DAY-31

①. Sort a stack using recursion

Algorithm:

```
→ to sort stack elements:
  sortStack (stack S)
      if stack is not empty:
          temp= pop(S)
          sortStack (S)
          sortedInsert(S, temp)
→ to insert elements in
  sorted order.

    sortedInsert (stack S, element):
      if stack is empty or element>topele
          push(S, elem)
      else:
          temp = pop(S)
          sortedInsert(S, element|
          push(S, temp).
```

* hold all the values in function call stack until the stack becomes empty.
→ then insert all held items one by one in sorted order.

*here, the sorted order is important.

code part:-

```
def    sortedInsert (s, element):
    if  len(s) ==0  or element > s[-1]:
         s. append(element)
         return.
    else:
         temp . pop( )
         sortedInsert (s, element).
          s. append (temp)

def  sortStack(s):

     if len(s) !=0 :

          temp = s.pop( )

        sortStack (s )

        sortedInsert (s, temp)
```

(2). Implement k stacks in a single array.

The idea is to use extra arrays for efficient implementation of k stacks in an array.

That two extra arrays are:

i) top[] ⇒ it's of size k and stores indexes of top elements in all stacks.

(ii) next[] ⇒ it's of size n and stores indexes of next item for the items in array arr[].

* arr[] is the actual array that stores k stacks.

→ together with k stacks, a stack of free slots in arr[] is also maintained.

The top of this stack is stored in a variable 'free'.

→ all entries in the top[] are initialized as -1 to indicate that all stacks are empty.

→ All entries next[i] are initialized as i+1 because all slots are free initially and pointing to next slot

→ top of free stack, 'free' is initialized as 0.

Code part Implementation:-

```
class kstacks:
        .....
        ......
    def push(self, item, sn):
        if self.isFull():
            print ("stack overflow")
            return
        insert_at = self.free
        self.free = self.next[self.free]

        self.arr[insert_at] = item
        self.next[insert_at] = self.top[sn]

        self.top[sn] = insert_at

    def pop (self, sn):
        if self.isEmpty(sn):
            return None
        top_of_stack = self.top[sn]
        self.top[sn] = self.next[self.top[sn]]

        self.next[top_of_stack] = self.free
        self.free = top_of_stack
        ~~ret~~
        return self.arr[top_of_stack]
```