In [7]:

```
l=list(map(int,input().split()))
n=len(l)
for i in range(n-1):                          #Bubble sort-------Swapping side by side
    for j in range(n-i-1):
        if l[j]>l[j+1]:
            l[j],l[j+1]=l[j+1],l[j]
print(l)
```

```
23 56 2 1 78 90
[1, 2, 23, 56, 78, 90]
```

In [13]:

```
#selection sort-------find min and swap
l=list(map(int,input().split()))
n=len(l)
for i in range(n):
    min=i
    for j in range(i+1,n):
        if l[j]<l[min]:
            min=j
    l[i],l[min]=l[min],l[i]
print(l)
```

```
23 56 2 1 78 90
[1, 2, 23, 56, 78, 90]
```

In [17]:

```
#Insertion sort---
l=list(map(int,input().split()))                        #2,34,4,1
n=len(l)                                                 #2,34,4,1
for i in range(1,n):
    a=l[i]
    j = i - 1
    while j >= 0 and a < l[j]:
        l[j + 1] = l[j]
        j -= 1
    print(l)
    l[j + 1] = a
print(l)
```

```
12 3 45 1 56
[12, 12, 45, 1, 56]
[3, 12, 45, 1, 56]
[3, 3, 12, 45, 56]
[1, 3, 12, 45, 56]
[1, 3, 12, 45, 56]
```

In [32]:

```
#merge sort

def mergesort(arr):
    if(len(arr)>1):
        r=len(arr)//2
        L=arr[:r:]
        M=arr[r::]
        mergesort(L)
        mergesort(M)
        i=j=k=0
        while(i<len(L) and j<len(M)):
            if(L[i]<M[j]):
                arr[k]=L[i]
                i+=1
            else:
                arr[k]=M[j]
                j+=1
            k+=1
        while(i<len(L)):
            arr[k]=L[i]
            i+=1
            k+=1
        while(j<len(M)):
            arr[k]=M[j]
            j+=1
            k+=1
arr=list(map(int,input().split()))
mergesort(arr)
print(arr)
```

```
12 3 2
[2, 3, 12]
```

In [1]:

```
#quicksort
arr=list(map(int,input().split()))
def partition(l, r, nums):

def quicksort(l, r, nums):

n=len(arr)
quicksort(0,n-1,arr)
print(arr)
```

```
12 4 2
[2, 4, 12]
```

In [48]:

```
#class
class circle:
    pi=3.14
    def __init__(self,radius):
        self.r=radius
    def cir(self):
        print(2*self.pi*self.r)
c1=circle(12)
c1.cir()
```

```
75.36
```

