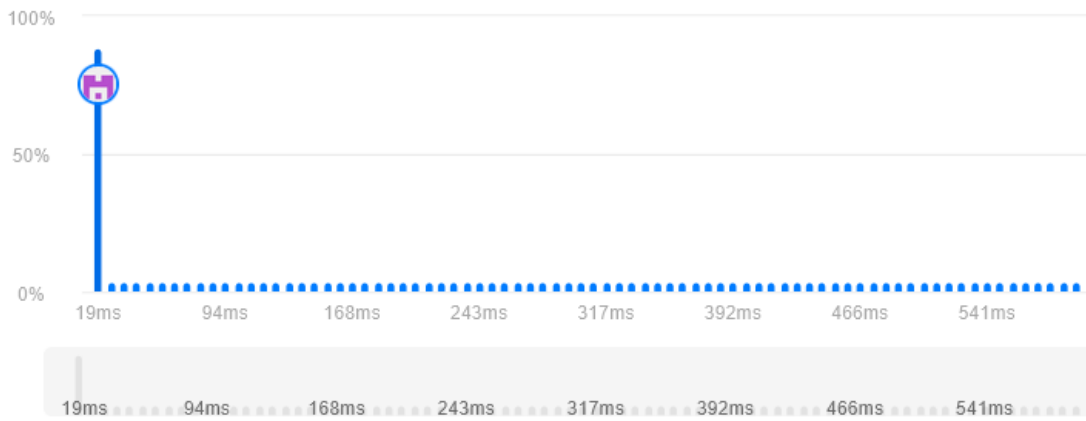


22 ms | Beats 33.44%

 [Analyze Complexity](#)

⚙️ Memory

54.24 MB | Beats 57.39% 



225. Implement Stack using Queues

Easy Topics Compare

Implement a last-in-first-out (LIFO) stack using only two queues. The implemented stack should support all the functions of a normal stack (`push`, `top`, `pop`, and `empty`).

Implement the `MyStack` class:

- `void push(int x)` Pushes element `x` to the top of the stack.
- `int pop()` Removes the element on the top of the stack and returns it.
- `int top()` Returns the element on the top of the stack.
- `boolean empty()` Returns `true` if the stack is empty, `false` otherwise.

Notes:

- You must use **only** standard operations of a queue, which means that only `push` to back, `peek/pop` from front, `size` and `is empty` operations are valid.
- Depending on your language, the queue may not be supported natively. You may simulate a queue using a list or deque (double-ended queue) as long as you use only a queue's standard operations.

Example 1:


Input

```


1 class MyStack {
2     Queue<Integer> q = new LinkedList<>();
3
4     public void push(int x) {
5         q.add(x);
6         for (int i = 1; i < q.size(); i++) {
7             q.add(q.poll());
8         }
9     }
10
11     public int pop() {
12         return q.poll();
13     }
14
15     public int top() {
16         return q.peek();
17     }
18
19     public boolean empty() {
20         return q.isEmpty();
21     }
22 }
23
24 /**
25  * Your MyStack object will be instantiated and called as such:
26  * MyStack obj = new MyStack();
27  * obj.push(x);
28  */

```

Accepted 18 / 18 testcases passed

 kas-sim submitted at Sep 18, 2025 12:08

Editorial

 Solution


Runtime

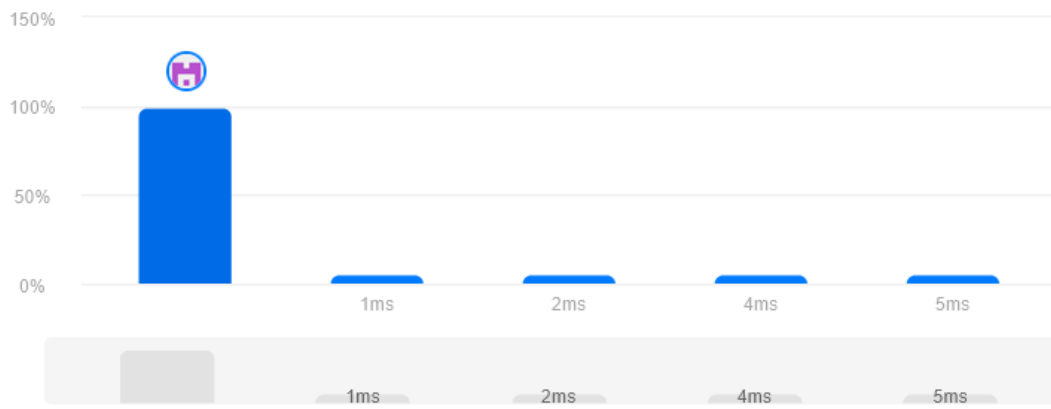


0 ms | Beats 100.00% 

 [Analyze Complexity](#)

Memory

41.28 MB | Beats 82.87% 



DescriptionEditorialSolutionsSubmissions

2073. Time Needed to Buy Tickets

EasyTopicsCompaniesHint

There are n people in a line queuing to buy tickets, where the 0^{th} person is at the **front** of the line and the $(n - 1)^{\text{th}}$ person is at the **back** of the line.

You are given a **0-indexed** integer array `tickets` of length n where the number of tickets that the i^{th} person would like to buy is `tickets[i]`.

Each person takes **exactly 1 second** to buy a ticket. A person can only buy **1 ticket at a time** and has to go back to the **end** of the line (which happens **instantaneously**) in order to buy more tickets. If a person does not have any tickets left to buy, the person will **leave** the line.

Return the **time taken** for the person **initially** at position k (0-indexed) to finish buying tickets.

Example 1:

Input: `tickets = [2,3,2]`, `k = 2`

Output: 6

Explanation:

- The queue starts as `[2,3,2]`, where the k^{th} person is underlined.
- After the person at the front has bought a ticket, the queue becomes `[3,2,1]` at 1 second.

</ Code

JavaAuto

```
1 class Solution {
2     public int timeRequiredToBuy(int[] tickets, int k) {
3         Queue<Integer> q = new LinkedList<>();
4         for (int i = 0; i < tickets.length; i++) q.add(i);
5
6         int time = 0;
7         while (!q.isEmpty()) {
8             int person = q.poll();
9             time++;
10            tickets[person]--;
11            if (tickets[person] > 0) q.add(person);
12            if (person == k && tickets[person] == 0) return time;
13        }
14        return time;
15    }
16 }
17
```

Saved

TestcaseTest Result

DescriptionAccepted XEditorialSolutionsSubmissions

All Submissions

Accepted 65 / 65 testcases passed

kas-sim submitted at Sep 18, 2025 12:14

EditorialSolution

Runtime

10 ms | Beats 19.78%

Analyze Complexity

Memory

44.08 MB | Beats 26.53%

Runtime (ms)	Percentage
0	~50%
2	~15%
4	~5%
6	~5%
8	~5%
10	~10%

Code | Java