Assignment on Queue and Stack

PROBLEM 1. IMPLEMENT A STACK USING TWO QUEUES Q1 AND Q2.

```
Example 1:

Input:
push(2)
push(3)
pop()
push(4)
pop()
Output: 3 4
Explanation:
push(2) the stack will be {2}
push(3) the stack will be {2 3}
pop()   poped element will be 3 the
        stack will be {2}
push(4) the stack will be {2 4}
pop()   poped element will be 4
Example 2:

Input:
push(2)
pop()
pop()
push(3)
Output: 2 -1
Your Task:

Expected Time Complexity: O(1) for push() and O(N) for pop() (or vice-versa).
Expected Auxiliary Space: O(1) for both push() and pop().

Constraints:
1 <= Number of queries <= 100
1 <= values of the stack <= 100
```

PROBLEM 2. IMPLEMENT A QUEUE USING 2 STACKS S1 AND S2 .

A Query **Q** is of 2 Types
(i) 1 x (a query of this type means pushing **'x'** into the queue)
(ii) 2 (a query of this type means to pop element from queue and print the poped element)
Example 1:

```
Input:5
1 2 1 3 2 1 4 2
Output: 2 3
Explanation: In the first testcase
```

```
1 2 the queue will be {2}
1 3 the queue will be {2 3}
2   poped element will be 2 the queue
    will be {3}
1 4 the queue will be {3 4}
2   poped element will be 3.
```

Example 2:

```
Input:4
1 2 2 2 1 4
Output: 2 -1
Explanation: In the second testcase
1 2 the queue will be {2}
2   poped element will be 2 and
    then the queue will be empty
2   the queue is empty and hence -1
1 4 the queue will be {4}.
```

Your Task:
You are required to **write** two methods **push** which take one argument an integer **'x'** to be pushed into the queue and **pop** which returns a integer poped out from other queue(-1 if the queue is empty).
Expected Time Complexity : O(1) for **push()** and O(N) for **pop()** or O(N) for **push()** and O(1) for **pop()**
Expected Auxilliary Space : O(1).
Constraints:
1 <= Q <= 100
1 <= x <= 100

Note: The **Input/Ouput** format and **Example** given are used for system's internal purpose, and should be used by a user for **Expected Output** only. As it is a function problem, hence a user should not read any input from stdin/console. The task is to complete the function specified, and not to write the full code.

PROBLEM 3. GIVEN AN ARRAY ARR[] OF SIZE N AND AN INTEGER K. FIND THE MAXIMUM FOR EACH AND EVERY CONTIGUOUS SUBARRAY OF SIZE K.

Example 1:

```
Input:
N = 9, K = 3
arr[] = 1 2 3 1 4 5 2 3 6
Output:
3 3 4 5 5 5 6
Explanation:
1st contiguous subarray = {1 2 3} Max = 3
2nd contiguous subarray = {2 3 1} Max = 3
3rd contiguous subarray = {3 1 4} Max = 4
4th contiguous subarray = {1 4 5} Max = 5
5th contiguous subarray = {4 5 2} Max = 5
6th contiguous subarray = {5 2 3} Max = 5
7th contiguous subarray = {2 3 6} Max = 6
```

Example 2:

```
Input:
N = 10, K = 4
arr[] = 8 5 10 7 9 4 15 12 90 13
Output:
10 10 10 15 15 90 90Explanation: 1st contiguous subarray = {8 5 10 7}, Max = 10
2nd contiguous subarray = {5 10 7 9}, Max = 10
3rd contiguous subarray = {10 7 9 4}, Max = 10
4th contiguous subarray = {7 9 4 15}, Max = 15
5th contiguous subarray = {9 4 15 12},
Max = 15
6th contiguous subarray = {4 15 12 90},
Max = 90
7th contiguous subarray = {15 12 90 13},
Max = 90
```

Your Task:
Write a function which takes the array, N and K as input parameters and returns a list of integers denoting the maximum of every contiguous subarray of size K.
Expected Time Complexity: O(N)
Expected Auxiliary Space: O(k)
Constraints:

$1 \le N \le 10^5$

$1 \le K \le N$

$0 \le arr[i] \le 10^7$

PROBLEM 4. DESIGN A DATA STRUCTURE THAT WORKS LIKE A LRU CACHE. HERE CAP DENOTES THE CAPACITY OF THE CACHE AND Q DENOTES THE NUMBER OF QUERIES. QUERY CAN BE OF TWO TYPES:

1. **SET x y**: sets the value of the key **x** with value **y**
2. **GET x**: gets the key of **x** if present else returns **-1**.

The LRUCache class has two methods **get**() and **set**() which are defined as follows.

1. **get(key)**: returns the value of the key if it already exists in the cache otherwise returns **-1.**
2. **set(key, value)**: if the key is already present, update its value. If not present, add the key-value pair to the cache. If the cache reaches its capacity it should invalidate the least recently used item before inserting the new item.
3. In the **constructor** of the class the capacity of the cache should be initialized.

Example 1:

```
Input:cap = 2
Q = 2
Queries = SET 1 2 GET 1Output: 2
Explanation: Cache Size = 2

SET 1 2 GET 1
SET 1 2 : 1 -> 2
```

```
GET 1 : Print the value corresponding
to Key 1, ie 2.
```

Example 2:

```
Input:cap = 2
Q = 8
Queries = SET 1 2 SET 2 3 SET 1 5
SET 4 5 SET 6 7 GET 4 SET 1 2 GET 3Output: 5 -1
Explanation: Cache Size = 2
SET 1 2 : 1 -> 2

SET 2 3 : 1 -> 2, 2 -> 3 (the most recently
used one is kept at the rightmost position)

SET 1 5 : 2 -> 3, 1 -> 5

SET 4 5 : 1 -> 5, 4 -> 5 (Cache size is 2, hence
we delete the least recently used key-value pair)

SET 6 7 : 4 -> 5, 6 -> 7

GET 4 : Prints 5 (The cache now looks like
6 -> 7, 4->5)

SET 1 2 : 4 -> 5, 1 -> 2
(Cache size is 2, hence we delete the least
recently used key-value pair)

GET 3 : No key value pair having
key = 3. Hence, -1 is printed.
```

Your Task:
Write the constructor and get(), and set() methods of the class LRUcache.