In [1]:

```
#single linked list
class node:
    def __init__(self,value):
        self.data=value
        self.next=None
class linkedlist:
    def __init__(self):
        self.head=None
        self.tail=None
    def insertionatbegin(self,value):
        newnode=node(value)
        if self.head==None:
            self.head=newnode
            self.tail=newnode
        else:
            newnode.next=self.head
            self.head=newnode
    def insertionatend(self,value):
        newnode=node(value)
        if self.head==None:
            self.head=newnode
            self.tail=newnode
        else:
            self.tail.next=newnode
            self.tail=newnode
    def printing(self):
        temp=self.head
        print("---data----")
        while(temp!=None):
            print(temp.data,end=" ")
            temp=temp.next
        print("\n---data----")
    def insertat(self,value,pos):
        newnode=node(value)
        i=0
        temp=self.head
        while(i!=pos and temp.next!=None):
            temp=temp.next
            i+=1
            if(temp.next==None):
                break
        if(temp.next==None):
            print("Not possible")
        else:
            newnode.next=temp.next
            temp.next=newnode
    def search(self,ele):
        temp=self.head
        flag=0
        while(temp.data!=ele):
            if(temp.next==None):
                flag=1
                break
            else:
                temp=temp.next
        if flag==1:
            return 0
        else:
            return temp
    def insertaftele(self,ele,value):
        newnode=node(value)
        x=self.search(ele)
        if not x:
            print("Not possible")
        else:
            newnode.next=x.next
            x.next=newnode
    def deletebegin(self):
        temp=self.head
        self.head=self.head.next
    def deleteend(self):
        temp=self.head
        while(temp.next.next!=None):
            temp=temp.next
        self.tail=temp
    def deletepos(self,pos):
        i=0
        temp=self.head
        while(i!=pos-1 and temp.next!=None):
            temp=temp.next
            i+=1
            if(temp.next==None):
                break
        if(temp.next.next==None):
            temp.next=None
        else:
            temp.next=temp.next.next
            r=temp
if __name__:
    list1=linkedlist()
```

```
    option=1
    while(option!=0):
        print("select the option")
        print("0.Exit 1.Insertionatbegin 2.Display 3.Insertionatend 4.Insertatpos 5.Insertaftele\n6.search 7.deletebegin 8.deleteatend 9.d
        option=int(input())
        if(option==1):
            value=int(input("Enter value:"))
            list1.insertionatbegin(value)
        if(option==2):
            list1.printing()
        if(option==3):
            value=int(input("Enter value:"))
            list1.insertionatend(value)
        if(option==4):
            value=int(input("Enter value:"))
            pos=int(input("Enter pos:"))
            list1.insertat(value,pos)
        if(option==5):
            ele=int(input("enter element:"))
            value=int(input("Enter value:"))
            if list1.search(ele):
                list1.insertaftele(ele,value)
            else:
                print("Not possible")
        if(option==6):
            ele=int(input("enter element:"))
            if list1.search(ele):
                print("Found")
            else:
                print("Not found")
        if(option==7):
            list1.deletebegin()
        if(option==8):
            list1.deleteend()
        if option==9:
            pos=int(input("Enter position:"))
            if pos==0:
                list1.deletebegin()
            else:
                list1.deletepos(pos)
```

```
select the option
0.Exit 1.Insertionatbegin 2.Display 3.Insertionatend 4.Insertatpos 5.Insertaftele
6.search 7.deletebegin 8.deleteatend 9.deletepos 10.reverse: 3
Enter value:12
select the option
0.Exit 1.Insertionatbegin 2.Display 3.Insertionatend 4.Insertatpos 5.Insertaftele
6.search 7.deletebegin 8.deleteatend 9.deletepos 10.reverse: 3
Enter value:45
select the option
0.Exit 1.Insertionatbegin 2.Display 3.Insertionatend 4.Insertatpos 5.Insertaftele
6.search 7.deletebegin 8.deleteatend 9.deletepos 10.reverse: 3
Enter value:89
select the option
0.Exit 1.Insertionatbegin 2.Display 3.Insertionatend 4.Insertatpos 5.Insertaftele
6.search 7.deletebegin 8.deleteatend 9.deletepos 10.reverse: 2
---data----
12 45 89
---data----
select the option
0.Exit 1.Insertionatbegin 2.Display 3.Insertionatend 4.Insertatpos 5.Insertaftele
6.search 7.deletebegin 8.deleteatend 9.deletepos 10.reverse: 6
enter element:45
Found
select the option
0.Exit 1.Insertionatbegin 2.Display 3.Insertionatend 4.Insertatpos 5.Insertaftele
6.search 7.deletebegin 8.deleteatend 9.deletepos 10.reverse: 10
select the option
0.Exit 1.Insertionatbegin 2.Display 3.Insertionatend 4.Insertatpos 5.Insertaftele
6.search 7.deletebegin 8.deleteatend 9.deletepos 10.reverse: 0
```

