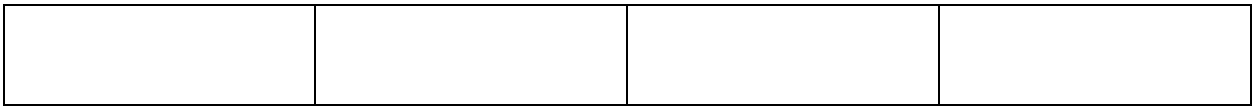
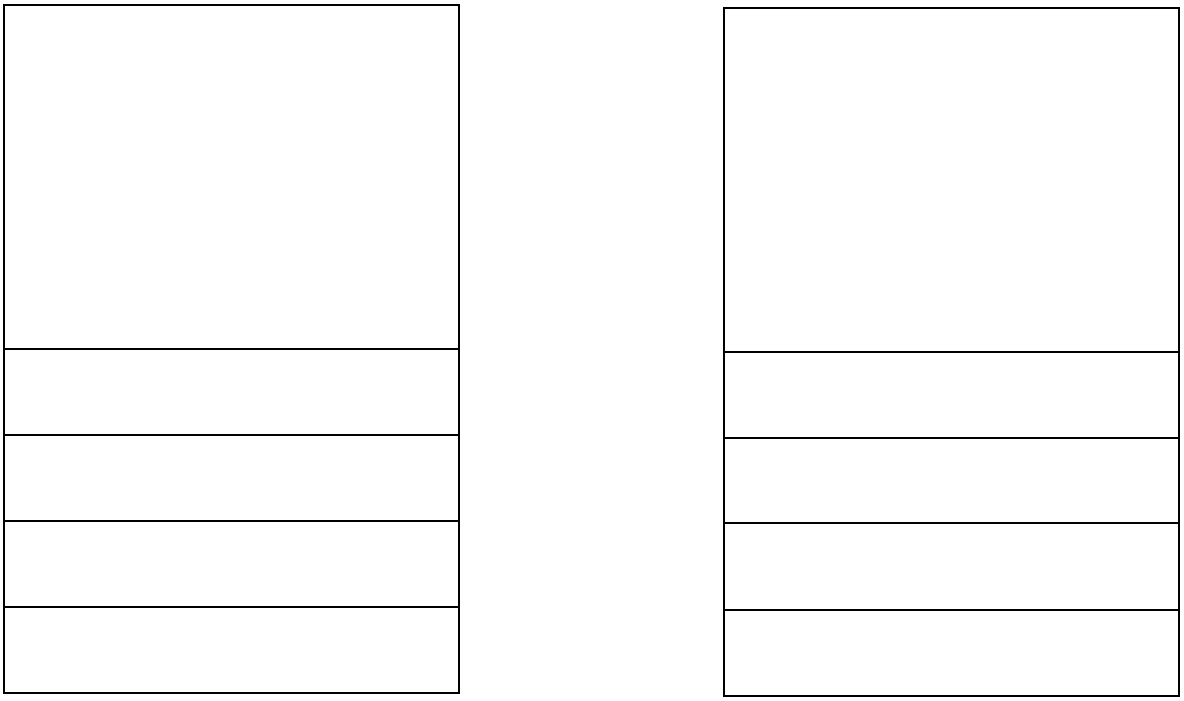
Queue Implementation

1) Using two stacks

Queue





Stack 1(enqueue) Stack 2(dequeue)

1st method

- Dequeue costly

2nd method

Enqueue costly

1ST METHOD

Code part

```
#define N S
Int S1[N],S2[N];
Int top1=-1, top2=-1;
Int count=0;
void enqueue(int x){
  push1(x);
count++;
}
```

