CSE 4304-Data Structures Lab. Winter 2022
Date: September 15, 2022
Target Group: All
Topic: Heap, Heapsort, Priority Queues

Instructions:

- Task naming format: fullID_T01L05_1B.c/CPP
- If you find any issues in problem description/test cases, comment in the google classroom.
- If you find any test case that is tricky that I didn't include but others might forget to handle, please comment! I'll be happy to add.
- Use appropriate comments in your code. This will help you to easily recall the solution in the future.
- Obtained marks will vary based on the efficiency of the solution.

- Modified sections will be marked with BLUE color.

Task-01: Implementing the basic operations of a **Heap**.

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.

Take input as long as you don’t get -1. For each test case, show the state of Max-Heap and the Sorted array.

Input	**Output**
4 1 3 2 16 9 10 14 8 7 -1	Max Heap: 16 14 10 8 7 9 3 2 4 1 Sorted: 16 14 10 9 8 7 4 3 2 1
7 9 6 19 8 17 11 2 5 3 13 -1	Max Heap:19 13 17 9 8 6 11 2 5 3 7 Sorted: 19 17 13 11 9 8 7 6 5 3 2

Note:

- STL not Allowed
- Use Separate functions for ‘heapify’, ‘Build_max_heap’, ‘Heap_sort’.

Task 2

Use the Heap that you created in Task 1 and convert it into a **'Min Priority Queue'** and implement the following functionalities:

1. int **Heap_Minimim**(int heap[]): Returns the minimum value.
2. int **Heap_extract_min**(int heap[]): Removes the minimum value and returns it.
3. **Min_heap_insert**(int value, int heap[]): Inserts the 'value' into the heap and makes necessary arrangements.

Input

First line of input will contain a set of numbers. Show the corresponding min-heap for that.

After that the input will be like 'function_id necessary_params (if any)'. Show the output and 'state of the heap' after each function call.

Input	**Output**
70 90 60 190 80 170 110 20 50 30 130 -1	Min Heap: 20 30 60 50 70 170 110 190 90 80 130
1	20 20 30 60 50 70 170 110 190 90 80 130
2	20 30 50 60 90 70 170 110 190 130 80
1	30 30 50 60 90 70 170 110 190 130 80
3 45	30 45 60 90 50 170 110 190 130 80 70
3 47	30 45 47 60 90 50 170 110 190 130 80 70

Note:

- Assume that, we are using 1-based indexing.

(Self-study)

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 3:

Mark loves cookies. He wants the sweetness of all his cookies to be greater than the value of **K**. To do this, Mark 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 Mark'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** representing 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 Mark'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.

Task 4

Given the description of ***N*** meetings i.e start time and end time of the meetings respectively, return the *minimum number of conference rooms required to arrange the meetings.*

Input	**Output**
3 0 30 5 10 15 20	2
2 7 10 2 4	1

Note:

- 0 < ***N*** <= 1000
- You must use **priority queue.**

Task 5

You are given an array of integers ‘stones’ where ‘storen[i]’ is the weight of the i-th stone.
We are playing a game with the stones. On each turn, we choose the **heaviest two stones** and smash them together. Suppose the heaviest two stones have weights x and y, with x<=y. The result of the smash is:

- If x==y, both stones are destroyed.
- If x!=y, the stone of weight x is destroyed, and the stone of weight y has a new weight (y-x).

At the end of the game, there is **at most one stone left**. Return the weight of the last remaining stone. If there are no stones left, return 0.

Input	**Output**	**Explanation**
2 7 4 1 8 1 -1	1	Combine 7,8. State: (2 4 1 1 1) Combine 2,4. State: (2 1 1 1) Combine 2,1. State: (1 1 1) Combine 1,1. State: (1) That's the value of the last stone.
10 10 10 10 10 -1	10	
10 10 5 10 10 10 -1	5	
50 30 10 40 20 -1	10	
50 30 10 40 60 20 -1	10	
10 50 30 10 40 60 20	0	

-1		
1 7 5 4 2 2 1 4 8 1 -1	1	
1 7 5 4 2 2 1 4 8 -1	0	
3 3 -1	0	
1 -1	1	