Given a reference to the head of a doubly-linked list and an integer, $data$, create a new DoublyLinkedListNode object having data value $data$ and insert it at the proper location to maintain the sort.

Example

$head$ refers to the list $1 \leftrightarrow 2 \leftrightarrow 4 \rightarrow NULL$

$data = 3$

Return a reference to the new list: $1 \leftrightarrow 2 \leftrightarrow 3 \leftrightarrow 4 \rightarrow NULL$.

Function Description

Complete the sortedInsert function in the editor below.

sortedInsert has two parameters:

- DoublyLinkedListNode pointer head: a reference to the head of a doubly-linked list
- int data: An integer denoting the value of the $data$ field for the DoublyLinkedListNode you must insert into the list.

Returns

- DoublyLinkedListNode pointer: a reference to the head of the list

Note: Recall that an empty list (i.e., where $head = \texttt{NULL}$) and a list with one element are sorted lists.

Input Format

The first line contains an integer t, the number of test cases.

Each of the test case is in the following format:

- The first line contains an integer n, the number of elements in the linked list.
- Each of the next n lines contains an integer, the data for each node of the linked list.
- The last line contains an integer, $data$, which needs to be inserted into the sorted doubly-linked list.

Constraints

- $1 \leq t \leq 10$
- $1 \leq n \leq 1000$
- $1 \leq DoublyLinkedListNode.data \leq 1000$

Sample Input

```
STDIN   Function
-----   --------
1       t = 1
4       n = 4
1       node data values = 1, 3, 4, 10
3
4
10
5       data = 5
```

Sample Output

```
1 3 4 5 10
```

Explanation

The initial doubly linked list is: $1 \leftrightarrow 3 \leftrightarrow 4 \leftrightarrow 10 \rightarrow NULL$.

The doubly linked list after insertion is: $1 \leftrightarrow 3 \leftrightarrow 4 \leftrightarrow 5 \leftrightarrow 10 \rightarrow NULL$

Change Theme Language Python 3

```python
def sortedInsert(llist, data):
    # Write your code here
    head = llist
    node = llist
    previous = None

    # if an empty list is given
    if head is None:
        print('Creating a new linked list')
        return DoublyLinkedListNode(data)

    # if the new node should be inserted before head
    if head.data > data:
        print(f'head: {head.data} > data: {data}')
        print('Inserting at the start')
        new_head = DoublyLinkedListNode(data)
        new_head.next = head
        new_head.prev = None
        head.prev     = new_head
        return new_head

    # find where the new node should be inserted
    while node is not None and node.data < data:
        previous = node
        node = node.next

    # first check if the node is None
    # that means we reached the end of the doubly linked list
    # insert the new node at the end
    if node is None:
        print('Inserting at the end')
        print(f'tail: {previous.data} < data: {data}')
        new_tail = DoublyLinkedListNode(data)
        previous.next = new_tail
        new_tail.prev = previous
        return head

    # we are not at the end of the linked list
    # means, we are somewhere between head and before tail
    # now, the current node in variable node
    # has data value greater than the given data value
    # so we can safely insert our new node
    # before this node
    # if we are inserting in between
    print('Inserting in between')
    print(f'after: {node.data} > data: {data}')
    new_node = DoublyLinkedListNode(data)
    new_node.prev       = node.prev
    new_node.prev.next  = new_node
    new_node.next       = node
    node.prev           = new_node
    return head

```

Line: 58 Col: 1

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

Congratulations

You solved this challenge. Would you like to challenge your friends?

Test case 0

Compiler Message

Success