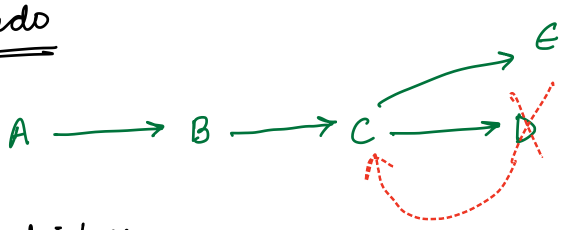
LIFO :- Last In first out

undo / redo



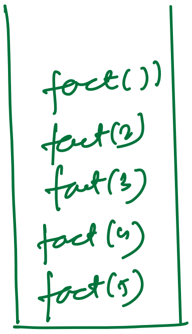
browser history

⇐ ⇒

Stack → DS which follows LIFO behaviour

```
int fact ( int n)
{
        if( n==0 || n==1) return 1;

        return n* fact(n-1);
}
```

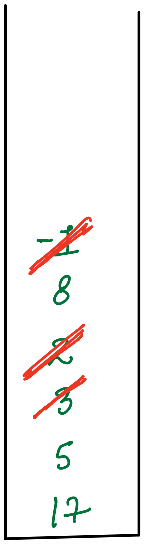
fact(5)
↓
5*fact(4)
↓
4*fact(3)
↓
3*fact(2)
⋮



operations by stack :-

push(x) :- insert the data x at top
pop() :- delete the data from top
top() :- access to top-most element
isEmpty() :- if stack is empty

push	17
push	5
push	3
push	2
pop	
top	→ 3
pop	
push	8
push	-1
pop	
top	→ 8

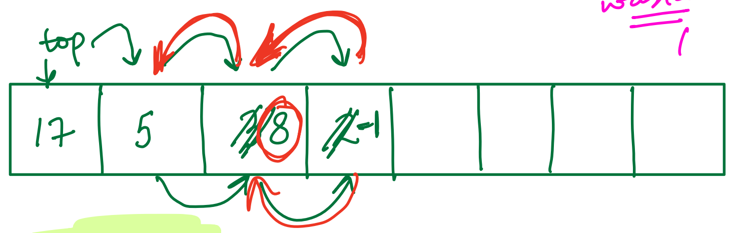


Implementation of stack : → linear

fixed size
↓
waste

- arrays

push	17
push	5
push	3
push	2
pop	
top	
pop	
push	8
push	-1
pop	
top	



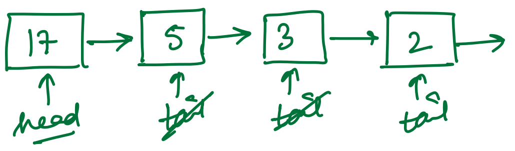
top = -1;

```
push( x )
{
    top++;
    arr[top] = x;
}

pop( )
{  top--;
}

top( )
{
    return arr[top];
}

isempty()
{  if (top==-1)
      return true;
   return false;
}
```

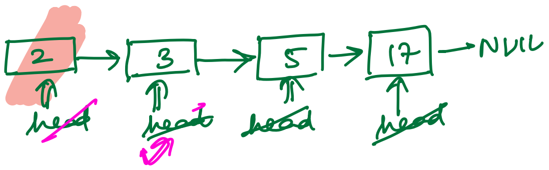
- linked list

push	17
push	5
push	3
push	2
pop	
top	
pop	
push	8
push	-1
pop	
top	

head = NULL;   tail;

