WEEK2

```
# def bubble_sort(arr):
#     for i in range(len(arr)):
#         for j in range(len(arr)-1):
#             if arr[j]>arr[j+1]:
#                 arr[j],arr[j+1]=arr[j+1],arr[j]
#     print(arr)
#
# bubble_sort([2,3,4,5,6,7,1,3,8,2,11])



# def insertion_sort(arr):
#     for i in range(len(arr)):
#         for j in range(i):
#             if arr[i]<arr[j]:
#                 arr[i],arr[j]=arr[j],arr[i]
#     print(arr)
#
# insertion_sort([4,3,2,1,4,55,3,2,9,1])



# def selection_sort(arr):
#     for i in range(len(arr)):
#         min=i
#         for j in range(i,len(arr)):
#             if arr[j]<arr[min]:
```

```
#               min=j
#               arr[i],arr[min]=arr[min],arr[i]
#     print(arr)
#
# selection_sort([4,3,21,3,45,43,32,0])

# def binary(high,low,arr,x):
#     try:
#         mid=((high+low)//2)
#         if arr[mid]==x:
#             print('found')
#         elif arr[mid]>x:
#             return binary(mid-1,low,arr,x)
#         else:
#             return binary(high,mid+1,arr,x)
#     except:
#         print('not found')
#
# a=[1,2,3,4,5]
# x=9
# binary(a[-1],a[0],a,x)

# def quick_sort(arr):
#     if len(arr)<=1:
#         return arr
#     pivot=arr[-1]
#     left=[]
#     right=[]
#     mid=[]
#     for x in arr[:-1]:
#         if x>pivot:
#             right.append(x)
#         elif x<pivot:
#             left.append(x)
#         else:
#             mid.append(x)
#     return quick_sort(left)+mid+[pivot]+quick_sort(right)
#
# a=[3,2,13,21,0]
# i=quick_sort(a)
# print(i)
```

WEEK 3

```
def merge_sort(arr):
    if len(arr)<=1:
        return arr
    else:
        mid=len(arr)//2
        left=merge_sort(arr[:mid])
        right=merge_sort(arr[mid:])
        return merge(left,right)
def merge(left,right):
    i,j=0,0
    l=[]
    while(i<len(left) and j<len(right)):
        if left[i]<right[j]:
            l.append(left[i])
            i+=1
        else:
            l.append(right[j])
            j+=1
    while (i<len(left)):
        l.append(left[i])
        i+=1
    while (j<len(right)):
        l.append(right[j])
        j+=1
    return l
arr=[42,78,64,31,43]
k=merge_sort(arr)
print("Sorted array:",k)

"Counting Sort"
# def counting_sort(arr):
#     m=max(arr)
#     count_array=[0]*(m+1)
#     output_array=[0]*(len(arr))
#     for i in arr:
```

```
#        count_array[i]+=1
#    for i in range(1,m+1):
#        count_array[i]+=count_array[i-1]
#    for i in range(len(arr)):
#        output_array[count_array[arr[i]]-1]=arr[i]
#        count_array[arr[i]]-=1
#    return output_array
# arr=[3,4,6,5,0,1,2,3]
# print(counting_sort(arr))

"Radix Sort"
# def radix_sort(arr):
#    x=1
#    m=max(arr)
#    while m/x>=1:
#        counting_sort(arr,x)
#        x*=10
#    return arr

# def counting_sort(arr,x):
#    output=[0]*len(arr)
#    count_array=[0]*10
#    for i in arr:
#        index=(i//x)%10
#        count_array[index]+=1
#    for i in range(1,10):
```

```
#        count_array[i]+=count_array[i-1]
#    i=len(arr)-1
#    while i>=0:
#        index=(arr[i]//x)%10
#        output[count_array[index]-1]=arr[i]
#        count_array[index]-=1
#        i-=1
#    for i in range(len(arr)):
#        arr[i]=output[i]
# arr=[189,67,34,98]
# print(radix_sort(arr))
```

WEEK4&5

```
"Implement stacks using list"
class Stack():
    def _init_(s):
        s.stack=[]
    def push(s,e):
        s.stack.append(e)
        return
    def pop(s):
        if s.stack:
            return s.stack.pop()
        else:
            return "Stack is empty"
    def is_Empty(s):
        return len(s.stack)==0
    def peek(s):
        return s.stack[-1]
    def display(s):
        return s.stack

def menu():
    print("Choose an option:")
    print("1.Push()\n2.Pop()\n3.is_Empty()\n4.Peek()\n5.Display\n6.Exit")
    s=Stack()
    while True:
        n=int(input("Enter option number"))
        if n==1:
            e=int(input("Enter element to push: "))
            s.push(e)
        elif n==2:
            print(s.pop())
        elif n==3:
            print(s.is_Empty())
        elif n==4:
            print(s.peek())
        elif n==5:
            print(s.display())
        elif n==6:
            return False
        else:
```

```
        print("Invalid option number")
menu()


"Stack implementation using numpy"
# import numpy as np
# class Stack():
#   def _init_(s):
#       s.stack=np.array([])
#   def push(s,e):
#       s.stack=np.append(s.stack,e)
#       return "Element has been pushed"
#   def pop(s):
#       if len(s.stack)==0:
#           return "Stack is empty"
#       else:
#           end=s.stack[-1]
#           s.stack=s.stack[:-1]
#           return end
#   def peek(s):
#       return s.stack[-1]
#   def is_Empty(s):
#       return len(s.stack)==0
#   def display(s):
#       return s.stack
# def menu():
#   print("Choose an option:")
#    print("1.Push()\n2.Pop()\n3.is_Empty()\n4.Peek()\n5.Display\n6.Exit")
#    s=Stack()
#    while True:
#        n=int(input("Enter option number:"))
#        if n==1:
#            e=int(input("Enter element to push: "))
#            s.push(e)
#        elif n==2:
#            print(s.pop())
#        elif n==3:
#            print(s.is_Empty())
#        elif n==4:
#            print(s.peek())
#        elif n==5:
#            print(s.display())
#        elif n==6:
#            return False
#        else:
#            print("Invalid option number")
# menu()

"Stack using ADT"