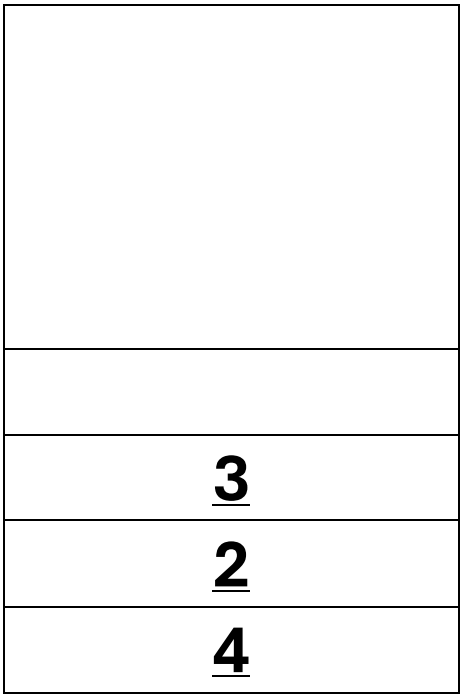
void main(){

enqueue(4;

enqueue(2);

enqueue(3);

}



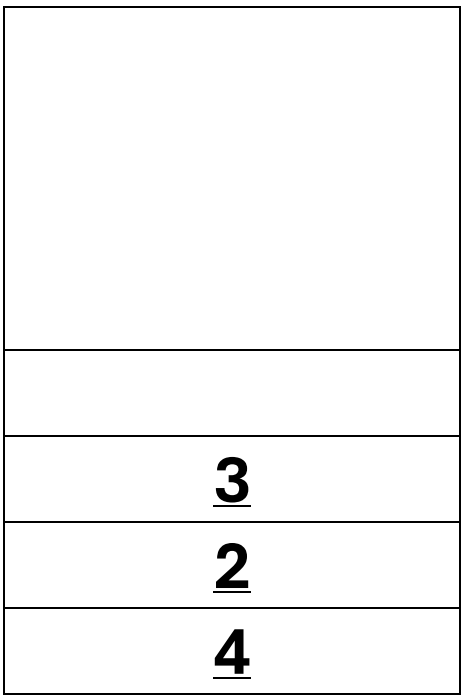
Stack 1(enqueue)

```
void dequeue(){
    int k;
    if(top=-1 && top2=-1){
    {print("Queue is Empty);
    }
Else{
    for(i=0;i<count;i++){
     b=pop1();
    push2(a);
    }
```

Void main(){

Dequeue();

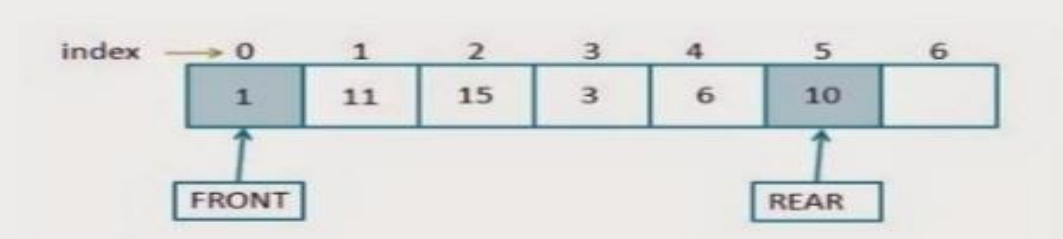
}



Stack2(dequeue)

2) Using an array

Use an array to store the elements of the queue. This array will have a fixed size, which determines the maximum number of elements the queue can hold.



- **Front Index**: Keeps track of the position of the first element in the queue.

- **Rear Index**: Keeps track of the position where the next element will be inserted.

- Initialize the array with the given capacity.
- Set the front index to 0 and the rear index to -1.
- Initialize the size of the queue to 0.

3) Using linked list

Implementing a FIFO queue using a linked list in Java involves using a Node class to represent each element in the queue and a LinkedList class to manage the queue operations. This approach leverages the natural properties of linked lists to efficiently handle enqueue (add to the end) and dequeue (remove from the front) operations

front: Points to the first node in the queue.
rear: Points to the last node in the queue.
size: Tracks the number of elements in the queue..