1) Write a program to find all pairs of an integer array whose sum is equal to a given number?

In [2]:
```
def getPairsCount(arr, n, sum):
    count = 0
    for i in range(0, n):
        for j in range(i + 1, n):
            if arr[i] + arr[j] == sum:
                count += 1

    return count
arr = [1, 5, 7, -1, 5]
n = len(arr)
sum = 6
print("Count of pairs is",getPairsCount(arr, n, sum))
```

```
Count of pairs is 3
```

2) Write a program to reverse an array in place? In place means you cannot create a new array. You have to update the original array.

In [3]:
```
def reverseList(A, start, end):
    while start < end:
        A[start], A[end] = A[end], A[start]
        start += 1
        end -= 1
A = [1, 2, 3, 4, 5, 6]
print(A)
reverseList(A, 0, 5)
print("Reversed list is")
print(A)
```

```
[1, 2, 3, 4, 5, 6]
Reversed list is
[6, 5, 4, 3, 2, 1]
```

3) Write a program to check if two strings are a rotation of each other?

In [5]:
```
def checkRotation(s1, s2):
    temp = ''
    if len(s1) != len(s2):
        return False

    # concatenating both strings
    temp = s1 + s1
    if s2 in temp:
        return True #returning true if 2nd string is present in concatenated string
    else:
        return False
# Driver
string1 = "CRG"
string2 = "GCR"

if checkRotation(string1, string2):
    print("Given Strings are rotations of each other.")
else:
    print("Given Strings are not rotations of each other.")
```

```
Given Strings are rotations of each other.
```

4) Write a program to print the first non-repeated character from a string?

In [8]:
```
# using while loop
```

```
s = "charanrajgowda"
while s != "":
   slen0 = len(s)
   ch = s[0]
   s = s.replace(ch, "")
   slen1 = len(s)
   if slen1 == slen0-1:
      print ("First non-repeating character is: ",ch)
      break;
   else:
    print ("No Unique Character Found!")
```

```
First non-repeating character is:  c
```

In [10]:

```
#using HashMap
NO_OF_CHARS = 256
def getCharCountArray(string):
    count = [0] * NO_OF_CHARS
    for i in string:
        count[ord(i)]+= 1
    return count

def firstNonRepeating(string):
    count = getCharCountArray(string)
    index = -1
    k = 0

    for i in string:
        if count[ord(i)] == 1:
            index = k
            break
        k += 1

    return index

# Driver
string = "charanrajgowda"
index = firstNonRepeating(string)
if index == 1:
    print("Either all characters are repeating or string is empty")
else:
    print("First non-repeating character is " + string[index])
```

```
First non-repeating character is c
```

5) Read about the Tower of Hanoi algorithm. Write a program to implement it.

In [13]:

```
def TowerOfHanoi(n , from_rod, to_rod, aux_rod):
    if n == 1:
        print("Move disk 1 from rod",from_rod,"to rod",to_rod)
        return
    TowerOfHanoi(n-1, from_rod, aux_rod, to_rod)
    print("Move disk",n,"from rod",from_rod,"to rod",to_rod)
    TowerOfHanoi(n-1, aux_rod, to_rod, from_rod)

# Driver code
n = 4
TowerOfHanoi(n, 'A', 'C', 'B')
```

```
Move disk 1 from rod A to rod B
Move disk 2 from rod A to rod C
Move disk 1 from rod B to rod C
Move disk 3 from rod A to rod B
Move disk 1 from rod C to rod A
```

```
Move disk 2 from rod C to rod B
Move disk 1 from rod A to rod B
Move disk 4 from rod A to rod C
Move disk 1 from rod B to rod C
Move disk 2 from rod B to rod A
Move disk 1 from rod C to rod A
Move disk 3 from rod B to rod C
Move disk 1 from rod A to rod B
Move disk 2 from rod A to rod C
Move disk 1 from rod B to rod C
```

6) Read about infix, prefix, and postfix expressions. Write a program to convert postfix to prefix expression.

