1. In the reading, we saw how to make a Stack data structure. There is also a "bonus lecture" available about this if you would like to see it presented that way. Suppose we have a Stack that can grow indefinitely (for example, the push method has been fixed to double the size of the array when at capacity instead of throwing a StackFullException). We want to create a second Stack data structure, which I promise will always contain only comparable items (e.g., integers, strings, although you **may not** assume anything about the data other than that it is a comparable type). We also want to add a function `findMin` to the Stack interface that will return the smallest element currently in the stack. We could implement this by searching the array, but that takes time linear in the number of elements in the Stack.

   Explain how you would change the Stack data structure to allow for this function to run in $\mathcal{O}(1)$ time. If you are storing additional private member data, state what else you are storing. If you are changing existing functions `push`, `pop`, or `top` (or the constructor/size functions), explain briefly how you are changing them. Their running times must still be $\mathcal{O}(1)$; for example, you cannot search the full stack for the newest minimum value at every push and pop. You should also explain in a few sentences how the change works and how it achieves the goal.

   Your explanation should be sufficient that if, six months from now, you had to write the necessary modifications to a Stack data structure written in your favorite programming language, using *only* your written description, you could do so.

   You may assume that there will never be a duplicate item pushed to the Stack.

First of all, we define a private variable min for stack class.

then we modify the push and pop method, the psendo code is below

```
void push (int ele)
{  if (size of stack is 0)
      { min = ele;}
   else
      { if (ele < min)
           { min = 2·ele - min;}
      }
   return;
}

void pop ( )
{ if (this.front() < min)
     { min = 2·min - this.front();
       delete the front;
     }
  else
  {
     delete the front)
  }
}
```

then for findmin method, we just return the variable min in the stack class.

2. Suppose I want to implement the public member functions of a Stack (`push`, `pop`, `top`, `size`, `isEmpty`). However, instead of the private member data we had in class, I have *only* a single Queue. The Queue has unbounded capacity

   Explain how you would implement each of those functions using just that Queue. You may use $\mathcal{O}(1)$ additional space within each function. The only Queue functions you may call are enqueue, dequeue, front, size, and isEmpty.

   Your explanation should be sufficient that if, six months from now, you had to write the necessary modifications to a Stack data structure written in your favorite programming language, using *only* your written description, you could do so.

   For each function, give the running time in $\mathcal{O}$ notation in terms of n, the number of elements currently in the stack.

Assume the Queue is called Q

```
1. bool   isEmpty()
   { return Q.isempty();}            O(1)

2. unsigned  size()

   { return Q.size();}               O(1)

3. T  top()
   { return Q.front();}              O(1)

4. void  push(T ele)

   { Q.enqueue(ele);}                O(1)

5. void  pop()

   { for(i=0; i<n; i++)              O(n)
      { if (i != n-1)
```

```
        { Q.enqueue( Q.dequeue() );}
        else
        { Q.dequeue();}
     }
}
```