North South University

CSE 225: Data Structures & Algorithms

Take Home Mid Problem

Your task in this assignment is to design a generic/template data structure called *WhimsicalStorage*. It has this name because it is capable of behaving whimsically, as per its user's command. Specifically, it should be capable of behaving either as a (i) Stack, (ii) Queue, (iii) Sorted List, or a (iv) Priority Queue, depending on a parameter supplied by the user in the constructor of *WhimsicalStorage* class. For e.g. suppose the user (*i.e.*, writer of the driver.cpp file) writes the following code:

```
WhimsicalStorage<int> *st = new WhimsicalStorage('S');

//'S' means stack, 'Q': queue, 'L': sorted list, 'P': priority queue
```

Then the object `st` should behave as a stack.

The *WhimsicalStorage* class must support at least the following operations:

- *insert*: this function must behave as the *push* operation of Stack when the *WhimsicalStorage* object represents a stack; it must behave as the *enqueue* operation of a Queue or a priority queue when the object represents a queue or a priority queue, respectively; and it must behave as the *insertItem* function of the sorted list when the object represents a sorted list.

- *delete:* this function must behave as the *pop* operation of Stack when the *WhimsicalStorage* object represents a stack; it must behave as the de*queue* operation of a Queue or a priority queue when the object represents a queue or a priority queue, respectively; and it must behave as the *deleteItem* function of the sorted list when the object represents a sorted list. This function must return the deleted item.

- *firstItem:* Just like delete function, this function returns the item that would get deleted if delete is called; but unlike delete, it doesn't actually delete the item (this function is similar to the front function of the queue or top function of the stack data structure)

- *search:* for searching an item among the items stored by the *WhimsicalStorage object.*

- *edit:* for replacing one item by another; for e.g. to replace the item 5 by item 7, we shall call

  `st->edit(5,7);` inside the main function (assume that duplicate items won't ever be inserted by the user in a *WhimsicalStorage* data structure).

- *print:* for printing the items in the *WhimsicalStorage* object in the order of deletion without actually deleting any item. For e.g. if we use a *WhimsicalStorage* object as a stack and insert into it 5, 6, 3 respectively and call print function on this object, then print function should print "3, 6, 5" because that is the order of their deletions; but no item gets deleted due to print function call.

- *change:* to change the behavior of current object from one type of data structure to another. For e.g. according to our previous code `st` behaves as a stack; if we write `st->change('P');` later in the program then from that point `st` should behave as a priority queue instead.

Write the necessary files for declaring this class (*WhimsicalStorage.h),* for implementing the functions in this class (*WhimsicalStorage.cpp*), and for testing the functionality of this class (*driver.cpp*).