In [2]:

```
class node:
    def _init_(self,value):
        self.data=value
        self.prev=None
        self.next=None
class Dlist:
    def _init_(self):
        self.head=self.tail=None
    def insertatbegin(self,value):
        newnode=node(value)
        if self.head is None:
            self.head=self.tail=newnode
        else:
            newnode.next=self.head
            self.head.prev=newnode
            self.head=newnode
    '''def insertatend(self,value):
        newnode=node(value)
        self.tail.next=new
        new.prev=tail
        new.data=value
        new.next=None
        tail=new'''
    def display(self):
        temp=self.head
        while temp is not None:
            print(temp.data,end="->")
            temp=temp.next
    #def insertatpos(self,value,pos):


list1=Dlist()
option=1
while(option):
    print("select operation")
    print("1.insert at start pos")
    print("2 is display")
    print("3 is insert at end")
    print("4 is insert at pos")
    print("5 is searching")
    print("6 is delete at begin")
    print("7 is delete at end")
    print("8 is delete at pos")
    print("0.exit")
    option=int(input())
    if(option==1):
        print("enter value")
        value=int(input())
        list1.insertatbegin(value)
    if option==2:
        list1.display()
    if option==3:
        print("enter value")
        value=int(input())
        list1.insertatend(value)
    if option==4:
        print("enter pos")
        pos=int(input())
        print("enter value")
        value=int(input())
        list1.insertatpos(value,pos)
    if option==5:
        print("enter searching element")
        key=int(input())
        list1.searching(key)
    if option==6:
        list1.deleteatbegin()
    if option==7:
        list1.deleteatend()
    if option==8:
        print("enter position to delete")
        pos=int(input())
        list1.deleteatpos(pos)
```

```
select operation
1.insert at start pos
2 is display
3 is insert at end
4 is insert at pos
5 is searching
6 is delete at begin
7 is delete at end
8 is delete at pos
0.exit
0
```

infix to prefix expression

.reverse the expr .if the input is an operand then print it .if the input is ')' push into stack .if the input is an operator check if the stack is empty or contains ')' on top of stack then push into stack .if the input is '('.pop the stack and print the operators until the ')' is found .if the incoming operator has higher precedence than the top of stack then push on top of stack .if the incoming operator has equal precedence then use associativity rule .if associativity is from left to right then pop and print

*ab+(c^d)e/f/gh+i i+h*g/f/e*)d^c(+b*a +*ab+*^cd/e/f*ghi

In [ ]:

```
*+-++*44111685
58611144*++-+*
```

In [5]:

```
Operators = set(['+', '-', '*', '/', '(', ')', '^'])  # collection of Operators

Priority = {'+':1, '-':1, '*':2, '/':2, '^':3} # dictionary having priorities of Operators

def infixToPostfix(expression):

    stack = [] # initialization of empty stack

    output = ''


    for character in expression:

        if character not in Operators:  # if an operand append in postfix expression

            output+= character

        elif character=='(':  # else Operators push onto stack

            stack.append('(')

        elif character==')':

            while stack and stack[-1]!= '(':

                output+=stack.pop()

            stack.pop()

        else:

            while stack and stack[-1]!='(' and Priority[character]<=Priority[stack[-1]]:

                output+=stack.pop()

            stack.append(character)

    while stack:

        output+=stack.pop()

    return output


expression = input('Enter infix expression ')

print('infix notation: ',expression)

print('postfix notation: ',infixToPostfix(expression))
```

```
Enter infix expression a+b
infix notation:  a+b
postfix notation:  ab+
```

In [11]:

```
exp=input("Enter expression:")
stack=[]
postfix=""
Priority = {'+':1,'-':1,'*':2,'/':2,'^':3}
op={'(','+','-','*','/',')'}
for i in range(len(exp)):
    if exp[i] not in op:
        postfix+=exp[i]
    else:
        if exp[i]=='(':
            stack.append('(')
        if exp[i]==')':
            x=stack.pop()
            while x!='(':
                postfix+=x
                x=stack.pop()
        else:
            while Priority[i]<=Priority[stack[-1]] and len(stack)!=0 and stack[-1]!='(':
                postfix+=stack.pop()
            stack.append(i)
while stack:
    postfix+=stack.pop()
print(postfix)
```

```
Enter expression:a+b

---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-11-c11f0b49f90b> in <module>
     16                 x=stack.pop()
     17         else:
---> 18             while Priority[i]<=Priority[stack[-1]] and len(stack)!=0 and stack[-1]!='(':
     19                 postfix+=stack.pop()
     20             stack.append(i)

KeyError: 1
```