# class Stack():
#    def _init_(self,capacity):
#        self.capacity=capacity
#        self.stack=[0]*self.capacity
#         self.top=-1
#     def push(self):
#         if self.top==self.capacity-1:
#             return "Stack is full"
#         else:
#             e=int(input("Enter element to push: "))
#             self.top=self.top+1
#             self.stack[self.top]=e
#             return "Element has been pushed"
#     def pop(self):
#         if len(self.stack)<self.capacity:
#             end=self.stack[self.top]
#             self.top-=1
#             self.stack=self.stack[:self.top]
#             return end
#         else:
#             return "Stack is Empty"
#     def is_Empty(self):
#         return len(self.stack)==0
#     def peek(self):
#         return self.stack[self.top]
#     def display(self):
#         return self.stack
# def menu():
#     capacity=int(input("Enter capacity of stack"))
#     print("Choose an option:")
#     print("1.Push()\n2.Pop()\n3.is_Empty()\n4.Peek()\n5.Display\n6.Exit")
#     s=Stack(capacity)
#     while True:
#         n=int(input("Enter option number:"))
#         if n==1:
#             print(s.push())
#         elif n==2:
#             print(s.pop())
#         elif n==3:
#             print(s.is_Empty())
#         elif n==4:
#             print(s.peek())
#         elif n==5:
#             print(s.display())
#         elif n==6:
#             return False
#         else:
#             print("Invalid option number")
# menu()

"Valid parenthesis"
# def parenthesis(str):
#     s=Stack()
#     for i in str:
#         if i=="(":
#             s.push(i)
#         elif i==")":
#             if s.stack and s.stack[-1]=="(":
#                 s.pop()
#             else:
#                 s.push(i)
#                 break
#     if s.stack:
#         print("Invalid")
#     else:
#         print("Valid")
# str=input("Enter:")
# parenthesis(str)

"Infix to postfix expression"
# def operators(char):
#     if char=="+" or char=="-":
#         return 1
#     elif char=="*" or char=="/":
#         return 2
#     else:
#         return -1

"postfix to arithematic"
# def arithematic(n):
#     s=Stack()
#     result=[]
#    for i in n:
#        if len(result)==0 and (i=="+" or i=="-" or i=="*" or i=="/"):
#            right=s.pop()
#            left=s.pop()
#            result.append(left+i+right)
#        elif i=="+" or i=="-" or i=="*" or i=="/":
#            result.append(i+s.pop())
#        else:
#            s.push(i)
#    for i in result:
#        print(i,end="")
# n=input("Enter expression:")
# arithematic(n)


"Postfix Eval"
# def arithematic(n):
#    s=Stack()
#    result=[]
#    for i in n:
#        if (i=="+" or i=="-" or i=="*" or i=="/"):
#            right=int(s.pop())
#            left=int(s.pop())
#            if i=="+":
#                s.push(left+right)
#            elif i=="-":
#                s.push(left-right)
#             elif i=="*":
#                 s.push(left*right)
#             elif i=="/":
#                 s.push(left/right)
#         else:
#             s.push(i)
#     if len(s.stack)==1:
#         print(s.stack[-1])
#     else:
#         print("Invalid")
# n=input("Enter expression:")
# arithematic(n)

def infix_to_postfix(expression):
    precedence = {'+': 1, '-': 1, '*': 2, '/': 2, '^': 3}  # Define operator precedence
    output = ""  # Output as a string
    operators = []  # Stack for operators

    def precedence_of(op):
        return precedence.get(op, 0)

    def is_operator(c):
        return c in precedence

    for char in expression:
        if char.isalnum():  # If the character is an operand (alphanumeric), add it to the output
```

```
            output += char
        elif char == '(':
            operators.append(char)
        elif char == ')':
            while operators and operators[-1] != '(':
                output += operators.pop()
            operators.pop()  # Remove the '(' from the stack
        elif is_operator(char):
            while (operators and operators[-1] != '(' and
                   precedence_of(char) <= precedence_of(operators[-1])):
                output += operators.pop()
            operators.append(char)
        else:
            raise ValueError(f"Unknown character: {char}")

    # Pop all remaining operators in the stack
    while operators:
        output += operators.pop()

    return output

# Example usage
infix_expr = "a*k+b-l*j"
postfix_expr = infix_to_postfix(infix_expr)
print(f"Infix: {infix_expr}")
print(f"Postfix: {postfix_expr}")
```

```
def valid(n,B):
    j=0
    stack=[]
    for i in range(1,n+1):
        stack.append(i)
        while stack and stack[-1]==B[j]:
            stack.pop()
            j+=1
    return True if not stack else False
def permutations(arr):
    stack=[]
    stack.append(([],arr))
    result=[]
    while stack:
        current,remaining=stack.pop()
        if not remaining:
            if valid(len(arr),current):
                result.append(current)
        for i in range(len(remaining)):
            new_current=current+[remaining[i]]
            new_remaining=remaining[:i]+remaining[i+1:]
            stack.append((new_current,new_remaining))
    return result
permutations([1,2,3])
```