Expected Time Complexity: O(1) for both **get()** and **set().**
Expected Auxiliary Space: O(1) for both **get()** and **set().**
(Although, you may use extra space for cache storage and implementation purposes).

Constraints:
1 <= cap <= 10^3
1 <= Q <= 10^5
1 <= x, y <= 10^4

PROBLEM 5. GIVEN A SORTED DECK OF CARDS NUMBERED 1 TO N.

1) We pick up 1 card and put it on the back of the deck.
2) Now, we pick up another card, it turns out to be card number 1, we put it outside the deck.

3) Now we pick up 2 cards and put it on the back of the deck.
4) Now, we pick up another card and it turns out to be card numbered 2, we put it outside the deck. ...
We perform this step until the last card.

If such an arrangement of decks is possible, output the arrangement, if it is not possible for a particular value of N then output -1.
Example 1:

```
Input:
N = 4Output: 2 1 4 3Explanation:
We initially have [2 1 4 3]
In Step1, we move the first card to the end.
Deck now is: [1 4 3 2]
In Step2, we get 1. Hence we remove it. Deck
now is: [4 3 2]
In Step3, we move the 2 front cards ony by one
to the end  ([4 3 2] -> [3 2 4] -> [2 4 3]).
Deck now is: [2 4 3]
In Step4, we get 2. Hence we remove it. Deck
now is: [4 3]
In Step5, the following sequence follows:
[4 3] -> [3 4] -> [4 3] -> [3 4]. Deck now
is: [3 4]
In Step6, we get 3. Hence we remove it. Deck
now is: [4] Finally, we're left with a single
card and thus, we stop.
```

Example 2:

```
Input:
N = 5
Output: 3 1 4 5 2
```

Your Task:
Your task is to write a function which takes the integer N as input parameters and returns If such arrangement of decks is possible, return the arrangement, if it is not possible for a particular value of n then return -1.

Expected Time Complexity: O(N^2)
Expected Auxiliary Space: O(1)

PROBLEM 6. GIVEN A STRING CONTAINING ONLY PARENTHESES, DETERMINE IF THE STRING IS VALID. AN INPUT STRING IS VALID IF:

- Open brackets must be closed by the same type of brackets.
- Open brackets must be closed in the correct order.

```
Example input: "()[]{}"
Expected output: True
```

```
Example input: "([)]"
Expected output: False
```

Expected Time Complexity: O(n), where n is the length of the input string.

PROBLEM 7. GIVEN AN ARRAY, FIND THE NEXT GREATER ELEMENT (NGE) FOR EVERY ELEMENT IN THE ARRAY. THE NEXT GREATER ELEMENT FOR AN ELEMENT X IS THE FIRST GREATER ELEMENT ON THE RIGHT SIDE OF X IN THE ARRAY. IF THERE IS NO GREATER ELEMENT ON THE RIGHT SIDE, THEN THE OUTPUT FOR THAT ELEMENT SHOULD BE -1.

```
Example input: [4, 5, 2, 25]
Expected output: [5, 25, 25, -1]
```

```
Example input: [13, 7, 6, 12]
Expected output: [-1, 12, 12, -1]
```

Expected Time Complexity: O(n), where n is the length of the input array.

PROBLEM 8. GIVEN A STRING S CONSISTING OF LOWERCASE LETTERS, REMOVE ADJACENT DUPLICATES FROM S. THE FINAL OUTPUT SHOULD BE IN LEXICOGRAPHICALLY SMALLEST ORDER.

```
Example input: "abbaca"
Expected output: "ca"
```

Expected Time Complexity: O(n), where n is the length of the input string.

PROBLEM 9. WRITE A CLASS STOCKSPANNER WHICH COLLECTS DAILY PRICE QUOTES FOR SOME STOCK, AND RETURNS THE SPAN OF THAT STOCK'S PRICE FOR THE CURRENT DAY. THE SPAN OF THE STOCK'S PRICE TODAY IS DEFINED AS THE MAXIMUM NUMBER OF CONSECUTIVE DAYS (STARTING FROM TODAY AND GOING BACKWARDS) FOR WHICH THE PRICE OF THE STOCK WAS LESS THAN OR EQUAL TO TODAY'S PRICE.

```
Example input:
stockSpanner = StockSpanner()
stockSpanner.next(100) # returns 1
stockSpanner.next(80) # returns 1
stockSpanner.next(60) # returns 1
stockSpanner.next(70) # returns 2
stockSpanner.next(75) # returns 4
stockSpanner.next(85) # returns 6
```

Expected Time Complexity: O(n), where n is the number of calls to the next() method.

PROBLEM 10. EVALUATE THE VALUE OF AN ARITHMETIC EXPRESSION IN REVERSE POLISH NOTATION (RPN). VALID OPERATORS ARE +, -, *, AND /. EACH OPERAND MAY BE AN INTEGER OR ANOTHER EXPRESSION.

```
Example input: ["2", "1", "+", "3", "*"]
Expected output: 9
```

```
Example input: ["4", "13", "5", "/", "+"]
Expected output: 6
```

Expected Time Complexity: O(n), where n is the length of the input list.