In [2]:

```
stack=[]
def push(v):
    if len(stack)==5:
        print("Stack overflow")
    else:
        stack.append(v)
def pop():
    if stack==[]:
        print("stack underflow")
    else:
        print(stack.pop())
def peek():
    if stack==[]:
        print("stack underflow")
    else:
        print(stack[-1])
def display():
    if stack==[]:
        print("stack underflow")
    else:
        print(stack)

op=1
while op!=0:
    print("0.exit 1.push 2.pop 3.peek 4.display")
    op=int(input("Enter your option:"))
    if op==1:
        v=int(input("Enter value:"))
        push(v)
    if op==2:
        pop()
    if op==3:
        peek()
    if op==4:
        display()
```

```
0.exit 1.push 2.pop 3.peek 4.display
Enter your option:1
Enter value:12
0.exit 1.push 2.pop 3.peek 4.display
Enter your option:1
Enter value:56
0.exit 1.push 2.pop 3.peek 4.display
Enter your option:4
[56, 12]
0.exit 1.push 2.pop 3.peek 4.display
Enter your option:3
56
0.exit 1.push 2.pop 3.peek 4.display
Enter your option:2
56
0.exit 1.push 2.pop 3.peek 4.display
Enter your option:4
[12]
0.exit 1.push 2.pop 3.peek 4.display
Enter your option:2
12
0.exit 1.push 2.pop 3.peek 4.display
Enter your option:2
stack underflow
0.exit 1.push 2.pop 3.peek 4.display
Enter your option:1
Enter value:12
0.exit 1.push 2.pop 3.peek 4.display
Enter your option:3
12
0.exit 1.push 2.pop 3.peek 4.display
Enter your option:1
Enter value:34
0.exit 1.push 2.pop 3.peek 4.display
Enter your option:3
34
0.exit 1.push 2.pop 3.peek 4.display
Enter your option:4
[34, 12]
0.exit 1.push 2.pop 3.peek 4.display
Enter your option:1
Enter value:45
0.exit 1.push 2.pop 3.peek 4.display
Enter your option:1
Enter value:67
0.exit 1.push 2.pop 3.peek 4.display
Enter your option:1
Enter value:43
0.exit 1.push 2.pop 3.peek 4.display
Enter your option:1
Enter value:89
Stack overflow
0.exit 1.push 2.pop 3.peek 4.display
Enter your option:4
[43, 67, 45, 34, 12]
0.exit 1.push 2.pop 3.peek 4.display
Enter your option:3
43
0.exit 1.push 2.pop 3.peek 4.display
Enter your option:2
43
0.exit 1.push 2.pop 3.peek 4.display
Enter your option:2
67
0.exit 1.push 2.pop 3.peek 4.display
Enter your option:2
45
0.exit 1.push 2.pop 3.peek 4.display
Enter your option:2
34
0.exit 1.push 2.pop 3.peek 4.display
Enter your option:2
12
0.exit 1.push 2.pop 3.peek 4.display
Enter your option:2
stack underflow
0.exit 1.push 2.pop 3.peek 4.display
Enter your option:2
stack underflow
0.exit 1.push 2.pop 3.peek 4.display
Enter your option:2
stack underflow
0.exit 1.push 2.pop 3.peek 4.display
Enter your option:2
stack underflow
0.exit 1.push 2.pop 3.peek 4.display
Enter your option:2
stack underflow
0.exit 1.push 2.pop 3.peek 4.display
Enter your option:0
```

1. Full Binary Tree

A full Binary tree is a special type of binary tree in which every parent node/internal node has either two or no children.

2. Perfect Binary Tree

A perfect binary tree is a type of binary tree in which every internal node has exactly two child nodes and all the leaf nodes are at the same level.

3. Complete Binary Tree

A complete binary tree is just like a full binary tree, but with two major differences

Every level must be completely filled
All the leaf elements must lean towards the left.
The last leaf element might not have a right sibling i.e. a complete binary tree doesn't have to be a full binary tree.

4. Degenerate or Pathological Tree

A degenerate or pathological tree is the tree having a single child either left or right.

5. Skewed Binary Tree

A skewed binary tree is a pathological/degenerate tree in which the tree is either dominated by the left nodes or the right nodes. Thus, there are two types of skewed binary tree: left-skewed binary tree and right-skewed binary tree.

6. Balanced Binary Tree

It is a type of binary tree in which the difference between the height of the left and the right subtree for each node is either 0 or 1.

