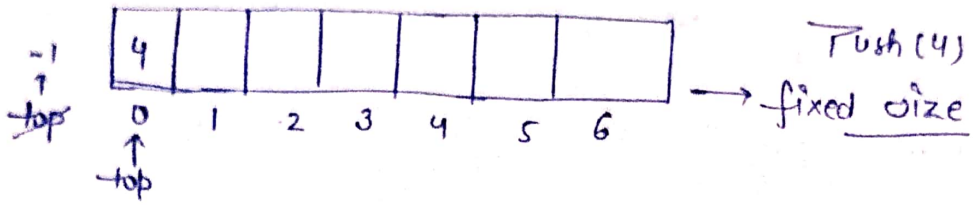
[Stack & Queue]

Stack using Arrays :-

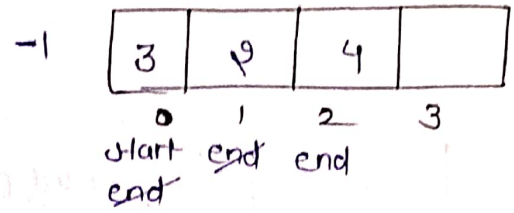



```
→ class stImpl {
      top=-1, int st[10];
      push(x) {
          if(top>=10) return;
            top++;                    → O(1)
            st[top]=n;
      }
      top() {
          if (top==-1) return;
          return st[top];             → O(1)
      }
      pop() {
          if (top==-1) return;
          top--;
      }                               → O(1)
      size() {
          return top+1;               → O(1)
      }
}
```

TC:- O(1)

SC:- O(10)

Queue using arrays :- (FIFO)



Push(3)
Push(2)
Push(4)

⇒ class queueImpl {

size=10, q[size], cursize=0, start = end = -1

push(x) {

if(cursize == size) return;

if(cursize == 0) {

start = end = 0;

}

else

end = (end+1) % size

q[end] = n, cursize += 1

}

TC :- O(1)

SC :- O(4)

pop() {

if(cursize == 0) return;

else { el = q[start]

if(cursize == 1)

start = end = -1

else

start = (start+1) % size

~~return el;~~ cursize--; }

}

top() {

if(~~start~~cursize == 0) return;

return q[start];

}

size() {

return cursize;

}

}

Stack using Linked List :-

```
Class st {
        Node * temp; size=0;
        Push(x) {
            Node * temp = new Node(x);
            temp → next = top
            top = temp
            size = size + 1
        }
        Pop() {
            temp = top
            top = top → next
            delete temp
            size -= 1;
        }
        top() {
            return top → data
        }
        size() {
            return size;
        }
}
```

TC :- O(1)

SC :- O(N)

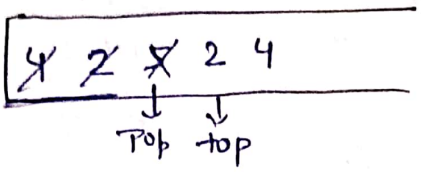
Queue using Linked List :-

```
class queueLL {

    Node * start, end;
    size=0

    push(x) {

        Node * temp = new Node (x);
        if(start == null)   start = end = temp
        else   end → next = temp
        size++;

    }

    pop() {

        if(start == null)   return;
        temp = start
        start = start → next
        delete temp
        size--;

    }

    top() {

        if (start == null)   return -1;
        return   start → val;

    }

    size() {

        return size;

    }

}
```

TC :- O(1)

SC :- O(N)

Implement stack using queue :-

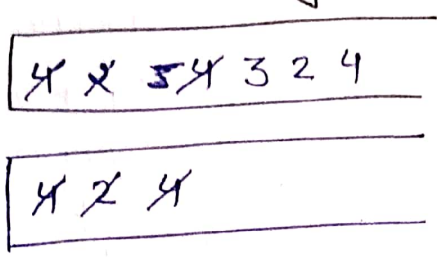


Push(4)
Push(2)
Push(5)
Pop()
top()

```
Class stack {
    queue <int> q;
    Push(x) {
        S = q.size
        q.push(x)
        for(i=1 → s) {
            q.push(q.top(1))
            q.pop()
        }
    }
    pop() {
        q.pop();
    }
    top {
        return q.top();
    }
}
```

TC :- O(1)
SC :- O(N)

Implement queue using stack :- (Approach-1)



Push(4)
Push(2)
Push(3)

- $S_1 \rightarrow S_2$
- $x \rightarrow S_1$
- $S_2 \rightarrow S_1$

```
class queue {
    stack<int> s1, s2;
    push(x) {
        while(s1.size()) {
            s2.push(s1.top())
            s1.pop()
        }
        s1.push(x)
        while(s2.size()) {
            s1.push(s2.top())
            s2.pop()
        }
    }

    top() {
        s1.top()
    }

    pop() {
        s1.pop()
    }
}
```

TC :- O(2N)

SC :- O(2N)

Implement ~~stack~~ queue using stack :- (Approach 2)

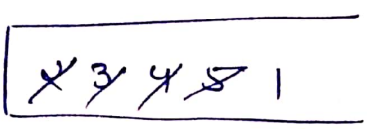
⇒ class Queue {

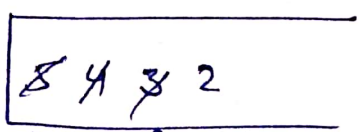
```
    push(x) {
        x → S1;
    }

    pop() {
        if(S2 != empty)
            S2.pop()
        else
            S1 → S2
            S2.pop
    }

    top() {
        if(S2 != empty)
            S2.top
        else {
            S1 → S2
            S2.top
        }
    }
}
```

TC :- O(N)

SC :- O(2N)

⇒





top top pop top
pop pop

Push(2)
Push(3)
Push(4)
Push(5)
top → 5
pop → 5
Push(1)
top → 4
top → 3

Balanced Parantheses :-

{ ( ) [ ] }

```
func (s) {
    stack s;
    for(int i=0 ⟶ n-1) {
        if(s[i]=='(' or s[i]=='[' or s[i]=='{')
            st.push(s[i]);
        else {
            if(st.empty()) return false;
            char ch = st.top()  st.pop()
            if(s[i]==')' && ch=='(')
                or
            if(s[i]==']' && ch=='[')
                or
            (s[i]=='}' && ch=='{')
                continue
            else
                return false
        }
    }
    return st.empty();
}
```

TC :- O(N)

SC :- O(N)

* It stores open braces & whenever a closing braces found it compare it with top of the stack.

minStack :-

1) you can use another stack to track the min element

2) TC :- O(1) SC :- O(2N)

3) 2 x val - prevmini = newVal

10 < 12

val < mini

val - mini < 0

val + val - mini < val + 0

2 x val - mini < val

newval < val → always lesser than the val

→ code :-

```
class MinStack {
        stack<int> st
        push(val) {
            if(st.empty()) {
                mini = val
                st.push(val)
            }
          else {
            if(val > mini) st.push(val)
            else {
                st.push(2*val - mini)
                mini = val;
            }
        }
    }
```

```
pop(){
    if(st.empty()) return;
    x = st.top();
    st.pop();
    if(x < mini){
        mini = (2x mini - x)
    }

top(){
    if(st.empty()) return -1;
    x = st.top();
    if(mini < x) return x
    return mini;
}

getMin(){
    return mini;
}

}
```

⇒ TC :- O(1)

SC :- O(N)