DATA STRUCTURE #4

yes

#7

① While (!isEmpty (Q2))

```
    I ← rear (Q1)
    while ( front (Q1)! = x
    {   if (front (Q1)! == front (Q2))
            Q1.dequeue
        else
            Q1.enqueue (Q1.dequeue)
    Q2.dequeue
```

There are 2 "while" loop, therefore, the runtime complexity of the algorithm is $O(n^2)$. The algorithm runs through Q_2 as long as its not empty. In the inner loop, I runs through Q_1 and removes the first of its values if its also in Q_2. If not, Q_1 first value goes to the end. After the inner loops finishes, Q_2's first value is removed and the loops runs again, until all common values between Q_1 & Q_2 are removed of Q_1.

② ⓐ A serie of push & push & pop can produce this result.

```
S = Stack create()
push (S,1)
push (S,2)
push (S,3)
pop (S) → 3
push (S,4)
pop (S) → 4
pop (S) → 2
push (S,5)
pop (S) → 5
pop (S) → 1
push (S,6)
pop (S) → 6
```

the serie we got: 6,1,5,2,4,3

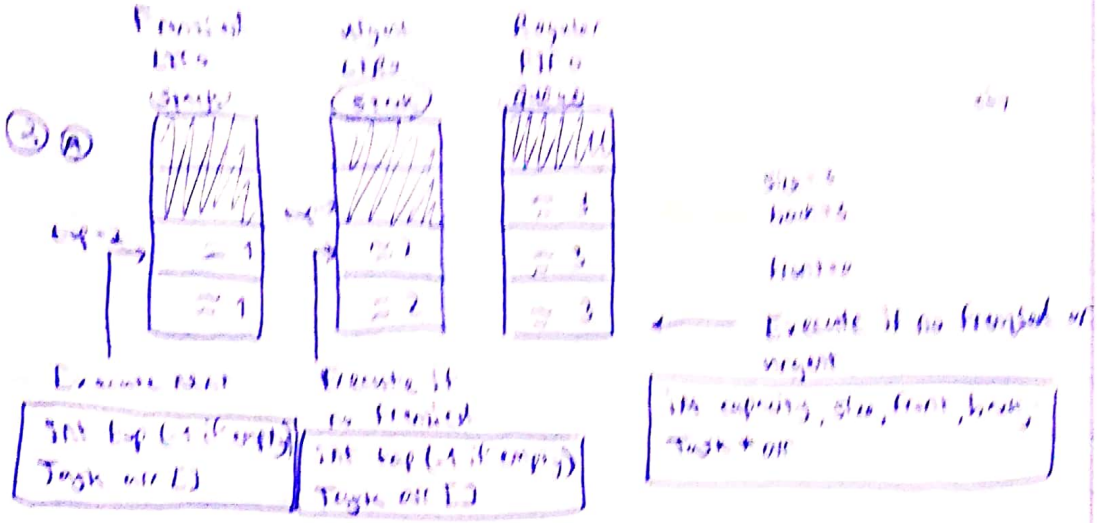
ⓑ Its not possible!

```
S = Stack (create()
push (S,1)
push (S,2)
push (S,3)
pop (S) → 3
pop (S) → 2
push (S,4)
push (S,5)
pop (S) → 5
```

4 is before the 1, so we cannot pop the 1, Therefore, we cannot get the serie:

3,5,1,4,6

(2) (A)



(B)
```
void Insert (Task new)
{   if(new.code == 1)
    {   Frenzied.push(new);
        return;
    }
    if(new.code == 2)
    {   Urgent.push(new);
        return;
    }
    if(new.code == 3)
    {   Regular.enqueue(new);
    }
}
```

Stack functions:

```
void push (Task new)
{   arr[++top] = new;
}
Task Pop()
{
  Task t = arr[top--];
  return t;
}
bool isEmpty()
{
  return (top<0);
}
```

(C)
```
Task Execute()
{   if(!Frenzied.isEmpty())
        return Frenzied.pop();
    if(!Urgent.isEmpty())
        return Urgent.pop();
    if(!Regular.isEmpty())
        return Regular.dequeue();
    return NULL;
}
```

Queue functions:

```
void enqueue (Task new)
{   back = (back+1) % capacity;
    array[back] = new;
    ++size;
}
Task dequeue()
{
  Task t = array[front];
  front = (front+1) % capacity;
  --size;
  return t;
}
bool isEmpty()
{   return (size==0);
}
```