In [2]:

```
#Binary tree
class node:
    def __init__(self,data):
        self.data=data
        self.left=None
        self.right=None
class tree:
    def __init__(self):
        self.root=None
    def create(self,v):
        newnode=node(v)
        if self.root==None:
            self.root=newnode
        else:
            self.insert(self.root,newnode)
    def insert(self,root,newnode):
        print("0.Left 1.Right:",end="")
        x=int(input())
        if x==0:
            if root.left==None:
                root.left=newnode
            else:
                self.insert(root.left,newnode)
        else:
            if root.right==None:
                root.right=newnode
            else:
                self.insert(root.right,newnode)
    def display(self):
        print("1.preorder 2.inorder 3.postorder",end="")
        ch=int(input())
        if ch==1:
            self.preorder(self.root)
        elif ch==2:
            self.inorder(self.root)
        else:
            self.postorder(self.root)
        print()
    def inorder(self,root):
        if root!=None:
            self.inorder(root.left)
            print(root.data,end=" ")
            self.inorder(root.right)
    def preorder(self,root):
        if root!=None:
            print(root.data,end=" ")
            self.inorder(root.left)
            self.inorder(root.right)
    def postorder(self,root):
        if root!=None:
            self.inorder(root.left)
            self.inorder(root.right)
            print(root.data,end=" ")
t=tree()
op=1
while op!=0:
    print("1.insert 2.display:",end="")
    op=int(input())
    if op==1:
        v=int(input("Enter value:"))
        t.create(v)
    if op==2:
        t.display()
```

```
1.insert 2.display:1
Enter value:12
1.insert 2.display:1
Enter value:34
0.Left 1.Right:0
1.insert 2.display:1
Enter value:45
0.Left 1.Right:1
1.insert 2.display:2
1.preorder 2.inorder 3.postorder2
34 12 45
1.insert 2.display:0
```

In [5]:

```
#Binary Search Tree
class node:
    def __init__(self,data):
        self.data=data
        self.left=None
        self.right=None
class bst:
    def __init__(self):
        self.root=None
    def create(self,v):
        newnode=node(v)
        if self.root==None:
            self.root=newnode
        else:
            self.insert(self.root,newnode)
    def insert(self,root,newnode):
        if newnode.data<root.data:
            if root.left==None:
                root.left=newnode
            else:
                self.insert(root.left,newnode)
        elif newnode.data>root.data:
            if root.right==None:
                root.right=newnode
            else:
                self.insert(root.right,newnode)
    def search(self,key):
        if self.root==None:
            print("No Elements")
        else:
            self.sr(self.root,key)
    def sr(self,root,key):
        if root.data==key:
            return 1
        elif key<root.data:
            self.sr(root.left,key)
        elif key>root.data:
            self.sr(root.right,key)
    def display(self):
        print("Root:",self.root.data)
        print("1.preorder 2.inorder 3.postorder:",end="")
        ch=int(input())
        if ch==1:
            self.preorder(self.root)
        elif ch==2:
            self.inorder(self.root)
        else:
            self.postorder(self.root)
    def inorder(self,root):
        if root!=None:
            self.inorder(root.left)
            print(root.data)
            self.inorder(root.right)
    def preorder(self,root):
        if root!=None:
            print(root.data)
            self.preorder(root.left)
            self.preorder(root.right)
    def postorder(self,root):
        if root!=None:
            self.postorder(root.left)
            self.postorder(root.right)
            print(root.data)
    def minimum(self):
        if self.root==None:
            return (self.root.data)
        else:
            self.minm(self.root)
    def minm(self,root):
        if root.left==None:
            return (root.data)
        else:
            self.minm(root.left)
    def maximum(self):
        if self.root==None:
            return (self.root.data)
        else:
            self.maxm(self.root)
    def maxm(self,root):
        if root.right==None:
            return (root.data)
        else:
            self.maxm(root.right)
    def delete(self,key):
        self.d(self.root,key)
    def d(self,root,key):
        if(root==None):
            return root
        elif key<root.data:
            root.left=d(root.left,key)
```

```
        elif key>root.data:
            root.right=d(root.right,key)
        else:
            if root.left==None and root.right==None:
                root=None
            elif root.right==None:
                root=root.left
            elif root.left==None:
                root=root.right
            else:
                t=self.minm(root.right)
                root.data=t.data
                root.right=d(root.right,t.data)
        return root
t=bst()
op=1
while op!=0:
    print("1.insert 2.display 3.Minimum 4.Maximum 5.Delete 6.search:",end="")
    op=int(input())
    if op==1:
        v=int(input("Enter value:"))
        t.create(v)
    if op==2:
        t.display()
    if op==3:
        v=t.minimum()
        print(v)
    if op==4:
        v=t.maximum()
        if v:
            print(v)
        else:
            print("Not found")
    if op==5:
        key=int(input("Enter key value:"))
        t.delete(key)
    if op==6:
        key=int(input("Enter search element:"))
        v=t.search(key)
        if v:
            print("Found")
        else:
            print("Not found")
```

```
1.insert 2.display 3.Minimum 4.Maximum 5.Delete 6.search:1
Enter value:12
1.insert 2.display 3.Minimum 4.Maximum 5.Delete 6.search:1
Enter value:2
1.insert 2.display 3.Minimum 4.Maximum 5.Delete 6.search:2
Root: 12
1.preorder 2.inorder 3.postorder:2
2
12
1.insert 2.display 3.Minimum 4.Maximum 5.Delete 6.search:3
None
1.insert 2.display 3.Minimum 4.Maximum 5.Delete 6.search:6
Enter search element:2
Not found
1.insert 2.display 3.Minimum 4.Maximum 5.Delete 6.search:0
```

