class node:

  def \_\_init\_\_(self,v):

    self.left=None

    self.right=None

    self.value=v

#insertion

def insert(t,v):

  #is tree empty

  if t is None:

    t=node(v)

    return t

  #is the insertion value and root equal

  if(t.value==v):

    return t

  #insertion value less than root

  if (v<t.value):

    #is left subtree empty

    if(t.left is None):

      t.left=node(v)

    else:

      insert(t.left,v)

      return t

  #insertion value greater than root

  #is right subtree empty

  elif(t.right is None):

    t.right=node(v)

  else:

    insert(t.right,v)

  return t

#deletion

def deletion(t, v):

    #is tree empty

    if t is None:

        return t

    # deletion value smaller than root

    if v < t.value:

        t.left = deletion(t.left, v)

    # deletion value greater than root

    elif(v > t.value):

        t.right = deletion(t.right, v)

    # deletion value which is root

    else:

        # Node with only one child or no child

        if t.left is None:

            temp = t.right

            t = None

            return temp

        elif t.right is None:

            temp = t.left

            t = None

            return temp

        # Node with two children:

        temp = minval(t.right)

        t.value = temp.value

        # Delete the inorder successor

        t.right = deletion(t.right, temp.value)

    return t

#minimum value

def minval(t):

  while(t.left is not None):

    t=t.left

  return t

#traversal

def in\_traverse(t):

  if t:

    in\_traverse(t.left)

    print(t.value)

    in\_traverse(t.right)

#search

def find(t,v):

  if t is None:

    return False

  if t.value == v:

    return True

  if v<t.value:

    return find(t.left,v)

  else:

    return find(t.right,v)

n=int(input("enter the no.of nodes:"))

y=int(input("enter the value:"))

x=node(y)

for i in range(n-1):

  y=int(input("enter the value:"))

  x=insert(x,y)

in\_traverse(x)

key=int(input("enter the number to be deleted:"))

x=deletion(x,key)

in\_traverse(x)

k=int(input("enter number to be searched:"))

print(find(x,k))

git clone https://github.com/username/repository-name

**def** **min\_heapify**(A,k):

l = left(k)

r = right(k)

**if** l < len(A) **and** A[l] < A[k]:

smallest = l

**else**:

smallest = k

**if** r < len(A) **and** A[r] < A[smallest]:

smallest = r

**if** smallest != k:

A[k], A[smallest] = A[smallest], A[k]

min\_heapify(A, smallest)

**def** **left**(k):

**return** **2** \* k + **1**

**def** **right**(k):

**return** **2** \* k + **2**

**def** **build\_min\_heap**(A):

n = int((len(A)//**2**)-**1**)

**for** k **in** range(n, -**1**, -**1**):

min\_heapify(A,k)

A = [**3**,**9**,**2**,**1**,**4**,**5**]

build\_min\_heap(A)

**print**(A)