1. Queue. — FIFO. (BFS).

Classic problem: Tree print out by levels

Sliding Window problems.

{ p_2 p_3 p_4 }.

head tail.

2. Stack — LIFO. (DFS).

operations: push(); pop(); top().

4 top of the stack (LI)

3

2

1 bottom of the stack. (FI)

} like a box

Q1. Sort w/ 2 stacks.

Soln: Stack 1: store unsorted numbers.

Stack 2: left | right

└ store all sorted numbers

└ buffer to find the current smallest element.

while (stack2.size > 0 && stack2.peek() >= globalMIN)

move stack2.top back to stack 1.

assumption: no duplicates.

Q1.2. Acumption: duplicates ✓. (use counter).

{ record times when find the globalMIN. count the times

```
while ( ! input.isEmpty() )
    set curMin. & count = 0.
    while ( ! input.isEmpty() )
        cur = input.pollfirst();
        compare w/ curMin.
        update count & curMin.
        offerfirst to buffer.
    while ( ! buffer.isEmpty() && buffer.peekfirst() >= MIN)
        buffer.pollfirst();
        check if != MIN:
            offerfirst back to input.
    while ( count-- > 0 )
        offerfirst to buffer.
while( ! buffer.isEmpty() ).
    input.offerfirst ( buffer.pollfirst() );
```

Q2. Implement a Queue using 2 stacks.

Stack 1: store new elements when adding

Stack 2: pop old elements

push – O(1). just push to S1.

pop – Amortized O(1) Worst O(n).

if S2 is Empty:

move from S1 to S2.

return S2.pop()

Amortized Analysis:

if S2 is Empty:

1st poll: n* pop from S1 + n* push from S2 + 1 pop

2nd poll: 1

3rd poll: 1

⋮

nth poll: 1

Amortized Poll: $\frac{2n+1+1*(n-1)}{n} = \frac{3n}{n} = O(1)$

Q3. Stack w/ min():

normal two stacks:

Stack: regular operations.

mins: store min each time pushing.

improved two stacks:

Stack: regular operations.

mins: int[]. 1st: min. 2nd: count.

when pop, decrement count.

if count == 0, pop mins.

when push, if equals curMIN, increment count