In [1]:

```
class BSTNode:
    def __init__(self, val=None):
        self.left = None
        self.right = None
        self.val = val

    def insert(self, val):
        if not self.val:
            self.val = val
            return

        if self.val == val:
            return

        if val < self.val:
            if self.left:
                self.left.insert(val)
                return
            self.left = BSTNode(val)
            return

        if self.right:
            self.right.insert(val)
            return
        self.right = BSTNode(val)

    def get_min(self):
        current = self
        while current.left is not None:
            current = current.left
        return current.val

    def get_max(self):
        current = self
        while current.right is not None:
            current = current.right
        return current.val

    def delete(self, val):
        if self == None:
            return self
        if val < self.val:
            if self.left:
                self.left = self.left.delete(val)
            return self
        if val > self.val:
            if self.right:
                self.right = self.right.delete(val)
            return self
        if self.right == None:
            return self.left
        if self.left == None:
            return self.right
        min_larger_node = self.right
        while min_larger_node.left:
            min_larger_node = min_larger_node.left
        self.val = min_larger_node.val
        self.right = self.right.delete(min_larger_node.val)
        return self

    def exists(self, val):
        if val == self.val:
            return True

        if val < self.val:
            if self.left == None:
                return False
            return self.left.exists(val)

        if self.right == None:
            return False
        return self.right.exists(val)

    def preorder(self, vals):
        if self.val is not None:
            vals.append(self.val)
        if self.left is not None:
            self.left.preorder(vals)
        if self.right is not None:
            self.right.preorder(vals)
        return vals

    def inorder(self, vals):
        if self.left is not None:
            self.left.inorder(vals)
        if self.val is not None:
            vals.append(self.val)
        if self.right is not None:
            self.right.inorder(vals)
        return vals
```

```
    def postorder(self, vals):
        if self.left is not None:
            self.left.postorder(vals)
        if self.right is not None:
            self.right.postorder(vals)
        if self.val is not None:
            vals.append(self.val)
        return vals
def main():
    nums = [12, 6, 18, 19, 21, 11, 3, 5, 4, 24, 18]
    bst = BSTNode()
    for num in nums:
        bst.insert(num)
    print("preorder:")
    print(bst.preorder([]))
    print("#")

    print("postorder:")
    print(bst.postorder([]))
    print("#")

    print("inorder:")
    print(bst.inorder([]))
    print("#")

    nums = [2, 6, 20]
    print("deleting " + str(nums))
    for num in nums:
        bst.delete(num)
    print("#")

    print("4 exists:")
    print(bst.exists(4))
    print("2 exists:")
    print(bst.exists(2))
    print("12 exists:")
    print(bst.exists(12))
    print("18 exists:")
    print(bst.exists(18))
```

In [ ]:

In [ ]: