**The data structure:**

**SequenceWithMinimum:** We create a new class with two elements of the type “Node”. The sequence consists of a series of Nodes. The most left node is called “head”, and the most right node is called “tail”. In addition, the class contains two integers:  
- The first integer is called “min” to refer the minimum value in the sequence.  
- The second integer is called “occurrenceOfMin”. It is a counter to define the number of the occurrence of the minimum value in the sequence.  
Each **Node** has a reference to the left node and the right node,and an integer to present the value.

**Implementation and the complexity:**

**– public void insertRight(Integer value)**

|  |  |  |
| --- | --- | --- |
| IF (value == null) | O(1) | O(1) |
| return; |  |
| ENDIF |  |
| IF (sequence is empty) | O(1) | O(1) |
| head = new Node(value) | O(1) |
| tail = head | O(1) |
| min = value | O(1) |
| occurrenceOfMin = 1 | O(1) |
| ELSE | O(1) | O(1) |
| newNode = new Node(value) | O(1) |
| left node of newNode = tail | O(1) |
| right node of tail = newNode | O(1) |
| tail = newNode | O(1) |
| IF value lesser than min | O(1) |
| min = value | O(1) |
| occurrenceOfMin = 1 | O(1) |
| ELSE IF (value == min) | O(1) |
| occurrenceOfMin++ | O(1) |
| ENDIF |  |  |
| ENDIF |  |  |

**complexity =** max**(** O(1), max(O(1),O(1)) **)** = O(1)

**– public Integer removeRight()**

|  |  |  |
| --- | --- | --- |
| IF (sequence is empty) | O(1) | O(1) |
| return null |  |
| ENDIF |  |  |
| int temp = the value of tail | O(1) | O(1) |
| IF left node of tail is null | O(1) | O(1) |
| tail is null | O(1) |
| head is null | O(1) |
| ELSE |  | O(1) |
| tail = left node of tail | O(1) |
| right node of tail = null | O(1) |
| ENDIF |  |  |
| IF (temp == min) | O(1) | O(1) |
| occurrenceOfMin-- | O(1) |
| IF (no more occurrence of min) | O(1) |
| min = null | O(1) |
| ENDIF |  |
| ENDIF |  |
| return temp |  |  |

**complexity =** O(1)+ O(1)+max(O(1), O(1)) + O(1) = O(1)

**– public void insertLeft(Integer value)**

|  |  |  |
| --- | --- | --- |
| IF (value == null) | O(1) | O(1) |
| return; |  |
| ENDIF |  |
| IF (sequence is empty) | O(1) | O(1) |
| head = new Node(value) | O(1) |
| tail = head | O(1) |
| min = value | O(1) |
| occurrenceOfMin = 1 | O(1) |
| ELSE |  |  |
| newNode = new Node(value) | O(1) | O(1) |
| right node of newNode = head | O(1) |
| left node of head = newNode | O(1) |
| head = newNode | O(1) |
| IF value < min | O(1) |
| min = value | O(1) |
| occurrenceOfMin = 1 | O(1) |
| ELSE IF (value == min) | O(1) |
| occurrenceOfMin++ | O(1) |
| ENDIF |  |  |
| ENDIF |  |  |

**complexity =** max**(** O(1), max(O(1),O(1)) **)** = O(1)

**– public Integer removeLeft()**

|  |  |  |  |
| --- | --- | --- | --- |
| IF (sequence is empty) | O(1) | | O(1) |
| return null |  | |
| ENDIF |  | |  |
| int temp = the value of head | O(1) | | O(1) |
| IF right node of head is null | O(1) | | O(1) |
| tail is null | O(1) | |
| head is null | O(1) | |
| ELSE | O(1) | | O(1) |
| head = right node of head | O(1) | |
| left node of head is null | O(1) | |
| ENDIF |  | |  |
| IF (temp == min) | | O(1) | O(1) |
| occurrenceOfMin-- | | O(1) |
| IF (no more occurrence of min) | | O(1) |
| min = null | | O(1) |
| ENDIF | |  |
| ENDIF | |  |
| return temp |  | |  |

**complexity =** O(1)+ O(1)+max(O(1), O(1)) + O(1)= O(1)

**– public Integer findMinimum()**

|  |  |  |
| --- | --- | --- |
| IF min is not null | O(1) | O(1) |
| return min |  |  |
| ELSE IF (sequence is not empty) | O(1) | O(N) |
| searchNewMin() | O(N) see next table |
| ENDIF |  |  |
| return min |  |  |

**complexity (worst-case) = max (**O(1), O(N)) = O(N)  
**complexity (amortized) = O(1)**

New search will happen only if all occurrences for the previous min were removed. Therefore, the call for the method **searchNewMin()** is rarely happens. As result:

**complexity (amortized) = O(1)**

**– searchNewMin()**

|  |  |  |
| --- | --- | --- |
| IF (sequence is not empty) | O(1) | O(1) |
| Node current = head | O(1) | O(1) |
| min = value of current node | O(1) | O(1) |
| occurrenceOfMin = 1 | O(1) | O(1) |
| WHILE right node of current is not null | N TIMES | O(N) |
| current = right node of current | O(1) |
| IF the value of current < min | O(1) |
| min = value of current | O(1) |
| occurrenceOfMin = 1 | O(1) |
| ELSE IF the value of current == min | O(1) |
| occurrenceOfMin++ | O(1) |
| ENDIF |  |  |
| ENDWHILE |  |  |
| ENDIF |  |  |
| return min |  |  |

**complexity =** O(1)+ O(1)+ O(1)+ O(1)+ O(N) = O(N)