CSE 4304-Data Structures Lab. Winter 2021
Lab-03
Date: May 29, 2021 (Saturday)
Target Group: All Lab groups
Topic: Queue, Priority Queue, Heap

Instructions:

- Task naming format: fullID_T01L03_2A.c/cpp
- If you find any issues in problem description/test cases, comment in the google classroom.
- If you find any tricky test case which I didn't include and others might forget to handle, please comment! I'll be happy to add.
- Modified sections will be marked with BLUE colour.

[For task-01,02,03 use the implementations taught in lectures]

Task 1

Implement Enqueue & Dequeue operations using

A. Linear Queue
B. Circular Queue

Task 2

Suppose an arbitrary array of size N is given as input. Your task is to build a **max-heap** from the set of numbers and sort them using **Heap-sort**.

Note: Use different functions for 'heapify', 'Build_max_heap', 'Heap_sort'.

Task 3

Use the Heap that you created in **Task 2** as a 'Min Priority Queue' and implement the following functionalities:

A. Heap_Minimim()
B. Heap_extract_min()
C. Min_heap_insert()
D. Heap_decrease_key()
E. Heap_increase_key()

C++ has Some built-in functions for performing operations on Queue, Heap/ Priority Queue. Check the following links for better understanding:

- Basic STL functions to use queues: https://www.geeksforgeeks.org/queue-cpp-stl/
- https://www.geeksforgeeks.org/heap-using-stl-c/
- STL function to swap two queues: https://www.geeksforgeeks.org/queue-swap-cpp-stl/
- https://www.geeksforgeeks.org/heap-using-stl-c/

Task 4

Jesse loves cookies. He wants the sweetness of all his cookies to be greater than value **K**. To do this, Jesse repeatedly mixes two cookies with the least sweetness. He creates a special combined cookie with:

Sweetness = (1 x Least sweet cookie + 2 x *2nd least sweet cookie*).

He repeats this procedure until all the cookies in his collection have a sweetness ≥ **K**

You are given Jesse's cookies. Print the number of operations required to give the cookies a sweetness ≥ **K** Print **-1** if this isn't possible.

Input format

The first line consists of integers **N**, the number of cookies and **k**, the minimum required sweetness, separated by a space.

The next line contains **N** integers describing the array ***A*** where A_i is the sweetness of the i^{th} cookie in Jesse's collection.

Output format

Output the number of operations that are needed to increase the cookie's sweetness ≥ **K**

Output **-1** if this isn't possible.

Sample Input	**Sample Output**
6 7 12 9 1 3 10 2	2

Explanation

Combine the first two cookies to create a cookie with *sweetness* = 1 x 1 + 2 x 2 = 5

After this operation, the cookies are (3, 5, 9, 10, 12)

Then, combine cookies with sweetness and sweetness, to create a cookie with resulting *sweetness* = 1 x 3 + 2 x 5 = 13

Now, the cookies are (9, 10, 12, 13).

All the cookies have a sweetness ≥7

Thus, **2** operations are required to increase the sweetness.

Note: You should use ***Heap*** to solve this problem. Sorting might be another way of solving this problem, but that will take $O(nlogn)$ in the worst case. But Heap can lead us to a linear solution.

You can submit the task here.

Task 5

A ***queue*** is an abstract data type that means the order in which elements were added to it, allowing the oldest element 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 for the longest) is always the first one to be removed.

A basic queue has the following operation:

- 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 linear queue using two stacks.*** Then process ***q*** queries, where each query is one of the following 2 types:

1. 1 x: Enqueue element ***x*** into the end of the queue, prints the size of the queue with all the elements.
2. 2: Dequeue the element from the front, prints the size of the queue with all the elements.

Input Format

The first line contains a single integer, ***q***, denoting the number of queries.
Each line '***i***' of the ***q*** subsequent lines contains a single query in the form described in the problem statement above. All three queries start with an integer denoting the query ***type***␣ but only query **1** is followed by an additional space-separated value ***x*** denoting the value to be enqueued.

Output Format

For each query perform the Enqueue/Dequeue & print the elements of the queue on a new line.

Sample Input	**Sample Output**
5 10	Size:1 elements: 42 Size:0 elements: Null

1 42 2 1 14 1 25 1 33 2 1 10 1 22 1 99 1 75	Size:1 elements: 14 Size:2 elements: 14 25 Size:3 elements: 14 25 33 Size:2 elements: 25 33 Size:3 elements: 25 33 10 Size:4 elements: 25 33 10 22 Size:5 elements: 25 33 10 22 99 Size:5 elements: Overflow !

Note: You can submit the task here.