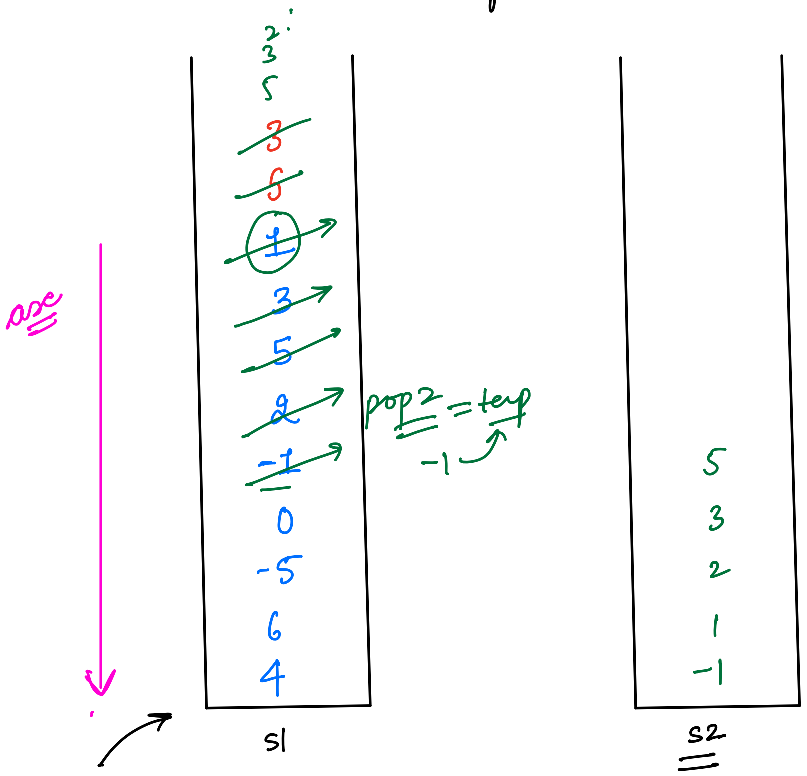


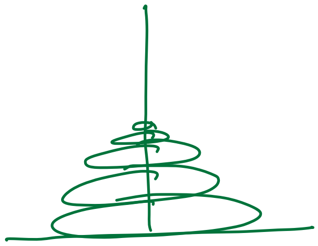
insert always at head!



head = head.next;

- Sort the stack using another stack.





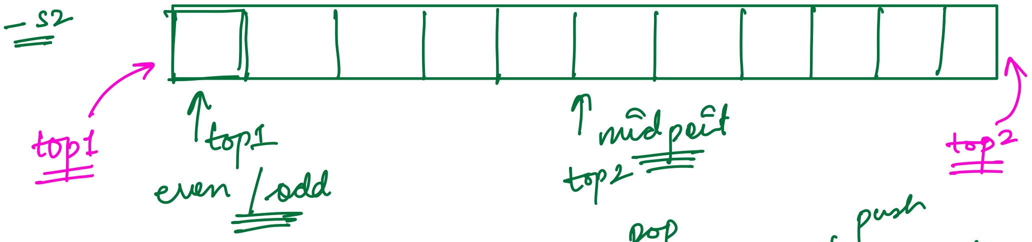
S2 → descend order
↓
Always maintain your s2 in desc order

T.C: $O(n^2)$

```
while( !s1.empty() )
{
        int x = s1.top();
            s1.pop();
                         !s2.empty() &&
    while ( x < s2.top())
    {
          temp = s2.top()
                s1.push(temp);
              s2.pop();
    }
    s2.push(x);
}  // move all the element back to s1.
```

Q Implement 2 stacks using a single arrays.

push x – s1
push x' – s2



```
pop
{
   top++;
}
```

```
{ push
   top--;
}  arr[top]=x
```

push → s1 → x
push → s2 → x
pop – s1
pop s2
top s1

Q. Implement a stack — one more operation

all : O(1)

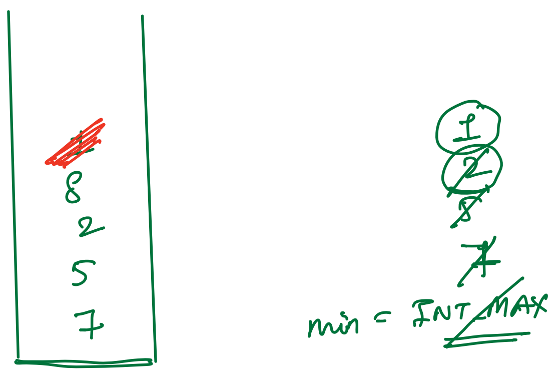
T.C : O(1) ⟸ getMin() } → return min of all elements currently present in stack.

