ICS 46 Problem Set 1

- Ruohuai Pan, 90029752
- 1. Answer: We can add a new private array (dynamically allocated, the element type stored in this array is the same as the ones stored in the main array of the stack object) called ***minimumArray*** which is to keep track of the history of the minimum value (act like a stack) , ***minimumArraySize*** as an unsigned integer to keep track of the size of the *minimumArray,* and ***minimumArrayCapacity*** to track the capacity of *minimumArray*.
  - **a. new procedures added to the public functions**
    - 1. Constructor: dynamically allocate the *minimumArray* with default size of 100, and initialize the minimumArraySize to 0.
    - 2. **push** function:
      - **Case 1: if stack object is empty, while pushing the first element into the main array of stack, the function also needs to:**
        - minimumArraySize = 1
        - set this first element to the minimumArray[minimumArraySize - 1]
      - **Case 2: if the stack object is not empty and the new element is less than the previous minimum value (which is minimumArray[minimumArraySize - 1]). While pushing the new element into the main array of stack, the function also needs to:**
        - minimumArraySize += 1
        - **Case: if the minimumArrayCapacity > minimumArraySize**

- dynamically allocate a new minimumArray with doubled size, and copy all the previous values to the new array
- set this first element to the minimumArray[minimumArraySize - 1]

- **Case 3: if the stack object is not empty and the new element is not less than the previous minimum value (which is minimumArray[minimumArraySize - 1]). We don't need to change anything else of the older function.**

- 3. **pop** function:
  - **Case 1: if stack object is empty, the minimumArray is also empty. We don't need to change anything else of the older function.**
  - **Case 2: if the stack object is not empty and the popped element is equal to the previous minimum value (which is minimumArray[minimumArraySize - 1]). While popping the new element into the main array of stack, the function also needs to:**
    - minimumArraySize -= 1
  - **Case 3: if the stack object is not empty and the popped element is not equal to the previous minimum value (which is minimumArray[minimumArraySize - 1]). We don't need to change anything else of the older function.**

- **b. define the findMin() function**
  - **Case 1: if minimumArraySize== 0, meaning that the stack is also empty.**
    - throw StackEmptyException
  - **Case 2: if minimumArraySize > 0**
    - return minimumArray[minimumArraySize - 1]

- 2. Answer: we use name *qu* to keep track of the Queue object, Object type denote the type of the Stack object

stores

- 1. push (O(1), constant time, number of elements: qu.size())
  - qu.enqueue(new lement)
- 2. pop (O(n), linear time, number of elements: qu.size())
  - for (qu.size() - 1) times:
    - Object poppedValue = qu.end()
    - qu.dequeue()
    - qu.enqueue(poppedValue)
  - qu.dequeue() // pop the top of the stack
- 3. top (O(1), constant time, number of elements: qu.size())
  - return qu.front()
- 4. size() (O(1), constant time, number of elements: qu.size())
  - return qu.size()
- 5. isEmpty() (O(1), constant time, , number of elements: qu.size())
  - return qu.size() == 0