In [14]:

```
def isOperator(x):
    if x == "+":
        return True
    if x == "-":
        return True
    if x == "/":
        return True
    if x == "*":
        return True
    return False

#postfix to Prefix
def postToPre(post_exp):
    s = []
    # length of expression
    length = len(post_exp)
    # reading from right to left
    for i in range(length):
        # check if symbol is operator
        if (isOperator(post_exp[i])):
            # pop two operands from stack
            op1 = s[-1]
            s.pop()
            op2 = s[-1]
            s.pop()
            # concat the operands and operator
            temp = post_exp[i] + op2 + op1
            # Push string temp back to stack
            s.append(temp)
        # if symbol is an operand
        else:
            # push the operand to the stack
            s.append(post_exp[i])
    ans = ""
    for i in s:
        ans += i
    return ans

# Driver Code
if __name__ == "__main__":

    post_exp = "AB+CD-"

    # Function call
    print("Prefix : ", postToPre(post_exp))
```

```
Prefix :  +AB-CD
```

7) Write a program to convert prefix expression to infix expression

In [15]:

```
def prefixToInfix(prefix):
    stack = []

    # read prefix in reverse order
    i = len(prefix) - 1
    while i >= 0:
        if not isOperator(prefix[i]):

            # symbol is operand
            stack.append(prefix[i])
            i -= 1
        else:

            # symbol is operator
            str = "(" + stack.pop() + prefix[i] + stack.pop() + ")"
            stack.append(str)
            i -= 1

    return stack.pop()

def isOperator(c):
    if c == "*" or c == "+" or c == "-" or c == "/" or c == "^" or c == "(" or c ==
        return True
    else:
        return False

# Driver code
if __name__=="__main__":
    str = "*-A/BC-/AKL"
    print(prefixToInfix(str))
```

```
((A-(B/C))*((A/K)-L))
```

8) Write a program to check if all the brackets are closed in a given code snippet.

In [16]:

```
def areBracketsBalanced(expr):
    stack = []

    # Traversing the Expression
    for char in expr:
        if char in ["(", "{", "["]:

            # Push the element in the stack
            stack.append(char)
        else:

            # IF current character is not opening
            # bracket, then it must be closing.
            # So stack cannot be empty at this point.
            if not stack:
                return False
            current_char = stack.pop()
            if current_char == '(':
                if char != ")":
                    return False
            if current_char == '{':
                if char != "}":
                    return False
            if current_char == '[':
                if char != "]":
                    return False

    # Check Empty Stack
    if stack:
```

```
        return False
    return True


# Driver Code
if __name__ == "__main__":
    expr = "{()}[]"

    # Function call
    if areBracketsBalanced(expr):
        print("Balanced")
    else:
        print("Not Balanced")
```

```
Balanced
```

9) Write a program to reverse a stack

In [21]:

```
class Stack:
    def __init__(self):
        self.items = []

    def is_empty(self):
        return self.items == []

    def push(self, data):
        self.items.append(data)

    def pop(self):
        return self.items.pop()

    def display(self):
        for data in reversed(self.items):
            print(data)
def insert_at_bottom(s, data):
    if s.is_empty():
        s.push(data)
    else:
        popped = s.pop()
        insert_at_bottom(s, data)
        s.push(popped)


def reverse_stack(s):
    if not s.is_empty():
        popped = s.pop()
        reverse_stack(s)
        insert_at_bottom(s, popped)


s = Stack()
data_list = input('Please enter the elements to push: ').split()
for data in data_list:
    s.push(int(data))
print('The stack:')
s.display()
reverse_stack(s)
print('After reversing:')
s.display()
```

```
Please enter the elements to push: 123
The stack:
123
After reversing:
123
```

10) Write a program to find the smallest number using a stack.

In [22]:

```
from collections import deque

class MinStack:
    def __init__(self):
        self.s = deque()
        self.min = None

    def push(self, x):
        if not self.s:
            self.s.append(x)
            self.min = x
        elif x > self.min:
            self.s.append(x)
        else:
            self.s.append(2*x - self.min)
            self.min = x

    def pop(self):

        if not self.s:
            self.print("Stack underflow!!")

        top = self.s[-1]
        if top < self.min:
            self.min = 2*self.min - top
        self.s.pop()

    def minimum(self):

        return self.min

if __name__ == '__main__':

    s = MinStack()

    s.push(6)
    print(s.minimum())

    s.push(7)
    print(s.minimum())

    s.push(5)
    print(s.minimum())

    s.push(3)
    print(s.minimum())

    s.pop()
    print(s.minimum())

    s.pop()
    print(s.minimum())
```

```
6
6
5
3
5
6
```