7 5 2 getMin() 8 getMin() 1 getMin()

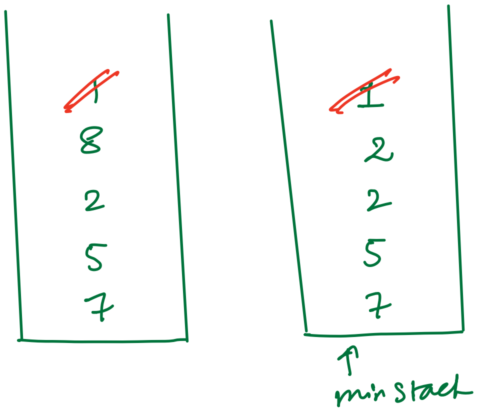
↑ push

⇓ 2 ⇓ 2 ⇓ 1

pop() getMin()

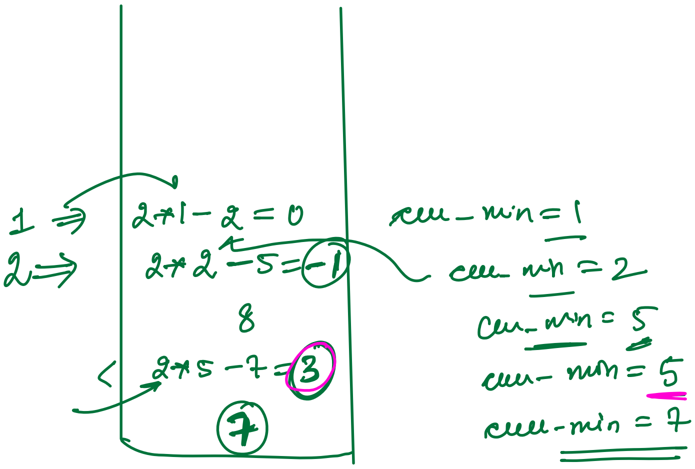


```
getMin()
{
    return s2.top();
}
```



```
push( )
{
    s1.push(x);
    minStack.push( );
    curr_min =
        min(curr_min, x);
}
```

$prev\_min = 2 * curr\_min - s.top();$



getMin()

```
{
    return curr_min;
}
```

$2 * x - curr\_min$

↑ increaj delta

$x < curr\_min$

$2x - curr\_min < x$

```
push(x)
{
    if (x < curr_min)
    {
        s.push(2*x - curr_min);
        curr_min = x;
    }
    else { s.push(x); curr_min = min(cu, x); }
}
```

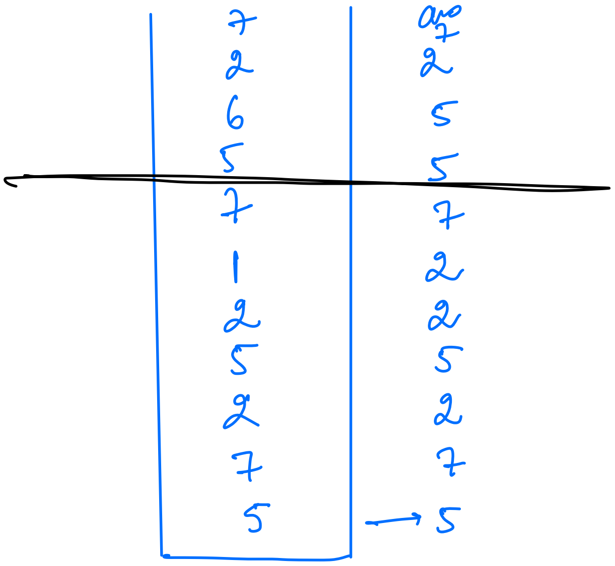
pop()

```
{
    if (s.top() < curr_min)
    {   int prev_min;
        prev_min = 2*curr_min
                   - s.top();
        curr_min = prev_min;
    }
    s.pop();
}
```

Q Maximum frequency stack → operation

O(1) → return the element with max freq & if two elements have same freq return whichever is closest to the top

5 7 2 5 2 1 7 5 6 2 7



pop will cause issues

HM + stack
↓
freq