Offline

Heap Data structure

Implement Max Heap Data Structure with the following operations:

***You must use Class for implementing the heap and also write test cases in Main function to check each one of the operations.

Basic

- *find-max*: find a maximum item of a max-heap
- *insert*: adding a new key to the heap (a.k.a., *push*[3])
- *extract-max* : returns the node of maximum value from a max heap after removing it from the heap (a.k.a., *pop*[4])
- *delete-max* : removing the root node of a max heap
- *replace*: pop root and push a new key. More efficient than pop followed by push, since only need to balance once, not twice, and appropriate for fixed-size heaps.[5]

Creation

- *create-heap*: create an empty heap
- *heapify*: create a heap out of given array of elements
- *merge* (*union*): joining two heaps to form a valid new heap containing all the elements of both, preserving the original heaps.
- *meld*: joining two heaps to form a valid new heap containing all the elements of both, destroying the original heaps.

Inspection

- *size*: return the number of items in the heap.
- *is-empty*: return true if the heap is empty, false otherwise.

Internal

- *increase-key* or *decrease-key*: updating a key within a max heap
- *delete*: delete an arbitrary node (followed by moving last node and sifting to maintain heap)
- *sift-up*: move a node up in the tree, as long as needed; used to restore heap condition after insertion. Called "sift" because node moves up the tree until it reaches the correct level, as in a sieve.
- *sift-down*: move a node down in the tree, similar to sift-up; used to restore heap condition after deletion or replacement.

(These are pretty much common operations, details can be found online).

In your online evaluation sessional, you have to modify your offline to solve a new problem.